



## V. Tasks

This section describes tasks and scripted tasks and sets.

---

## Introduction to Tasks

VR-Forces simulation objects can execute a variety of movement and tactical tasks. These tasks can be assigned in plans, through the Task menu, or in the Command Panel. Tasks assigned outside of a plan are called independent tasks.

Some tasks are mutually exclusive with other tasks, for example, a simulation object cannot execute two different movement tasks at the same time. Other tasks, such as Send Text Message, can run concurrently with movement tasks. When you give a task to a simulation object, if the new task is mutually exclusive with the current task, the simulation object immediately stops the current task and begins the new one. If the new task is not mutually exclusive with the current task, the simulation object executes the new task and continues with the previous task. For information about how to configure mutually exclusive tasks, see "2.11. Configuring Task Execution Rules" on page 36 .

If a simulation object has a plan, when you give it an independent task, it abandons the rest of the plan.

Independent tasks are useful for assigning tasks to simulation objects without the need to edit a plan. You can assign them interactively, so simulation participants can respond immediately to events in a simulation by assigning new tasks to a simulation object as they might have to do in the field. General issues for assigning tasks are described in detail in "Chapter 2. Assigning Tasks" on page 5. Specific task procedures are in the succeeding chapters.

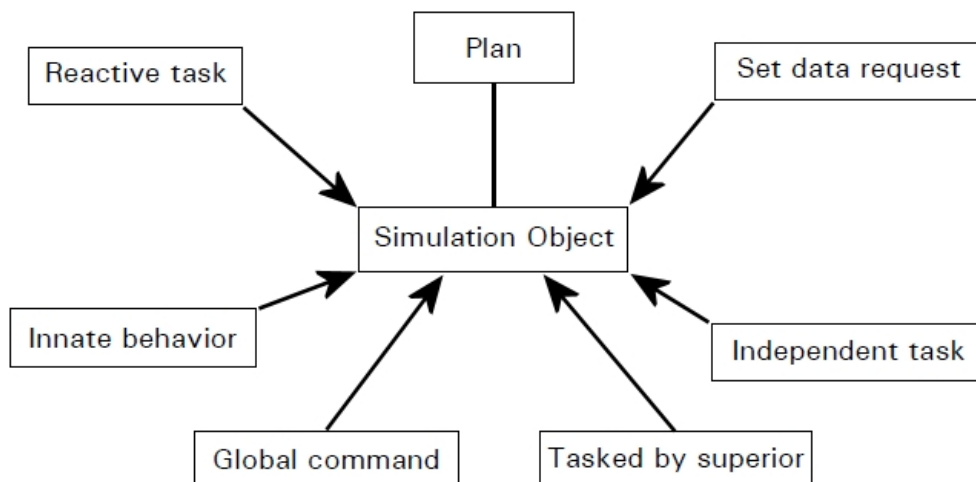
Many tasks are implemented as part of the VR-Forces code (C++ tasks) and cannot be added to or changed by end-users. Other tasks, called scripted tasks, are implemented using Lua scripts. You can edit scripted tasks and write new ones. For details, see "Chapter 9. Creating Scripted Tasks and Sets" on page 189.

VR-Forces has a special type of scripted task, called a reactive task, that is not explicitly assigned to simulation objects. Reactive tasks monitor the simulation and run only if certain conditions are met. For information about reactive tasks, see "2.5. Reactive Tasks" on page 19.

A third type of scripted task is the background process. Background processes run continuously to handle certain types of ongoing aggregate-level activities, such as consuming or receiving supplies, calculating a unit's footprint, and automatically attacking opposing forces. For information about background processes, see "10.11. Background Processes" on page 257.

Figure 1 illustrates the many ways that a simulation object can have its task, behavior, and state changed.

**Figure 1. Inputs to simulation object actions**









## 2. Assigning Tasks

This chapter provides general details about how to assign tasks to simulation objects. For the procedures for assigning tasks, please see the rest of the chapters in this section.

|                                                                          |    |
|--------------------------------------------------------------------------|----|
| 2.1. Assigning Tasks to Simulation Objects .....                         | 6  |
| 2.1.1 The Command Panel .....                                            | 6  |
| 2.1.2 Task Procedures .....                                              | 8  |
| 2.1.3 C++ Tasks and Scripted Tasks .....                                 | 8  |
| 2.1.4 Concurrent Task Execution .....                                    | 9  |
| 2.1.5 How Do I Know Which Simulation Objects Can Execute a Task? .....   | 10 |
| 2.1.6 Escaping the Task Assignment Process .....                         | 10 |
| 2.1.7 Specifying Parameters for Tasks .....                              | 11 |
| 2.1.8 Viewing Task Status .....                                          | 12 |
| 2.1.9 Filtering the Object Selection Lists .....                         | 13 |
| 2.1.10 Skipping (Stopping) a Task .....                                  | 13 |
| 2.2. The Entity Control Panel .....                                      | 14 |
| 2.2.1 Specifying Entity Movement Using the Entity Control Panel .....    | 15 |
| 2.2.2 Using the Buttons in the Entity Control Panel .....                | 16 |
| 2.3. Using a Keyboard Shortcut to Assign a One-Click Task (or Set) ..... | 17 |
| 2.4. Assigning Tasks to Units .....                                      | 18 |
| 2.4.1 Convoy Tasks .....                                                 | 18 |
| 2.4.2 Independently Tasking Unit Members .....                           | 18 |
| 2.5. Reactive Tasks .....                                                | 19 |
| 2.5.1 Enabling Reactive Tasks .....                                      | 20 |
| 2.5.2 Disabling Reactive Tasks .....                                     | 21 |
| 2.5.3 Setting the Priority of a Reactive Task .....                      | 21 |
| 2.5.4 Managing Reactive Tasks .....                                      | 22 |
| 2.5.5 Canceling a Reactive Task .....                                    | 22 |
| 2.6. Using Behavior Sets to Manage Scripts .....                         | 23 |
| 2.6.1 Creating Behavior Sets .....                                       | 24 |

|                                                                         |    |
|-------------------------------------------------------------------------|----|
| 2.6.2 Editing Behavior Sets .....                                       | 25 |
| 2.6.3 Assigning a Behavior Set to a Force .....                         | 26 |
| 2.7. Entity Movement On Roads and Off Roads .....                       | 26 |
| 2.7.1 Road Driving Behavior .....                                       | 27 |
| 2.7.2 Specifying the Side of the Road to Drive On .....                 | 27 |
| 2.8. Fixed-Wing Entity Tasks and Behaviors .....                        | 28 |
| 2.8.1 Placement of Newly Created Fixed-Wing Entities .....              | 28 |
| 2.8.2 Fly Task Behavior Is Different from Move Task Behavior .....      | 28 |
| 2.8.3 Specifying and Maintaining Altitude for Fixed-Wing Entities ..... | 29 |
| 2.8.4 Fixed-Wing Entity Movement on the Ground and in the Air .....     | 29 |
| 2.8.5 How Fixed-Wing Entities Take Off .....                            | 30 |
| 2.8.6 How Fixed-Wing Entities Land .....                                | 31 |
| 2.9. Rotary-Wing Entity Tasks and Behaviors .....                       | 32 |
| 2.10. Suppressive Fire Tasks .....                                      | 33 |
| 2.10.1 Configuring Suppression Effects .....                            | 34 |
| 2.11. Configuring Task Execution Rules .....                            | 36 |
| 2.11.1 Configuring Action Categories .....                              | 36 |

## 2.1. Assigning Tasks to Simulation Objects

You can assign tasks to a simulation object as part of its plan, as part of a global plan, and independently of any plan. For conceptual information about tasks, see "Introduction to Tasks" on page 2. For information about how to add a task to a plan, see "Adding a Task or Set Data Request to a Plan".

You cannot assign tasks to non-VR-Forces simulation objects. However, you can use a remote simulation object as a parameter in a task, for example, follow or target a remote simulation object. For more information, see "Using Non-VR-Forces Simulation Objects in Plans".

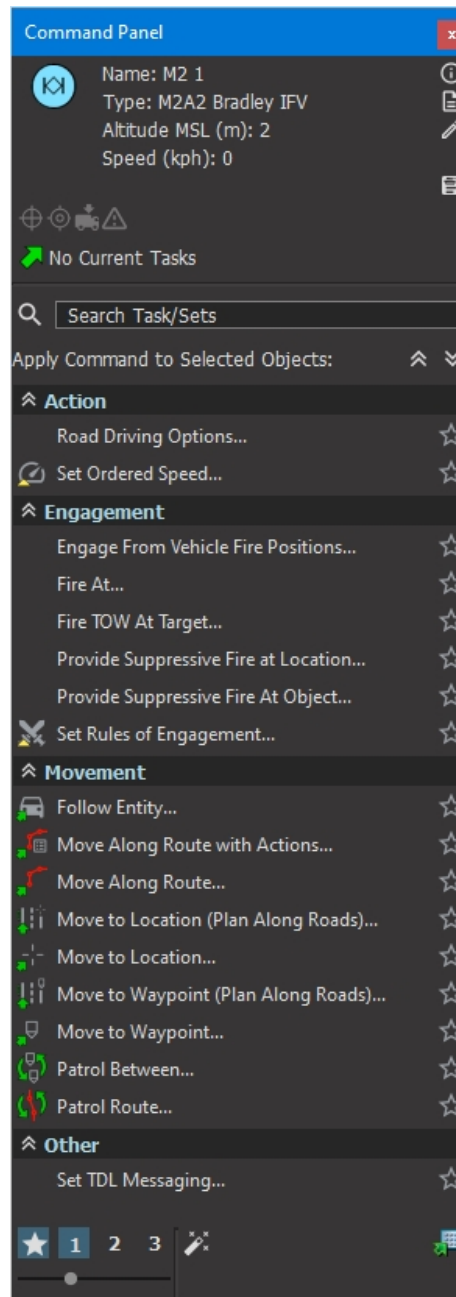
### 2.1.1 The Command Panel

The Command Panel brings together all of the tasks and sets available to a simulation object, along with basic status information, and presents them in a categorized menu that you can control to display more or fewer options. You can use it both to assign independent tasks during run time and to build plans.

By default, the Command Panel (Figure 2) opens on the right side of the VR-Forces window.

- To display the Command Panel if it does not open automatically—or to close it—choose **View > Command Panel**.
- To populate the Command Panel, select a simulation object in any observer mode or in the Objects List.

**Figure 2. The Command Panel populated with commands for an M2A2 Bradley IFV**



► Use the controls on the Command Panel to display more or fewer task and set options:

- Click the star (☆) next to a task or set to make it a favorite that is always visible.
- Click the star, 1, 2, or 3 at the bottom of the panel to display favorites (★), most common (1), advanced (2), or detailed (3) control commands.
- Click the magic wand (🪄) at the bottom of the panel to display options that are not possible in the real world (such as Set Location, which can move an entity to a distant location instantly).

- ▶ To create a plan for the selected simulation object or open the existing plan, click the plan icon (📄) near the top of the Command Panel. For instructions on creating and editing plans, see "Writing Plans".
- ▶ Click any command on the panel to view the parameters for that task or set data request.

See the subsections that follow for general information on assigning tasks. For details on particular tasks, see the remaining parts of this Tasks section: "Chapter 3. Movement Tasks for Entity-Level Scenarios" on page 39, "Chapter 4. Engagement Tasks for Entity-Level Scenarios" on page 75, "Chapter 5. Human and Crowd Tasks" on page 101, and so on.

 To see icons instead of text for the tasks and set data requests, choose **Settings > Application**, select the GUI Settings page, and then select the Show Command Panel Icons check box. You can then choose whether to view them as small or large icons.

### 2.1.2 Task Procedures

For the procedures for assigning tasks, see "Chapter 3. Movement Tasks for Entity-Level Scenarios" on page 39, "Chapter 4. Engagement Tasks for Entity-Level Scenarios" on page 75, "Chapter 5. Human and Crowd Tasks" on page 101, and "Chapter 7. Embarkation, Wait, Radio, and Other Tasks" on page 131.

If your installation uses the VR-Forces API to add additional tasks, consult with your software engineers for procedural information. See "2.8. Fixed-Wing Entity Tasks and Behaviors" on page 28 and "2.9. Rotary-Wing Entity Tasks and Behaviors" on page 32 for additional information about assigning tasks to these simulation object types.

The instructions for creating tasks direct you to use options on the Task menu. However, all the options available on the main menu are also available on context-sensitive menus and the Command Panel. A limited number of tasks are available on the Tasks Toolbar and on the Last Selected Object Panel.

### 2.1.3 C++ Tasks and Scripted Tasks

The VR-Forces application includes a basic set of tasks for movement, weapons fire, and other actions, that were developed using the VR-Forces Toolkit and the C++ API. System integrators often use the C++ API to add additional tasks before delivering VR-Forces to their customers. These tasks are called C++ tasks to distinguish them from another type of task, the scripted task. Scripted tasks, written in the Lua programming language, use the lower-level C++ tasks and the VR-Forces Lua API to build higher-order tasks for simulation objects. You do not need the VR-Forces Toolkit or a developers license to write scripted tasks.

To the VR-Forces end user, scripted tasks, most of which are listed on the Task menu, are indistinguishable from C++ tasks. For information about scripted tasks, see "Chapter 9. Creating Scripted Tasks and Sets" on page 189.

VR-Forces also supports two special categories of scripted tasks called reactive tasks and background processes. These tasks are not on the Task menu. Reactive tasks execute

when a condition becomes true. For details, see "2.5. Reactive Tasks" on page 19. Background processes run continuously. (For details, see "8.9. Background Process List" on page 188.)

### 2.1.4 Concurrent Task Execution

Some tasks can execute at the same time. Some are mutually exclusive. If you assign a simulation object a task that is mutually exclusive with its current task, the current task is abandoned. If you assign a task that is not mutually exclusive with the current task, it executes the new task while continuing to execute the previous task.

**i** When you assign a mutually exclusive independent task, VR-Forces displays a prompt that asks you to confirm interruption of the current task. You can disable this prompt if you always want a new task to interrupt the current task. For details, see "Enabling and Disabling the Task Confirmation Prompt" below.

C++ tasks are organized into these groups:

- Weapon
- Movement
- Radio
- Depth control

All tasks within a group are mutually exclusive among themselves. For example, all movement tasks are mutually exclusive. Simulation objects that are moving can usually execute a task from one of the other groups at the same time. For example, simulation objects can execute the various Send tasks while they are executing movement tasks. When you create a scripted task, you can specify which groups it conflicts with. For information about how to configure mutually exclusive tasks, see "2.11. Configuring Task Execution Rules" on page 36. For information about scripted tasks, see "Chapter 9. Creating Scripted Tasks and Sets" on page 189. For information about how to use concurrent tasks in plans, see "Using Concurrent Tasks in Plans".

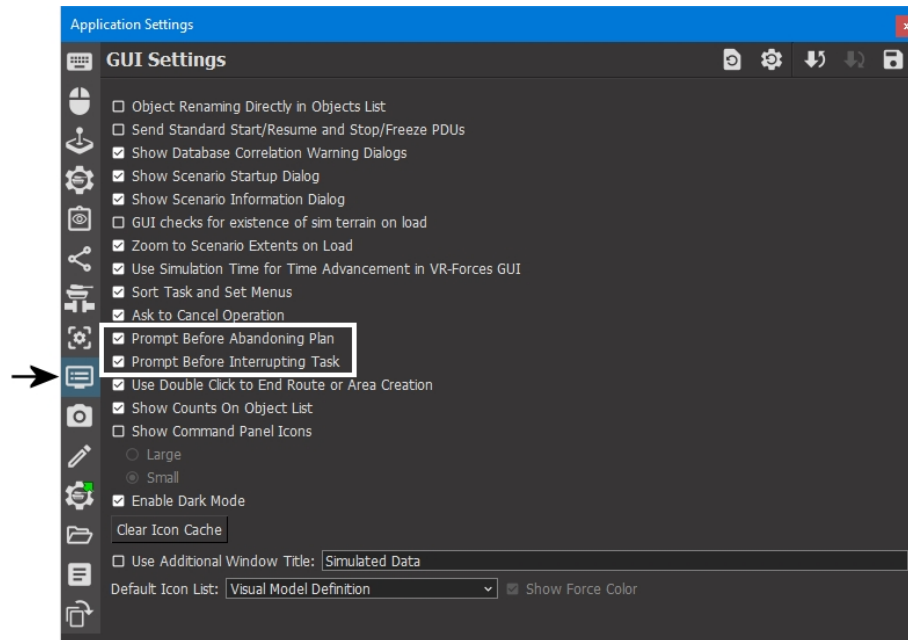
#### Enabling and Disabling the Task Confirmation Prompt

When you give a simulation object an independent task, if it is executing a task or a plan, VR-Forces informs you that the new task will override the task or plan and asks you to confirm the action. You can enable and disable the confirmation prompt.

##### ► To enable or disable the task override confirmation prompt:

1. Choose **Settings > Application**. The Application Settings dialog box opens.
2. Select the GUI Settings page (Figure 3).

Figure 3. GUI Settings page



3. To enable or disable the prompt for abandoning a plan, select or clear the Prompt Before Abandoning Plan check box.
4. To enable or disable the prompt for interrupting a task, select or clear the Prompt Before Interrupting Task check box.
5. Click Close.

### 2.1.5 How Do I Know Which Simulation Objects Can Execute a Task?

Some VR-Forces tasks apply to almost all simulation objects, such as Radio tasks and basic movement tasks. Others are valid only for particular platforms or even particular entity types. Sometimes it is obvious which entity types can execute a task. For example, you would expect that Rotary Wing Land only applies to rotary-wing entities. Similarly, Raise Periscope applies to submarines. However, for other tasks, it is not always evident which simulation object can execute a task.

Simulation objects can execute tasks for which they have an appropriate controller. They access most task controllers through systems. Therefore, simulation objects can execute the tasks supported by their various movement, weapon, sensor, and other systems. For example, to deploy naval mines, an entity needs a Naval Mine Deployment system.

To see which systems a simulation object has, and thereby infer which tasks it can execute, you can examine the simulation object's systems in the Simulation Object Editor. You can also refer to the *VR-Forces Entity Model Catalog*, which lists the systems for each simulation object configured in the Simulation Object Editor. "Systems and System Usage" lists each system and the scripted tasks associated with it.

### 2.1.6 Escaping the Task Assignment Process

You can escape the task assignment process at any point before you complete it.

- To escape the task assignment process, in the task parameters, click Cancel.

### 2.1.7 Specifying Parameters for Tasks

In VR-Forces, you specify task parameters in fields, dialog boxes, or by selecting tactical graphic or a simulation object on the terrain. When you select the appropriate object, the selection is reflected in the task parameters.

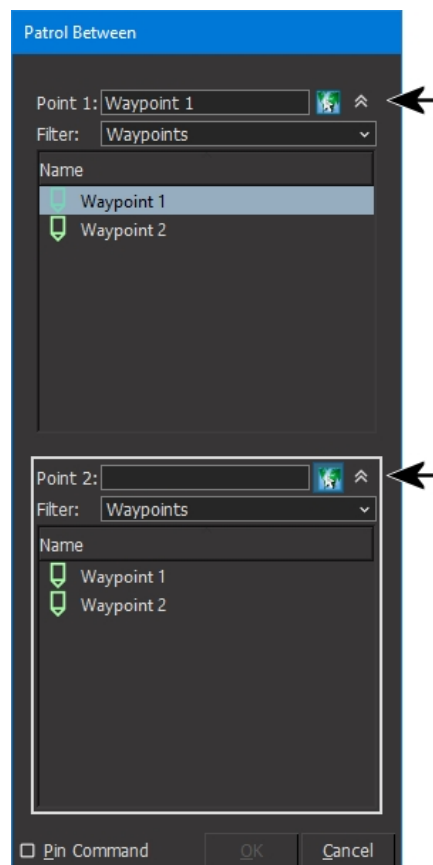
If you prefer, you can specify all parameters in the task parameters. Depending on the zoom level of the terrain and the number of simulation objects and tactical graphics you have created, it may be easier to select an object from a list than to select it from a densely packed group of icons.

#### Selecting Objects in Multiple List Parameters

Some tasks' parameters, such as those for Patrol Between, require you to select multiple objects, each of which is selected from a different list.

You can select the objects in the list window or in the display window. At any given moment, one of the lists is the active list and a selection in the display window is reflected in that list. The active list is indicated by a white rectangle around the list and the button next to the selection box (Figure 4).

**Figure 4. Multiple list dialog box**



The active list is determined as follows:

- When you initially open the parameters, the first list window is automatically active. If you select an object in the display, it is selected in the first list window. Then the second list window automatically becomes active and the next object you select in the display is selected in the second list window. This sequence continues until all list windows have a selection. Then the last window stays as the active list.
- Selecting an object in a list window makes it active.
- Clicking the Select Object button (👤) beside the box makes that list window active. It stays active until you click the button to deactivate it or click in another list window.
- If all lists are inactive (by clicking the Select Object button to put it into the unselected state), you can click objects in the display without affecting the selections in the list windows.

### Pinning Task Parameters

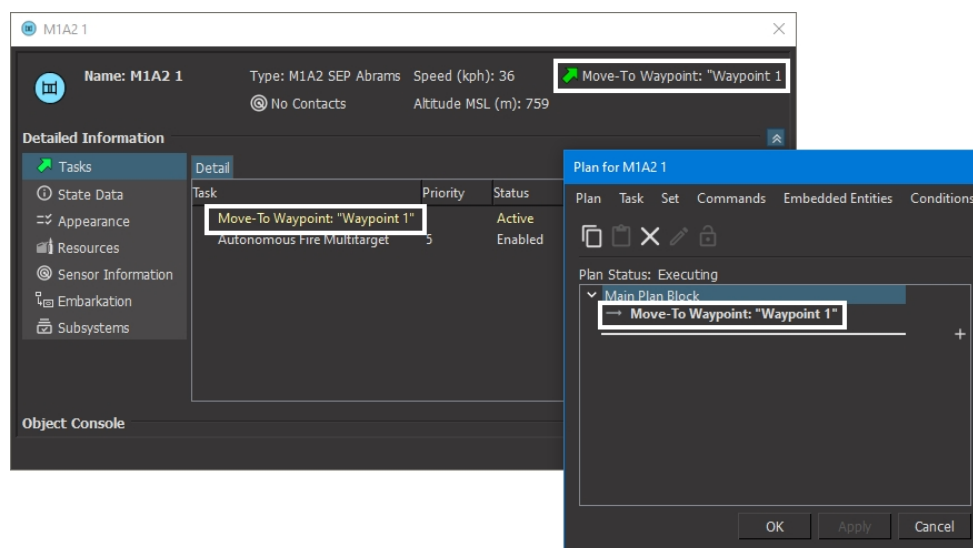
Use the Pin Command check box to pin the task parameters in the dialog box or the Command Panel. The OK button changes to an Execute button, and the entity will perform the task but the parameters will remain open. This allows you to quickly reassign the same task.

 This option is also available for set data requests.

### 2.1.8 Viewing Task Status

The Information dialog box for a simulation object lists its current task on the summary section. It also lists the state of the reactive tasks that are enabled for the simulation object. If a simulation object is executing a task as part of its plan, you can see which task it is executing by opening its Plan window (Figure 5). You can also see the current task on the Last Selected Object Panel and the Tasks page of the Information dialog box.

**Figure 5. Display of simulation object task status**

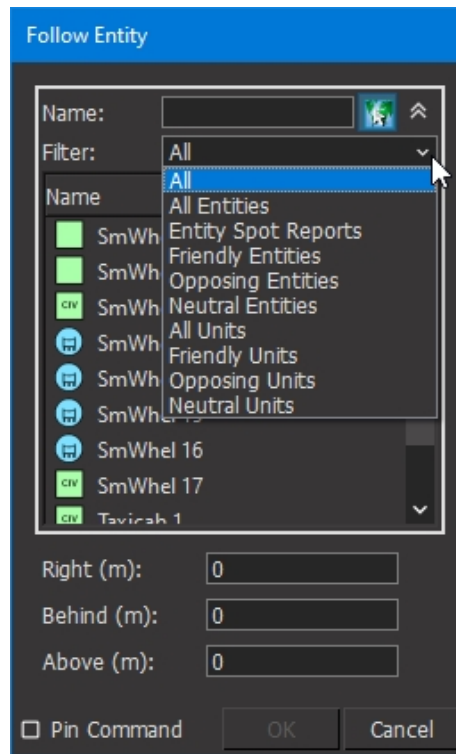




### 2.1.9 Filtering the Object Selection Lists

If a task requires you to select objects, you can filter the object selection list to make it easier to find the particular objects you need to select (Figure 6). If you are selecting simulation objects, for example in a Follow Entity task, you can display all simulation objects, individual simulation objects only, or units only. Within the latter two groups, you can display friendly or opposing simulation objects. In a Move To task, you can filter for waypoints, friendly simulation objects, or opposing simulation objects.

**Figure 6. Simulation object filter list**



- To filter a simulation object list, in a task or set data request parameters, choose the filter type from the list.

### 2.1.10 Skipping (Stopping) a Task

You can instruct a simulation object to stop executing its current task. If the simulation object is executing a plan, it moves to the next task in its plan. If there are no further tasks, the simulation object waits for further orders.

- To order a simulation object to skip a task, choose **Objects > Manage Plan > Skip Task**.



- Skip Task does not apply to members of a unit that are executing tasks assigned to them by the unit leader.
- You can also stop tasks from the Task Status page of the Information dialog box. For details, see "2.5.5 Canceling a Reactive Task" on page 22.

## 2.2. The Entity Control Panel

The Entity Control Panel is specific to a single entity. You can create one for each entity that you want to control.

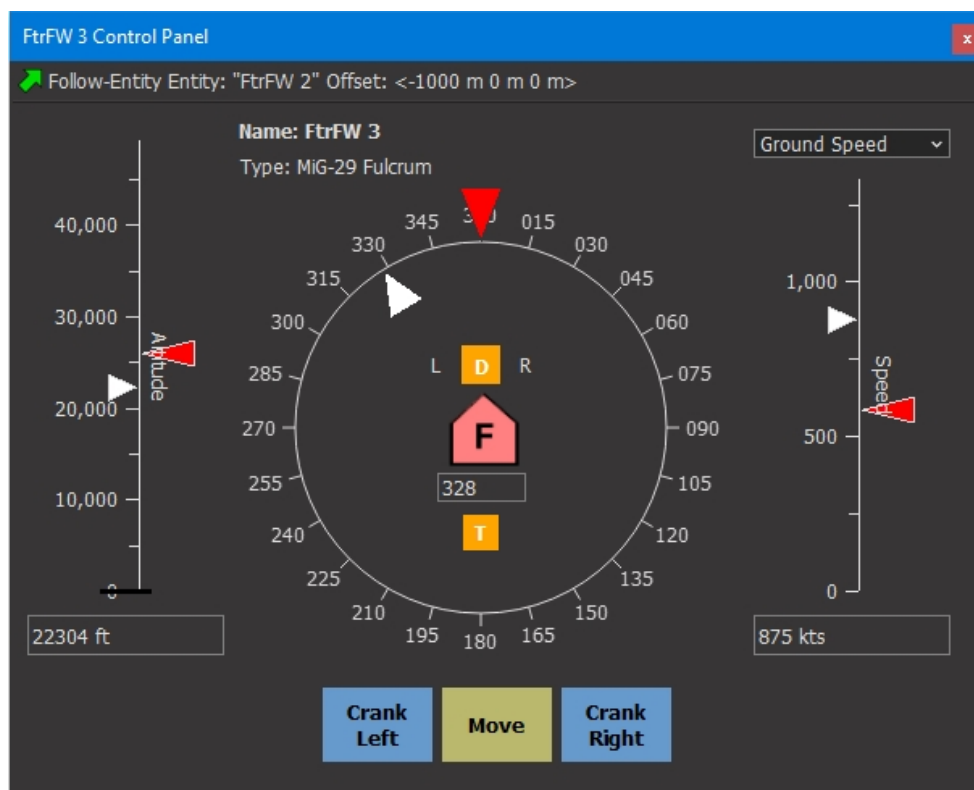
For aircraft, surface watercraft, or subsurface watercraft, you use the panel to quickly assign:

- Fly Heading and Altitude task (aircraft only)
- Sail Heading task (surface and subsurface watercraft only)
- Move to Depth task (subsurface watercraft only)
- Ordered Speed set data request

Using the configurable buttons at the bottom of the panel, you can assign entities other tasks or set data requests.

The Entity Control Panel (Figure 7) displays the name, type, altitude, heading, and speed of the selected entity with white arrows. Text boxes also show the current values. Red arrows show the desired values for aircraft, surface, and subsurface entities. The panel also shows the entity's current task.

**Figure 7. Entity Control Panel**



The Entity Control Panel displays the same information for ground entities and lifeforms as it does for aircraft and marine entities, but you can control them only by using the configurable buttons.

 Using the fly or sail heading, fly altitude, move to depth, and ordered speed controls that comprise most of the Entity Control Panel do not interfere with any plans that the entity might have. The buttons at the bottom can cause the entity to abandon its current plan. You are given a chance to confirm whether that is what you want.

### 2.2.1 Specifying Entity Movement Using the Entity Control Panel

After opening an Entity Control Panel for an aircraft, surface watercraft, or subsurface watercraft, you can use the arrows and text boxes to specify its movement.

When you change the altitude or heading of those types of entities using the panel, you are issuing a movement task. Like any other task assignment, this action causes the entity to stop its current task and begin executing the new task.

By default, when you change heading, VR-Forces chooses the direction in which to turn. However, you can specify which direction you want it to turn.

You can specify that the heading be aligned with either true north or magnetic north. For information about magnetic north in VR-Forces, see "Specifying True or Magnetic North".

If you change the ordered speed, the entity accelerates or decelerates to the desired speed. Changing the speed does not change the task.

When you single-click to change an entity's heading, speed, or altitude, then the entity makes the change at a normal speed. If you double-click, this indicates an emergency maneuver and the entity makes the change more rapidly than normal.

► **To assign a movement task from the Entity Control Panel:**

1. In the object list or terrain, select one or more entities to command.
2. Choose **Objects > Entity Control Panel** The Entity Control Panel opens for each selected entity.

 You can also open the control panel for an entity by clicking Show Entity Control Panel () on the Last Selected Object panel.

3. To change altitude or depth, click along the Altitude line, click and drag the read arrow, or type an altitude in the text box.
4. To change heading, click a heading on the compass, click and drag the red arrow, or type a heading in the text box. To have the entity change heading at the maximum speed possible, double-click the heading. To specify the direction of turn, click L (left) or R (right).
5. The button below the heading text box indicates whether the heading is true north (T) or magnetic north (M). Click the button to change the setting.
6. To set the ordered speed, click along the Speed line, click and drag the red arrow, or type a value in the text box. For aircraft, you can use the list at the top right to specify the measurement of the speed: Ground Speed, TAS (true airspeed), IAS (indicated air speed), or Mach. You cannot set a speed greater than the entity's maximum speed.

## 2.2.2 Using the Buttons in the Entity Control Panel

You can click a button at the bottom of the Entity Control Panel to quickly assign tasks and set data requests to an individual entity. These buttons are configurable.

### Assigning a Task or Set

► **To assign a movement task from the Entity Control Panel:**

1. In the object list or terrain, select one or more entities to command.
2. Choose **Objects > Entity Control Panel** The Entity Control Panel opens for each selected entity.

 You can also open the control panel for an entity by clicking Show Entity Control Panel () on the Last Selected Object panel.

3. Click the button for the task or set that you want to assign.

### Configuring the Buttons

One file determines what buttons are available:

`./data/simulationModelSets/EntityLevel/gui/unifiedControlToolbar.xml`. VR-Forces includes some buttons by default. Only the buttons that are relevant to the entity display in its Entity Control Panel. For example, the Move To Waypoint task displays for all entities but Patrol Area is only available to watercraft.

The default configuration for the buttons in the Entity Control Panel is:

```
<DtUnifiedControlToolbar buttonwidth="70" buttonheight="48"
spacing="4" fontSize="10" fontBold="true" justify="Center">
  <item task="crank_left" icon="false" background="#6699cc"/>
  <item task="move-to" icon="false" background="#BAB86C"
title="Move"/>
  <item task="crank_right" icon="false" background="#6699cc"/>
</DtUnifiedControlToolbar>
```

Table 1 describes the available parameters for customizing a button on the Entity Control Panel.

**Table 1. unifiedControlToolbar.xml parameters**

| Keyword     | Description                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| task<br>set | ID of the C++ or Lua task or set. (string, required)                                                                                           |
| icon        | Whether the button should show the icon for the task or set (if available) instead of the text. The default is not to show the icon. (Boolean) |
| background  | Hexadecimal color to use as the background for the button.                                                                                     |

Table 1. unifiedControlToolbar.xml parameters (continued)

| Keyword | Description                                                                                                             |
|---------|-------------------------------------------------------------------------------------------------------------------------|
| title   | Text for the button that is different from the task or set's name (when not displaying the icon). (string)              |
| justify | Justification for the text. You can specify Left, Center, or Right. Does not apply to icons; those are always centered. |

► **To configure the buttons in the Entity Control Panel:**

1. Open `./data/simulationModelSets/EntityLevel/gui/unifiedControlToolbar.xml` in your preferred tool for editing XML files, such as Notepad++.

2. Enter a new item entry or edit an existing one as desired, using the format:

```
<item = [task="task_ID"|set="set_ID"] (icon="[true|false]")
      (background="#nnnnnn") (title="title")
      (justify="[Left|Right|Center]")/>
```



- There is no limit to the number of buttons that you can add, but do note that they will not wrap. The Entity Control Panel simply grows wider to accommodate the buttons.
- You cannot add multiple buttons for the same task or set.
- Any changes you make go into effect the next time you start VR-Forces. If you make changes while the application is running, you must shut it down and restart it to see the new button configuration.

## 2.3. Using a Keyboard Shortcut to Assign a One-Click Task (or Set)

You can quickly task an individual entity with a "one-click" task. VR-Forces is configured with two such tasks: move to a location or fire on a target or location. These one-click tasks do not require you to enter any values: you simply click the entity on the terrain to select it (not in the Objects List), press the appropriate keyboard combination, and then click a simulation object or a location on the terrain.

The setting to allow users to assign a Lua script task (or set data request) with a keyboard shortcut and a mouse click is One Click Support. See "9.2. Creating a New Script" on page 193 to learn more about the parameters for Lua scripts.

► **To assign a one-click task:**

1. Select the entity on the terrain.
2. Press the appropriate keyboard combination, for example:
  - **Ctrl+M** to move to a location.
  - **Ctrl+F** to fire on a target or location.

3. Click the location or simulation object on the terrain or in the Objects List. While the scenario is running, the entity immediately performs the task (or set) if able.

 Note that these shortcuts only allow you to task entities to move to or fire at anything that they are configured for. It does not override any restrictions. For example, an entity configured only to fire at simulation objects cannot fire on a location.

## 2.4. Assigning Tasks to Units

- To assign a task to a unit, select the unit, then assign it a task as you would any individual entity.

### 2.4.1 Convoy Tasks

VR-Forces has two tasks for moving in convoys: Convoy Along and Convoy To. These tasks can only be assigned to units made up of ground vehicles. When a unit receives a convoy task, it organizes itself into a column based on the echelon IDs. Then it executes the task.

Once the task begins, the members of the unit try to maintain a consistent distance apart from each other. You can configure this distance by editing the Ground Convoy Movement system. The distance is specified by the **separation-distance** parameter. You can also specify an amount by which the separation distance can vary using the **separation-tolerance** parameter.

 If VR-Forces determines that a convoy task is not appropriate for a given unit—for example, a rotary-wing unit—the convoy tasks are not available on the Task menu or in the Command Panel. In other cases, however, such as mixed units or when you aggregate disparate entities as Ground Unit, it is up to you to know whether a unit consists only of ground vehicles or contains platforms that do not support convoy tasks.

### 2.4.2 Independently Tasking Unit Members

Members of a unit can have individual plans or be assigned tasks independently of the unit. If a member of a unit is independently tasked, once it completes its task, it then automatically becomes tasked by its superior. You do not need to give it a Tasked by Superior set data request. You can override this behavior by setting an object's **tasked-by-superior-upon-task-complete** parameter in the object parameter database to False.

If a Skip Task command is given to an independently tasked subordinate, it remains in the independently tasked state until it either completes a task or it is given a Set Tasked by Superior set data request.

When a simulation object becomes tasked by its superior, it does not execute the task that the unit is currently executing. It receives the next task that is sent out by the unit.

## 2.5. Reactive Tasks

Reactive tasks are a special category of scripted tasks. Unlike other tasks (whether scripted or C++), reactive tasks do not begin to execute as soon as they are assigned. They monitor events in a simulation and execute in response to conditions specified in the script. In this way they are similar to collision avoidance, which takes over entity movement in response to an imminent collision, or trigger statements in plans, which are triggered when a condition becomes true.

Because reactive tasks are scripted tasks, you are not restricted to the few conditions available in plans. You have the full VR-Forces Lua API available to script the conditions and the resulting behaviors. Like other scripted tasks, reactive tasks are either scenario-specific or system reactive tasks.

Reactive tasks have the following behaviors and characteristics:

- Reactive tasks can be automatically enabled for a category of simulation object. They can also be individually enabled and disabled.
- When a reactive task activates, the current task is suspended. When the reactive task concludes, the simulation object resumes the previous task, if possible. There may be cases where the previous task is no longer meaningful due to changes in the scenario.
- Reactive tasks support task concurrency.
- Reactive tasks can be interrupted by other reactive tasks. Precedence among reactive tasks is based on a task's priority. If a reactive task is executing and a reactive task with a higher priority is triggered, it suspends the current reactive task. When the higher priority task completes, the lower priority task resumes. When it completes, the original task resumes. Multiple nesting of tasks in this way is permitted. Reactive tasks with the same or lower priority do not interrupt the active task.
- Reactive tasks can have parameters, just like any other task. If a reactive task is enabled and the parameters are not set, it may not work as expected. A well-written reactive task should have default values for parameters, but VR-Forces does not enforce this.
- Users can change the priority of a reactive task while a scenario is running. This affects the next activation of the task.
- If a reactive task is interrupted by a non-concurrent independent task assignment from the Task menu, the Command Panel, or a global plan, the reactive task and all suspended tasks are canceled and the new task is executed. However, it is still possible that the new task could be interrupted by the same or a different reactive task.
- Active reactive tasks can be canceled by a VR-Forces user.
- After a reactive task is complete, it enters the enabled state and can become triggered again.
- The status of a reactive task displays on the Tasks page of the Information dialog box, along with the status of the current or suspended task.

The following sections describe how to manage reactive tasks in a scenario. VR-Forces includes a collection of reactive tasks, listed in "7.14. Reactive Tasks" on page 158. For information about how to write and configure reactive tasks, see "Chapter 10. Writing Scripts" on page 229.

### 2.5.1 Enabling Reactive Tasks

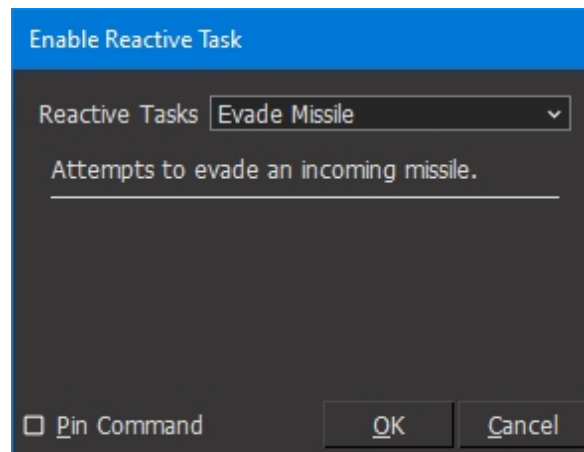
Like other scripted tasks, when a reactive task is written, it is specified as being valid for one or more simulation object types. It can also be configured to be enabled by default. If a reactive task is not enabled by default, you can enable it for specific simulation objects.

**i** This procedure applies to the currently selected simulation object. If you have multiple Information windows open, selecting a reactive task on the Task Status page of a simulation object does not select the simulation object and does not make it the task affected by this procedure.

► **To enable a reactive task:**

1. Select the simulation object for which you want to enable the task.
2. Choose **Set > Reactive Tasks > Enable**. The Enable Reactive Task dialog box opens (Figure 8). If there are no reactive tasks available, the dialog box states this. This condition occurs if all appropriate reactive tasks are already enabled or there are no reactive tasks for this entity type.

**Figure 8. Enable Reactive Task dialog box**



3. In the Reactive Tasks list, select the task you want to enable. If the task requires parameters, the dialog box redisplay and lists the parameters.
4. If necessary, specify the parameters.
5. Click OK.

**i** You can also enable reactive tasks in the Manage Reactive Tasks dialog box. For details, see "2.5.4 Managing Reactive Tasks" on page 22.



## 2.5.2 Disabling Reactive Tasks

You can disable reactive tasks on a per-simulation object basis. When you disable a reactive task, any active tasks continue to execute. However, no new instances of the disabled task will be activated.

**i** This procedure applies to the currently selected simulation object. If you have multiple Information windows open, selecting a reactive task on the Task Status page of a simulation object does not select the simulation object and does not make it the task affected by this procedure.

► **To disable a reactive task for a simulation object:**

1. Select the simulation object for which you want to disable a reactive task.
2. Choose **Set > Reactive Tasks > Disable**. The Disable Reactive Task dialog box opens. If there are no reactive tasks available, the dialog box states this. This condition occurs if all appropriate reactive tasks are already disabled or there are no reactive tasks for this simulation object type.
3. In the Reactive Tasks list, select the task you want to disable.
4. Click OK.

**i** You can also disable reactive tasks in the Manage Reactive Tasks dialog box. For details, see "2.5.4 Managing Reactive Tasks" on the next page.

## 2.5.3 Setting the Priority of a Reactive Task

When a reactive task is created, it is assigned a priority. The priority determines which reactive tasks it can override or be overridden by. Priorities are expressed as integers, with 1 being the highest priority. You can change the priority of a reactive task on a per-simulation object basis.

**i** This procedure applies to the currently selected simulation object. If you have multiple Information windows open, selecting a reactive task on the Task Status page of a simulation object does not select the simulation object and does not make it the task affected by this procedure.

► **To change the priority of a reactive task:**

1. Select the simulation object for which you want to change a reactive task's priority.
2. Choose **Set > Reactive Tasks > Priority**. The Reactive Task Priority dialog box opens. If there are no reactive tasks available, the dialog box states this.
3. In the Reactive Tasks list, select the task whose priority you want to change.
4. Select a priority in the box. The lower the number, the higher its priority.
5. Click OK.

**i** You can also change priority in the Manage Reactive Tasks dialog box. For details, see "2.5.4 Managing Reactive Tasks" below.

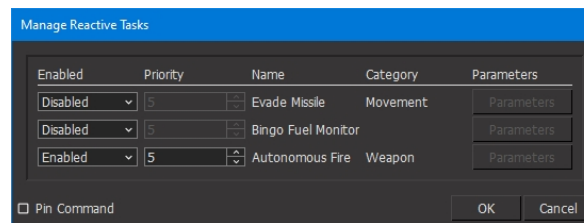
## 2.5.4 Managing Reactive Tasks

You can enable, disable, and change priority for all the reactive tasks that apply to a simulation object from the Manage Reactive Task dialog box.

► **To manage multiple reactive tasks for a simulation object:**

1. Select the simulation object for which you want to manage reactive tasks.
2. Choose **Set > Reactive Tasks > Manage** or choose **Objects > Manage Reactive Tasks**. The Manage Reactive Task dialog box opens (Figure 9). It lists each reactive task configured for this simulation object type, its status, and its priority. If a task requires parameters, you can set them.

**Figure 9. Manage Reactive Task dialog box**



3. For each task, to enable or disable, select an option from the list in the Enabled column.
4. For each task, change the priority by selecting an option in the box in the Priority column.
5. For each task that has parameters, click Parameters and set its parameters. (A task must be enabled to set its parameters.)
6. Click OK.

## 2.5.5 Canceling a Reactive Task

You can cancel a reactive task while it is executing. When you cancel a reactive task, the simulation object resumes the suspended task. However, if the condition that caused the reactive task to activate is still true, when you cancel a reactive task, it will usually immediately activate again.

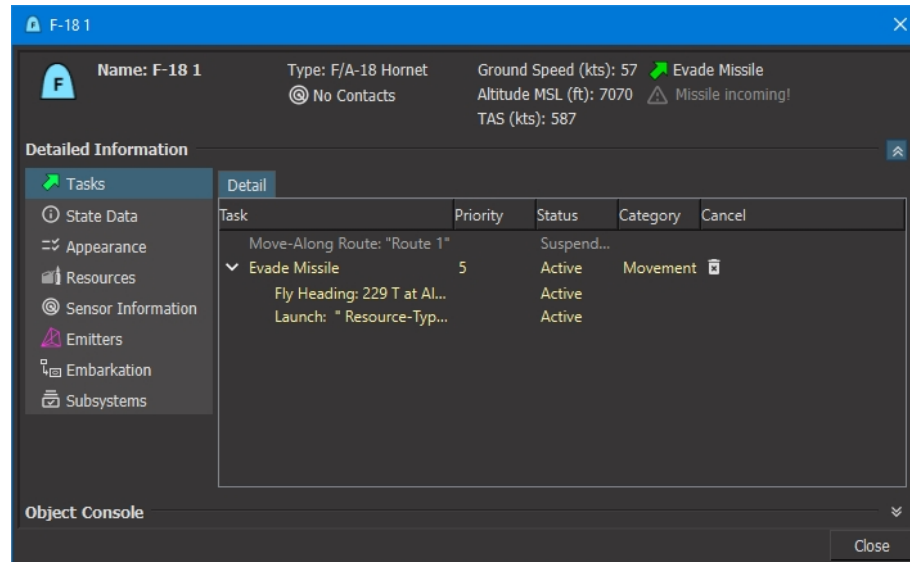
**i** You can also use this procedure to cancel (skip) a regular task.

► **To cancel a reactive task:**

1. Select the simulation object whose task you want to cancel.
2. Choose **Objects > Information**, or press **I**. The Information dialog box opens.

3. Select the Tasks page. If a reactive task is active, it has a button in the Cancel column (Figure 10).

Figure 10. Active reactive task



4. Right-click the active task and choose **Cancel Reactive Task**, or click the button in the Cancel column.

## 2.6. Using Behavior Sets to Manage Scripts

Behavior sets let you organize scripts so that a set of related tasks and set data requests can be enabled or disabled at the force level.

If a script is not assigned to a behavior set, its availability to a simulation object is determined by the following settings:

- The Valid for Entity Types list for the script.
- For tasks specific to systems, configuration of that system on a simulation object.
- For reactive tasks, the Enable/Disable setting in the Manage Reactive Tasks dialog box for the simulation object.

If a script is assigned to a behavior set, then in addition to the requirements in the previous list, it is available to simulation objects only if the behavior set is assigned to a force and only to simulation objects of that force. (You can assign a behavior set to more than one force.)

As an example of how you might use behavior sets, suppose that dismounted infantry react differently to an attack depending on whether they are operating in rural terrain or in urban terrain. You have a set of scripted tasks that you have written for reacting to attack in each of these two situations. If you are not using behavior sets, then to manage which tasks are used when the simulation objects are under attack, you might use one of these strategies:

- Tell users not to use the inappropriate tasks.
- Enable and disable the correct set of tasks in the Scripts dialog box before the start of the scenario.
- Enable and disable the tasks on a per simulation object basis in the Manage Reactive Tasks dialog box.

Shifting back and forth between simulations in different terrains might become a management problem.

Using behavior sets, you could:

1. Create a behavior set for rural terrain operations and one for urban operations.
2. Assign the relevant scripts to each behavior set.
3. Before you run a scenario, you would assign the behavior set you want to use to the force.

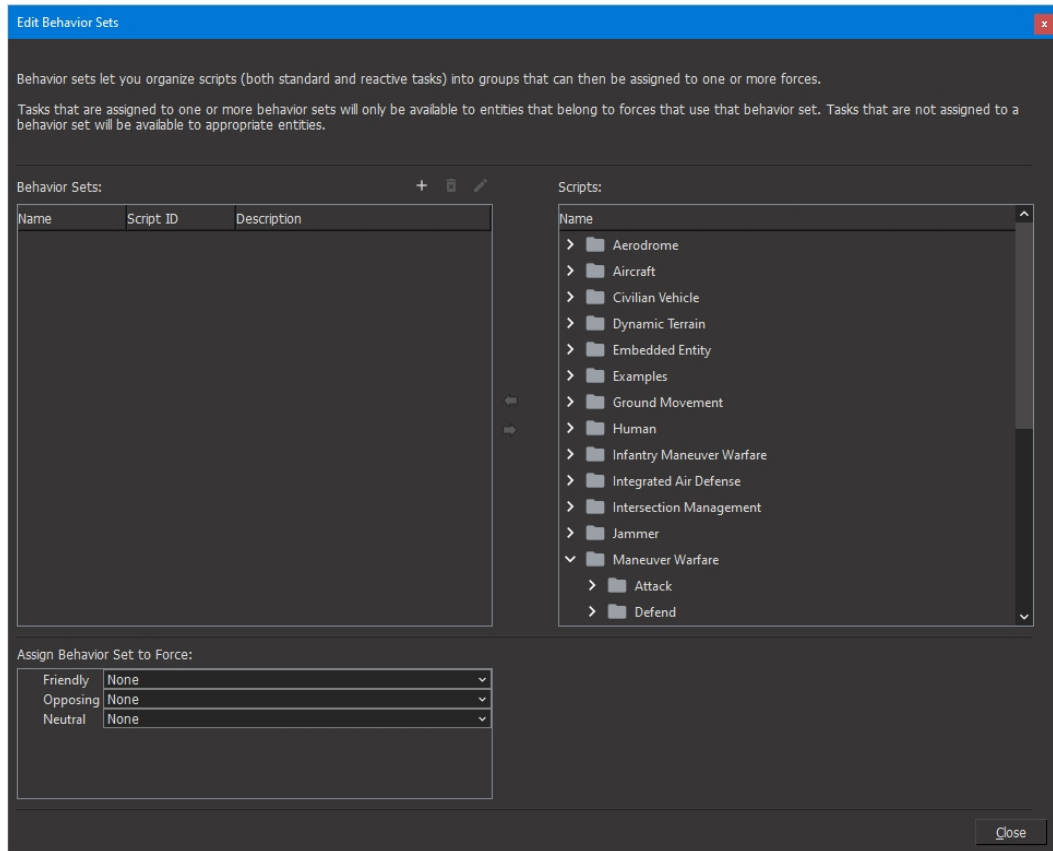
The tasks in the behavior set would be enabled and those in the unused behavior set would be disabled. There would be no further configuration required.

### 2.6.1 Creating Behavior Sets

► **To create a behavior set:**

1. Choose **Simulation > Behavior Sets**. The Edit Behavior Sets dialog box opens (Figure 11).

Figure 11. Edit Behavior Sets dialog box



2. Click the Add button (+). The New Behavior Set Name dialog box opens.
3. Enter a name for the behavior set.
4. Click OK.
5. In the Scripts list, select the scripts that you want to assign to this behavior set.
6. Click the left-facing arrow. The scripts are added to the behavior set. If you select a folder, all tasks in the folder are added to the behavior set. (You can also assign tasks to behavior sets in the Edit Script dialog box. For details, see "9.2. Creating a New Script" on page 193.)
7. Optionally, assign the behavior set to a force.

### 2.6.2 Editing Behavior Sets

You can edit behavior sets, rename them, and delete them.

► **To edit behavior sets:**

1. Choose **Simulation > Behavior Sets**. The Edit Behavior Sets dialog box opens (Figure 11).
2. Expand the behavior set that you want to edit.
3. Add and remove scripts by selecting them and clicking the appropriate arrow button.

- ▶ To rename a behavior set, select it in the Behavior Sets list, click the Rename button () and then enter a new name.
- ▶ To delete a behavior set, select it in the Behavior Sets list and click the Delete button ()

### 2.6.3 Assigning a Behavior Set to a Force

You can only assign one behavior set to a force at any given time.

▶ **To assign a behavior set to a force:**

1. Choose **Simulation > Behavior Sets**. The Edit Behavior Sets dialog box opens (Figure 11).
2. In the Assign Behavior Set to Force window, click the list for the force to which you want to assign a behavior set.
3. Select the behavior set that you want to assign. If you do not want any behavior sets assigned, select the blank line in the list.

## 2.7. Entity Movement On Roads and Off Roads

Some terrains have vector road networks. Ground vehicles can use these networks to move precisely through the terrain. The tasks for moving on roads are distinct from the tasks for moving without respect to road networks. (For a description of how entities move when they use road driving, see "2.7.1 Road Driving Behavior" on the facing page.) If you want entities to move both on and off roads, you need to understand how these tasks interact.

The road driving tasks are Move to Waypoint (Plan Along Roads) and Move to Location (Plan Along Roads). The generic movement tasks are Move to Location and Move to Waypoint.

 Road driving tasks are valid only for ground vehicles. If you give a road driving task (including Move To (Plan Along Roads), Treat Route as Road, and Use Roads in pattern of life default plans) to any other type of entity, it will fail.

When you give a simulation object a generic Move To task, it moves directly towards its destination. If navigation data is available, it plans a path using that data. If no navigation data is available, the entity avoids obstacles, but otherwise does not pay attention to the terrain except to the extent that it cannot move on certain soil types or terrains that are too steep.

When you give an entity a Plan Along Roads task, it looks for a road network. If it finds one:

- It moves to the nearest point on the road.
- It moves along the road network to the point nearest to the final destination.
- It moves to the final destination.

❗ The road driving feature in VR-Forces requires that the road vectors making the network be connected at their edges. If your terrain's data does not have accurate positions for road ends, causing them to be disconnected, VR-Forces might not be able to plan a path along those roads.

Vehicles follow vector features and ground clamp to the terrain. If there is a crater in the vehicle's path, the vehicle drives through it. If a bridge is destroyed, the road vector remains and the vehicle drives across the terrain that the bridge previously spanned.

### 2.7.1 Road Driving Behavior

The Move To (Plan Along Roads) tasks use road driving behaviors intended to provide realistic behavior of multiple cars on a road network. When an entity is given one of these tasks:

- It clamps to the road and does not deviate from it.
- It goes around corners accurately.
- It ignores any obstacles that are near to the road and which would ordinarily trigger obstacle avoidance. The vehicle only responds to obstacles that block the road. This does not include craters.
- If a vehicle is blocking its progress and there is an adjacent lane in the road that is clear, the entity passes the blocking vehicle.

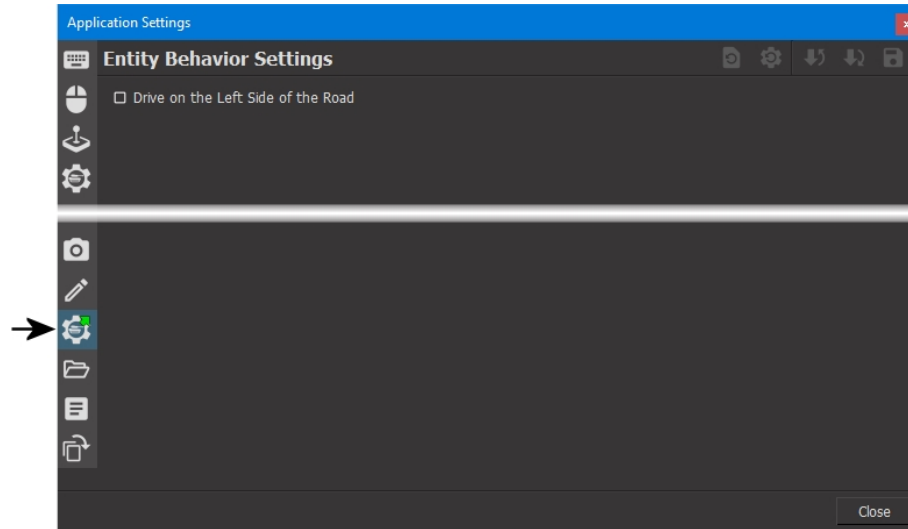
In the Move Along Route task, you can specify that VR-Forces treat the route as a road. The entity would then use the road driving system to follow the route.

### 2.7.2 Specifying the Side of the Road to Drive On

By default, entities that use road driving behavior drive on the right side of the road. You can specify that entities drive on the left side of the road. For this setting to take effect, you must make the change before you load a scenario or before you rewind it. If you change the setting after you rewind a scenario or while the scenario is running, the behavior does not change until the next time you rewind the scenario.

► **To specify the side of the road to drive on:**

1. Choose **Settings > Application**. The Application Settings dialog box opens.
2. Select the Entity Behavior Settings page (Figure 12).

**Figure 12. Entity Behavior Settings page**

3. Select or clear the Drive on the Left Side of the Road check box.
4. Click Close.

## 2.8. Fixed-Wing Entity Tasks and Behaviors

This section describes the behaviors of fixed-wing entities supplied with VR-Forces and the issues you should keep in mind when assigning tasks and set data requests.

### 2.8.1 Placement of Newly Created Fixed-Wing Entities

When you create a fixed-wing entity, it is placed on the ground. If you want a fixed-wing entity to start a scenario in the air, you must set its altitude at the beginning of the scenario. It will loiter at the specified altitude.

### 2.8.2 Fly Task Behavior Is Different from Move Task Behavior

Fixed-wing entities can move in response to Fly tasks (Fly Altitude, Fly Heading, and so on) and Move tasks (Move to Waypoint, Move Along Route, and so on). There are important differences between how fixed-wing entities respond to the different classes of movement tasks. Fixed-wing entities can move on the ground (taxi), in which case they move like ground vehicles.

For Move and Patrol tasks, if a fixed-wing entity is in the air, when it arrives at a point or the end point of a route, it loiters.

Fly tasks are only for entities that are in the air. You cannot use them to cause an entity to take off or land. For Fly tasks:

- Entities move to a new altitude at the specified heading, at the specified climb or descent rate.
- When they reach the new altitude, they continue flying at their current heading.



Given these differences, if you want to change heading, altitude, or both for an entity that is in the air and not executing a Move or Patrol task, use a Fly task. These tasks are not designed to have an entity move to a particular location or along a route. To have an entity move to a particular location or along a route, use a Move or Patrol task. In most cases, Fly Altitude will be preferable to Move to Altitude.

### 2.8.3 Specifying and Maintaining Altitude for Fixed-Wing Entities

You can explicitly change a fixed-wing entity's altitude:

- The Altitude set data request moves an entity immediately to the specified altitude.
- The Move to Altitude task causes an entity to spiral to the specified altitude at the same coordinates that it had when it received the task.
- The Fly Altitude and Fly Heading and Altitude tasks cause the entity to move smoothly to the new altitude while continuing to fly at its current or newly specified heading.

An entity's altitude changes implicitly if it is given a Move or Patrol task in which the altitude of the target point or vertex is different from the entity's current altitude. When moving along a route, the entity changes altitude as it flies along a line between vertices.

 When you assign a task that includes a location and you specify the location by clicking on the terrain, the default altitude is zero. If you do not specify a non-zero altitude in the task dialog box, VR-Forces uses the altitude of the entity at the time the task is assigned. This prevents you from inadvertently specifying a location that would cause the entity to crash.

### 2.8.4 Fixed-Wing Entity Movement on the Ground and in the Air

Fixed-wing aircraft support takeoff and landing. Since they can move on the ground and in the air, you need to understand their behavior in these different environments. Table 2 and Table 3 describe how fixed-wing entities respond to Move To, Move Along Route, and Patrol Route tasks depending on their current location. The rest of this section provides additional details about take-off and landing.

**Table 2. Fixed-wing entity Move To behavior**

| If entity is on/in the: | and waypoint is on/in the: | the entity:                  |
|-------------------------|----------------------------|------------------------------|
| Ground                  | Ground                     | taxis to point.              |
| Ground                  | Air                        | takes off towards the point. |
| Air                     | Air                        | flies to point.              |
| Air                     | Ground                     | crashes.                     |

**Table 3. Fixed-wing entity route-following behavior**

| If entity is on/in the: | and next vertex is on/in the: | the entity:                  |
|-------------------------|-------------------------------|------------------------------|
| Ground                  | Ground                        | taxis to vertex.             |
| Ground                  | Air                           | takes off toward the vertex. |
| Air                     | Air                           | flies to the vertex.         |
| Air                     | Ground                        | tries to land at the vertex. |

Fixed-wing entities have a look-ahead capability. If a route segment goes underground, a fixed-wing entity tries to maintain altitude above the terrain rather than following the route into the ground. This is not nap-of-the-earth flight. The aircraft looks ahead a configured distance in its current flight direction. If any terrain blocks the path between it and its destination vertex on the route, it ascends to avoid the terrain. When no blocking terrain is found, it resumes flying toward the destination vertex directly.

### 2.8.5 How Fixed-Wing Entities Take Off

You can have fixed-wing aircraft take off using the Takeoff On Runway task (for details, see "3.46. Takeoff On Runway" on page 72) or by using innate behavior. This topic describes takeoff using innate behavior.

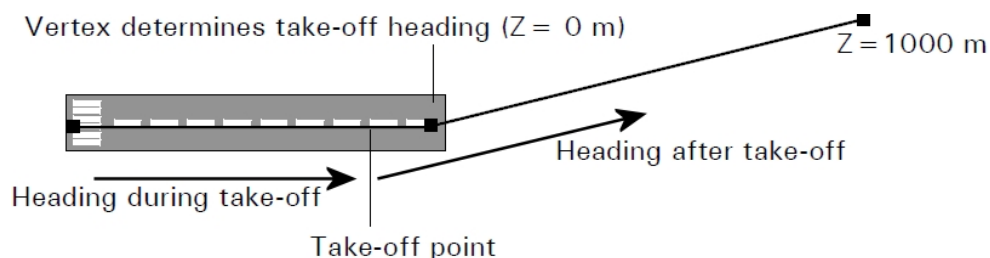
Fixed-wing entities take off by moving to a waypoint that is above ground level or by following a route that has a vertex above ground level.

When you task a fixed-wing entity to move to a waypoint that is above ground level, the entity orients itself directly towards that point and accelerates in the direction of the point. When it reaches its minimum lift speed, it takes off and rises to the altitude of the point it is moving to. You can control the distance required to take off by setting the **max-thrust parameter**.

When a fixed-wing entity takes off, it does not take into account the soil type or terrain, except that if the slope of the terrain along which the entity must move to take off is greater than the **max-slope** parameter, the entity halts.

When you create a waypoint towards which an entity will take off, or create a route along which an entity will take off, you are responsible for ensuring that the entity is properly oriented towards the runway.

If an entity takes off as part of route following, you can define a runway with two points on the ground. If an entity detects that the point in the route after the one it is moving towards is in the air, it uses the current point as the take-off direction, but takes off as soon as possible and flies to the point in the air. Figure 13 illustrates this behavior.

**Figure 13. Defining a runway for take-off**

## 2.8.6 How Fixed-Wing Entities Land

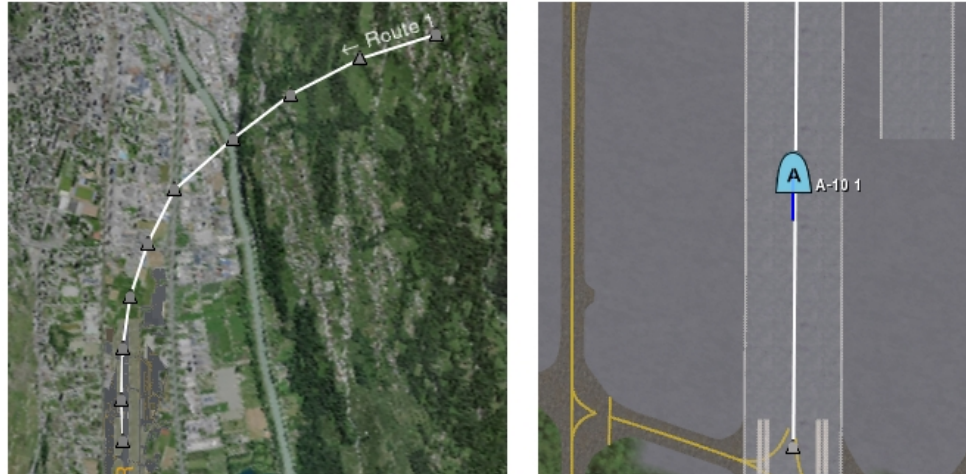
You can have aircraft land using a fixed-wing landing task (see "3.15. Land At Airport" on page 50, "3.16. Land at Airport Near Location" on page 50, "3.17. Land On Route" on page 51, or "3.18. Land On Runway" on page 51), or as described in this section. Using a fixed-wing landing task automates the process of setting up the approach route and landing.

When a fixed-wing entity follows a route and determines that the next vertex in the route is on the ground, it decelerates as it reduces altitude towards that vertex. If it does not have enough distance from the previous vertex to decelerate sufficiently, it may crash. To ensure adequate deceleration, you may want to create a trigger that sets the entity's speed when it approaches the runway.

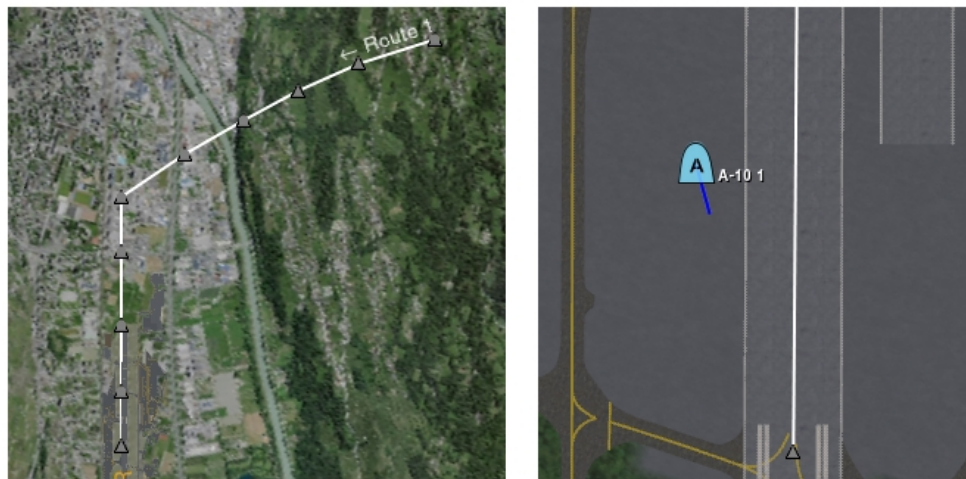
 Any time an entity is within the **on-ground-tolerance** distance of the terrain and it is moving within 10 percent of its minimum lift speed, it will try to land. The **on-ground-tolerance** for most fixed-wing entities is 10 meters.

To orient an entity for landing, you may have to create several vertices prior to the runway to properly line up the entity. Figure 13 shows two different routes ending at an airport (the routes have been edited to improve visibility). The first route has vertices that line up the entity with the runway. The second route curves in towards the runway, so the entity does not approach the runway in a straight line. Therefore, as it reaches the landing point, it is not lined up with the runway and it lands on the taxiway.

Figure 14. Landing fixed-wing entities



Entity lands on runway



Entity lands beside runway

## 2.9. Rotary-Wing Entity Tasks and Behaviors

This topic describes the behaviors of rotary-wing entities supplied with VR-Forces and the issues you should keep in mind when assigning tasks and set data requests.

Rotary-wing entities are created on the ground unless you specify an altitude at creation. (For details about specifying an altitude when you create an air platform, see "Specifying an Object's Altitude".)

If a rotary-wing entity is airborne and it is not assigned a task, it hovers at the specified location and altitude. When it completes a task, it hovers in the location where it completed the task.

The `terrain-check` parameter determines how a rotary-wing entity interacts with the terrain. If `terrain-check` is `False`, a rotary-wing entity ignores the terrain. It could appear to fly below ground. However, when `terrain-check` is set to `False`, VR-Forces does not have to check ground points, so performance may improve.

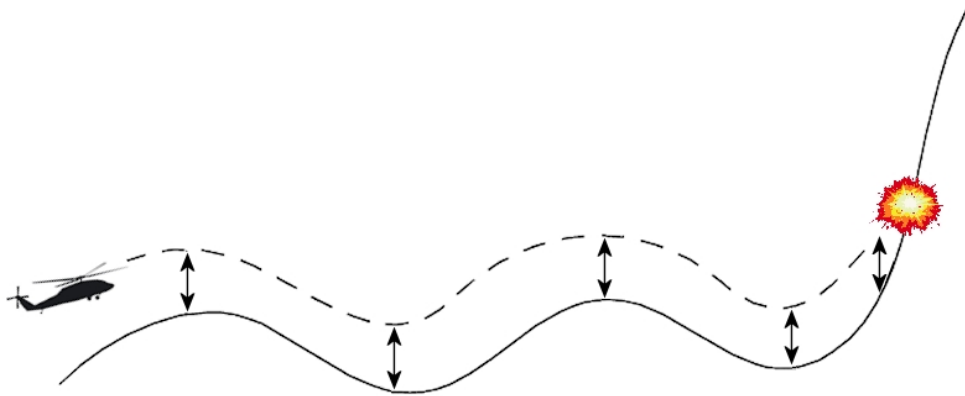
When `terrain-check` is `True`, rotary-wing entities interact with the terrain. If a rotary-wing entity approaches a ground point at three meters per second or less, it lands. If it approaches the ground at a greater speed, it crashes.

**i** When you assign a task that includes a location and you specify the location by clicking on the terrain, the default altitude is zero. If you do not specify a non-zero altitude in the task dialog box, VR-Forces uses the altitude of the entity at the time the task is assigned. This prevents you from inadvertently specifying a location that would cause the entity to crash.

Helicopters have a basic terrain avoidance algorithm that helps prevent crashing if they are sent to waypoints or follow routes that do not have an altitude. The terrain avoidance logic maintains a distance above the ground; it does not look ahead. Therefore, a helicopter could crash if the terrain rises suddenly (Figure 15), such as a steep hill or building. You can view the terrain profile of the route to check for intersections with the terrain. (For details, see "Displaying a Line's Terrain Profile".)

Rotary-wing aircraft take wind resistance and the weight of the aircraft, including the weight of any embarked objects, into account when calculating fuel consumption. If an aircraft's power plant appearance is set to off and it is tasked to move, the rotors spin up and the aircraft will not move until they reach full speed.

**Figure 15. Helicopter terrain avoidance**



## 2.10. Suppressive Fire Tasks

VR-Forces has two suppressive fire tasks.

- The Provide Suppressive Fire at Object task tells an entity to fire at a simulation object such as a point, along a line, or in an area, targeting one meter above the terrain. If the entity must stand or move to the side to see the target, it does so.
- The Provide Suppressive Fire at Location task tells an entity to fire at a location, one meter above that location.

Fire is rapid at first, then sustained for a period of time and maximum rounds to fire, or until the entity runs out of ammunition. The rapid and sustained rates of fire are configured in the weapon's system definition file. If you want to suppress a floor of a

building, use the Provide Suppressive Fire at Location task and set the altitude at or slightly above the altitude of the floor of the building. Suppressive fire is executed even if line of sight is blocked near the target location, so you can place the target in a building and have fire on the wall in front of it.

Suppressive fire insults on an entity are accumulated as part of the internal state of the entity. This suppression value also continually decays over time. Therefore large insults, or many small insults in a short time, raises the suppression value, but the value naturally drops back down to its starting level.

An entity is considered suppressed when its suppression value rises above a certain threshold. The entity is no longer suppressed when its decaying suppression value falls below a different, lower threshold. This allows the model to produce a delay in suppression recovery and at the same time continue handling new insults.

The model represents three suppression levels (in addition to "not suppressed") so that behavior can be changed for more severe suppression. The number of suppression levels was chosen arbitrarily. Higher levels of suppression are reached when the suppression value passes higher thresholds. The State Data page in the Information dialog box displays the suppression level of the entity, a value between 0 and 3.

Suppressive fire causes humans that are near the detonation locations to temporarily stop shooting and lose some ability to sense entities. When an entity is suppressed:

- Its sensors may have their sensitivity reduced.
- Its guns may be prevented from firing.
- It may be forced into a crouching posture.
- It may be forced to stop moving and go prone.

The default configuration for humans is:

- Visual sensor sensitivity is reduced to 0.
- Guns are disabled.
- Level 1 suppression forces a crouch.
- Level 2 and 3 suppression stops movement and makes the entity go prone.

The Lifeform Suppression system enables two reactive behaviors. The first one makes the human adopt a crouching posture when it is suppressed. This does not interrupt other movement. The second reaction makes the entity go prone when it is suppressed to level 2 or 3. This interrupts any ongoing movement task. Both reactions restore the original posture when the reaction ends.

Several vehicles are configured so that they cannot fire their machine guns if they are suppressed.

An entity is susceptible to suppression if it has a suppression system. Lifeforms can have a Lifeform Suppression system. Vehicles can have a Vehicle Exposed Crew Suppression system. For details about configuring suppression, see "2.10.1 Configuring Suppression Effects" below.

### **2.10.1 Configuring Suppression Effects**

Suppression effects are configured by parameters in several components.

The suppression system contains a suppression actuator with several parameters to control its behavior.

- **suppression-insult-level.** The level of insult necessary to suppress the entity, that is, reach the level 1 threshold. This value should be set appropriately for the amount of insult caused by munitions, as defined in the damage-model portion of this descriptor. You can set this parameter in the Simulation Object Editor.
- **suppression-recovery-time.** The time required to recover from the level 1 suppression threshold to a not-suppressed state. This same time is required to recover to level 2 from level 3, and to level 1 from level 2. You can set this parameter in the Simulation Object Editor.
- **passing-rounds-suppress.** Determines whether rounds passing by the entity, but not necessarily detonating near it, are considered when detecting insults. You can turn off this effect to save computation time. Default: true.
- **max-detonation-distance-for-passing-rounds.** The maximum distance that the actuator looks for detonations. A line of fire that passes near the entity may detonate some distance beyond the entity. To find such lines, the actuator must look at distant detonations. This may result in the actuator processing many detonations. (The detonation manager makes a distance test for each entity for each detonation, so not much computation is saved by reducing this parameter value.)
- **hostile-fire-only.** If true, the actuator does not consider detonations and passing rounds as insults if they are from a friendly force. Default: true. While the default may be unrealistic in the case of HE shells landing near the entity, it avoids unintended suppression when nearby friendly entities are firing at the enemy.

The actuator descriptor includes a damage-model section that is just like the damage-model in a damage system. For each munition there are two parameters:

- **suppression-detonation-insult.** The maximum insult that this munition can cause, if it detonates at the entity location or passed directly over the entity.
- **suppression-detonation-range.** The maximum range at which the munition can have effect. The insult from the munition decreases linearly until it is 0 at the maximum range.

In ballistic gun controllers, **disabled-by-suppression** indicates whether the gun is susceptible to suppression. If true, the gun is not allowed to fire while the entity is suppressed. This is true for the handheld weapon systems, but it can also be selectively set for guns on vehicles. For example, tank machine guns are often operated by an exposed crew member, and could be suppressed by small arms fire even if the tank is otherwise unaffected by such fire. The handheld systems have this parameter set True, while other machine guns use a variable and so the parameter can be set in the Simulation Object Editor.

In sensor components, **suppression-degrades-sensor-performance-by** indicates how they are affected by suppression. It is a value from 0 - 1.0. A value of 0 means that the sensor is not degraded at all by suppression, while a value of 1.0 means that sensitivity is reduced to 0 by suppression. The value can be set in the Simulation Object Editor.



## 2.11. Configuring Task Execution Rules

Each simulation object OPE file contains a parameter called **task-execution-rules** that specifies a task execution rule configuration file. (The default rules file is `./data/simulationModelSets/<model_set>/vrfSim/taskRules/default-task-rules.tsk`.) The task execution rule file specifies whether tasks can be executed concurrently, and if so, which tasks are mutually exclusive.

The task execution rules file has these parameters:

- **all-tasks-exclusive**. This parameter enables (False) or disables (True) use of concurrent execution of tasks. When True, the Task Manager executes only one task at a time, and the **mutually-exclusive-task-groups** parameter is ignored.
- **mutually-exclusive-task-groups**. This parameter is a list of groups. Each group contains a list of tasks that cannot be executed at the same time. Any time a task from a particular group is started by the Task Manager, any currently running task that is part of the same group is interrupted. Tasks that are part of other groups, or not part of any group, are allowed to continue running.

The task names that are listed in the **mutually-exclusive-task-groups** parameter are the names that are registered with the task factory when a task is added to the system. Groups can contain any number of tasks, and there can be any number of groups.

Tasks are always considered to be mutually exclusive with themselves. For that reason, it is impossible to execute two tasks of the same type at the same time on a single simulation object.

 *default-task-rules.tsk* configures the tasks that come with VR-Forces and which were developed in C++. It does not include scripted tasks. Concurrency for scripted tasks is set when they are created.

### 2.11.1 Configuring Action Categories

In scripted tasks, action categories serve the same function as task rules for enforcing concurrency. Action categories are configured in `./data/simulationModelSets/<model_set>/vrfSim/taskRules/actionCategories.tsk`. An action category is specified with the entry:

```
<item>
  <first>Movement</first>
  <second>
    <first>Movement</first>
    <second>1</second>
  </second>
</item>
```

If the value in the `<first>` element matches a group in `./data/simulationModelSets/<model_set>/vrfSim/taskRules/default-task-rules.tsk`, scripted tasks that use that action category are unioned with the tasks specified in *default-task-rules.tsk* to form the complete set of mutually exclusive tasks.



 If you edit *actionCategories.tsk*, be sure to update the <count> element to reflect additions or deletions.





## 3. Movement Tasks for Entity-Level Scenarios

This chapter describes the movement tasks supplied with VR-Forces that apply to entity-level scenarios. Some of these tasks are also used in aggregate-level scenarios. "Chapter 5. Human and Crowd Tasks" on page 101 describes some movement tasks that are specific to human characters. "6.5. Movement Tasks for Unit Behaviors" on page 124 describes movement tasks for more advanced unit behaviors. "Generating Traffic (Pattern of Life)" discusses methods of generating air, sea, and ground traffic to provide realistic background activity in your scenarios.

Models with wheels and treads have the ability to rotate those at speeds relative to the entity's movement speed. Wheels can also articulate when the entity makes a turn to realistically represent the steering.

|                                                      |    |
|------------------------------------------------------|----|
| 3.1. Airdrop Along Route .....                       | 40 |
| 3.2. Animated Movement .....                         | 41 |
| 3.3. Come to Stop .....                              | 42 |
| 3.4. Convoy Along .....                              | 42 |
| 3.5. Convoy To .....                                 | 43 |
| 3.6. Crank Left and Crank Right .....                | 43 |
| 3.7. Establish Refueling Station .....               | 43 |
| 3.8. Fly Altitude .....                              | 45 |
| 3.9. Fly Heading and Altitude .....                  | 45 |
| 3.10. Fly Heading .....                              | 46 |
| 3.11. Fly Loiter Pattern .....                       | 47 |
| 3.12. Follow Entity .....                            | 48 |
| 3.12.1 How Fixed-Wing Entities Follow Entities ..... | 48 |
| 3.13. Hover .....                                    | 48 |
| 3.14. Keep Station .....                             | 49 |
| 3.15. Land At Airport .....                          | 50 |
| 3.16. Land at Airport Near Location .....            | 50 |
| 3.17. Land On Route .....                            | 51 |
| 3.18. Land On Runway .....                           | 51 |
| 3.19. Land (Vertical) .....                          | 52 |

|                                                      |    |
|------------------------------------------------------|----|
| 3.20. Move Along Route .....                         | 52 |
| 3.21. Move Along Route with Actions .....            | 53 |
| 3.22. Move Into Formation .....                      | 56 |
| 3.23. Move to Altitude .....                         | 56 |
| 3.24. Move to Bullseye Offset .....                  | 57 |
| 3.25. Move to Depth .....                            | 57 |
| 3.26. Move To .....                                  | 58 |
| 3.26.1 Adding Multiple Move To Tasks to a Plan ..... | 58 |
| 3.27. Move To (Plan Along Roads) .....               | 59 |
| 3.28. Move To (Plan Path) .....                      | 59 |
| 3.29. Orbit .....                                    | 60 |
| 3.30. Orbit Object .....                             | 61 |
| 3.31. Patrol Area .....                              | 61 |
| 3.32. Patrol Between .....                           | 61 |
| 3.33. Patrol Route .....                             | 62 |
| 3.34. Pattern Hold (Location) .....                  | 62 |
| 3.35. Pattern Hold (Waypoint) .....                  | 63 |
| 3.36. Refuel From .....                              | 63 |
| 3.37. Return To Base .....                           | 64 |
| 3.38. Rotary Wing Airlift Cargo to Location .....    | 64 |
| 3.39. Rotary Wing Land .....                         | 64 |
| 3.40. Rotary Wing Retrieve Cargo .....               | 65 |
| 3.41. Rotary Wing Unload Cargo .....                 | 65 |
| 3.42. Sail Heading .....                             | 65 |
| 3.43. Search and Rescue .....                        | 66 |
| 3.43.1 Fly Creeping Line Pattern .....               | 66 |
| 3.43.2 Fly Expanding Square Pattern .....            | 68 |
| 3.43.3 Fly Parallel Pattern .....                    | 69 |
| 3.44. Sweep Naval Mines .....                        | 71 |
| 3.45. Takeoff On Route .....                         | 71 |
| 3.46. Takeoff On Runway .....                        | 72 |
| 3.47. Takeoff (Vertical) .....                       | 73 |
| 3.48. Turn to Heading .....                          | 73 |
| 3.49. Zig-Zag to Heading .....                       | 74 |

## 3.1. Airdrop Along Route

The Airdrop Along Route task causes an aircraft to move along a specific route or to a specific plan variable. Upon reaching the route or variable, the aircraft disembarks paratroopers or entities such as vehicles. It specifies only when the entities embarked on the aircraft will parachute, and it can specify the number of "drop slots" available to those objects.

Drop slots are not the same as embarkation slots. On an aircraft, drop slots indicate the number of intervals at which entities can exit the aircraft with the Airdrop from Aircraft task. On entities, the drop slot indicates the order in which they will perform that task. Drop slots do not support units.

The entities must have a task for deploying the parachute or they will not exit the aircraft. See "7.1.5 Airdrop from Aircraft" on page 136 for details.

 The movement task Parachute to Target does not coordinate with Airdrop Along Route.

► **To assign an Airdrop Along Route task:**

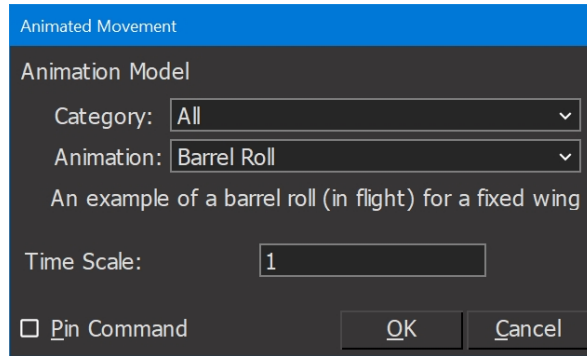
1. Select an aircraft that has entities embarked on it.
2. Choose **Task > Movement > Airdrop Along Route**. The Airdrop Along Route parameters display.
3. Specify the time to wait between each entity drop in seconds.
4. Use Drop Route to specify where the aircraft should begin the airdrop. It can be a route or a plan variable.
5. Optionally, you can select the Use Drop Slots check box and then specify the number of drop slots. One or more entities can exit the aircraft at each drop slot, as specified in the Airdrop from Aircraft task.
6. Click OK.

## 3.2. Animated Movement

Use the Animated Movement task to assign a scripted behavior to an entity. To use this task, you must first create a file that describes the movements that you want an entity to make. Animated movement files can be in comma-separated values (CSV) format or 3DS ASCII Export (ASE) format. You configure animated movement tasks in the Simulation Object Editor. For details, see "Configuring Animated Movement".

► **To assign an animated movement task:**

1. Select the entity to task.
2. Choose **Task > Movement > Animated Movement**. The Animated Movement parameters display (Figure 16).

**Figure 16. Animated Movement dialog box**

3. Optionally, filter the animation list by selecting a category in the Category list.
4. Select an animation in the Animation list. You are responsible for knowing that the animation is appropriate to the entity you selected.
5. Optionally, specify a time scale. By changing the time scale you can make the animation run faster or slower than specified in the animation file.
6. Click OK.

### 3.3. Come to Stop

The Come to Stop task stops a surface watercraft entity using a normal stop, an emergency stop, or a slow drift.

► **To assign a Come to Stop task:**

1. Select the entity.
2. Choose **Task > Movement > Come to Stop**. The Come to Stop parameters display.
3. Select an option for how to stop.
4. Click OK.

### 3.4. Convoy Along

The Convoy Along task commands a unit to move along a route in a convoy. When the task starts, the lead entity moves far enough along the route for the rest of the unit to create a column that ends approximately at the beginning of the route. Then the unit begins moving along the route.

 The Convoy Along task is valid only for ground-vehicle aggregates.

For more details about convoys, see "2.4.1 Convoy Tasks" on page 18.

► **To assign a Convoy Along task:**

1. Select the unit that you want to send in a convoy.
2. Choose **Task > Movement > Convoy Along**. The Convoy Along parameters display.

3. Specify the route to convoy along.
4. Click OK.

### 3.5. Convoy To

The Convoy To task commands a unit to move to a waypoint in a convoy. If you specify use of roads, VR-Forces uses the information in the road network to plan the convoy's movement. If you do not specify use of roads, the convoy moves directly across the terrain as an individual entity would move in a Move To task.

 The Convoy To task is valid only for ground-vehicle aggregates.

For more details about convoys, see "2.4.1 Convoy Tasks" on page 18.

► **To assign a Convoy To task:**

1. Select the unit that you want to send in a convoy.
2. Choose **Task > Movement > Convoy To**. The Convoy To parameters display.
3. Specify the waypoint to convoy to.

 When you specify use of roads, VR-Forces looks for roads over a fairly wide area and if one is found, will try to use it even if this results in an indirect approach to the waypoint. If there are no roads close to the route you would expect the unit to take to reach the waypoint, do not select this option.

4. Click OK.

### 3.6. Crank Left and Crank Right

The Crank Left and Crank Right movement tasks cause a fixed-wing aircraft to immediately turn to and fly a heading. The default is to change by 60 degrees, turning at 6 degrees per second.

- To assign the Crank Left or Crank Right task, select the entity and then choose **Task > Movement > Crank Left** or **Crank Right**. The entity changes its heading accordingly.

 You can also use the Crank Left and Crank Right buttons in the Entity Control Panel. See "2.2. The Entity Control Panel" on page 14.

### 3.7. Establish Refueling Station

Entities with a refueling system can establish a refueling station where other aircraft can refuel using the Refuel From task. The Establish Refueling Station task requires two waypoints: one to use as the Air Refueling Control Point (ARCP) and the other to use as the Air Refueling Initial Point (ARIP). These should be placed no farther than 50 kilometers apart in the terrain.

While performing the Establish Refueling Station task, the tanker aircraft first moves to the specified ARCP waypoint and then flies a holding pattern relative to that waypoint. The tanker then waits for aircraft to request refueling and queues the aircraft in the order it received the requests. It flies the holding pattern for a specified length of time.

An aircraft performing the Refuel From task (see "3.36. Refuel From" on page 63) then flies to the tanker's ARIP waypoint. When it is about four minutes away from the waypoint, that aircraft sends a request. The tanker either accepts or rejects the request. If it accepts the request, the refueling proceeds as described below. If it rejects the request, the tanker returns to its station and the aircraft performs the next task in its plan, if any. Reasons that the tanker would reject a request are:

- Its aviation-fuel resource is depleted.
- The aircraft has a different fuel supply type than what the tanker is carrying. The fuel supply type is identified by its SISO entity type.
- The aircraft requires a different refuel connection than is available on the tanker, either because the tanker is not equipped with it or it is disabled.

Tankers can be equipped with an air-to-air refueling system: boom, boom wingtip drogue, wingtip drogue, centerline boom drogue attachment, or multi-point. Aircraft can be equipped with an air refueling receiver boom or drogue system.

If the tanker accepts the request, it sends a message to the aircraft that indicates a rendezvous location and an approximate time of arrival. The aircraft makes altitude and speed adjustments to arrive at the location on time. Once the two aircraft are near the ARCP waypoint, the tanker sends a refueling location to the aircraft. The aircraft then moves to that location and the tanker and aircraft fly in a straight line. The tanker deploys its air-to-air refueling system and the aircraft moves to connect with it. During the refueling, the aircraft's movement|fuel resource increases as the tanker's aviation-fuel resource decreases.

When the aircraft's fuel is nearly full, they separate. The aircraft moves to the next statement in its plan, if any. The tanker returns to refueling station holding pattern and checks for other refuel requests. If another aircraft placed a request while the tanker was refueling the first aircraft, that aircraft orbits the ARIP waypoint until its request is accepted or rejected.

 You can also task a tanker to refuel itself. See "7.2. Internal Transfer of Fuel" on page 144.

► **To assign an Establish Refueling Station task:**

1. Select the entity.
2. Choose **Task > Movement > Establish Refueling Station**. The Establish Refueling Station parameters display.
3. Specify the ARCP waypoint.
4. Specify the ARIP waypoint.
5. Specify a heading, in degrees, and indicate whether it is relative to true or magnetic north.



6. Enter the speed at which the entity should travel.
7. Enter the altitude it should fly.
8. Select the desired Turn Direction.
9. Enter the length of time that the aircraft should spend at the refueling station waiting for refuel requests.
10. Specify the Pattern Type, either Time or Distance.
  - When using the Time pattern, you specify the length of time to complete each leg of the flight pattern.
  - When using the Distance pattern, you specify the length and width of the pattern.
11. Click OK.

### 3.8. Fly Altitude

The Fly Altitude task commands a fixed-wing or rotary-wing entity to fly to the specified altitude at a specified ascent or descent rate, using its current heading and ordered speed. When it reaches the altitude it continues to fly at that altitude at the current heading until it receives other commands. The entity will not exceed the limits for altitude or ascent/descent rate set in the OPD for the entity type.

You cannot use a Fly Altitude task to cause a fixed-wing entity to take off. A fixed-wing entity that is on the ground ignores this task.

If a rotary-wing entity is on the ground and is given this task, it rises to the specified altitude as it moves forward at its current heading. If a rotary-wing entity is in the air and is given this task with an altitude of zero, it descends to approximately 33 M and continues flying at that altitude at its current heading.

► **To assign a Fly Altitude task:**

1. Select a fixed-wing or rotary-wing entity.
2. Choose **Task > Movement > Fly Altitude**, or click the Fly Altitude button (). The Fly Altitude parameters display.
3. In the Altitude box, specify the altitude above sea level to fly to.
4. In the Climb/Descent Rate box, specify the rate.
5. Click OK.

### 3.9. Fly Heading and Altitude

The Fly Heading and Altitude task commands a fixed-wing or rotary-wing entity to fly to a specified altitude, at a specified ascent or descent rate, while turning to a specified heading at a specified turn rate. If you do not specify a heading or altitude, the entity continues to use the current value for that parameter.

While executing this task, the entity maintains its current speed. When it completes the command, it continues to fly at the new altitude and heading until given other commands. The entity will not exceed the limits on altitude, turn rate, and ascent/descent rate set in the OPD.

You cannot use a Fly Heading and Altitude task to cause a fixed-wing entity to take off. A fixed-wing entity that is on the ground ignores this task.

If a rotary-wing entity is on the ground and is given this task, it takes off and moves forward while it rises to the specified altitude and turns to the specified heading. If a rotary-wing entity is in the air and is given this task with an altitude of zero, it moves forward and turns to the new heading while it descends to approximately 33 M and continues flying at that altitude on the new heading.

 You can also quickly assign or change this task using the Entity Control Panel. For details, see "2.2. The Entity Control Panel" on page 14.

► **To assign a Fly Heading and Altitude task:**

1. Select a fixed-wing or rotary-wing entity.
2. Choose **Task > Movement > Fly Heading and Altitude**, or click the Fly Heading and Altitude button (). The Fly Heading and Altitude parameters display.
3. In the Heading box, specify the heading to fly to.
4. Next to the Heading box, indicate whether the entity should fly to the heading relative to true north or magnetic north.
5. In the Altitude box, specify the altitude above sea level to fly to.
6. In the Turn Rate box, specify the rate.
7. In the Climb/Descent Rate box, specify the rate.
8. Click OK.

## 3.10. Fly Heading

The Fly Heading task commands a fixed-wing or rotary-wing entity to turn to the specified heading at the specified turn rate. It maintains its current altitude and speed. When it is finished turning to the new heading, it continues to fly at this heading until it receives other commands. The entity will not exceed the turn rate specified in the OPD.

A fixed-wing entity that is on the ground ignores this task.

If a rotary-wing entity is on the ground and is given this task, it rises to approximately 25 meters above the terrain while simultaneously moving forward and turning to the specified heading. After turning to the new heading, it continues moving forward. Its altitude varies somewhat as it follows the terrain.

► **To assign a Fly Heading task:**

1. Select a fixed-wing or rotary-wing entity.
2. Choose **Task > Movement > Fly Heading**, or click the Fly Heading button (). The Fly Heading parameters display.
3. In the Heading box, specify the heading to fly to.
4. Next to the Heading box, indicate whether the entity should fly to the heading relative to true north or magnetic north.

5. In the Turn Rate box, specify the rate.
6. Click OK.

### 3.11. Fly Loiter Pattern

The Fly Loiter Pattern task commands an aircraft to fly in a loiter pattern, which can be a circle, figure eight, racetrack, cloverleaf, or a hold pattern. The entity loiters around the control point you specify. You also determine the turn direction, altitude, speed, and the duration of the loiter pattern flight. For the figure eight, racetrack, cloverleaf, and hold patterns, you specify the bearing and length to indicate the direction of the main axis and the space between loop centers.

The racetrack and hold patterns are both ovals. The approaches that an aircraft takes to enter them are more complicated than the other loiter patterns, and they also differ from each other.

- The hold pattern represents an air-traffic-control hold pattern, which requires the entity to fly a direct, parallel, or teardrop approach to that pattern.
- The racetrack pattern represents a typical military-style pattern. The aircraft flies along a tangent line to a point on either the inbound or outbound turn circle, then executes that turn to start the pattern.

The entity will not exceed the limits on altitude, air speed, or maximum lateral acceleration set in the OPD.

► **To assign a Fly Loiter Pattern task:**

1. Select a fixed-wing entity.
2. Choose **Task > Movement > Fly Loiter Pattern**. The Fly Loiter Pattern parameters display.
3. In the Loiter Point section, select the waypoint that marks the reference point for the entity's loiter pattern. This is the center of every pattern except for the oval patterns, racetrack and hold pattern. For those patterns, it is the point in the pattern where one of the turns begins.
4. In the Radius box, specify the radius of the loops made in the pattern.
5. Select the desired Turn Direction.
6. In the Altitude box, specify the altitude above sea level to fly the pattern.
7. In the Speed box, enter the speed to fly the pattern.
8. Specify the desired Duration Type:
  - Duration in Laps: Enter the number of laps the entity should fly the pattern. This must be a whole number.
  - Duration in Time: Enter the length of time that the entity should fly the pattern. The first box is the number of days, and the second box is hours, minutes, and seconds.
9. Specify the desired Loiter Pattern.
10. If the entity will fly in a figure 8, cloverleaf, racetrack, or hold pattern:

- a. In the Bearing box, enter the bearing of the main axis of the pattern.
  - b. In the Length box, enter the space between loop centers.
11. Click OK.

## 3.12. Follow Entity

You can order an entity to follow a simulation object. You can specify a distance and offset between the simulation objects. If you tell two or more entities to follow the same simulation object, make sure that you do not give them the same offset (see "Following Simulation Objects").

An entity continues to follow its target entity until:

- The task is overridden.
- The followed entity is destroyed.

If entity A is assigned to follow entity B, but it cannot find entity B, entity A continues to look for entity B until entity B becomes available, or until entity A is assigned another task.

The following entity tries to match the speed of the followed entity unless it has to hold back or catch up to accommodate changes in direction or terrain effects.

### ► To assign a Follow task:

1. Select the entity.
2. Choose **Task > Movement > Follow Entity**. The Follow Entity parameters display.
3. Optionally, filter the entity list.
4. In the Name box, type the name of the entity to follow, or select it from the list or on the terrain.
5. Optionally, type an offset in the Behind, Right, and Above text boxes. If you want an entity to follow in front, to the left, or below, put a minus sign (-) in front of the value.
6. Click OK.

### 3.12.1 How Fixed-Wing Entities Follow Entities

A fixed-wing entity can follow an entity that is on the ground, or might be on the ground, for example if it is following a fixed-wing entity that lands. In this case, to ensure that the follower does not crash, make sure that it is following above the followed entity and that the distance above is high enough to account for changes in the terrain.

## 3.13. Hover

Aircraft configured with the Fixed Wing VTOL movement system can perform a Hover task.

► **To assign a Hover task:**

1. Select the entity.
2. Choose **Task > Movement > Hover**. The Hover parameters display.
3. Select the location where the aircraft should land:
  - Click the location or enter the coordinates.
  - Select the waypoint.
4. Enter the height at which the entity should hover.
5. Click OK.

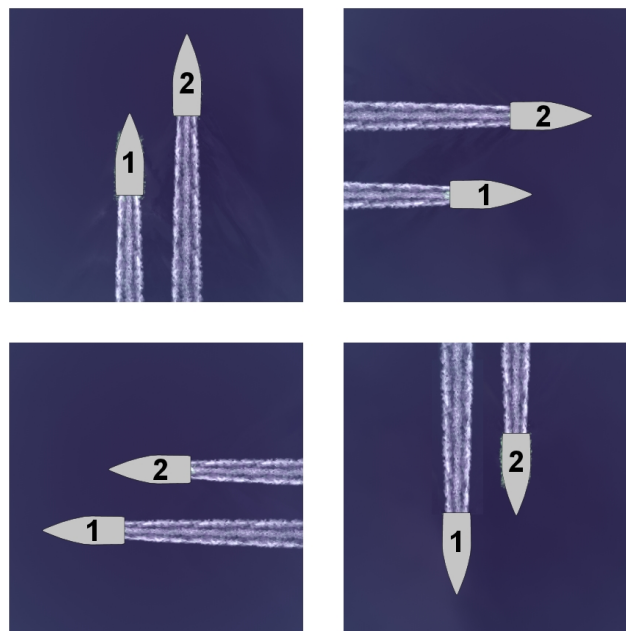
### 3.14. Keep Station

The Keep Station task directs a watercraft (surface or subsurface entity) to follow at a fixed distance and bearing from the leader's position, regardless of the leader's heading. If the station is located 200 yards to the northeast, the ship moves as needed to maintain that position, even if the leader changes the direction it is facing. For example, you can assign a destroyer to keep station between a carrier and the direction of probable enemy engagement, even when the task force changes course.

This task is different from the Follow Entity task, which positions the follower relative only to the leader's heading and causes the follower to adjust as needed when the leader changes direction.

Figure 17 illustrates a scenario where ship 1 is tasked to keep station with ship 2 at a 220° bearing relative to true north. This causes ship 1 to remain at a 220° offset from the center of ship 2, no matter what direction ship 2 is heading.

**Figure 17. Keep Station example**



The parameters you specify are:

- Name. The name of the entity to follow.
- Distance. Distance to maintain from the lead entity.
- Bearing. The bearing relative to the lead entity and either true or magnetic north where the watercraft should maintain its position.

► **To assign the Keep Station task:**

1. Select the entity that will be the follower.
2. Choose **Task > Movement > Keep Station**. The Keep Station parameters display.
3. Enter the parameters as desired.
4. Click OK.

### 3.15. Land At Airport

The Land at Airport task causes VR-Forces to land at specified aircraft on an appropriate runway based on weather conditions. This task requires use of a terrain database that has airport and runway data. For details, see "Aerodromes".

► **To assign a Land At Airport task:**

1. Select the entity.
2. Choose **Task > Movement > Land At Airport**, or click the Land At Airport button (). The Land At Airport parameters display.
3. Select the aerodrome where you want the aircraft to land.
4. Click OK.

### 3.16. Land at Airport Near Location

The Land at Airport Near Location task causes VR-Forces to find an airport near the specified location and land the specified fixed-wing aircraft on an appropriate runway based on weather conditions. This task requires use of a terrain database that has airport and runway data. For details, see "Aerodromes".

► **To assign a Land at Airport Near Location task:**

1. Select the fixed-wing entity.
2. Choose **Task > Movement > Land at Airport Near Location**, or click the Land At Airport button (). The Land At Airport Near Location parameters display.
3. Specify a range.
4. Specify the location in the dialog box or on the terrain. The list of aerodromes populates based on the location and range.
5. Select the aerodrome where you want the aircraft to land.
6. Click OK.

## 3.17. Land On Route

"2.8.6 How Fixed-Wing Entities Land" on page 31 describes how fixed-wing entities land and the steps to design a landing approach. The Land On Route task automates the process of landing a fixed-wing aircraft. However, you must specify a route to serve as the runway.

When the task is invoked, VR-Forces defines an approach route for the plane to move along. The length of this route is either the plane's altitude or the length needed to slow the plane down to landing speed, whichever is greater. The route is created and the plane moves along the route, coming to landing speed. At the end of the approach route, the plane is tasked to move along the runway.

 Although it is possible to land an entity on another entity, such as an aircraft carrier, if you want to embark an entity, you must use the Embark task, not a landing task.

► **To assign a Land On Route task:**

1. Select the entity.
2. Choose **Task > Movement > Land On Route**. The Land On Route parameters display.
3. Select the route in the list or on the terrain, or enter the name of the route where you want it to land in the Route box.
4. Click OK.

## 3.18. Land On Runway

The Land On Runway task automates the process of landing a fixed-wing aircraft. There must be a runway in the terrain for the aircraft to land on, however; see "Aerodromes". If the terrain does not have an aerodrome with runways that you want to use, you can add a route to the scenario and use that instead; see "3.17. Land On Route" above.

When the task is invoked, VR-Forces defines an approach route for the plane to move along. The length of this route is either the plane's altitude or the length needed to slow the plane down to landing speed, whichever is greater. The route is created and the plane moves along the route, coming to landing speed. At the end of the approach route, the plane is tasked to move along the runway.

- 
- Although it is possible to land an entity on another entity, such as an aircraft carrier, if you want to embark an entity, you must use the Embark task, not a landing task.
  - There is no logic to determine whether the runway is appropriate for the aircraft. For example, VR-Forces will not prevent a C-5 Galaxy from landing at a small municipal airport.

► **To assign a Land On Runway task:**

1. Select the entity.
2. Choose **Task > Movement > Land On Runway**, or click the Land On Runway button (  ). The Land On Runway parameters display.
3. In the Airport box, enter the ICAO code of the aerodrome. You can type the code, select a managed aerodrome in the Objects List, or select from the list of aerodromes at the bottom of the window. In the last case, any managed aerodromes are listed at the top of the list, and then all other aerodromes in the terrain are listed below those.
4. Select the specific runway on which the entity should land.
5. Click OK.

 If you want to design an explicit landing using a route, see "2.8.6 How Fixed-Wing Entities Land" on page 31, which describes how fixed-wing entities land and the steps to design a landing approach.

### 3.19. Land (Vertical)

Aircraft configured with the Fixed Wing VTOL movement system can perform a vertical landing task.

► **To assign a Land (Vertical) task:**

1. Select the entity.
2. Choose **Task > Movement > Land (Vertical)**. The Land (Vertical) parameters display.
3. Select the location where the aircraft should land:
  - Click the location or enter the coordinates.
  - Select the waypoint.
4. Click OK.

### 3.20. Move Along Route

By default, a simulation object moves along a route from its nearest vertex to its end point.

- If you edit a route while a simulation object is moving to the starting point of that route as part of a Move Along Route task, the simulation object moves to the nearest vertex, whether it is the starting point, or not.
- If a human is moving along a route outside of a navigation area and you move it away from the route or change the route, the entity plans a path to the route vertex it is moving to rather than moving to the nearest point on the route and then proceeding to the target vertex.



- If you want a simulation object to repeatedly move along the same route from beginning to end (for example, to follow a circular route), be sure to clear the Start at Closest Vertex check box. If you do not, after the first time the simulation object moves along the route, it keeps going to the end vertex, because that is the closest vertex to where it is starting the new Move Along Route task.
- When you task an air platform entity to follow a route, be sure to take into account the altitude of the route's vertices and the terrain the entity has to pass over to complete the task. (For more information, see "2.8.4 Fixed-Wing Entity Movement on the Ground and in the Air" on page 29 and "2.9. Rotary-Wing Entity Tasks and Behaviors" on page 32.)
- Units do not generate subordinate routes until they reach the beginning of the route.
- A unit starts moving along a route when the leading edge of its formation is at the first point of the route. It is considered finished when the leading edge reaches the last point of the route.
- In addition to routes, you can use other linear tactical graphics in this task, including corridors, lines, pedestrian paths, and supply routes.

► **To assign a Move Along Route task:**

1. Select the simulation object.
2. Choose **Task > Movement > Move Along Route**, or click the Move Along Route button . The Move Along Route parameters display.
3. In the Name box, enter the route you want the simulation object to follow.
4. To have the simulation object move from the end of the route to the beginning, select Reverse Direction.
5. To have a ground vehicle use the road driving movement strategy, select the Treat Route as Road check box. The route is treated as a two-lane bidirectional road. Ground vehicles compute curves to go around the vertices of the route. (This applies only to entity-level scenarios.)



If you select Treat Route as Road, the entity must be within 10 meters of the route or the task will fail.

6. To have the simulation object start the route at the beginning, clear the Start at Closest Vertex check box.
7. Click OK.

## 3.21. Move Along Route with Actions

Use Move Along Route with Actions task to assign actions to an entity that is following a route. The tasked entity performs the actions at points in the route that you specify, then continues to follow the route.

- By default, a simulation object executing a Move Along Route with Actions task goes to the beginning of the route, rather than starting the route at its nearest vertex.
- If a human entity is moving along a route outside of a navigation area and you move it away from the route or change the route, the entity plans a path to the route vertex it is moving to rather than moving to the nearest point on the route and then proceeding to the target vertex.
- If you want a simulation object to repeatedly move along the same route from beginning to end (for example, to follow a circular route), be sure to clear the Start at Closest Vertex check box. If you do not, after the first time the simulation object moves along the route, it keeps going to the end vertex, because that is the closest vertex to where it is starting the new Move Along Route with Actions task.
- The Move Along Route with Actions task includes a Specify Time on Target option that lets you set the time when the tasked entity must reach the target location on the route (TL). When you set a time on target, the task employs a hold location (HL) where the tasked entity can wait or circle to use up extra time and complete the task at the specified time.
- You can control the tasked entity's speed by setting Ordered Speed, which offers Enforced and Suggested options. However, if you set both a time on target and an enforced ordered speed, the entity might not be able to complete the task.
- The Abandon if Time on Target Cannot Be Achieved option is available when you use Specify Time on Target. If you select the option, the tasked entity abandons the task if it determines that it cannot reach the target location within 10 seconds of the specified time.
- When you task an air platform entity to follow a route, take into account the altitude of the route's vertices and the terrain the entity has to pass over to complete the task. If you are using an existing ground route, you can set an ordered altitude that puts the tasked entity at the height you want, or raise vertices to enable the entity to follow the route. (For more information, see "2.8.4 Fixed-Wing Entity Movement on the Ground and in the Air" on page 29 and "2.9. Rotary-Wing Entity Tasks and Behaviors" on page 32.)
- For air platforms, the flight path shown in the terrain profile in the Move Along Route with Actions dialog box is an approximation, not a precise rendering.
- In addition to routes, you can use other linear tactical graphics in this task, including air axes, air corridors, lines, and supply routes.

► **To prepare to assign the task:**

1. Identify the entity that will follow the route, or use the Create Object Palette to create one.

 If you create an airplane in the air, set its altitude to a few hundred meters to keep it from stalling.

2. Identify the route (or other linear tactical graphic) to be followed, or use the Create Object Palette to create one.

3. If the action in the task you are creating relies on tactical graphics such as target points or areas, use the Create Object Palette to create them along the route.

 To drop a bomb, you must create a target point on the route. You cannot drop a bomb on the route's target location, which is simply the end point of the route.

► **To assign a Move Along Route with Actions task:**

1. Select the entity that will move along the route and perform the actions.
2. Choose **Plan > Task > Movement > Move Along Route with Actions**, or click the Move Along Route With Actions button . The Move Along Route with Actions parameters display.
3. In the Name box, enter the route or other linear tactical graphic that the entity should follow. The dialog box displays a terrain profile of the route you selected and a numbered list of the route's vertices.
4. Right-click any of these to add an action:
  - In the Move Along Route with Actions dialog box, any location on the route in the terrain profile.
  - In the Action list, any vertex.
  - In Plan View, any location on the route.
5. In the context menu, choose **Insert Task** or **Insert Set** and select an option. For example, you might choose **Insert Task > Engagement > Release Bomb on Target** or **Insert Set > Action > Ordered Altitude**.
6. In the new dialog box, either:
  - Select the target of the action, or
  - Enter required information and click OK.

The action appears in the Action list in the Move Along Route with Actions dialog box.

7. Optionally, set a speed for the entity:
  - a. Right-click a point on the route and then select **Insert Set > Action > Ordered Speed**.
  - b. In the Ordered Speed dialog box, enter a speed. For aircraft entities, this is the ground speed.
  - c. Optionally, select Enforced under Ordered Speed. (If you also Specify Time on Target below, the entity might not be able to achieve the time on target at the ordered speed unless you change or remove the ordered speed later in the task. You can do that by adding another **Insert Set > Action > Ordered Speed** for a different speed, or even for the same speed by selecting Suggested rather than Enforced.)
  - d. Click OK. VR-Forces updates the estimated times in the Move Along Routes with Actions dialog box.

8. Optionally, select Specify Time on Target and set the time at which you want the simulation object to reach the target point in the route. When you enable Specify Time on Target, VR-Forces recalculates the estimated times in the Action list, counting backward from the Time on Target at the route's target location. (Estimated times before the hold location do not change.) When it is not possible for the tasked entity to achieve an action in the time allowed, the numbers in the Time on Target field and the estimates for the actions in question turn red.
9. Optionally, select Abandon if Time on Target Cannot Be Achieved if you want the tasked entity to give up on the task if it estimates that it cannot complete the task by the time you specified in Time on Target (within 10 seconds).
10. To have the simulation object move from the end of the route to the beginning, click Reverse Route.
11. To reverse the order of the actions along the route, click Reverse Actions. You can also reorder by right-clicking an action or the target location in the Actions list and selecting Move.
12. To have the simulation object start the route at the nearest vertex rather than moving to the beginning of the route, select the Start at Closest Vertex check box.
13. Click OK.

## 3.22. Move Into Formation

You can order a ground unit to move to a formation at a specified location. Moving to a formation is not the same as setting formation. Setting formation changes the formation instantly. Moving to a formation causes the members of the unit to travel through the terrain into the new formation.

► **To assign a Move Into Formation task:**

1. Select the unit.
2. Choose **Task > Movement > Move Into Formation**. The Move Into Formation parameters display.
3. Select the desired formation from the list.
4. Specify the location at which the unit is to move into formation by entering coordinates, or by clicking on the terrain.
5. Specify the heading the unit is to assume at the target location.
6. Click OK.

## 3.23. Move to Altitude

You can order an aircraft entity to move to an altitude. When you assign this task, the entity moves to the new altitude at the same coordinates. Fixed-wing aircraft spiral to the new coordinates. Rotary-wing entities rise or descend. Moving to an altitude is not the same as setting altitude. Setting altitude moves an entity instantaneously. It is also not the same thing as Fly Altitude. At the conclusion of Move to Altitude, an aircraft circles or hovers at the new altitude. At the conclusion of Fly Altitude, an aircraft continues moving at its current heading.

 Fixed-wing entities that are on the ground cannot respond to a Move to Altitude task. If they are given this task, a message is printed to the object console and the task is ignored.

► **To assign a Move to Altitude task:**

1. Select the entity.
2. Choose **Task > Movement > Move to Altitude**, or click the Move to Altitude button . The Move to Altitude parameters display.
3. Type the altitude that you want the entity to move to.
4. Click OK.

## 3.24. Move to Bullseye Offset

Use the Move to Bullseye Offset task to send an aircraft to a bearing and range relative to a bullseye tactical graphic in the scenario.

► **To assign a Move to Bullseye Offset task:**

1. Select the entity.
2. Choose **Task > Movement > Move to Bullseye Offset**. The Move to Bullseye Offset parameters display.
3. Select the bullseye.
4. Enter the bearing from the bullseye to the location where you want the aircraft to move to. Indicate whether the heading is relative to true or magnetic north.
5. Enter the range.
6. Click OK.

## 3.25. Move to Depth

The Move to Depth task lets you specify the depth for subsurface entities. When an entity moves to a depth, it continues any horizontal movement that was in effect prior to starting the task.

 You can also quickly assign or change this task using the Entity Control Panel. For details, see "2.2. The Entity Control Panel" on page 14.

► **To assign a Move to Depth task:**

1. Select the entity.
2. Choose **Task > Movement > Move to Depth**. The Move to Depth parameters display.
3. Select an option for the depth. If you choose a specific depth, type a depth in the Target Depth box.
4. Specify the ascent/descent rate.
5. Click OK.

## 3.26. Move To

You can order an entity to move to a location or point. If navigation data is present, simulation objects take it into account in planning their path to the location.

If you add Move To tasks to a plan, you have more options than when you give an independent task. You can

Moving to a location or point is not the same as setting location with a set data request. Setting location moves a simulation object instantaneously. Moving to a location or point causes a simulation object to travel through the terrain.

If you want a ground platform entity to use the road network to move to a location, use Move To (Plan Along Roads). For details about road movement, see "2.7. Entity Movement On Roads and Off Roads" on page 26.

 Rotary-wing entities are not able to reach moving waypoints or land on moving ships. A rotary-wing entity will approach a given waypoint or landing point, but will never completely reach the goal.

### ► To assign a Move To task:

1. Select the simulation object.
2. Choose **Task > Movement > Move To**, or click the Move To button (). The Move To parameters display.
3. Optionally, click Select Location or Select Object.
4. Select the destination:
  - Click the location to move to or enter the coordinates.
  - Select the waypoint or point tactical graphic.
5. For a rotary- or fixed-wing aircraft moving to a location, you can set the altitude, as described in "Specifying an Object's Altitude".

 If the destination is a location and the task is for an aircraft that is in the air, the default is for it to remain at the same altitude. If there are obstacles in the terrain, such as buildings, you might want to change the altitude to a height that will prevent the entity from crashing.

6. Click OK.

### 3.26.1 Adding Multiple Move To Tasks to a Plan

When you add a Move to Location task to a plan, you can add multiple locations without clicking OK and selecting the task again from the menu. Also, since in this method you cannot specify the altitude for each location, you can specify a height above the terrain for the altitude of the succeeding points.

► **To add multiple Move To tasks to a plan:**

1. In the Plan window for a simulation object, choose **Task > Movement > Move To**. The Move To parameters display.
2. Select Each Click Generates A Task.
3. Select the destination:
  - Click the location to move to or enter the coordinates.
  - Select the waypoint or point tactical graphic.
4. For a rotary- or fixed-wing aircraft moving to a location, you can set the altitude, as described in "Specifying an Object's Altitude".
5. Click OK.

### 3.27. Move To (Plan Along Roads)

The Move to Waypoint (Plan Along Roads) task directs an entity to move to a waypoint and use the road network if it is available.

When an entity receives this task, it determines if there is a road network near by. If there is, it moves directly to the nearest point on the road network. Then it moves along the roads using road driving behavior until it gets to the point on the road that is closest to the destination. Then the entity moves off the road directly to the destination (unless you select Stay On Road When Complete).

For more information about road driving and how the path planning tasks work, see "2.7. Entity Movement On Roads and Off Roads" on page 26.

► **To assign a Move To (Plan Along Roads) task:**

1. Select the entity.
2. Choose **Task > Movement > Move To (Plan Along Roads)**. The Move To (Plan Along Roads) parameters display.
3. Select the destination:
  - Click the location to move to or enter the coordinates.
  - Select the waypoint or point tactical graphic.
4. Optionally, you can select Stay On Road When Complete. If the destination is not on a road, the entity stops when it reaches the point on the road network that is closest to the destination.
5. Click OK.

### 3.28. Move To (Plan Path)

The Move to (Plan Path) task applies to surface and subsurface entities. It plots a route to the waypoint based on the depth of water between the starting point and ending point. You specify the minimum depth required for the ship to maneuver. You can specify that the depth be based on the ship's draft or you can enter a specific depth value. The ship's draft is determined by calculating the distance its bounding box extends below its origin, as set in the Simulation Object Editor.

If you give this task to a submarine, VR-Forces assumes that it is moving on the surface.



- For this task to work, your terrain must include S-57 data or depth information provided in a similar way.
- The formula for calculating the draft is:

$$\text{draft} = \text{height}/2 - \text{offset}(\text{Up})$$

where *height* and *offset* are set in the Size group box in the Simulation Object Editor.

► **To assign a Move to (Plan Path) task:**

1. Select the entity.
2. Choose **Task > Movement > Move to (Plan Path)**. The Move to (Plan Path) parameters display.
3. Select the destination:
  - Click the location to move to or enter the coordinates.
  - Select the waypoint or point tactical graphic.
4. Specify how you want VR-Forces to determine the minimum depth of water for the route.
5. If you want to see the path displayed, select Display Route.
6. Click OK.

## 3.29. Orbit

The Orbit task allows an aircraft to circle around a point without needing to create a route for it to follow.

► **To assign an Orbit task:**

1. Select the entity that you want to fly the orbit.
2. Choose **Task > Movement > Orbit**. The Orbit parameters display.
3. Specify the coordinates of a point on the terrain around which the entity should orbit. You can click on the terrain to specify the point. You can also expand the location group box by clicking the arrow next to the coordinates and altitude, then enter the location for the entity to orbit.
4. Specify the altitude at which to fly.
5. Specify the radius of the circle.
6. Select or clear the Clockwise option to specify the direction of travel.
7. Click OK.



### 3.30. Orbit Object

The Orbit Object task causes an aircraft to fly a circular pattern around a simulation object or point tactical graphic. If the object moves, the aircraft moves to maintain the orbital relationship.

► **To assign an Orbit Object Task:**

1. Select the entity that you want to orbit.
2. Choose **Task > Movement > Orbit Object**, or click the Orbit Object button (). The Orbit Object parameters display.
3. Select the simulation object or tactical graphic around which you want the entity to orbit.
4. In the Orbit Radius box, specify the radius of the orbit.
5. To orbit clockwise, select the Clockwise check box.
6. Click OK.

### 3.31. Patrol Area

The Patrol Area task is for watercraft (surface and subsurface entities). This allows you to have a ship patrol an area such as a harbor. The task uses a line between the two most distant points that falls within the area and then moves the ship in a zig-zag pattern across the area with that line as the spine of the pattern.

The ship can be outside of the area at the beginning of the task. It will move into the area to patrol it.

► **To assign a Patrol Area task:**

1. Select the entity.
2. Choose **Task > Movement > Patrol Area**. The Patrol Area parameters display.
3. Select the area for the ship to patrol.
4. Click OK.

### 3.32. Patrol Between

An entity can patrol between two point objects, simulation objects, or both. When a simulation object patrols between two objects, it goes to the starting object and then moves back and forth between the two objects.

► **To assign a Patrol Between task:**

1. Select the entity.
2. Choose **Task > Movement > Patrol Between**, or click the Patrol Between button (). The Patrol Between parameters display.
3. In the Name box for Point 1, type the name of the first object to patrol between, or select an object in the list or on the terrain.

4. In the Name box for Point 2, type the name of the second object to patrol between, or select an object in the list or on the terrain.
5. Click OK.

### 3.33. Patrol Route

When an entity patrols along a route, it moves to the nearest vertex of the route, to the end point, back to the beginning of the route, and then continues going back and forth along the route until given another command.

If a human is moving along a route outside of a navigation area and you move it away from the route or change the route, the entity plans a path to the route vertex it is moving to rather than moving to the nearest point on the route and then proceeding to the target vertex.

 For more information about moving along routes, see the issues discussed in "3.20. Move Along Route" on page 52.

► **To assign a Patrol Route task:**

1. Select the simulation object.
2. Choose **Task > Movement > Patrol Route**, or click the Patrol Route button (). The Patrol Along Route parameters display.
3. In the Name box, type the name of the route to patrol, or select the route in the list or on the terrain.
4. Click OK.

### 3.34. Pattern Hold (Location)

The Pattern Hold (Location) task causes a fixed-wing or rotary-wing entity to execute a racetrack hold pattern at the specified location.

In a standard racetrack, the aircraft tries to spend one minute per leg. Therefore, the length of the leg varies with the speed. The default turn radius is three degrees/second. In a custom racetrack, you specify the length and width of the legs.

The aircraft enters the pattern using direct, parallel, or teardrop entry. It determines the approach based on its location relative to the desired pattern. If you set the plane's object console to Info, the chosen entry method is displayed.

It may take an aircraft one or two loops before it settles into the desired pattern.

► **To assign a Pattern Hold (Location) task:**

1. Select a fixed-wing or rotary-wing entity.
2. Choose **Task > Movement > Pattern Hold (Location)**, or click the Pattern Hold (Location) button (). The Pattern Hold (Location) dialog box opens.
3. Enter the Holding Fix (location) or select it on the terrain.
4. Specify the heading, speed, and altitude for the hold pattern.

5. Specify the turn direction for the hold pattern.
6. For the Pattern Type, select Time or Distance. If you select Time, specify the amount of time per leg of the pattern. If you select Distance, specify a length and width.
7. Click OK.

### 3.35. Pattern Hold (Waypoint)

The Pattern Hold (Waypoint) task causes a fixed-wing or rotary-wing entity to execute a racetrack hold pattern at the specified waypoint.

In a standard racetrack, the aircraft tries to spend one minute per leg. Therefore, the length of the leg varies with the speed. The default turn radius is three degrees/second. In a custom racetrack, you specify the length and width of the legs.

The aircraft enters the pattern using direct, parallel, or teardrop entry. It determines the approach based on its location relative to the desired pattern. If you set the aircraft's object console to Info, the chosen entry method is displayed.

It may take an aircraft one or two loops before it settles into the desired pattern.

► **To assign a Pattern Hold (Waypoint) task:**

1. Select a fixed-wing or rotary-wing entity.
2. Choose **Task > Movement > Pattern Hold (Waypoint)**, or click the Pattern Hold (Waypoint) button . The Pattern Hold (Waypoint) parameters display.
3. In the Holding Fix section, select the waypoint.
4. Specify the heading, speed, and altitude for the hold pattern.
5. Specify the turn direction for the hold pattern.
6. For the Pattern Type, select Time or Distance. If you select Time, specify the amount of time per leg of the pattern. If you select Distance, specify a length and width.
7. Click OK.

### 3.36. Refuel From

The Refuel From task causes an aircraft with the Air Refueling Receiver system to rendezvous with an air tanker that is performing the Establish Refueling Station task. See "3.7. Establish Refueling Station" on page 43 for more information.

► **To assign a Refuel From task:**

1. Select a fixed-wing entity.
2. Choose **Task > Movement > Refuel From**. The Refuel From parameters display.
3. Select the air tanker that the aircraft should refuel from.
4. Click OK.

### 3.37. Return To Base

You can quickly task an entity to return to its assigned base aerodrome (RTB). This task requires that you assign a base aerodrome using the Base set data request, as described in "Base", or the Edit dialog box for the aircraft. If it is a managed aerodrome, at least one of its runways must be set to active to allow it to land. See "Runway Active".

When you task an entity to RTB, it immediately plans a route back to its base. It does not change altitude until it is 10 NM from the base, at which point it makes a landing. The entity is not removed from the scenario after it lands on a runway at its base.

► **To task an aircraft to return to base:**

1. Select the entity.
2. Choose **Task > Movement > Return to Base**, or click the Return to Base button (). The entity returns to its assigned air base.

### 3.38. Rotary Wing Airlift Cargo to Location

The Rotary Wing Airlift Cargo to Location task causes an unmanned rotary-wing aircraft to move to a list of objects, embark them onto itself, move to a location, and disembark the objects. The objects must be configured as embarkable.

If the total weight of the objects plus the weight of the aircraft is greater than the aircraft can carry, the aircraft will embark the objects, but be unable to fly.

The unmanned aircraft needs to use the Quadcopter or Quadcopter Heavy Lift movement dynamics system.

► **To assign a Rotary Wing Airlift Cargo to Location task:**

1. Select an unmanned aircraft.
2. Choose **Task > Movement > Rotary Wing Airlift Cargo to Location**. The Rotary Wing Airlift Cargo To Location parameters display.
3. Optionally, filter the list of object types.
4. Type the name of an object in the Name box, or in the Cargo Objects list select one or more objects.
5. In the Cruise Altitude box, specify the altitude at which you want the aircraft to fly.
6. Specify the destination location by typing in the coordinates, or click the Destination Location filter button () to give the Destination Location focus, and click on the terrain to specify the destination location.
7. Click OK.

### 3.39. Rotary Wing Land

The Rotary Wing Land task automates the process of landing a rotary-wing aircraft. Rotary-wing aircraft can land on points.

 Although it is possible to land an entity on another entity, such as an aircraft carrier, if you want to embark an entity, you must use the Embark task, not a landing task.

► **To assign a Rotary Wing Land task:**

1. Select the entity.
2. Choose **Task > Movement > Rotary Wing Land**. The Rotary Wing Land parameters display.
3. In the Name box, type the name of the waypoint on which you want the entity to land, or select the waypoint in the list or on the terrain.
4. Click OK.

### 3.40. Rotary Wing Retrieve Cargo

The Rotary Wing Retrieve Cargo task causes an unmanned aircraft to move to one or more cargo items and embark them onto itself. It hovers at the location of the last item retrieved until given further tasks.

► **To assign a Rotary Wing Retrieve Cargo task:**

1. Select an unmanned aircraft.
2. Choose **Task > Movement > Rotary Wing Retrieve Cargo**. The Rotary Wing Airlift Cargo To Location parameters display.
3. Optionally, filter the list of object types.
4. Type the name of an object in the Name box or in the Cargo Objects list, select one or more objects.
5. In the Cruise Altitude box, specify the altitude at which you want the aircraft to fly.
6. Click OK.

### 3.41. Rotary Wing Unload Cargo

The Rotary Wing Unload Cargo task causes a rotary-wing aircraft to unload (disembark) its cargo at the current location of the aircraft.

► **To assign a Rotary Wing Unload Cargo task:**

1. Select an unmanned aircraft.
2. Choose **Task > Movement > Rotary Wing Unload Cargo**. The aircraft unloads its cargo.

### 3.42. Sail Heading

The Sail Heading task causes a surface or subsurface entity to sail towards the specified heading. After turning to the heading, the entity moves at its ordered speed indefinitely.

 You can also quickly assign or change this task using the Entity Control Panel. For details, see "2.2. The Entity Control Panel" on page 14.

► **To assign a Sail Heading task:**

1. Select the entity.
2. Choose **Task > Movement > Sail Heading**, or click the Sail Heading button (). The Sail Heading parameters display.
3. Specify a heading in the indicated units.
4. Next to the Final Heading box, indicate whether the entity should sail to the heading relative to true north or magnetic north.
5. Specify the diameter of the turn.
6. Specify the direction of the turn:
  - Toward Heading. Chooses the most direct turn.
  - Port. Turns to port until it reaches the heading.
  - Starboard. Turns to starboard until it reaches the heading.
7. Click OK.

## 3.43. Search and Rescue

There are three search and rescue patterns available to aircraft: creeping line, expanding square, and parallel.

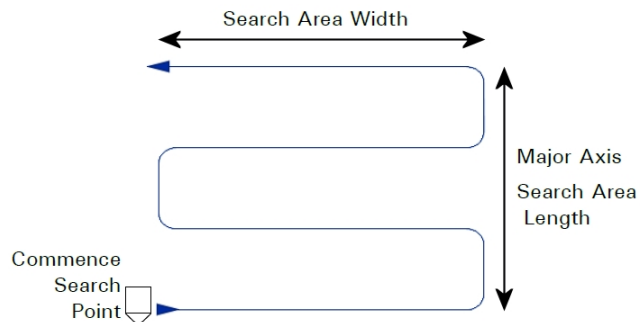
### 3.43.1 Fly Creeping Line Pattern

The Fly Creeping Line Pattern search and rescue task causes an aircraft to fly a pattern starting at the Commence Search Point. You specify the Major Axis for the direction in which the search expands (creeps). The search legs are perpendicular to this axis. The Search Area Length is the size of the search area (box) in the direction of the Major Axis, and the Search Area Width is the size perpendicular to the Major Axis. The search area is either to the right or the left of the Commence Search Point.

The entity will not exceed the limits on altitude and air speed set in the OPD.

Figure 18. Fly Creeping Line Pattern dialog box

Figure 19. Creeping Line Pattern, search right of CSP



► **To assign a Fly Creeping Line Pattern search and rescue task:**

1. Select an aircraft.
2. Choose **Task > Movement > Search and Rescue > Fly Creeping Line Pattern**. The Fly Creeping Line Pattern parameters display (Figure 18).
3. Specify the parameters:
  - In the Commence Search Point section, select the waypoint.
  - In the Major Axis box, enter the axis for the search. Next to this box, indicate whether the entity should fly relative to true north or magnetic north.
  - In the Search Area Length box, enter the length of the search area. This is measured along the Major Axis.
  - In the Track Spacing box, enter the spacing between the search legs.

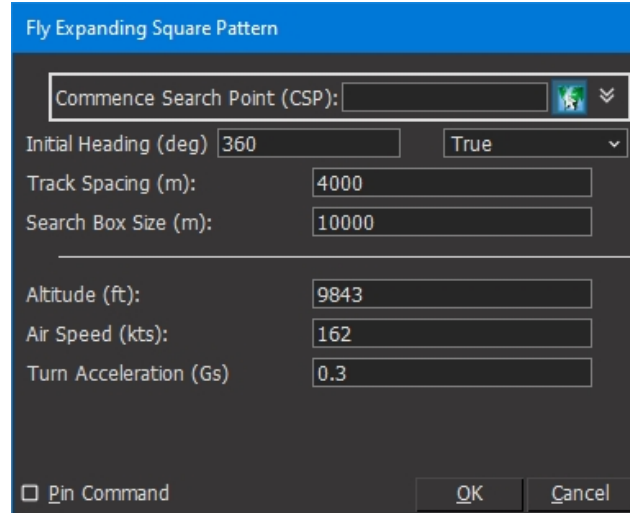
- Select the Search Box Is Right of CSP check box to search the area to the right of the Commence Search Point. Clear the check box to search the area to the left of the waypoint.
  - In the Search Area Width dialog box, enter the width of the search area. This is the width measured from the Commence Search Point.
  - In the Altitude box, specify the altitude above sea level to fly the search and rescue pattern.
  - In the Air Speed box, enter the speed to fly the pattern.
  - In the Turn Acceleration box, enter the maximum Gs for the turns. This acceleration, together with the selected air speed, determines the radius of the loops in the pattern. You must set the air speed and turn acceleration such that the aircraft can turn 180 degrees within the track spacing distance.
4. Click OK.

### 3.43.2 Fly Expanding Square Pattern

The Fly Expanding Square Pattern search and rescue task causes an aircraft to fly a pattern that starts at the Commence Search Point. That point is the center of the search, and the entity moves around it in an expanding series of right turns until the search box is covered.

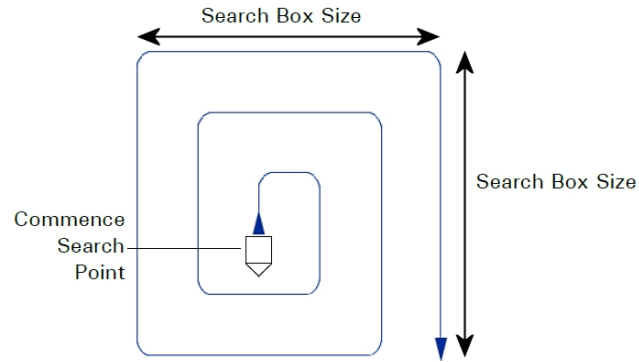
The entity will not exceed the limits on altitude and air speed set in the OPD.

**Figure 20. Fly Expanding Square Pattern dialog box**

The image shows a software dialog box titled "Fly Expanding Square Pattern". It contains several input fields and a checkbox. The "Commence Search Point (CSP)" field has a map icon and a dropdown arrow. The "Initial Heading (deg)" field is set to 360, and there is a "True" checkbox next to it. The "Track Spacing (m)" field is set to 4000. The "Search Box Size (m)" field is set to 10000. The "Altitude (ft)" field is set to 9843. The "Air Speed (kts)" field is set to 162. The "Turn Acceleration (Gs)" field is set to 0.3. At the bottom left is a checkbox labeled "Pin Command". At the bottom right are "OK" and "Cancel" buttons.

| Fly Expanding Square Pattern         |                             |
|--------------------------------------|-----------------------------|
| Commence Search Point (CSP):         | [Map Icon] [Dropdown Arrow] |
| Initial Heading (deg)                | 360 [True] [Dropdown Arrow] |
| Track Spacing (m):                   | 4000                        |
| Search Box Size (m):                 | 10000                       |
| <hr/>                                |                             |
| Altitude (ft):                       | 9843                        |
| Air Speed (kts):                     | 162                         |
| Turn Acceleration (Gs)               | 0.3                         |
| <hr/>                                |                             |
| <input type="checkbox"/> Pin Command | [OK] [Cancel]               |



**Figure 21. Expanding Square Pattern**

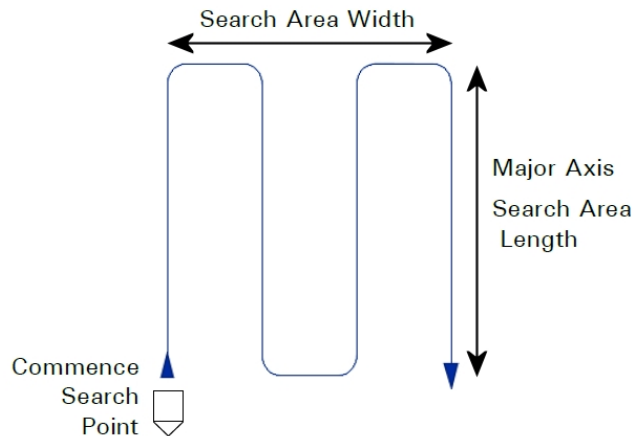
► **To assign a Fly Expanding Square Pattern search and rescue task:**

1. Select an aircraft.
2. Choose **Task > Movement > Search and Rescue > Fly Expanding Square Pattern**. The Fly Expanding Square Pattern parameters display (Figure 20).
3. Specify the parameters:
  - In the Commence Search Point section, select the waypoint.
  - In the Initial Heading box, enter the heading at which the entity should begin the pattern. Next to this box, indicate whether the entity should fly to that heading relative to true north or magnetic north.
  - In the Track Spacing box, enter the spacing between the search legs.
  - In the Search Box Size box, enter the length for the sides of the search square.
  - In the Altitude box, specify the altitude above sea level to fly the search and rescue pattern.
  - In the Air Speed box, enter the speed to fly the pattern.
  - In the Turn Acceleration box, enter the maximum Gs for the turns. This acceleration, together with the selected air speed, determines the radius of the loops in the pattern. You must specify an air speed and turn acceleration such that the aircraft can turn 180 degrees within the track spacing distance (the first two turns).
4. Click OK.

### 3.43.3 Fly Parallel Pattern

The Fly Parallel Pattern search and rescue task causes a aircraft to fly a pattern starting at the Commence Search Point. You specify a Major Axis. The search legs are in line with this axis, and the search area expands perpendicularly to the distance of the Search Area Width. The Search Area Length is the size of the search area (box) in the direction of the Major Axis. The search area is either to the right or the left of the Commence Search Point.

The entity will not exceed the limits on altitude and air speed set in the OPD.

**Figure 22. Fly Parallel Pattern dialog box**
**Figure 23. Parallel Pattern, search right of CSP**

► **To assign a Fly Parallel Pattern search and rescue task:**

1. Select a aircraft.
2. Choose **Task > Movement > Search and Rescue > Fly Parallel Pattern**. The Fly Parallel Pattern parameters display(Figure 22).
3. Specify the parameters:
  - In the Commence Search Point section, select the waypoint.
  - In the Major Axis Heading box, enter the axis for the search. Next to the this box, indicate whether the entity should fly relative to true north or magnetic north.
  - In the Search Area Length box, enter the length of the search area. This is measured along the Major Axis.

- In the Track Spacing box, enter the spacing between the search legs.
- Select the Search Box Is Right of CSP check box to search the area to the right of the Commence Search Point. Clear the check box to search the area to the left of the waypoint.
- In the Search Area Width dialog box, enter the width of the search area. This is the width measured from the Commence Search Point.
- In the Altitude box, specify the altitude above sea level to fly the search and rescue pattern.
- In the Air Speed box, enter the speed to fly the pattern.
- In the Turn Acceleration box, enter the maximum Gs for the turns. This acceleration, together with the selected air speed, determines the radius of the loops in the pattern. You must specify an air speed and turn acceleration such that the aircraft can turn 180 degrees within the track spacing distance.

4. Click OK.

### 3.44. Sweep Naval Mines

The Sweep Naval Mines task causes an entity to look for and disable naval mines along a route. It looks within a specified distance from the route. An error rate specifies a percentage chance that the entity will detect a mine, but not disable it. The Sweep Naval Mines task is available to surface entities that have a Naval Mine Sweep system.

► **To assign a Sweep Naval Mines task:**

1. Select the entity.
2. Choose **Task > Movement > Sweep Naval Mines**. The Sweep Naval Mines parameters display.
3. Select the route to follow.
4. Specify the sensing range. The default is 300 meters.
5. Optionally, specify the disarm failure probability. This is the percentage chance that a mine will be found, but not disabled.
6. Click OK.

### 3.45. Takeoff On Route

You can explicitly task a fixed-wing entity to take off. (For details about implicit take-off, see "2.8.5 How Fixed-Wing Entities Take Off" on page 30.) In a Takeoff On Route task, the entity moves along the specified route, reaches takeoff speed (30% of minimum lift), and then moves to a point defined by the controller some distance up and away from the end of the route.

► **To assign a Takeoff On Route task:**

1. Select the entity.
2. Choose **Task > Movement > Takeoff On Route**. The Takeoff On Route parameters display.

3. Select the route.
4. Optionally, select the Teleport to Runway Start check box to have the entity move immediately to the start of the runway. This saves some time by quickly positioning the entity for takeoff.
5. Click OK.

### 3.46. Takeoff On Runway

In a Takeoff on Runway task, the aircraft moves along either the specified runway or the closest one, reaches takeoff speed (30% of minimum lift), and then moves to a point defined by the controller some distance up and away from the end of the runway. If the entity is not at the end of the runway, and you do not choose to have it teleport to the runway or create a route for it to follow to the end of the runway, then it moves straight across the terrain to the runway's starting point. It does not move along the taxiways.

This task requires a terrain where there are aerodromes with runways for the aircraft to take off from; see "Aerodromes". If the terrain does not have an aerodrome with runways that you want to use, you can add a route to the scenario and use that instead; see "3.45. Takeoff On Route" on the previous page.

 There is no logic to determine whether the runway is appropriate for the aircraft. For example, VR-Forces will not prevent a B-52 Stratofortress from taking off on a small municipal airport's runway.

► **To assign a Takeoff On Runway task for a specific runway:**

1. Select the entity.
2. Choose **Task > Movement > Takeoff On Runway**, or click the Takeoff On Runway button . The Takeoff On Runway dialog box opens.
3. In the Airport box, enter the ICAO code of the aerodrome. You can type the code, select a managed aerodrome in the Objects List, or select from the list of aerodromes at the bottom of the window. In the last case, any managed aerodromes are listed at the top of the list, and then all other aerodromes in the terrain are listed below those.
4. Select the specific runway from which the entity should take off. The runways listed are those associated with the selected aerodrome.
5. Optionally, select the Teleport to Runway Start check box to have the entity move immediately to the start of the runway. This saves some time by quickly positioning the entity for takeoff.
6. Click OK.

► **To assign a Takeoff On Runway task for the closest runway:**

1. Select the entity.
2. Choose **Task > Movement > Takeoff On Runway**, or click the Takeoff On Runway button . The Takeoff On Runway parameters display.

3. Select the Use Closest Runway check box.
4. Optionally, select the Teleport to Runway Start check box.
5. Click OK.

 You can explicitly task a fixed-wing entity to take off. For details about innate take-off, see "2.8.5 How Fixed-Wing Entities Take Off" on page 30.

### 3.47. Takeoff (Vertical)

Aircraft configured with the Fixed Wing VTOL movement system can perform a vertical takeoff task.

► **To assign a Takeoff (Vertical) task:**

1. Select the entity.
2. Choose **Task > Movement > Takeoff (Vertical)**. The Takeoff (Vertical) parameter displays.
3. Enter the height to which the entity should move.
4. Click OK.

### 3.48. Turn to Heading

The Turn to Heading task provides a realistic way for ground vehicles to turn to a new heading. Rather than turning instantly, as happens with the Heading request, a vehicle pivots or makes a K-turn to the new heading. The turning behavior for an entity is controlled by its **can-pivot** parameter in the object parameter database.

► **To assign a Turn to Heading task:**

1. Select the entity.
2. Choose **Task > Movement > Turn to Heading**. The Turn to Heading parameters display.
3. Type a heading.
4. In the Relative To list, select one of the options:
  - True North. Turn to the specified absolute heading (true north).
  - Host Object (if embarked). If embarked, turn to the specified heading relative to the host simulation object.
  - Current Heading. Turn the specified number of degrees relative to the current heading.
  - Magnetic North. Turn to the specified heading relative to magnetic north.
5. Click OK.

## 3.49. Zig-Zag to Heading

The Zig Zag to Heading task causes a surface watercraft entity to move towards a heading using a zig-zag pattern.

► **To assign a Zig-Zag to Heading task:**

1. Select the entity.
2. Choose **Task > Movement > Zig-Zag to Heading**. The Zig-Zag to Heading parameters display.
3. Specify the heading.
4. Specify the amount of time (in minutes) to spend on each leg of the zig-zag pattern.
5. Specify the angle of direction for each leg.



## 4. Engagement Tasks for Entity-Level Scenarios

This chapter describes tasks in the engagement, offense, and defense categories that apply to entity-level scenarios.

|                                                       |    |
|-------------------------------------------------------|----|
| 4.1. Arm Mine at Depth .....                          | 76 |
| 4.2. Attack Aircraft .....                            | 76 |
| 4.3. Attack Aircraft with Guns .....                  | 77 |
| 4.4. Cease Fire .....                                 | 77 |
| 4.5. Drop Naval Depth Charge .....                    | 77 |
| 4.6. Drop Naval Depth Charge at Location .....        | 78 |
| 4.7. Drop Naval Mine .....                            | 78 |
| 4.8. Drop Naval Mines Along Route .....               | 79 |
| 4.9. Execute Gun Attack .....                         | 80 |
| 4.10. Execute Surface Attack .....                    | 82 |
| 4.11. Explode Charge at Depth .....                   | 83 |
| 4.12. Fire At .....                                   | 83 |
| 4.13. Fire Ballistic Gun .....                        | 83 |
| 4.14. Fire Cruise Missile .....                       | 84 |
| 4.15. Fire for Effect Tasks .....                     | 85 |
| 4.15.1 Firing Naval Guns .....                        | 86 |
| 4.16. Fire Laser Guided Missile .....                 | 86 |
| 4.17. Fire Missile Launcher .....                     | 87 |
| 4.18. Fire Team Suppress Target .....                 | 87 |
| 4.19. Lase and Fire Guided Missile .....              | 88 |
| 4.20. Lase Location .....                             | 88 |
| 4.21. Lase Target .....                               | 89 |
| 4.22. Launch Anti-Submarine Missile (Vertical) .....  | 89 |
| 4.23. Launch Ballistic Missile at Location .....      | 90 |
| 4.24. Launch Ballistic Missile at Target Object ..... | 90 |
| 4.25. Launch Countermeasures .....                    | 91 |
| 4.26. Launch Smoke .....                              | 91 |

|                                                  |    |
|--------------------------------------------------|----|
| 4.27. Launch Torpedo Tasks .....                 | 92 |
| 4.27.1 Anti-ship (Fixed Depth) .....             | 92 |
| 4.27.2 Anti-Submarine .....                      | 92 |
| 4.28. Loiter Munition Tasks .....                | 93 |
| 4.28.1 Launch Loiter Munition .....              | 93 |
| 4.28.2 Loitering Attack .....                    | 93 |
| 4.29. Place IED .....                            | 94 |
| 4.30. Provide Indirect Fire .....                | 95 |
| 4.31. Provide Suppressive Fire at Location ..... | 96 |
| 4.32. Provide Suppressive Fire at Object .....   | 97 |
| 4.33. Release Bomb Tasks .....                   | 97 |
| 4.34. Throw Grenade Tasks .....                  | 98 |
| 4.34.1 Throw Grenade at Location .....           | 99 |
| 4.34.2 Throw Grenade at Target .....             | 99 |

## 4.1. Arm Mine at Depth

If a naval mine has been deployed, Arm Mine at Depth allows you change the mine's parameters.

Arm Mine at Depth is a system scripted task. It is disabled by default.

► **To assign an Arm Mine at Depth task:**

1. Select the mine.
2. Choose **Task > Arm Mine at Depth**. The Arm Mine at Depth parameters display.
3. Specify the depth at which to arm the mine.
4. Specify the radius within which target entities will detonate the mine.
5. Optionally, specify a time when the mine will be armed, in days and hours:minutes:seconds.
6. In the Trigger Entity Types list, add the types of entities that can trigger the mine:
  - a. Click the Add button (+). The Entity Type dialog box opens.
  - b. Build an entity type by selecting options from the lists for each enumeration component or type an enumeration.
  - c. Click OK.
7. To specify that only hostile entities will set off the mine, select Hostile Only. If this setting is cleared, any entity will set off the mine.
8. Click OK.

## 4.2. Attack Aircraft

Use this task to cause one aircraft to attack another one. The aircraft entity assigned this task will fly toward and fire on the target. The aircraft must be configured with a weapon that is capable of attacking the target aircraft.



► **To assign an Attack Aircraft task:**

1. Select an aircraft entity that has a weapon system.
2. Choose **Task > Engagement > Attack Aircraft**. The Attack Aircraft parameters display.
3. Select the target entity.
4. Click OK.

### 4.3. Attack Aircraft with Guns

The Attack Aircraft with Guns task commands aircraft that have mounted guns, such as the A-10, to target a specified aircraft with its guns rather than air-to-air missiles.

► **To assign an Attack Aircraft with Guns task:**

1. Select an aircraft entity that has a gun.
2. Choose **Task > Engagement > Attack Aircraft with Guns**. The Attack Aircraft with Guns parameters display.
3. Select the target entity.
4. Click OK.

### 4.4. Cease Fire

The Cease Fire task passes a cease fire order to artillery systems.

**i** As with other independent tasks, if the artillery entity is firing as part of a plan and you task it to cease firing, the plan terminates. See "Introduction to Tasks" on page 2.

► **To assign the Cease Fire task:**

1. Select the entity that you want to stop firing.
2. Choose **Task > Engagement > Cease Fire**.

### 4.5. Drop Naval Depth Charge

The Drop Naval Depth Charge task causes a fixed-wing, rotary wing, or surface entity to drop a depth charge at its current location. To execute this task, an entity must have a Naval Depth Charge Deployment weapon system.

► **To assign a Drop Naval Depth Charge task:**

1. Select the entity.
2. Choose **Task > Engagement > Drop Naval Depth Charge**. The Drop Naval Depth Charge parameters display.
3. Select the type of depth charge to drop.
4. Specify the depth at which you want the depth charge to explode.
5. Click OK.

## 4.6. Drop Naval Depth Charge at Location

The Drop Naval Depth Charge at Location task causes a fixed-wing, rotary wing, or surface entity to move to the specified location and drop a depth charge. To execute this task, an entity must have a Naval Depth Charge Deployment weapon system.

 In this task, the location is the location of the entity when it drops the depth charge. Since the entity is flying, the location must include a reasonable altitude.

► **To assign a Drop Naval Depth Charge at Location task:**

1. Select the entity.
2. Choose **Task > Engagement > Drop Naval Depth Charge at Location**. The Drop Naval Depth Charge At Location parameters display.
3. In the Naval Depth Charge list, select the type of depth charge to drop.
4. Specify the depth at which you want the depth charge to explode.
5. Specify the location where you want to drop the depth charge. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.

By default, this task uses the altitude of the entity at the time the task is assigned as the altitude for the drop location. (Use Current Altitude check box.) If you specify an altitude and the check box is selected, VR-Forces ignores the altitude you entered and uses the current entity altitude. If you want to specify the altitude, clear the check box and make sure that you specify the altitude in the Altitude group box. If you click on the terrain to specify the X, Y location, the altitude is set to 0, so be sure to enter the altitude you want after clicking on the terrain.

6. Click OK.

## 4.7. Drop Naval Mine

The Drop Naval Mine task allows entities with a Naval Mine Deployment system (fixed-wing, surface, and subsurface) to drop naval (underwater) mines at their current location.

► **To assign a Drop Naval Mine task:**

1. Select the entity.
2. Choose **Task > Engagement > Drop Naval Mine**. The Drop Naval Mine parameters display (Figure 24).

**Figure 24. Drop Naval Mine dialog box**

The screenshot shows the 'Drop Naval Mine' dialog box. It features a blue header bar. Below it, the 'Naval Mine:' field is a dropdown menu currently set to 'weapon-2|Mk-65'. The 'Mine Depth (ft):' field is a text input with the value '33'. The 'Arming Delay:' field consists of a numeric input '0' and a time input '00:00:00'. The 'Detonate Proximity (m):' field is a text input with the value '25'. The 'Target Types:' field is a list box containing 'Platform Subsurface' and 'Platform Surface'. Below the list box is a checked checkbox labeled 'Hostile Only'. At the bottom left is an unchecked checkbox labeled 'Pin Command'. At the bottom right are 'OK' and 'Cancel' buttons.

3. In the Naval Mine list, select the type of mine to drop.
4. In the Mine Depth field, set the depth to drop the mine. Positive numbers are below sea level.
5. In the Arming Delay fields, set the time to elapse before the mine becomes armed in days and hours:minutes:seconds.
6. In the Detonate Proximity field, set the distance from the mine within which a target entity will detonate the mine.
7. In the Target Types list, add the types of entities that can trigger the mine:
  - a. Click the Add button (+). The Entity Type dialog box opens.
  - b. Build an entity type by selecting options from the lists for each enumeration component or type an enumeration. By default, surface and subsurface watercraft are targets.
  - c. Click OK.
8. To specify that only hostile entities will set off the mine, select Hostile Only. If this setting is cleared, any entity will set off the mine.
9. Click OK.

## 4.8. Drop Naval Mines Along Route

The Drop Naval Mines Along Route task allows entities with a Naval Mine Deployment system (fixed-wing, surface, and subsurface) to drop naval (underwater) mines while they travel along a route.

► **To assign a Drop Naval Mines Along Route task:**

1. Select the entity.
2. Choose **Task > Engagement > Drop Naval Mines Along Route**. The Drop Naval Mines Along Route parameters display.
3. Select the route to follow.
4. In the Mine Spacing field, specify the distance between each mine.
5. In the Naval Mine list, select the type of mine to drop.
6. In the Mine Depth field, set the depth to drop the mine. Positive numbers are below sea level.
7. In the Arming Delay fields, set the time to elapse before the mine becomes armed in days and hours:minutes:seconds.
8. In the Detonate Proximity field, set the distance from the mine within which a target entity will detonate the mine.
9. In the Trigger Entity Types list, add the types of entities that can trigger the mine:
  - a. Click the Add button (). The Entity Type dialog box opens.
  - b. Build an entity type by selecting options from the lists for each enumeration component or type an enumeration. By default, surface and subsurface watercraft are targets.
  - c. Click OK.
10. To specify that only hostile entities will set off the mines, select Hostile Only. If this setting is cleared, any entity will set off the mine.
11. Click OK.

## 4.9. Execute Gun Attack

Fixed-wing entities that have the appropriate armament can strafe ground targets. The Execute Gun Attack task takes these parameters:

- **Initial Point.** The point at which the aircraft reaches strafing altitude and lines up to strafe the target point. VR-Forces enforces a minimum distance from the initial point to the target. It varies based on the plane's dive angle, strafe range, entry speed, and other parameters. Two kilometers is a good minimum for most entities. If the initial point is too close to the target, VR-Forces sends an error message to the entity console.
- **Target.** The entity or object to target. If you specify none, the entity strafes the target location. If you specify an object type, the entity looks for that object type. If it finds the target type, it strafes it. If it does not find the target type, it either strafes the target point (default behavior) or aborts the strafing run, depending on the state of the Abort Without Target check box.
- **Egress Heading and Egress Maneuver.** The way the aircraft should exit the strafing run and the heading at which it exits.
- **Specify Time on Target.**

You can also initiate a strafing run from the Execute Surface Attack task. Table 4 shows how arguments to the Execute Surface Attack task map to the Execute Gun Attack task.

**Table 4. Surface to Gun Attack argument mapping**

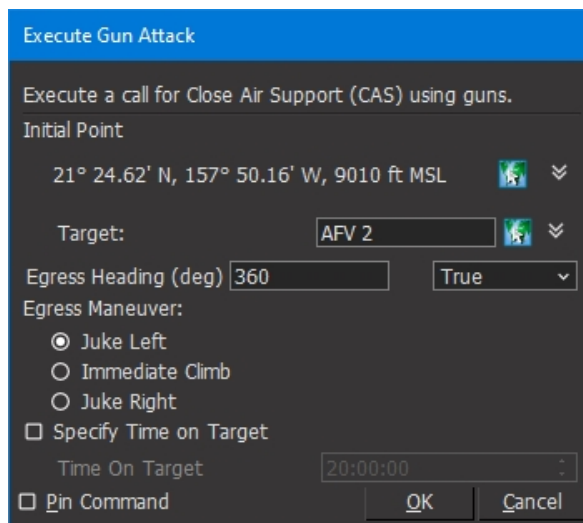
| Execute Surface Attack Argument | Execute Gun Attack Argument               |
|---------------------------------|-------------------------------------------|
| Initial Point                   | Initial Point.                            |
| Target: an entity               | Target Type: Target the indicated entity. |
| Offset: none                    | Egress Maneuver: Immediate Climb.         |
| Offset: left                    | Egress Maneuver: Juke Left.               |
| Offset: right                   | Egress Maneuver: Juke Right.              |
|                                 | Abort without target.                     |
| Egress Heading                  | Final Egress Heading.                     |

The effectiveness of a strafing run at killing entities around the target point depends in part on the value of the **dispersal-angle** parameter specified for the munition. You can edit this parameter in `./data/simulationModelSets/base/detonationParams.mtl`. The parameter is described in the comments at the top of the file.

► **To assign an Execute Gun Attack task:**

1. Select the entity that will make the strafing run.
2. Choose **Task > Engagement > Execute Gun Attack**. The Execute Gun Attack parameters display (Figure 25).

**Figure 25. Execute Ground Attack dialog box**



3. Click the terrain to specify the initial point or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Click the Select Object button (📍) and then specify the target entity.
5. Select an egress maneuver.

6. Optionally, specify an egress heading.
7. Optionally, select Specify Time on Target and specify the time at which the entity should fire upon the target. This controls the entity's speed as it approaches the drop site.
8. Click OK.

## 4.10. Execute Surface Attack

The Execute Surface Attack task assigns a fixed-wing fighter or bomber to release bombs on a ground target using coordinates, a laser designation code, or a specific target. If you specify multiple targets, all bombs are dropped simultaneously. The aircraft does not fly over each target. If you specify an initial point, that point displays as part of the task visualization when you select the entity.

 Due to the Time on Target option, this task is not intended for use with Move Along Route with Actions. It is intended to be a single use for dropping bombs on targets.

### ► To assign an Execute Surface Attack task:

1. Select the entity you want to execute the surface attack.
2. Choose **Task > Engagement > Execute Surface Attack**, or click the Execute Surface Attack button (). The Execute Surface Attack parameters display.
3. Optionally, select Initial Point and then enter the coordinates. (The coordinate system is specified in Display Units Settings.)
4. Click the Add button () to add a target to the list. The Target Configuration dialog box opens.
5. Choose the Target Type.
  - MGRS Coordinates or Lat/Lon Coordinates. Enter the Location coordinates.
  - Laser Designation Code. Enter a code.
  - Target. Use the target box to select the desired target.
6. Select the Weapon (Bomb) from the list.
7. Click OK. You return to the Execute Surface Attack, where the target is shown in the list.
8. You can add multiple targets by repeating steps 4–7.
9. Optionally, specify the Time on Target to indicate the time at which the entity should drop the bombs on the targets. This controls the entity's speed as it approaches the drop site.
10. Optionally, specify the Egress Heading.
11. Optionally, enter a Release Speed and Release Altitude. The speed is only the speed at the time of the drop, not the speed for the duration of the task. During the rest of the task, it uses the standard or ordered speed.
12. Click OK.

## 4.11. Explode Charge at Depth

The Explode Charge at Depth task explodes a depth charge at the specified depth. You can use this task to explode a depth charge that has been deployed, but has not yet exploded.

Explode Charge at Depth is a system scripted task. It is disabled by default.

► **To assign an Explode Charge at Depth task:**

1. Select the depth charge.
2. Choose **Task > Explode Charge at Depth**. The Explode Charge at Depth parameters display.
3. Specify the depth.
4. Click OK.

## 4.12. Fire At

The Fire At task commands an entity to fire at a specific target, either a location or simulation object. You can specify which weapon system to use if the entity has more than one.

When you assign this task, the entity ignores its rules of engagement and force types. Entities will fire at their own force if told to do so. The shooter must have an appropriate weapon system and ammunition for the target.

► **To assign a Fire At task:**

1. Select the entity you want to fire.
2. Choose **Task > Engagement > Fire At**. The Fire At parameters display.
3. Click Select Location or Select Object.
4. Select the target:
  - Click the location to fire at or enter the coordinates.
  - Select the simulation object.
5. Enter the number of rounds you want the entity to fire.
6. Optionally, select the Stop Firing When Target is Destroyed check box. This causes the entity to stop firing when the target is destroyed or when the maximum number of rounds has been fired, whichever comes first.
7. Optionally, select the Prefer Specific Weapon check box and then choose one from the list of the entity's available weapons.
8. Click OK.

## 4.13. Fire Ballistic Gun

The Fire Ballistic Gun task commands an entity to fire at a specific target, either a location or simulation object. You can specify which weapon system to use and the maximum number of rounds to fire. The entity must be equipped with a ballistic weapon system.

When you assign this task, the entity ignores its rules of engagement and force types. Entities will fire at their own force if told to do so.

► **To assign a Fire Ballistic Gun task:**

1. Select the entity you want to fire.
2. Choose **Task > Engagement > Fire Ballistic Gun**. The Fire Ballistic Gun parameters display.
3. Click Select Location or Select Object.
4. Select the target:
  - Click a location on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
  - Select the simulation object.
5. If the entity has more than one weapon system equipped, select the weapon you want the entity to fire.
6. Enter the maximum number of rounds the entity should fire.
7. Click OK.

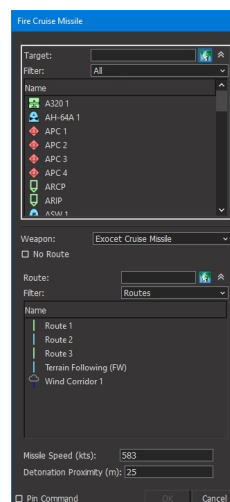
## 4.14. Fire Cruise Missile

Entities with a cruise missile launcher system can fire cruise missiles. A cruise missile can fly directly to a target or follow a route.

► **To assign a Fire Cruise Missile task:**

1. Select an entity that has cruise missiles.
2. Choose **Task > Engagement > Fire Cruise Missile**. The Fire Cruise Missile parameters display (Figure 26).

**Figure 26. Fire Cruise Missile dialog box**



3. Select the target from the list or on the terrain, or enter its name in the Target box. Targets can be entities or points.
4. If the entity is equipped with more than one weapon system, select the weapon that you want the entity to fire.



5. Select a route for the missile to follow or select the No Route check box. If you do not select a route, the missile flies directly to the target.
6. Specify the Missile Speed.
7. Specify the Detonation Proximity for the warhead. It will detonate within this range of the selected target.
8. Click OK.

## 4.15. Fire for Effect Tasks

You can task artillery entities and surface vessels with naval guns to fire for effect (FFE). The three tasks are:

- Fire for Effect on Entity.
- Fire for Effect on Location, where the location is a point on the terrain.
- Fire for Effect on Target, where the target is a waypoint.

When the entity executes the task, you can select it to see a countdown to when it will fire its weapon. After it fires, there is a countdown to the splash at the specified entity, location, or target.

### ► To assign a Fire for Effect task:

1. Select the entity to which you want to assign the task.
2. Choose **Tank > Engagement > Fire for Effect on [Entity | Location | Target]**. The appropriate parameters display.
3. Identify the target:
  - For FFE on Entity, select the target entity in the list, select it on the terrain, or type its name in the Name box.
  - For FFE on Location, click a location on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
  - For FFE on Target, select a point from the list, or select it on the terrain.

 For FFE on Entity and FFE on Target, you can filter the Name list.

4. If the entity is equipped with more than one weapon system, select the weapon that you want the entity to fire.
5. Select the munition.
6. Indicate how you want the entity to fire:
  - Specify the number of rounds.
  - Select Continuous to fire continuously until the target is destroyed or the entity has no more ammunition.
7. Optionally, specify the height above the terrain at which rounds should detonate.
8. Click OK.



- The Fire-for-Effect tasks are only meaningful when assigned to artillery entities or surface vessels that have naval guns.
- When you choose Fire-for-Effect on Entity, if the target entity is moving, VR-Forces dead reckons its position and the artillery fires at its expected location, not the location it was at the start of the task.
- Artillery entities have maximum and minimum ranges. To see these ranges, enable range rings for the entity. (For details, see "Displaying Threat and Sensor Range Rings".) The sim. engine console window prints messages when you select targets that are outside of the entity's range.
- The default preferred trajectory height for artillery and naval guns is 500 meters. If this height is not attainable given other parameters involved in calculating the trajectory, VR-Forces calculates a more appropriate height. You can specify a different preferred trajectory height with the Preferred Trajectory Height set data request. For details, see "Preferred Trajectory Height".

#### 4.15.1 Firing Naval Guns

Ordinarily, naval guns fire their munitions in a parabola, similar to artillery. However, they can also fire directly at a target if these conditions are true:

- The target's range is greater than the weapon's minimum range, but less than the first value in its range list.
- The **capable-of-direct-alignment** parameter for the weapon system's naval-gun-controller is set to True.
- The firing entity has line-of-sight to the target.

### 4.16. Fire Laser Guided Missile

This task causes an entity equipped with laser-guided missiles to launch a missile without lasing the target object or location. It is designed for use when a separate entity will lase the target, however it can fire the missile without a laser designator.

By default, the entity firing the missile waits to detect a designator before firing. You must specify the laser code that it will target, as described in "Laser Code".

You can specify that the entity fire the missile without a laser designator, in which case the entity immediately fires the missile.

► **To assign the Fire Laser Guided Missile task:**

1. Select the entity.
2. Choose **Task > Engagement > Fire Laser Guided Missile**. The Fire Laser Guided Missile task parameters display.
3. From the Weapon Selection list, choose the missile that you want the entity to fire.
4. Optionally, select the Fire Without Designator check box.
5. Click OK.

## 4.17. Fire Missile Launcher

The Fire Missile Launcher task commands an entity to launch a missile at a specific target, either a location or simulation object. You can specify which missile launcher system to use and the maximum number of rounds to fire. The entity must be equipped with a missile launcher system.

When you assign this task, the entity ignores its rules of engagement and force types. Entities will fire at their own force if told to do so. The shooter must have an appropriate weapon system and ammunition for the target.

 Cruise missiles are special, in that they can follow a specific route. If that is the type of missile you want to fire, see "4.14. Fire Cruise Missile" on page 84.

► **To assign a Fire Missile Launcher task:**

1. Select the entity you want to fire.
2. Choose **Task > Engagement > Fire Missile Launcher**. The Fire Missile Launcher parameters display.
3. Click Select Location or Select Object.
4. Select the target:
  - Click the location to fire at or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
  - Select the simulation object.
5. If the entity is equipped with more than one missile system, select the launcher you want the entity to fire.
6. Enter the maximum number of rounds the entity should fire.
7. Click OK.

## 4.18. Fire Team Suppress Target

The Fire Team Suppress Target task tells a fire team to fire at a simulation object such as a point, line, or area, one meter above the terrain. If one of the entities in the fire team must stand or move to the side to see the target, it does so. For more information, see "2.10. Suppressive Fire Tasks" on page 33.

► **To assign a Fire Team Suppress Target task:**

1. Select a fire team.
2. Choose **Task > Engagement > Fire Team Suppress Target**. The Fire Team Suppress Target parameters display.
3. Select the simulation object to fire at.
4. In the Rapid Fire Duration box, enter the number of seconds for the rapid fire.
5. In the Total Fire Duration box, enter the number of minutes the entity should provide suppressive fire.
6. Click OK.

## 4.19. Lase and Fire Guided Missile

This causes an entity equipped with laser-guided missiles to lase a target location or simulation object and then fire a missile at it.

► **To assign the Lase and Fire Guided Missile task:**

1. Select the entity.
2. Choose **Task > Engagement > Lase and Fire Guided Missile**. The Lase and Fire Guided Missile task parameters display.
3. Click Select Location or Select Object.
4. Select the target:
  - Click the location to lase and target or enter the coordinates.
  - Select the simulation object.
5. Click OK.

## 4.20. Lase Location

The Lase Location task simulates aiming a laser beam at a location to guide a laser guided weapon. Use this task if you want to lase a specific location. An entity can lase a location while it is executing a movement task.

To lase a location, an entity must have a Laser Designator system. For more information about lasing targets, see:

- "Lasing Targets".
- "Laser Code".
- "Synchronize Laser Code".



- The Lase Location task takes line of sight into account when looking at a specified location. If there are terrain obstacles between the lasing entity and the specified location, the laser designator spot is at the point blocking the line of sight.
- Laser designator systems have a maximum range. If the entity aiming the laser designator is too far away from the location it is trying to lase, the task is unsuccessful.

► **To assign a Lase Target task:**

1. Select the entity that will aim the laser.
2. Choose **Task > Engagement > Lase Location**. The Lase Location parameters display.
3. Click a location on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Click OK.

## 4.21. Lase Target

The Lase Target task simulates aiming a laser beam at a target to guide a laser guided weapon. Use this task if you want to lase a specific target. An entity can lase a target while it is executing a movement task.

To lase a target, an entity must have a Laser Designator system. For more information about lasing targets, see:

- "Lasing Targets".
- "Laser Code".
- "Synchronize Laser Code".



- The Lase Target task takes line of sight into account when looking at a target. If there are terrain obstacles between the lasing entity and the specified target, the task is unsuccessful.
- When an entity lases one side an entity, the direction of the laser is considered and weapons can only target the designated spot if they have line of sight to it. If the weapon does not have visibility to that side of the entity, the weapon cannot target it.
- Laser designator systems have a maximum range. If the entity aiming the laser designator is too far away from the location it is trying to lase, the task is unsuccessful.

### ► To assign a Lase Target task:

1. Select the entity that will aim the laser.
2. Choose **Task > Engagement > Lase Target**. The Lase Target parameters display.
3. Select the entity that you want to lase.
4. Click OK.

## 4.22. Launch Anti-Submarine Missile (Vertical)

The Launch Anti-Submarine Missile (Vertical) task commands a surface entity to launch a missile that flies to the vicinity of the target and then drops a homing torpedo. To execute this task, an entity needs an Anti-Submarine Missile (Vertically Launched) weapon system.

### ► To assign a Launch Anti-Submarine Missile (Vertical) task:

1. Select the surface entity.
2. Choose **Task > Engagement > Launch Anti-Submarine Missile (Vertical)**. The Launch Anti-Submarine Missile (Vertical) parameters display.
3. Select the target entity.
4. Specify the location at which to drop the homing torpedo.
5. Click OK.

## 4.23. Launch Ballistic Missile at Location

The Launch Ballistic Missile at Location task lets an entity that has a strategic missile weapon system launch a missile at a location.

**i** The Ballistic Missile tasks do not use physics-based modeling. They calculate a simple parabola for the trajectory and move the missile along the trajectory ignoring atmospheric, thrust stages, cargo weight, gravity, and the rotation of the earth.

► **To launch a ballistic missile at a location:**

1. Select an entity with a strategic missile weapon system.
2. Choose **Task > Engagement > Launch Ballistic Missile at Location**. The Launch Ballistic Missile at Location parameters display.
3. Click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. In the Missile Type list, select the missile to fire.
5. Click OK.

## 4.24. Launch Ballistic Missile at Target Object

The Launch Ballistic Missile at Target Object task lets an entity that has a strategic missile weapon system launch a missile at a point object, entity, or unit. The destination of the missile is the target's location when the missile is fired. The missile does not track moving targets.

**i** The Ballistic Missile tasks do not use physics-based modeling. They calculate a simple parabola for the trajectory and move the missile along the trajectory ignoring atmospheric, thrust stages, cargo weight, gravity, and the rotation of the earth.

► **To launch a ballistic missile at a target object:**

1. Select an entity with a strategic missile weapon system.
2. Choose **Task > Engagement > Launch Ballistic Missile at Target Object**. The Launch Ballistic Missile at Target Object parameters display.
3. Optionally, filter the list of objects.
4. Select the object to fire at.
5. In the Missile Type list, select the missile to fire.
6. Specify the altitude where the missile should detonate.
7. Click OK.

## 4.25. Launch Countermeasures

Fixed-wing and rotary-wing entities can launch countermeasures, such as chaff and flares. Launch Countermeasures is not an exclusive task. It does not interrupt an entity's current task. (For more information, see "Launching Countermeasures (Chaff and Flare)".)

The Evade Missile reactive task automatically launches chaff and flare when a fixed-wing entity evades a missile.

► **To assign a Launch Countermeasures task:**

1. Select the entity that you want to launch countermeasures.
2. Choose **Task > Engagement > Launch Countermeasures**. The Launch Countermeasures parameters display.
3. In the Countermeasure Types list, select the type of countermeasures you want to launch.
4. In the Number To Fire box, specify the number of countermeasures to launch.
5. In the Time Between Fires box, specify the amount of time, in seconds, between each countermeasure launch.
6. Click OK.

## 4.26. Launch Smoke

The Launch Smoke task causes an entity to create a smoke cloud. The cloud is created as an environmental object. It grows over time.

An entity can launch a smoke cloud towards a threat or in a specific direction. Smoke clouds affect sensors. They do not affect entity movement.



- Tactical smoke is delivered by the M250 smoke grenade launcher. An entity must have this system to execute this task.
- Wind and precipitation dissipate tactical smoke. You can adjust those dissipation factors by editing the Tactical Smoke simulation object in the Simulation Object Editor.

► **To assign a Launch Smoke task:**

1. Select the entity that you want to launch smoke.
2. Choose **Task > Engagement > Launch Smoke**. The Launch Smoke parameters display.
3. Select At Nearest Threat or Toward Heading.
4. If you selected Toward Heading, specify a compass heading and indicate whether it is relative to true or magnetic north.
5. Click OK.

## 4.27. Launch Torpedo Tasks

Subsurface entities can launch two types of anti-ship torpedoes: one moves at a fixed depth, and the other homes in on the center of the target. In both cases, the torpedo travels at an initial bearing for a fixed distance before it homes in on the target.

 Torpedoes rise to target depth at a fixed rate. If a submarine is quite close to a target when it fires, the torpedo might not be able to rise to target depth in the distance from the submarine to the target. If this happens, it circles until it reaches target depth.

### 4.27.1 Anti-ship (Fixed Depth)

► To assign a Launch Torpedo – Anti-ship (Fixed Depth) task:

1. Select a subsurface entity.
2. Choose **Task > Engagement > Launch Torpedo > Anti-ship (Fixed Depth)**. The Anti-ship (Fixed Depth) parameters display.
3. Optionally, filter the entity list.
4. Select the target entity.
5. In the Initial Bearing box, specify the bearing the torpedo should travel at until it begins homing in on the target.
6. In the Homing Start Delay box, enter the amount of time that the torpedo should travel on its initial bearing.
7. In the Cruise Depth box, specify the depth the torpedo should travel at. It will remain at this depth.
8. In the Proximity Trigger Distance box, set the distance from a target at which the torpedo will detonate.
9. Click OK.

### 4.27.2 Anti-Submarine

The anti-submarine torpedo homes in on the center of the target.

► To assign a Launch Torpedo – Anti-submarine task:

1. Select a subsurface entity.
2. Choose **Task > Engagement > Launch Torpedo > Anti-submarine**. The Anti-Submarine parameters display.
3. Optionally, filter the entity list.
4. Select the target entity.
5. In the Initial Bearing box, specify the bearing the torpedo should travel at until it begins homing in on the target.
6. In the Homing Start Delay box, enter the amount of time that the torpedo should travel on its initial bearing.



7. In the Proximity Trigger Distance box, set the distance from the target at which the torpedo will detonate.
8. Click OK.

## 4.28. Loiter Munition Tasks

### 4.28.1 Launch Loiter Munition

Entities configured with weapon systems for loiter munitions, such as the Switchblade 300 and 600 Launchers, can launch the munitions that will then orbit a location or strike a simulation object. When you use delayed targeting, you must then use the Target set data request after the munition is launched to specify which simulation object the munition should strike. See "Target".

► **To assign the Launch Loiter Munition task using Delayed Targeting:**

1. Select the launcher entity.
2. Choose **Task > Engagement > Launch Loiter Munition**. The Launch Loiter Munition parameters display.
3. Select Delayed Targeting and then specify the location that the munition should orbit.
4. Click OK. The munition is launched and then moves to the correct altitude and position to conduct its orbit task.
5. When you want the loiter munition to strike, select the munition.
6. Choose **Set > Engagement > Target** and then specify the simulation object. After you click OK, the loiter munition flies toward and crashes into the target.

► **To Assign the Loitering Attack task using Immediate Targeting:**

1. Select the launcher entity.
2. Choose **Task > Engagement > Launch Loiter Munition**. The Launch Loiter Munition parameters display.
3. Select Immediate Targeting and then specify the simulation object that the munition should strike.
4. Click OK. The munition is launched, then flies toward and crashes into the target.

### 4.28.2 Loitering Attack

Loiter munitions, such as the Switchblade 300 or 600, can perform the Loitering Attack task. You can task a loiter munition to orbit a location or strike a simulation object. When you use delayed targeting, you must then use the Target set data request to tell the munition which simulation object to strike when you want it to attack. See "Target".

► **To assign the Loitering Attack task using Delayed Targeting:**

1. Select the loiter munition entity.
2. Choose **Task > Engagement > Loitering Attack**. The Loitering Attack parameters display.
3. Select Delayed Targeting and then specify the location that the munition should orbit.
4. Click OK. The munition moves to the correct altitude and position to conduct its orbit task.
5. When you want the loiter munition to strike, select the munition if it is not still selected.
6. Choose **Set > Engagement > Target** and then specify the simulation object. After you click OK, the loiter munition flies toward and crashes into the target.

► **To Assign the Loitering Attack task using Immediate Targeting:**

1. Select the loiter munition entity.
2. Choose **Task > Engagement > Loitering Attack**. The Loitering Attack parameters display.
3. Select Immediate Targeting and then specify the simulation object that the munition should strike.
4. Click OK. The munition flies toward and crashes into the target.

## 4.29. Place IED

The Place IED task lets a human entity place an improvised explosive device (IED) at a particular location and move to a post-placement location. This task also sets the parameters for how the IED is armed.

► **To assign a Place IED task:**

1. Select the entity.
2. Choose **Task > Engagement > Place IED**. The Place IED parameters display (Figure 27).

**Figure 27. Place IED dialog box**

3. Click on the terrain where the entity should place the IED, or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. In the Placement Duration box, enter the amount of time it takes to place the IED.
5. Select a fuse type from the Fuse Type list.
6. For a timed delay fuse, specify the time to delay in the Time Delay box.
7. For a proximity fuse, specify the distance from the IED within which another entity will trigger the IED.
8. For the Post Placement Location, specify the location where the entity should move to after it places the IED, or click on the terrain to specify the location.
9. From the Arming Location list, select Arm at Placement Location to arm the IED as soon as it is placed, or Arm at Post Placement Location to arm it when the entity reaches the post-placement location.
10. Click OK.

## 4.30. Provide Indirect Fire

The Provide Indirect Fire task allows you to task artillery and surface watercraft entities that have naval guns to provide indirect fire. It was designed for an artillery officer who would fill in the information based on communication of a Call for Fire 6 line.

**i** The task was designed for specific use cases. It does not include some information such as a Warning Order, which can be understood in the mind of the person filling in the task's options.

There are a number of targeting methods available to choose:

- **Coordinates.** Specify either MGRS or Lat/Lon (DMS) coordinates, either by entering the information or clicking on the terrain.
- **Polar.** Select a Forward Observer entity and enter the location of the target in direction and distance from the observer.
- **Adjust.** Select a forward observer and enter the adjustment and direction and distance from the previously targeted point.
- **Known Point.** Use a known target point Specify the Forward Observer, Known Point, and Adjustment from the known point location based on the forward observer location. This requires a known target point, which is created by a previous run of the Provide Indirect Fire task with the Generate Known Target Point check box selected.
- **Target.** Select a simulation object.

► **To assign a Provide Indirect Fire task:**

1. Select the entity that will provide indirect fire.
2. Choose **Task > Engagement > Provide Indirect Fire**, or click the Indirect Fire button (). The Provide Indirect Fire parameters display.
3. Click the button for the method that you want to use to enter the target location.
4. Enter the parameters for the targeting method you selected.
5. Specify the munition.
6. Enter the number of Rounds to fire, or select the Continuous check box to fire until the entity runs out of ammunition for the selected weapon.
7. Choose the Fuse type. If you use Other, enter the Height Above Terrain.
8. Optionally, enter a Fire Mission Index. This is a unique identifier for the network PDU. It does not affect the VR-Forces task.
9. Optionally, select the Generate Known Target Point check box. This creates a target point with a 2-letter and 4-number designation, which is used by the Adjust target method.
10. Click OK.

## 4.31. Provide Suppressive Fire at Location

The Provide Suppressive Fire on Location task tells an entity to fire at a location, 1 meter above that location. For more information, see "2.10. Suppressive Fire Tasks" on page 33.

 The target of suppressive fire must be at least 25 meters away.

► **To assign a Provide Suppressive Fire at Location task:**

1. Select the entity that will deliver suppressive fire.
2. Choose **Task > Engagement > Provide Suppressive Fire at Location**. The Provide Suppressive Fire at Location parameters display.

3. Specify the location.
4. In the Duration of Rapid Fire box, type the number of seconds for the initial burst of rapid fire.
5. In the Total Fire Time box, type the number of minutes the entity should provide suppressive fire.
6. In the Maximum Round to Fire box, type the maximum number of rounds to fire.
7. In the Gun list, select the gun to use.
8. Click OK.

## 4.32. Provide Suppressive Fire at Object

The Provide Suppressive Fire at Object task tells an entity to fire at a simulation object such as a point, line, or area, 1 meter above the terrain. If the object is in cover, the entity moves 1 meter to the left or right or stands up, it does so. For more information, see "2.10. Suppressive Fire Tasks" on page 33.

 The target of suppressive fire must be at least 25 meters away.

### ► To assign a Provide Suppressive Fire at Object task:

1. Select the entity that will deliver suppressive fire.
2. Choose **Task > Engagement > Provide Suppressive Fire at Object**. The Provide Suppressive Fire from at Object parameters display.
3. Select the simulation object to fire at.
4. In the Rapid Fire Duration box, enter the number of seconds for the rapid fire.
5. In the Total Fire Duration box, enter the number of minutes the entity should provide suppressive fire.
6. From the Gun to Use list, select the weapon to prefer.
7. Enter the maximum number of rounds for the entity to fire.
8. Click OK.

## 4.33. Release Bomb Tasks

Fixed-wing entities can release bombs targeted to a laser spot, a location, or an entity. The Release Bomb on Laser Spot task requires that another entity lase the target. You must synchronize the laser code for the bomber with the laser code for the lasing entity. Use of lasers is described in:

- "Lasing Targets".
- "4.21. Lase Target" on page 89.
- "Laser Code".
- "Synchronize Laser Code".

The Release Bomb on Target and Release Bomb on Location tasks cause a bomb to be immediately dropped. The task assumes that the target is in front of the aircraft. The targetable area is a circle whose radius is defined by the altitude of the airplane and its maximum range.

The bomb navigates toward its target based on **max-range** and **max-range-altitude** parameters (configurable in the Simulation Object Editor). A bomb can navigate **max-range** meters laterally if it is at **max-range-altitude** meters above the target. As the altitude changes, the distance it can travel changes accordingly, so at half of **max-range-altitude** it can travel half of **max-range**.

The bomb follows a straight line path from its launch to the point it is navigating to, assuming it can reach it. If its target point is out of range, it gets as close to the target as it can. If the target point moves, the bomb changes course and heads for the new target point.

 Use of a Release Bomb task requires that you determine if the airplane is in range of the target. The Execute Surface Attack task ensures that the airplane is within range of the target before it drops its bomb. Therefore, we recommend using that task rather than one of the Release Bomb tasks. See "4.10. Execute Surface Attack" on page 82.

The Release Bomb on Laser Spot task steers the bomb towards the laser point.

► **To assign a release bomb task:**

1. Select the entity that will release the bomb.
2. Choose **Tank > Engagement > Release Bomb on [Laser Spot | Location | Target]**. The appropriate parameters display.
3. In the Bomb Types list, select the type of bomb you want to drop.
4. In the Detonation Proximity box, enter the distance from the target at which the bomb should detonate.
5. Enter other parameters as needed:
  - For Release Bomb on Location, specify the location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
  - For Release Bomb on Target, select the target simulation object.
6. Click OK.

## 4.34. Throw Grenade Tasks

Infantry characters can throw hand grenades at a target or at a location. An entity can throw available grenades, such as those included with VR-Forces:

- M18 - colored smoke.
- M67 - fragmentation.
- M7A3 - CS tear gas.

- M83 - white smoke.
- M84 - stun, or flashbang.

#### 4.34.1 Throw Grenade at Location

► **To assign a Throw Grenade at Location task:**

1. Select the entity.
2. Choose **Task > Engagement > Throw Grenade at Location**. The Throw Grenade at Location parameters display.
3. In the Grenade Type list, select the type of grenade to throw.
4. Specify the location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
5. Click OK.

#### 4.34.2 Throw Grenade at Target

► **To assign a Throw Grenade at Target task:**

1. Select the entity.
2. Choose **Task > Engagement > Throw Grenade at Target**. The Throw Grenade at Target parameters display.
3. In the Grenade Type list, select the type of grenade to throw.
4. Specify the target.
5. Click OK.







## 5. Human and Crowd Tasks

This chapter describes the tasks for humans and crowds that apply to entity-level scenarios. Generic movement and engagement tasks that also apply to humans are described in "Chapter 3. Movement Tasks for Entity-Level Scenarios" on page 39 and "Chapter 4. Engagement Tasks for Entity-Level Scenarios" on page 75.

|                                            |     |
|--------------------------------------------|-----|
| 5.1. Circle Around Location .....          | 102 |
| 5.2. Civilian Idle .....                   | 102 |
| 5.3. Civilian Pray .....                   | 102 |
| 5.4. Civilian Protest .....                | 103 |
| 5.5. Civilian Talk .....                   | 103 |
| 5.6. Close Door .....                      | 103 |
| 5.7. Close Window .....                    | 103 |
| 5.8. Create More Entities .....            | 103 |
| 5.9. Crowd Along Line .....                | 104 |
| 5.10. Crowd Around Location .....          | 104 |
| 5.11. Crowd Around Object .....            | 105 |
| 5.12. Crowd Around Waypoints .....         | 105 |
| 5.13. Crowd in Front of Entity .....       | 106 |
| 5.14. DI-Guy Animation .....               | 106 |
| 5.15. DI-Guy Gesture .....                 | 106 |
| 5.16. Disperse Crowd .....                 | 107 |
| 5.17. Find Cover .....                     | 107 |
| 5.18. Find Firing Position .....           | 108 |
| 5.19. Fire Team Bound to Location .....    | 109 |
| 5.20. Fire Team Occupy Fire Position ..... | 110 |
| 5.21. Fire Team Travel to Waypoint .....   | 110 |
| 5.22. Flee from Entities .....             | 111 |
| 5.23. Move Between Cover to Location ..... | 111 |
| 5.24. Open Door .....                      | 112 |
| 5.25. Open Window .....                    | 112 |

|                                      |     |
|--------------------------------------|-----|
| 5.26. Parachute to Target .....      | 112 |
| 5.27. Protest Along Line .....       | 113 |
| 5.28. Protest Around Location .....  | 113 |
| 5.29. Riot .....                     | 114 |
| 5.30. Rush to Location .....         | 114 |
| 5.31. Squad Travel to Waypoint ..... | 115 |
| 5.32. Visit Neighbor .....           | 115 |
| 5.33. Wait in Queue .....            | 115 |
| 5.34. Wander .....                   | 116 |
| 5.35. Wander In Area .....           | 117 |

## 5.1. Circle Around Location

The Circle Around Location task causes the members of a crowd or unit to form a circle around a location. They can face in (towards the location) or out (away from the location). This task only applies to the individually simulated members of a crowd (that is, leaf members). If a crowd has subordinate units, the task is not passed down to them.

If a crowd or unit has many members, using a small radius may result in overlapping entities.

► **To assign a Circle Around Location task:**

1. Select the crowd or unit.
2. Choose **Task > Movement > Circle Around Location**. The Circle Around Location parameters display.
3. Specify the location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Specify the direction to face.
5. Specify the radius of the circle to create.

## 5.2. Civilian Idle

The Civilian Idle task causes a human character to stand and cycle through DI-Guy animations typical of a person standing alone or in a group.

► **To assign a Civilian Idle task:**

1. Select a human character.
2. Choose **Task > Movement > Civilian Idle**.

## 5.3. Civilian Pray

The Civilian Pray task causes a human character to perform a praying animation. The entity performs the animation until something interrupts it.

- To assign a Civilian Pray task, select a human character and then choose **Task > Movement > Civilian Pray**.

## 5.4. Civilian Protest

The Civilian Protest task causes a human character to stand and cycle through DI-Guy animations typical of a person participating in a protest.

**i** If you need many people to protest, use the Protest Along Line or Protest Around Location crowd tasks rather than assigning this task to individuals. See "5.27. Protest Along Line" on page 113 or "5.28. Protest Around Location" on page 113.

- To assign a Civilian Protest task, select a human character and then choose **Task > Movement > Civilian Protest**.

## 5.5. Civilian Talk

The Civilian Talk task causes a human character to stand and cycle through DI-Guy animations typical of a person engaging in a conversation.

- To assign a Civilian Talk task, select a human character and then choose **Task > Movement > Civilian Talk**.

## 5.6. Close Door

If a human is within approximately one meter of an open door, this task causes the door to close. This task applies to doors controlled by switch nodes in the terrain, not articulated parts. For more information about opening and closing doors, see the chapter on [Dynamic Terrain in VR-Forces](#) in the *Adding Content to MAK ONE Applications Reference Manual*.

- **To assign a Close Door task:**
  1. Move a human to within one meter of an open door.
  2. Choose **Task > Terrain > Close Door**.

## 5.7. Close Window

If a human is within approximately one meter of an open window, this task causes the window to close. This task applies to windows controlled by switch nodes in the terrain, not articulated parts. For more information about opening and closing windows, see the chapter on [Dynamic Terrain in VR-Forces](#) in the *Adding Content to MAK ONE Applications Reference Manual*.

- **To assign a Close Window task:**
  1. Move a human to within one meter of an open window.
  2. Choose **Task > Terrain > Close Window**.

## 5.8. Create More Entities

The Create More Entities task adds more entities to a crowd or herd. If the crowd or herd is part of a pedestrian or animal area, the new entities wander within the area. If the crowd or herd is not part of an area, their wandering is unrestricted.

**i** When the Create More Entities task runs, it gives each new entity a task. It assigns humans a Move To Location task, followed by a Wander task. It assigns animals a Wander With Animations task. The Create More Entities task is not complete until all of the Wander tasks have been sent. Therefore, the task does not complete as quickly as you might expect simply based on when the entities are created.

► **To assign a Create More Entities task:**

1. Select a crowd or a herd, or select a pedestrian or herd area.
2. Choose **Task > Create More Entities**. The Create Pedestrians parameters display.
3. Specify the number of entities to create.
4. Select a placement option.
5. Click OK.

## 5.9. Crowd Along Line

The Crowd Along Line task causes a crowd to gather to the left or right of a line at its center point.

► **To assign a Crowd Along Line task:**

1. Select a crowd.
2. Choose **Task > Movement > Crowd Along Line**. The Crowd Along Line parameters display.
3. Select the line the crowd should crowd along.
4. Specify left or right of the line.
5. Click OK.

## 5.10. Crowd Around Location

The Crowd Around Location task causes a crowd to cluster around the specified location.

► **To assign a Crowd Around Location task:**

1. Select a crowd.
2. Choose **Task > Movement > Crowd Around Location**. The Crowd Around Location parameters display.
3. Specify the location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. In the Minimum Distance box, specify the minimum distance between a member of the crowd and the specified location. Some members of the crowd may stand further back, ensuring that they do not collide with others in the crowd
5. Click OK.

## 5.11. Crowd Around Object

The Crowd Around Object task causes a crowd unit to cluster around the specified simulation object or point.

 The Crowd Around Object task calculates target locations for the members of the crowd to move to that are greater than or equal to the minimum distance specified from the object. However, due to interactions between the crowd members as they move, it is possible that some will end up closer to the target object than the minimum distance.

► **To assign a Crowd Around Object task:**

1. Select a crowd.
2. Choose **Task > Movement > Crowd Around Object**. The Crowd Around Object parameters display.
3. In the Filter list, select All Entities or Points.
4. Select the simulation object or point you want the crowd to gather around.
5. In the Minimum Distance box, specify the minimum distance between a member of the crowd and the object. Some members of the crowd may stand further back, ensuring that they do not collide with others in the crowd.
6. Click OK.

## 5.12. Crowd Around Waypoints

The Crowd Around Waypoints task causes members of a crowd to gather at a waypoint and perform the selected behavior for a specified amount of time. You can choose one or more waypoints. If you choose multiple waypoints, the individual crowd members move to the waypoint closest to them.

► **To assign a Crowd Around Waypoints task:**

1. Select a crowd.
2. Choose **Task > Movement > Crowd Around Waypoints**. The Crowd Around Waypoints parameters display.
3. Select one or more waypoints where the crowd should gather.
4. In the Behavior list, select Talk or Pray to indicate what the members of the crowd should do at the waypoint.
5. In the Duration box, enter the number of minutes that the crowd should perform the selected behavior. Once the time has elapsed, the crowd is ready to perform another task. They continue performing the behavior until another task is assigned.
6. In the Minimum Distance box, specify the minimum distance between a member of the crowd and the waypoint. Some members of the crowd may stand further back, ensuring that they do not collide with others in the crowd.
7. Click OK.

## 5.13. Crowd in Front of Entity

The Crowd in Front of Entity task causes a crowd to gather in front of an entity, where the front of the entity is determined based on its heading.

► **To assign a Crowd in Front of Entity task:**

1. Select a crowd.
2. Choose **Task > Movement > Crowd in Front of Entity**. The Crowd in Front of Entity parameters display.
3. Select the entity you want the crowd to gather in front of.
4. Click OK.

## 5.14. DI-Guy Animation

VR-Forces can specify the movements and appearance of DI-Guy models used in the Stealth observer mode and in 3D visualization programs such as VR-Vantage Stealth. The DI-Guy Animation task directs the 3D model to engage in a one-time or repeated series of motions. VR-Forces does not simulate the actions that the 3D model performs; it just adjusts the location and orientation of the VR-Forces entity to synchronize it with the 3D model's animated behavior. To change a 3D model's appearance, use the DI-Guy Appearance set data request. For details, see "DI-Guy Characteristics (Appearance, Head, Body Weight)".

 The DI-Guy Animation task applies only to lifeform entities.

► **To assign a DI-Guy Animation task:**

1. Select the lifeform entity whose behavior you want to animate.
2. Choose **Task > Movement > DI-Guy Animation**. The DI-Guy Animation parameters display.
3. From the DI-Guy Animation list, select an animation.
4. In the Animate group box, specify the duration of the animation.
5. Optionally, select Preview to see the entity perform the animation in the 3D (Stealth) view before you assign the task.
6. Click OK.

## 5.15. DI-Guy Gesture

The DI-Guy Gesture task lets you assign a gesture to a human character. Gestures are motions that a character makes using part of its body. By contrast, animations may involve moving from one place to another.

► **To assign a DI-Guy Gesture task:**

1. Select the lifeform entity whose behavior you want to animate.
2. Choose **Task > Movement > DI-Guy Gesture**. The DI-Guy Gesture parameters display.

3. In the Gestures list, select a gesture.
4. Select the Wait Until Complete check box if you do not want the entity to start another gesture until this one is finished.
5. Optionally, select Preview to see the entity perform the animation in the 3D (Stealth) view before you assign the task.
6. Click OK.

## 5.16. Disperse Crowd

The Disperse Crowd task causes a crowd to wander freely. If the crowd was created as part of a pedestrian area, this task allows it to wander outside the area. (A Wander task would restrict it to the area.) The primary purpose of this task is to disperse a crowd that has previously crowded around an entity or location.

► **To assign a Disperse Crowd task:**

1. Select the crowd you want to disperse.
2. Choose **Task > Movement > Disperse Crowd**.

## 5.17. Find Cover

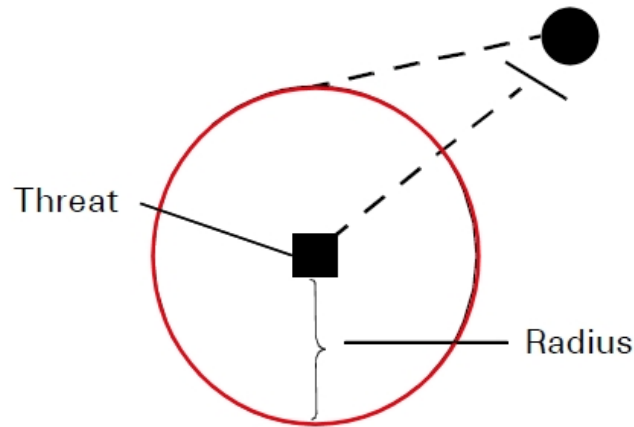
The Find Cover task causes a human entity to find a location from which a specified entity (the threat) cannot see it. The task takes a range, a distance from threat, and threat radius. The Find Cover task requires that the entity be in a navigation area.

The range is the distance from its starting location within which the tasked entity will seek cover. The distance from threat is the minimum distance to maintain between the threat and the tasked entity. For example, you wouldn't expect the entity to run right next to the threat to reach some point on the other side of it. You would expect it to maintain some distance from the threat entity.

The threat radius defines an area around the threat from which line-of-sight calculations are made. Figure 28 illustrates this concept. Line-of-sight between the threat and the covered entity is blocked. However, the covered entity is still visible from a point within the threat radius.

 The entity seeking cover cannot hide behind other entities. It only hides behind terrain features, such as walls or doors.

Figure 28. Threat radius



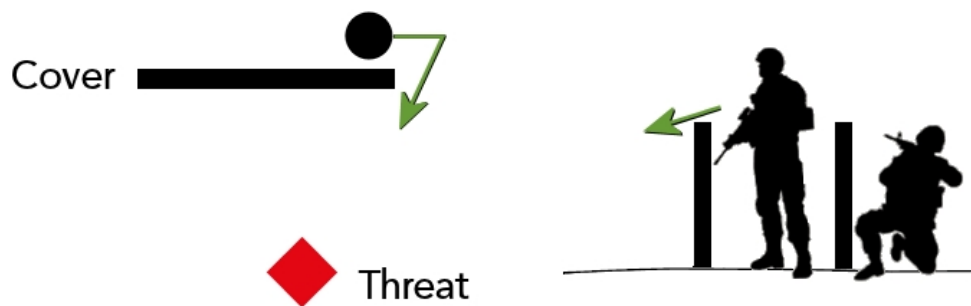
► To assign a Find Cover task:

1. Select an entity.
2. Choose **Task > Movement > Find Cover**. The Find Cover parameters display.
3. Select the entity to find cover from (the threat).
4. Specify the Threat Radius.
5. Specify the Range.
6. Specify the Distance From Threat.
7. Click OK.

## 5.18. Find Firing Position

The Find Firing Position task causes a human entity to find a location from which a specified entity (the threat) cannot see it, but from which it can easily fire on the threat entity. Figure 29 illustrates this concept. The entity under cover can easily move to a location that has line-of-sight to the threat. This could be by moving to the side or looking over the top of the terrain feature it is hiding behind. The Find Firing Position task requires that the entity be in a navigation area.

Figure 29. Firing position



**i** The entity seeking cover cannot hide behind other entities. It only hides behind terrain features, such as walls or doors.



The task takes a range, a distance from threat, and a threat radius value. The range is the distance from its starting position within which the tasked entity will seek cover. The distance from threat is the minimum distance to maintain between the threat and the tasked entity. For example, you wouldn't expect the entity to run right next to the threat to reach some point on the other side of it. You would expect it to maintain some distance from the threat entity.

The threat radius defines an area around the threat from which line-of-sight calculations are made. Figure 28 on the previous page illustrates this concept. Line-of-sight between the threat and the covered entity is blocked. However, the covered entity is still visible from a point within the threat radius.

► **To assign a Find Firing Position task:**

1. Select the entity.
2. Choose **Task > Movement > Find Firing Position**. The Find Firing Position parameters display.
3. Select the entity to find cover from (the threat).
4. Specify the Threat Radius.
5. Specify the Range.
6. Specify the Distance From Threat.
7. Click OK.

## 5.19. Fire Team Bound to Location

The Fire Team Bound to Location task causes a fire team to move in successive bounds to the specified destination while watching the location of a threat.

Initially, the team moves to points that they can use as fire positions while hidden from the threat location. Next, the team splits into two groups and each group then alternates between watching the threat (overwatch) and moving toward the threat. While moving toward the destination, the entities rush forward approximately 10-15 meters and then assume new overwatch positions. If you select the Always Fire check box, then the entities in the overwatch position will fire toward the threat location if they have line of sight.

► **To assign the Fire Team Bound to Location task:**

1. Select the fire team.
2. Choose **Task > Movement > Fire Team Bound to Location**. The Fire Team Bound to Location parameters display.
3. Click on the terrain where you want the fire team to move towards, or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Specify the location of the threat, and optionally the altitude, as described in "Specifying an Object's Altitude". The altitude can be useful if you want to specify a point above the terrain that is more visible to the fire team.
5. If you want the members of the team to fire at the threat location while they are in

the overwatch position, select the Always Fire check box.

6. Click OK.

## 5.20. Fire Team Occupy Fire Position

The Fire Team Occupy Fire Position task causes a unit to find firing positions. This is similar to the Find Firing Position task, but it works for a unit instead of individuals. See "5.18. Find Firing Position" on page 108 for details.

► **To assign the Fire Team Occupy Fire Position task:**

1. Select the unit.
2. Choose **Task > Movement > Fire Team Occupy Fire Position**. The Fire Team Occupy Fire Position parameters display.
3. Specify the Fire Position.
4. Specify the Range.
5. Select the entity to find cover from (the threat).
6. Specify the Threat Radius.
7. Specify the Distance From Threat.
8. Click OK.

## 5.21. Fire Team Travel to Waypoint

You can order a fire team to move to a waypoint in a specified formation. The formations and their spacing are defined in the Simulation Object Editor. You can override the spacing when you assign the task.

 To learn more about configuring formations, refer to "Editing Formations".

► **To assign the Fire Team Travel to Waypoint task:**

1. Select the fire team.
2. Choose **Task > Movement > Fire Team Travel to Waypoint**. The Fire Team Travel to Waypoint parameters display.
3. In the Name box, specify the name of the waypoint to move to.
4. From the Formation list, select the formation that the fire team should move in.
5. Optionally, select the Override Formation Spacing check box to have the entities in the team move with a different spacing than is specified in the SMS.
6. If you chose to override the formation spacing, enter the new spacing in the Minimum Entity Spacing box.
7. Click OK.

## 5.22. Flee from Entities

The Flee from Entities task can be assigned to individual humans and to crowds. It causes the human or crowd to flee from one or more specified entities until it is a specified distance away. The entities must be within a navigation area.

The version of the task for individual humans has an option, Allow Destroyed, which causes the entity continue to flee from the threat even if it is destroyed. In that case, the entity continues to flee to the specified distance. If you do not select that option, the Flee from Entities task ends if the threat is destroyed. Note that the entity might continue to run if it is within the Flee from Threat range when that reactive task is enabled. See "7.14. Reactive Tasks" on page 158.

► **To assign a Flee from Entities task:**

1. Select the entity, either a individual human or crowd.
2. Choose **Task > Movement > Flee From Entities**. The Flee From Entities parameters display.
3. Select each entity that you want the individual human or crowd members to flee from.
4. Specify the radius of the flight zone.
5. For an individual human, you can optionally select the Allow Destroyed check box to have the entity continue to flee from the threat after it is destroyed.
6. Click OK.

## 5.23. Move Between Cover to Location

The Move Between Cover to Location task causes an entity to move to a location by moving between cover points (locations where it is not visible to a specified threat entity). The task requires you to specify a distance from the threat entity and a threat radius.

**i** The entity seeking cover cannot hide behind other entities. It only hides behind terrain features, such as walls or doors.

The distance from threat is the minimum distance to maintain between the threat and the tasked entity. For example, you wouldn't expect the entity to run right next to the threat to reach some point on the other side of it. You would expect it to maintain some distance from the threat entity.

VR-Forces calculates cover points based on line-of-sight from a threat radius around the threat entity. Figure 28 illustrates this concept. Line-of-sight between the threat and the covered entity is blocked. However, the covered entity is still visible from a point within the threat radius.

**i** The default values for threat radius and distance from threat are fairly small. Unless you are playing in a congested area, you will probably want to change them.

► **To assign a Move Between Cover to Location task:**

1. Select the entity.
2. Choose **Task > Movement > Move Between Cover to Location**. The Move Between Cover to Location parameters display.
3. Click on the terrain to specify the location, or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Select the Threat entity.
5. In the Threat Radius box, specify the threat radius.
6. In the Distance From Threat box, specify the minimum distance to maintain between the threat entity and the moving entity.
7. Click OK.

## 5.24. Open Door

If a human is within approximately one meter of a closed door, this task causes the door to open. This task applies to doors controlled by switch nodes in the terrain, not articulated parts. For more information about opening and closing doors, see the chapter on [Dynamic Terrain in VR-Forces](#) in the *Adding Content to MAK ONE Applications Reference Manual*.

► **To assign an Open Door task:**

1. Move a human to within one meter of a closed door.
2. Choose **Task > Terrain > Open Door**.

## 5.25. Open Window

If a human is within approximately one meter of a closed window, this task causes the window to open. This task applies to windows controlled by switch nodes in the terrain, not articulated parts. For more information about opening and closing windows, see the chapter on [Dynamic Terrain in VR-Forces](#) in the *Adding Content to MAK ONE Applications Reference Manual*.

► **To assign an Open Window task:**

1. Move a human to within one meter of a closed window.
2. Choose **Task > Terrain > Open Window**.

## 5.26. Parachute to Target

The Parachute to Target task causes a paratrooper entity to disembark from an aircraft, deploy its parachute, and try to land at the target object or location. The paratrooper tries to land as close to the target as it can given the altitude at which it disembarks and the distance to the target. If you specify a muster point, the paratrooper walks to that point after landing.

- ①
- While an entity is parachuting, if you double-click it to try to move it, it snaps to the terrain.
  - The Air Drop task can also cause paratroopers to parachute. For details, see "3.1. Airdrop Along Route" on page 40.

► **To assign a Parachute to Target task:**

1. Select the paratrooper.
2. Choose **Task > Movement > Parachute to Target**. The Parachute to Target parameters display.
3. Specify a target point where the paratrooper will try to land.
4. Optionally, select Use Muster Point and then specify where the paratrooper should move after it lands.
5. Specify the altitude at which to deploy the parachute. If the aircraft is below the deployment altitude, the parachute deploys as soon as the entity disembarks.
6. By default, Deploy Altitude is AGL is selected to specify that the deployment altitude is above ground level instead of mean sea level (MSL). Unselect the check box to use MSL instead.
7. Optionally, select Deploy After Elapsed Time and enter the Elapsed Time to Deploy. The entity uses this instead of the deployment altitude to determine at what point to deploy the parachute.
8. Click OK.

## 5.27. Protest Along Line

The Protest Along Line task causes a crowd to gather along a specified line and continuously execute a series of protest animations.

► **To assign a Protest Along Line task:**

1. Select the crowd.
2. Choose **Task > Movement > Protest Along Line**. The Protest Along Line parameters display.
3. Specify a line.
4. Specify which side of the line to protest on.
5. Click OK.

## 5.28. Protest Around Location

The Protest Around Location task causes a crowd to gather around a specified location and continuously execute a series of protest animations.

► **To assign a Protest task:**

1. Select the crowd.
2. Choose **Task > Movement > Protest**. The Protest Around Location parameters display.
3. Specify a location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Click OK.

## 5.29. Riot

The Riot task causes a crowd to move into an area specified for the riot, going to Riot points or other places in the area. The members of the crowd run and perform angry gestures. You can also specify that the rioters should start fires, which are ignited at each riot point inside the area. Rioters ignore riot points placed outside the area.

The area where the riot occurs can be any Area, while Riot Points are a specific type of simulation effect object.

► **To assign a Riot task:**

1. Select a crowd.
2. Choose **Task > Movement > Riot**. The Riot parameters display.
3. Select an area where the riot will occur.
4. Optionally, select the Start Fires check box. Rioters will start fires at the riot points inside the specified area.
5. Click OK.

## 5.30. Rush to Location

The Rush to Location task causes an entity to move quickly to the specified location and then assume a prone position or, optionally, a kneeling position.

► **To assign a Rush to Location task:**

1. Select an entity.
2. Choose **Task > Movement > Rush to Location**. The Rush to Location parameters display.
3. Specify a location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Specify the location of the threat.
5. Optionally, select the Destination Is High Cover check box. This causes the entity to kneel at the specified location rather than lying prone.
6. Click OK.

## 5.31. Squad Travel to Waypoint

You can order a squad to move to a waypoint. The squad's formation is a lead fire team that moves in a wedge formation, followed by the squad leader and then the trailing fire team. The spacing of the formation is defined in the Simulation Object Editor but you can override it when you assign the task.

 To learn more about configuring formations, refer to "Editing Formations".

If you select the Use Overwatch check box, then the squad leader and the trailing fire team follow the lead fire team at a greater distance. Another waypoint is created just past the waypoint specified in the task. The lead fire team gathers at this new waypoint, while the squad leader and the trailing fire team gather at the task's waypoint.

► **To assign the Squad Travel to Waypoint task:**

1. Select the squad.
2. Choose **Task > Movement > Squad Travel to Waypoint**. The Squad Travel to Waypoint parameters display.
3. In the Destination box, enter the name of the waypoint to move to. You can select it from a list, on the terrain, or enter the name.
4. Optionally, select the Use Overwatch check box to increase the following distance between the lead fire team and the squad leader and the trailing fire team.
5. Optionally, select the Override Formation Spacing check box to have the entities in each fire team move with a formation spacing that is different than is specified in the SMS.
6. If you chose to override the formation spacing, enter the new spacing in the Minimum Entity Spacing box, in meters.
7. Click OK.

## 5.32. Visit Neighbor

Use the Visit Neighbor task to have a human entity move to a nearby human or ground entity.

► **To assign the Visit Neighbor task:**

1. Select the human entity.
2. Choose **Task > Movement > Visit Neighbor**. The Visit Neighbor parameters display.
3. Enter the Radius.
4. Click OK.

## 5.33. Wait in Queue

The Wait in Queue task causes a human to move to the beginning of a Queue simulation effect object and then move along the line until it stands behind the last human in the line. (The first human in line moves to the end point of the simulation effect object, which is

the beginning of the queue.) As people leave the top of the queue, the entity moves up in line.

Since this is a Wait task, the entity does not move to the next task in its plan until it is released from the queue. Entities are released from a queue when the Release Next Entity From Queue task is sent to the queue. For more information, see "7.12. Release Next Entity From Queue" on page 158.

 The Wait in Queue task only works with a Queue simulation effect object. It does not work with other types of lines.

► **To assign a Wait in Queue task:**

1. Select the entity.
2. Choose **Task > Wait > Wait in Queue**. The Wait in Queue parameters display.
3. Select the Queue that the entity should wait in.
4. Click OK.

## 5.34. Wander

The Wander task causes a lifeform entity to move to a series of randomly chosen locations for the specified amount of time. You can constrain wandering to a specified area. If the entity is using a preferred navigation mode and there is preferred road data for the terrain, the Wander task will only choose destinations that meet the preference criteria. (For information about setting navigation preferences, see "Navigation Preferences".) The entity must be in a navigation area.

► **To assign a Wander task:**

1. Select the entity.
2. Choose **Task > Movement > Wander**. The Wander parameters display.
3. Select a Duration option: Indefinitely or Timed. If you select Timed, specify the amount of time to wander in days, hours, minutes, and seconds.
4. Select a Movement option: Free or Restricted To Area. If you select Restricted To Area, enter the name of the area where you want the entity to wander.
5. From the Destination Restriction list, choose an option for how the entity will move.
6. Click OK.



## 5.35. Wander In Area

The Wander In Area task causes a crowd to wander in the specified area. The entity must be in a navigation area.

► **To assign a Wander In Area task:**

1. Select the crowd.
2. Choose **Task > Movement > Wander In Area**. The Wander In Area parameters display.
3. Select the area to wander in.
4. Click OK.





## 6. Unit Behavior Tasks

This chapter describes the complex tasks for units that apply to entity-level scenarios. These tasks involve a number of subtasks and are more complex than the tasks covered in other chapters.

|                                                          |     |
|----------------------------------------------------------|-----|
| 6.1. Overview of Unit Behaviors .....                    | 120 |
| 6.2. Drop Off Passengers at Location .....               | 120 |
| 6.3. Pick Up Passengers at Location .....                | 121 |
| 6.4. Tasks for Mounting and Dismounting Units .....      | 121 |
| 6.4.1 Dismount Headquarters .....                        | 121 |
| 6.4.2 Dismount Platoon .....                             | 122 |
| 6.4.3 Dismount Squad .....                               | 122 |
| 6.4.4 Dismount Team .....                                | 122 |
| 6.4.5 Mount Headquarters .....                           | 123 |
| 6.4.6 Mount Platoon .....                                | 123 |
| 6.4.7 Mount Squad .....                                  | 124 |
| 6.4.8 Mount Team .....                                   | 124 |
| 6.5. Movement Tasks for Unit Behaviors .....             | 124 |
| 6.5.1 Bound In Lanes .....                               | 124 |
| 6.5.2 Bounding Overwatch Successive Continuous .....     | 125 |
| 6.5.3 Follow Along Offset Route .....                    | 125 |
| 6.6. Defensive Tasks for Unit Behaviors .....            | 126 |
| 6.6.1 Assume Coil Formation .....                        | 126 |
| 6.6.2 Provide Mobile Overwatch .....                     | 126 |
| 6.6.3 Provide Stationary Overwatch .....                 | 126 |
| 6.7. Offensive Tasks for Unit Behaviors .....            | 127 |
| 6.7.1 Assault .....                                      | 127 |
| 6.7.2 Attack by Fire .....                               | 127 |
| 6.7.3 Attack By Fire in Rotation .....                   | 128 |
| 6.7.4 Clear .....                                        | 128 |
| 6.7.5 Engage From Vehicle/Soldier Firing Positions ..... | 128 |

|                                 |     |
|---------------------------------|-----|
| 6.7.6 Movement To Contact ..... | 129 |
| 6.7.7 Seize Objective .....     | 129 |
| 6.7.8 Suppressive Fire .....    | 130 |

## 6.1. Overview of Unit Behaviors

VR-Forces supports complex unit behaviors using the VR-Forces Lua Behavior Engine. Military unit behaviors often depend on the terrain. Having an armored company seize an objective in a desert is radically different than having a company seize an objective in a dense city or a forest. Things such as roads, rivers, railroads, bridges, and other natural and man-made obstacles can have a significant impact on how a task is performed.

MAK has designed these behaviors using open source United States doctrine to work in an open terrain such as the U.S. National Training Center at Fort Irwin. This is a fairly empty terrain designed for desert warfare. While the tasks may work well in other similar terrains, your results may vary. We do not support all tasks in all terrains. The VR-Forces behavior engine allows for a tremendous amount of complexity in behaviors, and we believe some modification to these behaviors may be needed for any deployed system. The scripts used to design these tasks are provided in VR-Forces for you to adjust and update to meet your doctrinal and terrain goals. Fully understanding how to extend these behaviors is complicated and MAK recommends developer training to best meet complex behavior needs. Please contact us at [sales@mak.com](mailto:sales@mak.com) so that we can help you further understand your options.

There are some concepts you should understand to use these tasks most effectively.

- **Vehicle Crew.** A vehicle crew is a notional subordinate of unit defined with the Vehicle Crew function. See "Defining Functions for Subordinates". Tasks for crews require that the unit knows what vehicles are associated with it, and so require that the unit has a vehicle crew in its unit composition.
- **Base of fire and maneuver elements.** When using an overwatch defense task, you specify one or more maneuver elements. These are the members of a unit who will move and shoot.

There are several Behavior Ground scenarios in `./userData/scenarios/Sample/VR-TheWorld_Online` that include several vignettes that demonstrate some of the unit behaviors.

 Field artillery units in entity-level scenarios are aware of the subordinates and the weapons in the unit, knowing their capabilities and remaining munition rounds. This allows the unit to determine which subordinates can be tasked in an assault.

## 6.2. Drop Off Passengers at Location

Use the Drop Off Passengers at Location task to instruct a vehicle crew to drop off their squad at a given location.

► **To task a vehicle crew to drop off the squad:**

1. Select the crew.
2. Choose **Task > Embarkation > Drop Off Passengers at Location**. The Drop Off Passengers at Location parameters display.
3. Specify the drop off location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. In the Objects List, select the passengers to drop off. They are added to the Passengers list.
5. Enter the heading that the crew should face when they arrive at the drop off location.
6. Click OK.

## 6.3. Pick Up Passengers at Location

Use the Drop Off Passengers at Location task to instruct a vehicle crew to pick up their squad at a given location.

► **To task a vehicle crew to drop off the squad:**

1. Select the unit's vehicle crew.
2. Choose **Task > Embarkation > Pick Up Passengers at Location**. The Pick Up Passengers at Location parameters display.
3. Specify the pickup location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Enter the heading that the crew should face when they arrive at the pickup location.
5. Click OK.

## 6.4. Tasks for Mounting and Dismounting Units

There are different tasks for mounting and dismounting units from vehicles. The one you should use depends on the type of unit.

### 6.4.1 Dismount Headquarters

Use the Dismount Headquarters task to tell an infantry platoon headquarters to exit a vehicle and gather at a rally point. Optionally, you can specify whether the platoon sergeant or other habitually mounted entities remain on board.

► **To task a headquarters section to dismount a vehicle:**

1. Select the unit.
2. Choose **Task > Embarkation > Dismount Headquarters**. The Dismount Headquarters parameters display.

3. Specify the location of the rally point. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Optionally, select or clear the Retain Habitual Mountees check box.
5. Click OK.

### 6.4.2 Dismount Platoon

Use the Dismount Platoon task to tell an infantry platoon to exit a vehicle and gather in formation around a point.

► **To task a platoon unit to dismount a vehicle:**

1. Select the platoon.
2. Choose **Task > Embarkation > Dismount Platoon**. The Dismount Platoon parameters display.
3. For the Dismount Point, specify the waypoint.
4. Enter the direction that the platoon should face. Also specify whether the direction is relative to True or Magnetic north.
5. Select the Vehicle Formation from the list.
6. From the Threat Level list, selected the expected threat from any nearby enemies. This determines the posture that the dismounted subunits will assume.
7. Optionally, select which subunits should dismount. Unselected units remain on the vehicle.
8. Click OK.

### 6.4.3 Dismount Squad

The Mount Headquarters task causes an infantry squad to exit a vehicle and gather at a rally point.

► **To task a squad unit to dismount a vehicle:**

1. Select the squad.
2. Choose **Task > Embarkation > Dismount Squad**. The Dismount Squad parameters only.
3. From the Threat Level list, selected the expected threat from any nearby enemies. This determines the posture that the dismounted entities will assume.
4. Optionally, select the Dismount Crew check box to also dismount the vehicle team.
5. Click OK.

### 6.4.4 Dismount Team

Use the Dismount Team task to tell an infantry team to exit a vehicle and gather at a rally point.

► **To task a team unit to dismount a vehicle:**

1. Select the team.
2. Choose **Task > Embarkation > Dismount Team**. The Dismount Team parameters display.
3. Specify the coordinates of the rally point or click a point on the terrain.
4. Click OK.

### 6.4.5 Mount Headquarters

The Mount Headquarters task causes an infantry platoon headquarters section to gather around a rally point and then board a vehicle. The vehicle is the one specified by the Assign Transport set data request. See "Assign Transport".

► **To task a headquarters section to mount a vehicle:**

1. Select the HQ section.
2. Choose **Task > Embarkation > Mount Headquarters**. The Mount Headquarters parameters display.
3. Specify the location of the rally point. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. In the Transports List or the window, select the entities to mount.
5. Click OK.

### 6.4.6 Mount Platoon

The Mount Platoon task causes an infantry platoon unit to gather around a point and then board a vehicle. The vehicle is the one specified by the Assign Transport set data request. See "Assign Transport".

► **To task a platoon to mount a vehicle:**

1. Select the platoon.
2. Choose **Task > Embarkation > Mount Platoon**. The Mount Platoon parameters display.
3. For the Mount Point, specify a point.
4. Enter the direction that the platoon should face when they arrive at the mount point. Additionally, specify whether the direction is relative to True or Magnetic north.
5. Select the Vehicle Formation from the list. The platoon assumes this formation.
6. From the Threat Level list, selected the expected threat from any nearby enemies. This determines the posture that the dismounted subunits will assume.
7. Optionally, select which subunits should mount the vehicle. Unselected units remain in formation at the Mount Point.
8. Click OK.

### 6.4.7 Mount Squad

The Mount Squad task causes an infantry squad to board a vehicle. You can choose to have the squad form up at a rally point before boarding the vehicle. The vehicle is the one specified by the Assign Transport set data request; see "Assign Transport". When tasking the squad, you can assign a different vehicle.

► **To task a squad to mount a vehicle:**

1. Select the squad unit.
2. Choose **Task > Embarkation > Mount Squad**. The Mount Squad parameters display.
3. Optionally, select the Reassign Transport check box and then specify a vehicle.
4. Specify the location of the rally point. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
5. Enter the direction that the squad should face. Additionally, specify whether the direction is relative to True or Magnetic north.
6. Select the Formation from the list.
7. From the Threat Level list, selected the expected threat from any nearby enemies. This determines the posture that the dismounted subunits will assume.
8. Click OK.

### 6.4.8 Mount Team

The Mount Team task causes an infantry squad to board a vehicle. You can choose to have the team form up at a rally point before boarding the vehicle. The vehicle is the one specified by the Assign Transport set data request; see "Assign Transport". When tasking the team, you can assign a different vehicle.

► **To task a team to mount a vehicle:**

1. Select the team unit.
2. Choose **Task > Embarkation > Mount Team**. The Mount Team parameters display.
3. Optionally, select the Reassign Transport check box and then specify a vehicle.
4. Optionally, specify the coordinates of the rally point or click a point on the terrain.
5. Click OK.

## 6.5. Movement Tasks for Unit Behaviors

Units can execute different movement tasks.

### 6.5.1 Bound In Lanes

The Bound In Lanes task causes a fire team to move in parallel lanes, in successive bounds, to assault a threat location.



► **To assign the Bound In Lanes task:**

1. Select the fire team.
2. Choose **Task > Movement > Bound In Lanes**. The Bound In Lanes parameters display.
3. Choose the Enemy Area.
4. Specify the Start Point for the movement. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
5. Click OK.

### 6.5.2 Bounding Overwatch Successive Continuous

The Bounding Overwatch Successive task causes a mechanized ground unit to move along a route in successive bounds. They will continue along the route, regardless of whether they contact opposing forces. You specify the minimum and maximum bounding lengths, as well as the possible threat locations that the unit should watch.

► **To assign the Bounding Overwatch Successive Continuous task:**

1. Select the platoon.
2. Choose **Task > Movement > Bounding Overwatch Successive Continuous**. The Bounding Overwatch Successive Continuous parameters display.
3. Select the Travel Route.
4. Enter a Minimum Bound Length.
5. Enter a Maximum Bound Length.
6. Select one or more Enemy Locations, which are area tactical graphics.
7. Click OK.

### 6.5.3 Follow Along Offset Route

The Follow Along Offset Route task causes one ground entity follow another at a specified offset. The leader is expected to be tasked to move. If you tell two or more entities to follow the same simulation object, make sure that you do not give them the same offset (see "Following Simulation Objects").

► **To assign the Follow Along Offset Route task:**

1. Select the entity that will follow another.
2. Choose **Task > Movement > Follow Along Offset Route**. The Follow Along Offset Route parameters display.
3. Select the Leader Route.
4. Select the Leader.
5. Enter the offset parameters.
6. Click OK.

## 6.6. Defensive Tasks for Unit Behaviors

Units can execute different defense tasks.

### 6.6.1 Assume Coil Formation

The Assume Coil Formation task causes a mechanized ground unit to move into a defensive coil, facing outward and distanced from a central point at the specified radius.

► **To assign the Assume Coil Formation task:**

1. Select the tank platoon.
2. Choose **Task > Defense > Assume Coil Formation**. The Assume Coil Formation parameters display.
3. Specify the Radius.
4. Click OK.

### 6.6.2 Provide Mobile Overwatch

The Provide Mobile Overwatch task tells member of a vehicle crew to provide overwatch of an enemy area, moving with the crew's squad.

When you assign the Provide Mobile Overwatch task, there are parameters to specify:

- Maneuver Elements. The members of the unit that are moving. The vehicle associated with the unit provides the overwatch support.
- Enemy area. The area where the enemy is located.

► **To task a vehicle crew to provide mobile overwatch:**

1. Select the crew.
2. Choose **Task > Defense > Provide Mobile Overwatch**. The Provide Mobile Overwatch parameters display.
3. Select a Maneuver Element object. Choose one or more entities or units in the Objects List or terrain.
4. For the Enemy Area, click the Select button () and then select the area.
5. Enter the Overwatch Distance at which the base of fire element will trail the Maneuver Elements.
6. Click OK.

### 6.6.3 Provide Stationary Overwatch

The Provide Stationary Overwatch task tells members of a vehicle crew to provide overwatch while remaining at the overwatch location.

When you assign the Provide Stationary Overwatch task, there are parameters to specify:

- Overwatch location. The vehicle moves to this location and provides overwatch defense from there.

- Maneuver Elements. The members of the unit that are moving. The vehicle associated with the unit provides the overwatch support.
- Enemy area. The area where the enemy is located.

► **To task a vehicle crew to provide stationary overwatch:**

1. Select the crew.
2. Choose **Task > Defense > Provide Stationary Overwatch**. The Provide Stationary Overwatch parameters display.
3. Specify the Overwatch Location. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Select a Maneuver Element object. Choose one or more entities or units in the Objects List or terrain.
5. Optionally, select the Enemy Area.
6. Click OK.

## 6.7. Offensive Tasks for Unit Behaviors

Units can execute different offense tasks.

### 6.7.1 Assault

The Assault task causes a mechanized ground unit to assault an area without moving past a specified simulation object.

► **To assign the Assault task:**

1. Select the unit.
2. Choose **Task > Offense > Assault**. The Assault parameters display.
3. Select the Enemy Area.
4. Select the line where the unit will set up its Support By Fire Position.
5. Specify the location of the Assault Position. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
6. Select the simulation object to be the Limit of Advance. The unit will not move past this point.
7. Click OK.

### 6.7.2 Attack by Fire

The Attack By Fire task causes a mechanized ground unit to move to a line where they take position and fire at an area. The unit can fire continuously, or stop after a specified length of time when it does not detect contacts in the area.

► **To assign an Attack By Fire task:**

1. Select the unit.
2. Choose **Task > Offense > Attack By Fire**. The Attack By Fire parameters display.
3. Select the Line of Fire that represents the hull-down position for the tank platoon.
4. Select the Enemy Area to use as the engagement zone.
5. In the Time Out box, indicate the length of time (in seconds) that the tank platoon should wait between contacts before ending the task.
6. Select the Do Fires Indefinitely check box to have the tank platoon continue firing until their ammunition is depleted.
7. Click OK.

### 6.7.3 Attack By Fire in Rotation

The Attack By Fire in Rotation task causes a mechanized ground unit to move to and rotate through a battle position, taking turns firing at an area.

► **To assign an Attack By Fire in Rotation task:**

1. Select the unit.
2. Choose **Task > Offense > Attack By Fire in Rotation**. The Attack By Fire in Rotation parameters display.
3. Specify the Battle Position. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Select the Target Area to use as the engagement zone.
5. Click OK.

### 6.7.4 Clear

A mechanized ground unit tasked to perform the Clear task lines up at a starting position and then moves to a line, attempting to remove all opposing forces along the way.

► **To assign the Clear task:**

1. Select unit.
2. Choose **Task > Offense > Clear**. The Clear parameters display.
3. Select the line that is the Limit of Advance. Surviving members of the platoon stop when they reach the line.
4. Select the Starting Point where the platoon will assemble.
5. Click OK.

### 6.7.5 Engage From Vehicle/Soldier Firing Positions

Use the Engage From Vehicle (or Soldier) Firing Positions to have a mechanized ground entity move first to a hide position, pause, continue on to a fire position, and then fire on an enemy location.

► **To assign the Engage From Vehicle (or Soldier) Firing Positions task:**

1. Select the entity.
2. Choose **Task > Engagement > Engage From Vehicle Firing Positions** (or **Engage From Soldier Firing Positions**). The parameters display.
3. Specify the location of the Hull Down Position (or Shoot Position). You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Specify the location of the Hide Position.
5. Select the Delayed Start check box to have the entity pause and allow another entity to engage first.
6. Select the Enemy Area.
7. Click OK.

### 6.7.6 Movement To Contact

The Movement To Contact task causes a mechanized ground unit to move towards an objective. Upon making contact with opposing forces, the task ends with the unit performing the specified action.

► **To assign the Movement To Contact task:**

1. Select the unit.
2. Choose **Task > Offense > Movement To Contact**. The Movement to Contact parameters display.
3. Select the Axis of Attack, which is a line.
4. Select the Objective, which is an ellipse or circle.
5. Specify the Action on Contact: Engage or Ignore.
6. Click OK.

### 6.7.7 Seize Objective

The Seize Objective task causes a tank company to move to take control of an area.

► **To assign the Seize Objective task:**

1. Select the company.
2. Choose **Task > Offense > Seize Objective**. The Seize Objective parameters display.
3. Select the Objective Area for the company to seize.
4. Select the Fire Positions, which are lines where platoons within the company should position themselves to provide supportive fire.
5. Select one Line of Departure.
6. Select the Seize Direction: Direct, Left, or Right.

7. If the platoons within the company should make the attack, select the Two Platoon Assault check box. If one platoon should provide supporting fire while the other seizes the objective, clear the check box.
8. If there is an artillery unit that can provide support for the company, select it in the Artillery box.
9. Optionally, specify the Assault Route that the company will follow to move to the objective area.
10. Click OK.

### 6.7.8 Suppressive Fire

The Suppressive Fire task causes the subordinates of a mechanized ground unit to fire at a specified simulation object. If other opposing forces are detected, a subordinate engages with them.

► **To assign the Suppressive Fire task:**

1. Select the unit.
2. Choose **Task > Offense > Suppressive Fire**. The Suppressive Fire parameters display.
3. Use the Filter to select whether you want to target a ground entity or a point.
4. Select the target.
5. Click OK.



## 7. Embarkation, Wait, Radio, and Other Tasks

This chapter describes the tasks supplied with VR-Forces that are categorized as Embarkation, Radio, Wait, Other, and uncategorized.

|                                        |     |
|----------------------------------------|-----|
| 7.1. Embarkation Tasks .....           | 133 |
| 7.1.1 Disembark .....                  | 133 |
| 7.1.2 Disembark All .....              | 133 |
| 7.1.3 Embark .....                     | 133 |
| 7.1.4 Airdrop .....                    | 135 |
| 7.1.5 Airdrop from Aircraft .....      | 136 |
| 7.1.6 Rope Sling Load Pick Up .....    | 137 |
| 7.1.7 Rope Sling Load Drop Off .....   | 137 |
| 7.1.8 Fast Rope Insertion .....        | 137 |
| 7.1.9 Rope Suspension Extraction ..... | 138 |
| 7.1.10 Extract Using Rope .....        | 139 |
| 7.1.11 Rope Suspension Insertion ..... | 140 |
| 7.1.12 Ladder Extraction .....         | 140 |
| 7.1.13 Extract Using Ladder .....      | 141 |
| 7.1.14 Ladder Insertion .....          | 142 |
| 7.1.15 Insert Using Ladder .....       | 143 |
| 7.1.16 Dismount .....                  | 143 |
| 7.2. Internal Transfer of Fuel .....   | 144 |
| 7.3. Transfer Supplies .....           | 144 |
| 7.4. IFF Tasks .....                   | 144 |
| 7.4.1 IFF Interrogate .....            | 145 |
| 7.4.2 IFF Sweep Interrogate .....      | 146 |
| 7.5. Radio Tasks .....                 | 147 |
| 7.5.1 Send Radio Set .....             | 147 |
| 7.5.2 Send Radio Task .....            | 148 |
| 7.5.3 Send Text Message .....          | 148 |
| 7.6. Wait Tasks .....                  | 149 |

|                                                       |     |
|-------------------------------------------------------|-----|
| 7.6.1 Wait .....                                      | 149 |
| 7.6.2 Wait Duration .....                             | 149 |
| 7.6.3 Wait Elapsed .....                              | 150 |
| 7.7. Tasks for Moving Articulated Parts .....         | 150 |
| 7.7.1 Open Car Door .....                             | 150 |
| 7.7.2 Close Car Door .....                            | 150 |
| 7.7.3 Open Cargo Door .....                           | 151 |
| 7.7.4 Close Cargo Door .....                          | 151 |
| 7.7.5 Deploy Refueling Boom .....                     | 151 |
| 7.7.6 Stow Refueling Boom .....                       | 152 |
| 7.7.7 Deploy Port or Starboard Refueling Drogue ..... | 152 |
| 7.7.8 Stow Port or Starboard Refueling Drogue .....   | 153 |
| 7.7.9 Extend Towed Array Decoy .....                  | 153 |
| 7.7.10 Retract Towed Array Decoy .....                | 154 |
| 7.7.11 Raise Periscope .....                          | 154 |
| 7.7.12 Lower Periscope .....                          | 154 |
| 7.8. Embedded Entity Tasks .....                      | 154 |
| 7.8.1 Deploying Embedded Entities .....               | 154 |
| 7.8.2 Recovering Deployed Entities .....              | 155 |
| 7.8.3 Sector Search Operation .....                   | 155 |
| 7.8.4 Deploying an Entity by Assigning a Task .....   | 156 |
| 7.9. Deploy Sonobuoy .....                            | 156 |
| 7.10. Deploy Sonobuoys Along Route .....              | 157 |
| 7.11. Sonar Dip .....                                 | 157 |
| 7.12. Release Next Entity From Queue .....            | 158 |
| 7.13. User Task .....                                 | 158 |
| 7.14. Reactive Tasks .....                            | 158 |
| 7.15. Background Processes .....                      | 160 |



## 7.1. Embarkation Tasks

This section describes tasks that simulate embarking and disembarking. Simulation objects move to the parent simulation object to embark, and move away after disembarking. For immediate embarkation and disembarkation, use the Embarked and Disembarked set data requests or the embarkation commands on the Objects menu. For a conceptual discussion of embarkation, see "Embarking and Disembarking Simulation Objects".

### 7.1.1 Disembark

The Disembark task causes a simulation object to move to the parent simulation object's egress (exit) point and disembark. If the selected simulation object is a unit, the disembark task is given to all embarked subordinates. The task is considered complete when all subordinates are disembarked.



- When a simulation object disembarks, it goes to the disembarkation location. However, it does not move away from that location. If you plan to disembark several simulation objects from the same parent by giving them disembark tasks, you are responsible for making sure that each simulation object moves from the disembarkation point before the next simulation object disembarks.
- The Fast Rope Insertion task causes infantry to disembark from a helicopter. For details, see "7.1.8 Fast Rope Insertion" on page 137.

► **To assign a Disembark task:**

1. Select the simulation object you want to disembark.
2. Choose **Task > Embarkation > Disembark**, or click the Disembark button ()

### 7.1.2 Disembark All

The Disembark All task is given to a parent simulation object. It gives a Disembark task to all embarked simulation objects except aggregates. When you issue a Disembark All task, VR-Forces tries to disembark the simulation objects sequentially and have them move to sufficiently dispersed locations to avoid collision problems.

► **To assign a Disembark All task:**

1. Select the parent simulation object.
2. Choose **Task > Embarkation > Disembark All**.

### 7.1.3 Embark

The Embark task causes a simulation object to move to the ingress (entry) point of the parent simulation object and embark on it. If the embarking simulation object is a unit, each available taskable subordinate is given an embark task. When all subordinates have embarked, the task is considered complete.

The Embark task is valid only if the specified parent simulation object is configured to accept the simulation object that wants to embark on it. If you task a simulation object to embark on a simulation object that is not configured to receive it, for example, you task a

DI to embark on an M1A2, the simulation object will not respond to the task. If you want to embark a simulation object on a simulation object that is not configured to receive it, you can use the immediate embarkation process. For details, see "Embarking and Disembarking Simulation Objects".

Human characters have a set of animations that allow them to open and close car doors and get into and out of vehicles that have interior geometry and articulated doors. For details, see "Embarking Humans in Cars" on the facing page.



- If you are using HLA and want to use the embarkation feature, you must use RPR FOM 2, draft 17 or later.
- If the embarkation spaces on the parent simulation object are full, VR-Forces sends a message to the sim. engine console and the simulation object does not try to embark.
- If you give multiple simulation objects embark tasks for the same parent, you are responsible for spreading the tasks over time so that the simulation objects do not all converge on the parent at the same time, which would result in problems due to collision avoidance.
- A simulation object is considered embarked as soon as it reaches its ingress point. In plan view mode, its icon is hidden immediately. However, in a 3D mode, the simulation object may need to continue moving to reach its slot on the parent. If you task the parent simulation object to move before a simulation object has moved to its slot, the embarking 3D model may end up in an odd position relative to the parent simulation object. To avoid this visual problem, allow time for the embarking simulation object to stop moving before the parent entity moves.

► **To assign an Embark task:**

1. Select the simulation object.
2. Choose **Task > Embarkation > Embark**, or click the Embark button (). The Embark On dialog box opens.



Unlike the Embark On dialog box that opens when you choose **Objects > Embarkation > Embark**, the Embark On dialog box for the Embark task does not allow you to choose the embarkation point.

3. Select the parent simulation object (the simulation object to embark on) or type its name in the Name box.
4. If the parent simulation object has slots of a specific type and you want the simulation object to embark in a slot of a particular type, select the type in the Slot Type list. Slots of that type are listed in the Slot Name list. (For information about slot types and slot names, see "Configuring Slots".)
5. Select a slot from the Slot Name list.
6. If you want the simulation object to use any available slot, select the Choose Any

Slot if Requested Slot Not Available check box.

7. Click OK.



### Embarking Humans in Cars

Some human characters have animations that allow them to open and close car doors and get into and out of cars. Humans get into and out of cars using the Embark and Disembark tasks.

When you give an entity an embark task for a car, you can specify that it embark in the driver's seat or a passenger seat. If you choose a passenger seat, you can specify which one to use. You do this by specifying the slot to embark in, as described in "7.1.3 Embark" on page 133.

If you do not specify the slot to use, the first entity that embarks in a car gets into the driver's seat. Additional entities embark in passenger seats until the number of slots available is used up.

The embark in cars capability is primarily intended for use with aerial surveillance scenarios. Viewing embarkation at ground level will not provide a completely realistic experience, including:

- You cannot change the speed at which entities enter and exit the vehicle and open and close the door. For example, they cannot run and jump into a car or jump out of a car and immediately run away.
- Contact between an entity's hand and the car door will not be entirely lifelike.
- If Show Occupants is enabled, an entity that embarks in the driver's seat will get in on top of the automatically displayed driver. We recommend you disable occupants for all cars or at least the car to be embarked on. For information about showing and hiding occupants, see "Hide Occupants" and "Displaying DI-Guys in Vehicles".
- The entry and exit animations are calibrated for civilian vehicles. Trying to embark on a large vehicle, such as a truck that has doors that are high off the ground, is not taken into account.
- This release does not support drivers in the right front seat.

#### 7.1.4 Airdrop

The Airdrop task is used by aircraft to disembark paratroopers or entities such as vehicles. It specifies only when the objects embarked on the aircraft will parachute, and it can specify the number of "drop slots" available to those entities.

On an aircraft, drop slots indicate the number of intervals at which entities can exit the aircraft with the Airdrop from Aircraft task. On entities, the drop slot indicates the order in which they will perform that task. Drop slots do not support units, and they are not the same as embarkation slots.

The aircraft performing the Airdrop task must have the Air Drop system (*airDrop.sysdef*). The embarked entities must have a task for deploying the parachute or they will not exit the aircraft. See "7.1.5 Airdrop from Aircraft" on the next page for details.

 The movement task Parachute to Target does not coordinate with Airdrop.

► **To assign an airdrop task:**

1. Select the aircraft that the parachuting entities are embarked upon.
2. Choose **Task > Embarkation > Airdrop**. The Airdrop parameters display.
3. Indicate the number of seconds between drop. Entities will exit the aircraft at that interval.
4. Optionally, you can select the Use Drop Slots check box and then define the number of drop slots. One or more entities can exit the aircraft at each drop slot, as specified in the Airdrop from Aircraft task.
5. Click OK.

### 7.1.5 Airdrop from Aircraft

Paratrooper entities and simulation objects with the Air Drop Capable system (such as vehicles) can parachute from an aircraft on which they are embarked. The aircraft must have a task for airdropping the entity. For more information, see "7.1.4 Airdrop" on the previous page or "3.1. Airdrop Along Route" on page 40.

Drop slots are optional. If you use them, you should specify the drop slots in both the tasks for the entities on the aircraft and the aircraft itself. If the aircraft's airdrop task does not use drop slots, then the entities will parachute one at a time (regardless of any specified drop slots). If the aircraft's airdrop task has fewer drop slots defined than the drop slot specified for an entity, then that entity will not perform the airdrop. Units cannot use drop slots.

 While an entity is parachuting, if you double-click it to try to move it, it can snap to the terrain. If the entity was on a palette, its palette continues to descend.

► **To assign an airdrop from aircraft task:**

1. Select the entity embarked on the aircraft.
2. Choose **Task > Embarkation > Airdrop from Aircraft**. The Airdrop from Aircraft parameters display.
3. Indicate whether the entity should deploy the parachute after a specific amount of time or at a particular altitude.
  - To specify that the entity should deploy the parachute seconds after exiting the aircraft, select the Deploy At Elapsed Time check box and then enter the number of seconds.
  - If you choose to deploy at altitude, you can enter the altitude in meters. You can use the check box to indicate that the altitude be above ground level (default is sea level).

Be sure that the entity deploys the parachute at a point where it will open and slow their descent to the terrain.

4. If the entity is a vehicle or another simulation object with the Air Drop Capable system, you can specify the number of parachutes that the entity should use from the Cargo Parachute Type list. This setting does not affect the parachute for Paratrooper entities.
5. Optionally, specify a point where the entity should try to land.
6. Optionally, specify a drop slot. This is the order in which the entity will exit the aircraft if the Airdrop or Airdrop Along Route task includes drop slots. Multiple entities can use the same slot to exit the aircraft simultaneously.
7. Click OK.

### 7.1.6 Rope Sling Load Pick Up

The Rope Sling Load Pick Up task causes a helicopter to move to a specified location and then move to a designated cargo to load on the rope. Helicopters require the Fast Roping system.

► **To assign a Rope Sling Load Pick Up task:**

1. Select the helicopter.
2. Choose **Task > Embarkation > Rope Sling Load Pick Up**. The Rope Sling Load Pick Up parameters display.
3. Optionally, filter the Pickup Point list to view either Waypoints or Plan Variables.
4. Specify the location that the helicopter should move to.
5. Specify the cargo for the helicopter to pick up.
6. Click OK.

### 7.1.7 Rope Sling Load Drop Off

The Rope Sling Load Drop Off task causes a helicopter to move to a specified location and drop off cargo that it picked with the Rope Sling Load Pick Up task.

► **To assign a Rope Sling Load Drop Off task:**

1. Select the helicopter.
2. Choose **Task > Embarkation > Rope Sling Load Drop Off**. The Rope Sling Load Drop Off parameters display.
3. Specify the location where the helicopter should drop off the cargo.
4. Click OK.

### 7.1.8 Fast Rope Insertion

The Fast Rope Insertion task causes a helicopter to move to a specified waypoint and disembark a specified number of embarked entities using a fast rope animation. The disembarked entities slide down the rope, then move to a specified muster point and assume a kneeling posture.

This task is only available to helicopters that have the Fast Roping system, such as the MH-60 Black Hawk or the CH-47 Chinook.

► **To assign a Fast Rope Insertion task:**

1. Embark entities on a helicopter that supports the Fast Rope Insertion task.
2. Select the helicopter.
3. Choose **Task > Embarkation > Fast Rope Insertion**. The Fast Rope Insertion parameters display.
4. Optionally, filter the Insertion Point list to view either Waypoints or Plan Variables.
5. Specify the location at which the insertion should take place.
6. Specify the altitude at which the helicopter should hover.
7. Specify the number of entities to be disembarked and inserted.
8. Optionally, filter the Muster Point list to view either Waypoints or Plan Variables.
9. Specify the destination where the entities should move to after disembarking.
10. Click OK.

### 7.1.9 Rope Suspension Extraction

Helicopters with the Fast Roping system can perform a rope extraction. The Rope Extraction task specifies the waypoint where the helicopter will lower the rope and which entities to extract. Those entities must be within 50 meters of the rope and also have the Extract Using Rope task or they will not embark on the rope. For details, see "7.1.10 Extract Using Rope" on the facing page.

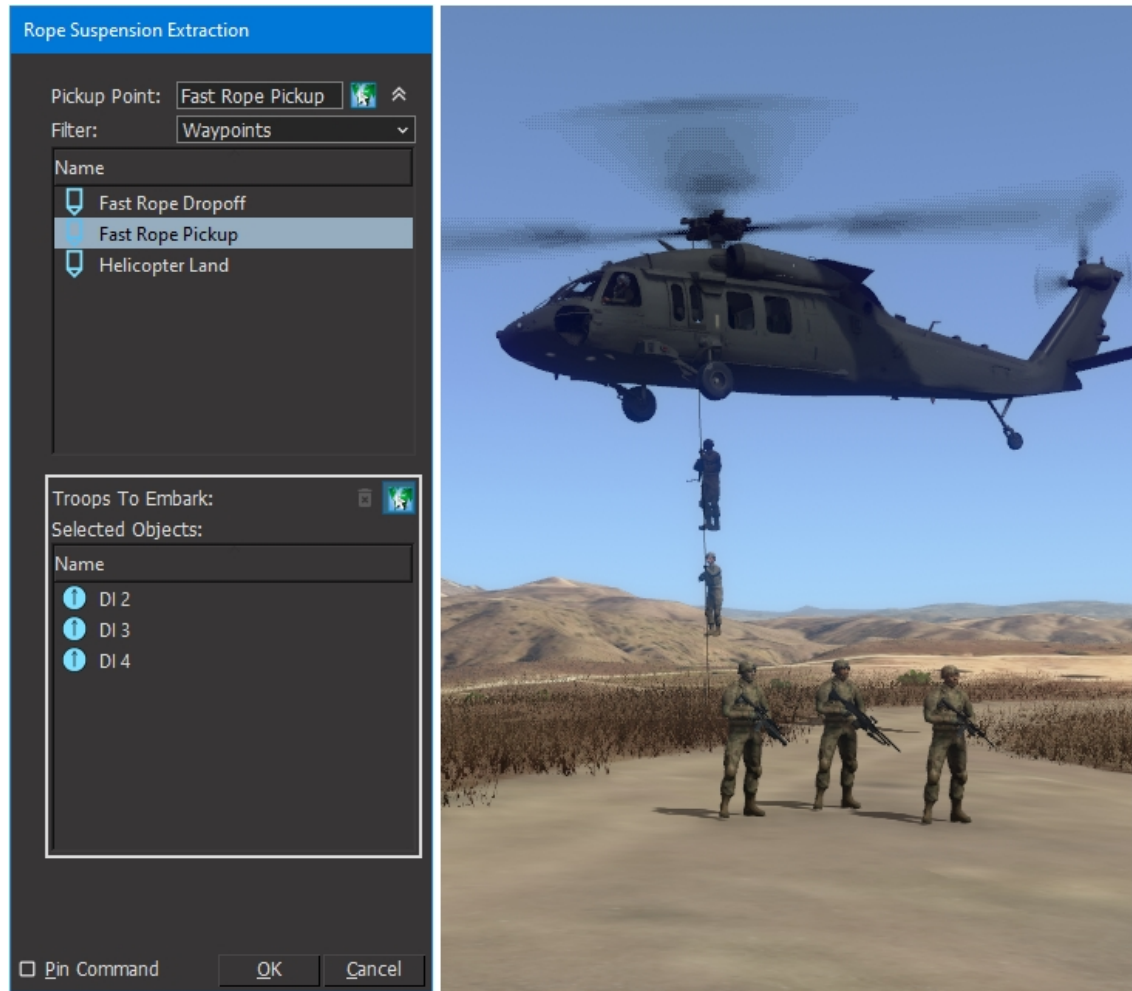
This task is only available to helicopters that have the Fast Roping system.

► **To have a helicopter extract entities with a rope:**

1. Select a helicopter that supports the Rope Extraction task.
2. Choose **Task > Embarkation > Rope Suspension Extraction**. The Rope Suspension Extraction parameters display (Figure 30).
3. Optionally, filter the Pickup Point list to view either Waypoints or Plan Variables.
4. In the Pickup Point list, specify the location from which the entities will be extracted.
5. Select the entities to extract in either the Objects List or in the window. They are added to the Troops To Embark list.
6. Click OK.

When the vehicle arrives at the waypoint and deploys the rope, entities on its Troops to Embark list that have the Extract Using Rope task and are within 50 meters of the rope will embark on the rope one at a time. After an entity attaches itself to the rope, the vehicle ascends slightly to allow the next entity to attach to the rope.

Figure 30. Rope Suspension Extraction



### 7.1.10 Extract Using Rope

Entities can be extracted via a rope deployed by a helicopter using the Rope Suspension Extraction task; see "7.1.9 Rope Suspension Extraction" on the previous page. In addition to the Extract Using Rope task, the entities must be included on the Troops to Embark list in the Rope Suspension Extraction task, and they must be within 50 meters of the point on the terrain where the rope was deployed.

Entities that have been extracted using the rope can only disembark from the rope via the Rope Suspension Insertion task. See "7.1.11 Rope Suspension Insertion" on the next page.

► **To specify that an entity will be extracted using a rope:**

1. Select the entity.
2. Choose **Task > Embarkation > Extract Using Rope**.

When the helicopter arrives and deploys the rope, surviving entities embark on the rope one at a time. After an entity attaches itself to the rope, the vehicle ascends slightly to allow the next entity to attach to the rope.

### 7.1.11 Rope Suspension Insertion

Helicopters with the Fast Roping system can perform an insertion via rope suspension using the Rope Suspension Insertion task. This applies to entities that are embarked on the helicopter's rope from an earlier extraction, and is the only way to disembark those entities. See "7.1.9 Rope Suspension Extraction" on page 138 to learn how to embark entities on the rope.

This task is different from the Fast Rope Insertion task, which is used for entities that begin while embarked inside a helicopter and then slide down a rope. For details, see "7.1.8 Fast Rope Insertion" on page 137.

This task is only available to helicopters that have the Fast Roping system.

► **To have a helicopter insert entities via rope suspension:**

1. Select a helicopter that supports the Rope Suspension Insertion task.
2. Choose **Task > Embarkation > Rope Suspension Insertion**. The Rope Suspension Insertion parameters display.
3. Optionally, filter the Dropoff Point list to see either Waypoints or Plan Variables.
4. In the Dropoff Point list, select the location where the entities will be inserted.
5. Click OK.

When the helicopter arrives at the waypoint, entities on the rope will disembark one at a time. After the entity at the bottom disembarks, the helicopter descends slightly so that the next entity on the rope can disembark.

### 7.1.12 Ladder Extraction

Helicopters with the Rope Ladder system can perform a ladder extraction. The Ladder Extraction task specifies the waypoint where it will drop the ladder and which entities to extract. Those entities must be within 50 meters of the ladder and also have the Extract Using Ladder task, or they will not climb the ladder and board the vehicle. For details, see "7.1.13 Extract Using Ladder" on the facing page.

This task is only available to helicopters that have the Rope Ladder system, such as the CH-47 Chinook.

 This task has the option to open the vehicle's rear cargo door. You can use the Simulation Object Editor to edit the Rope Ladder system and attach it to the rear of the helicopter instead of the front crew door (Figure 32). In that case, the ladder would deploy from the back and then the entities ascend the ladder and embark on the helicopter using the cargo door. See "Editing a System's Properties" for details.

► **To have a helicopter extract entities with a ladder:**

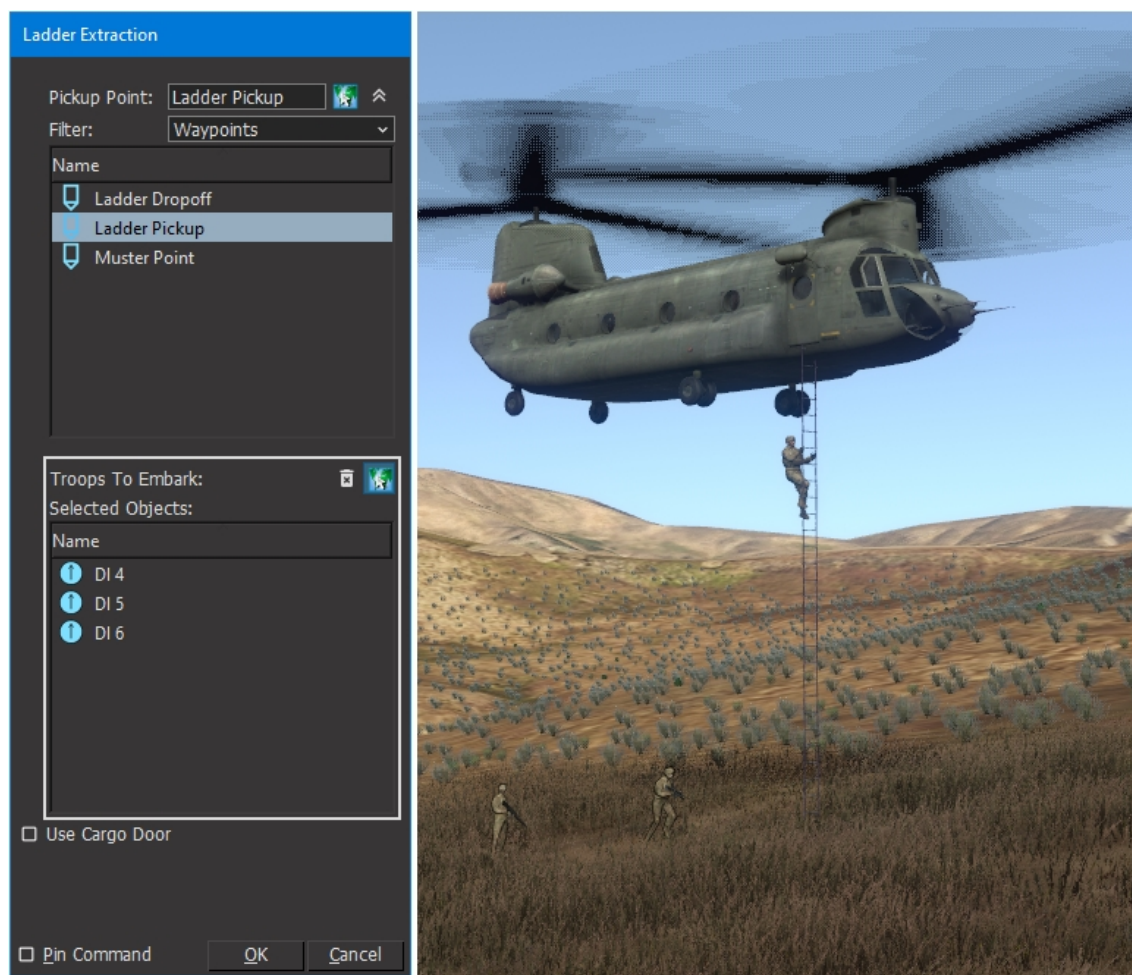
1. Select a helicopter that supports the Ladder Extraction task.
2. Choose **Task > Embarkation > Ladder Extraction**. The Ladder Extraction parameters display (Figure 31).
3. Optionally, filter the Pickup Point list to view either Waypoints or Plan Variables.



4. In the Pickup Point list, specify the location from which the entities will be extracted.
5. Optionally, filter the Troops to Embark list to see either All Entities or Plan Variables.
6. Select the entities to extract in either the Objects List or in the window. They are added to the Troops To Embark list.
7. Optionally, select Use Cargo Door to open the vehicle's rear cargo door.
8. Click OK.

When the helicopter arrives at the destination and deploys the ladder, entities on its Troops to Embark list that have the Extract Using Ladder task and are within 50 meters of the ladder will climb it and embark on the vehicle.

**Figure 31. Ladder Extraction**



### 7.1.13 Extract Using Ladder

Entities can be extracted via a ladder deployed by a helicopter using the Ladder Extraction task. See "7.1.12 Ladder Extraction" on the previous page. For an entity to successfully

execute the Extract Using Ladder task, the entity must be included on the Troops to Embark list in the Ladder Extraction task and must be within 50 meters of the extraction point.

► **To specify that an entity will be extracted using a ladder:**

1. Select the entity.
2. Choose **Task > Embarkation > Extract Using Ladder**.

When the helicopter arrives at the destination and deploys the ladder, surviving entities climb it and embark on the vehicle.

### 7.1.14 Ladder Insertion

Helicopters with the Rope Ladder system can perform a ladder insertion. The Ladder Insertion task specifies the waypoint where it will drop the ladder. Entities embarked on the helicopter with the Insert Using Ladder task will descend the ladder and move into formation. For details, see "7.1.15 Insert Using Ladder" on the facing page. There is no restriction on how the entities were embarked on the helicopter.

This task is only available to helicopters that have the Rope Ladder system.

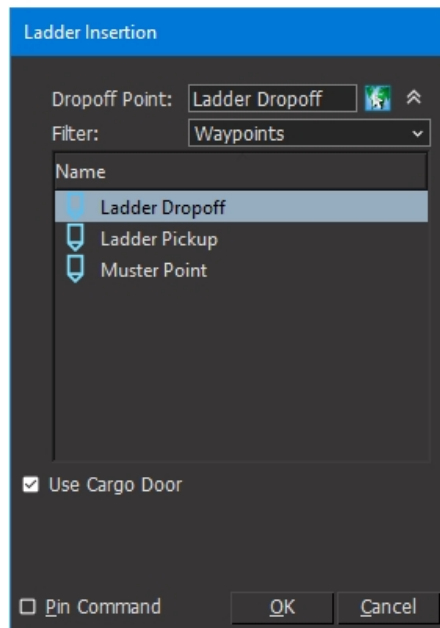
 This task has the option to open the vehicle's rear cargo door. You can use the Simulation Object Editor to edit the Rope Ladder system and attach it to the rear of the helicopter instead of the front crew door. In that case, the ladder deploys from the back and the entities descend the ladder from there (Figure 32). See "Editing a System's Properties" for details.

► **To have a helicopter insert entities with a ladder:**

1. Select a helicopter that supports the Ladder Insertion task.
2. Choose **Task > Embarkation > Ladder Insertion**. The Ladder Insertion parameters display (Figure 32).
3. Optionally, filter the Dropoff Point list to see either Waypoints or Plan Variables.
4. In the Dropoff Point list, specify the location where the entities will be inserted.
5. Optionally, select Use Cargo Door to open the vehicle's cargo door.
6. Click OK.

When the helicopter arrives at the destination and deploys the ladder, embarked entities that have the Insert Using Ladder task will climb down it and then move into formation.

**Figure 32. Ladder Insertion with Use Cargo Door and with Rope Ladder system positioned at rear of vehicle**



### 7.1.15 Insert Using Ladder

Entities can be inserted via a ladder deployed by a helicopter using the Ladder Insertion task. See "7.1.14 Ladder Insertion" on the previous page. The entities must be embarked on the helicopter and have the Insert Using Ladder task assigned to be included in the insertion. There is no restriction on how the entities were embarked on the helicopter.

► **To specify that an entity will be inserted using a ladder:**

1. Select an entity that is (or will be) embarked on the helicopter that has the Ladder Insertion task.
2. Choose **Task > Embarkation > Insert Using Ladder**.

When the helicopter arrives at the destination and deploys the ladder, each entity with this task will descend the ladder and then move into formation around its base.

### 7.1.16 Dismount

You can task squads or fire teams that are embarked on a rotary-wing aircraft or ground vehicle with the Dismount task. The entities will exit and assemble in formation around the aircraft or vehicle.

► **To task a squad or fire team to exit an aircraft or vehicle and assume formation:**

1. Select a squad or fire team unit.
2. Choose **Task > Embarkation > Dismount**.

## 7.2. Internal Transfer of Fuel

Entities with the Air Refueling Supplier system, typically tanker aircraft, can transfer their off-load fuel to their onboard fuel supply using the Internal Transfer of Fuel task. You can specify whether they should fill up to their full capacity or just a specific amount of fuel, as well as the rate at which the fuel supply should be replenished.

► **To task a tanker aircraft to transfer fuel to itself:**

1. Select the tanker aircraft.
2. Choose **Task > Logistics > Internal Transfer of Fuel**. The Internal Transfer of Fuel parameters display.
3. Specify the parameters for the transfer.
4. Click OK.

## 7.3. Transfer Supplies

Entities with an air refueling receiver systems can use the Transfer Supplies task to transfer fuel supplies from another entity to itself. The task uses the DIS and HLA RPR FOM Logistics Resupply protocol. The specified supplier must have that fuel resource for the task to be successful.

► **To assign the Transfer Supplies task:**

1. Select the entity that will receive the fuel supply.
2. Choose **Task > Logistics > Transfer Supplies**. The Transfer Supplies parameters display.
3. From the Supply Type list, select the fuel type.
4. If you want the entity to fill its fuel resource, select the Fill Up to Capacity check box.

If you do not want the entity to fill its fuel resource to capacity, clear the check box and then enter the amount it should take in the Supply Amount box.

5. Enter the rate at which the supply should transfer.
6. From the Supplier list, select the entity that will supply the fuel.
7. Click OK.

## 7.4. IFF Tasks

You can task entities configured with an IFF interrogator system can to interrogate other entities' IFF status. VR-Forces uses open standards such as RPR FOM and DIS to query the IFF transponder status of the other entities.

VR-Forces includes the ability to perform a Lethal Mode 5 interrogation on entities to check for Mode 5 enabled transmitters. It also displays a **V** next to any entities that are the target of a Lethal Mode 5 interrogation when they have not enabled Mode 5.

By default, there is no simulation effect in the scenario. The IFF information sent is the VR-Link interrogation response, which is simply the IFF modes that are enabled on the target. That information is only published to the network, making it available for higher fidelity IFF environments.

- Set entities' IFF codes using the IFF set data request as described in "IFF".
- Clutter aircraft include IFF information. See "Aircraft Clutter Area".

### 7.4.1 IFF Interrogate

You can task an aircraft entity to interrogate another entity for IFF transponders.

Any enabled modes display in the interrogating entity's Object Console as warnings (Figure 33). When you enable lethal interrogation for Mode 5, if the target of the interrogation has Mode 5 available but does not have it enabled, the interrogated entity briefly displays a **V** in the Plan View and the Object Console indicates it was a lethal interrogation (Figure 34).

**Figure 33. IFF interrogation results**

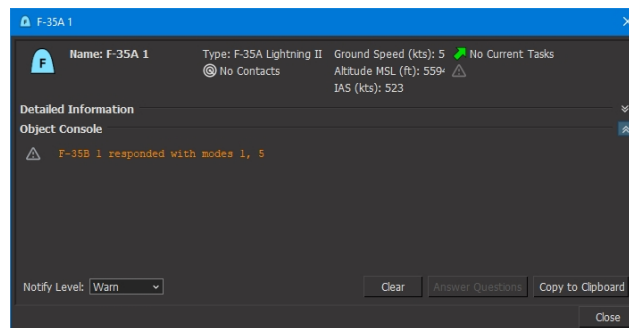
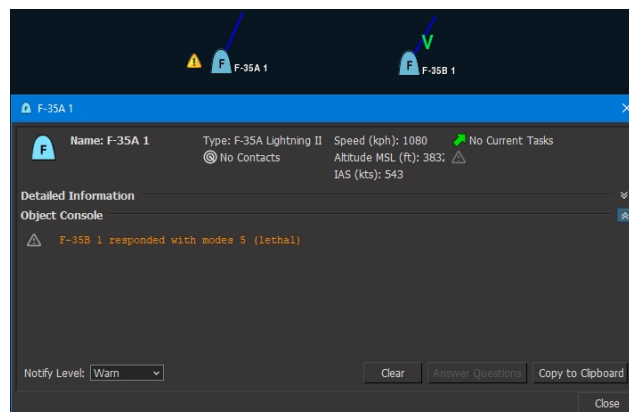


**Figure 34. IFF lethal interrogation results**



- VR-Forces implements the NETN FOM, based on RPR FOM 2.0, with IFF extensions from RPR FOM 3.0. If you use HLA Evolved, use the FOM provided by MAK.

There is no real simulation effect in the scenario. The resulting information is only published to the network, making it available for higher fidelity IFF environments.

► **To assign an IFF Interrogate task:**

1. Select the entity to task.
2. Choose **Task > IFF > IFF Interrogate**. The IFF Interrogate parameters display.
3. Use Name box to specify the entity to interrogate.
4. Select the modes that you want to check for.
5. Click OK.

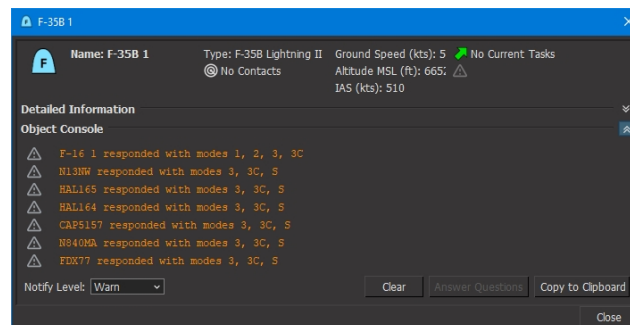
### 7.4.2 IFF Sweep Interrogate

You can task an aircraft entity to interrogate entities within range for their IFF transponders. It returns the modes that are present and on. Detected modes display in the interrogating entity's Object Console as warnings.

In the task, you specify the area that the entity should sweep. This is a combination of the range and the sweep. There are three methods for defining the sweep:

- 360 Degree Sweep. Interrogates all aircraft entities within the specified range.
- Sweep Range. Specify an offset from the orientation.
- From Boresight. Specify an angle from the entity's boresight.

**Figure 35. IFF sweep interrogation results**



❗ VR-Forces implements the NETN FOM, based on RPR FOM 2.0, with IFF extensions from RPR FOM 3.0. If you use HLA Evolved, use the FOM provided by MAK.

There is no real simulation effect in the scenario. The resulting information is only published to the network, making it available for higher fidelity IFF environments.

► **To assign an IFF Sweep Interrogate task:**

1. Select the entity to task.
2. Choose **Task > IFF > IFF Sweep Interrogate**. The IFF Sweep Interrogate parameters display.
3. Enter the parameters for the sweep.
4. Select the modes that you want to check for.
5. Click OK.

## 7.5. Radio Tasks

This section describes the tasks that allow you to use the VR-Forces radio network to send messages, tasks, and set data requests.

### 7.5.1 Send Radio Set

The Send Radio Set task lets you send a set data request to a simulation object using a VR-Forces radio message. The receiving entity executes the set data request as if you had issued it directly to the entity from the Set menu or a set data request in a plan.

**i** Since VR-Forces has no idea what type of entity you might want to give a set data request to, when you add this task the full list of possible sets is available. There is no context sensitivity like there is when you select a simulation object and view the Set menu. It is up to you to be sure that the set you are assigning to a simulation object is supported by that object type.

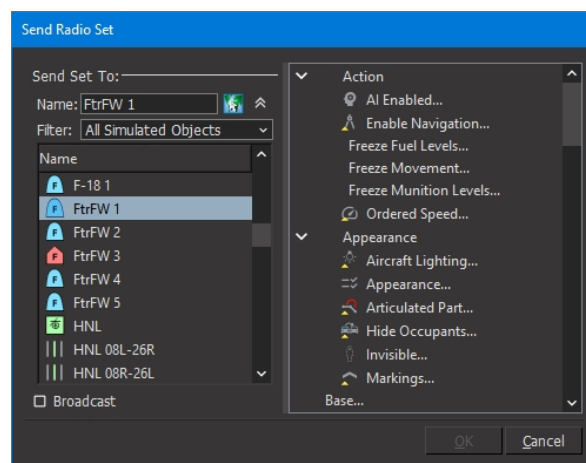
► **To assign a Send Radio Set task:**

1. Select the simulation object that will send the message.
2. Choose **Task > Radio > Send Radio Set**. The Send Radio Set parameters display.
3. To send the message to all simulation objects configured to receive messages from this simulation object's radio, select the Broadcast check box. (For details about configuring the radio model, see "Configuring Radios".)

To send the message to a specific simulation object, in the Send Set To group box, in the Name text box, type the name of the simulation object that will receive the set data request message, or select the simulation object in the list or on the terrain.

There is a list of set data requests to choose from (Figure 36).

**Figure 36. Send Radio Set dialog box**



4. Select the set data request you want to send.
5. Click OK. If the set data request requires input, an appropriate dialog box opens. If no input is required, the task setup is finished.



6. If a dialog box opens, complete the dialog box as described in chapters 34 and 35.
7. Click OK.

### 7.5.2 Send Radio Task

The Send Radio Task task lets you send a task to a simulation object using a VR-Forces radio message. The receiving simulation object executes the task as an independent task.

**i** Since VR-Forces has no idea what type of simulation object you might want to give a task to, when you add this task the full list of possible tasks is available. There is no context sensitivity like there is when you select a simulation object and view the Task menu. It is up to you to be sure that the task you are assigning to a simulation object is supported by that object type.

► **To assign a Send Radio Task:**

1. Select the simulation object that will send the message.
2. Choose **Task > Radio > Send Radio Task**. The Send Radio Task parameters display.
3. To send the message to all simulation objects configured to receive messages from this simulation object's radio, select the Broadcast check box. (For details about configuring the radio model, see "Configuring Radios".)

To send the message to a specific simulation object, in the Send Task To group box, in the Name text box, type the name of the simulation object that will receive the task message, or select the simulation object in the list or on the terrain.

A list of tasks displays (similar to Figure 36 on the previous page).

4. Select the task you want to send.
5. Click OK. If the task requires input, an appropriate dialog box opens. If no input is required, the task setup is finished.
6. If a dialog box opens, complete the dialog box as described in the various task procedure chapters.
7. Click OK.

### 7.5.3 Send Text Message

The Send Text Message task lets you send a text message to a simulation object using a VR-Forces simulated radio message. The receiving simulation object must be on the same radio network as the sending simulation object. Usually this means the same unit or same force. The receiving simulation object can test for receipt of the message in a conditional statement in its plan. (For details, see "Conditional Tests".)

**i** If you want to send a text message that can be received by simulation objects in a different radio network from the sender, use the Send Text Message command. (For details, see "Sending Text Messages from the Commands Menu".)



► **To assign a Send Text Message task:**

1. Select the simulation object that will send the message.
2. Choose **Task > Radio > Send Text Message**. The Send Text parameters display.
3. To send the message to all simulation objects configured to receive messages from this simulation object's radio, select the Broadcast check box. (For details about configuring the radio model, see "Configuring Radios".)  
To send the message to a specific simulation object, in the Name text box, type the name of the simulation object that will receive the message, or select the simulation object in the list or on the terrain.
4. In the Message Text box, type the message.
5. Click OK.

## 7.6. Wait Tasks

You can order a simulation object to wait indefinitely, for a specified period of time, or until a specified elapsed simulation time. You can also order a human to wait in a queue. For details, see "5.33. Wait in Queue" on page 115.

 The Wait task is not equivalent to an immediate stop task. If a simulation object is moving and you give it a Wait task, it might not stop immediately, particularly if a controller such as the collision avoidance controller is controlling its movement.

### 7.6.1 Wait

► **To assign a Wait task:**

1. Select the simulation object.
2. Choose **Task > Wait > Wait**.

 If you put an indefinite wait in a plan, the simulation object does not progress through its plan until it receives another command.

### 7.6.2 Wait Duration

► **To assign a Wait Duration task:**

1. Select the simulation object.
2. Choose **Task > Wait > Wait Duration**. The Wait Duration parameters display.
3. Specify the duration of the wait in hours, minutes, and seconds.
4. Click OK.

### 7.6.3 Wait Elapsed

► **To assign a Wait Elapsed task:**

1. Select the simulation object.
2. Choose **Task > Wait > Wait Elapsed**. The Wait Elapsed parameters display.
3. Specify the elapsed time from the start of the simulation, in hours, minutes, and seconds.
4. Click OK.

For information about simulation time and simulated exercise time, see "Simulation Time".

## 7.7. Tasks for Moving Articulated Parts

Some entity models have articulated parts, such as doors and periscopes, that you can move.

### 7.7.1 Open Car Door

Vehicles that have a Car Door system and whose model includes articulated parts for the car doors can open and close their doors. Open Car Door opens a specific door on a vehicle.

► **To open a car door:**

1. Select a vehicle that has a Car Door system.
2. Choose **Task > Car Door > Open Car Door**. The Open Car Door dialog box opens.
3. In the Door list, select the door that you want to open.
4. Optionally, in the Swing Rate box, specify the speed at which you want the door to open.
5. Click OK.

### 7.7.2 Close Car Door

Vehicles that have a Car Door system and whose model includes articulated parts for the car doors can open and close their doors. Close Car Door closes a specific door on a vehicle.

► **To close a car door:**

1. Select a vehicle that has an open door.
2. Choose **Task > Car Door > Close Car Door**. The Close Car Door dialog box opens.
3. In the Door list, select the door that you want to close.
4. Optionally, in the Swing Rate box, specify the speed at which you want the door to close.
5. Click OK.

### 7.7.3 Open Cargo Door

The Open Cargo Door task opens a cargo door on an entity that has a cargo door articulated part.

**Figure 37. CH-53E Super Stallion with open cargo door**



► **To assign an Open Cargo Door task:**

1. Select the entity.
2. Choose **Task > Cargo Door > Open Cargo Door**.

### 7.7.4 Close Cargo Door

The Close Cargo Door task closes a cargo door on an entity that has a cargo door articulated part.

► **To assign a Close Cargo Door task:**

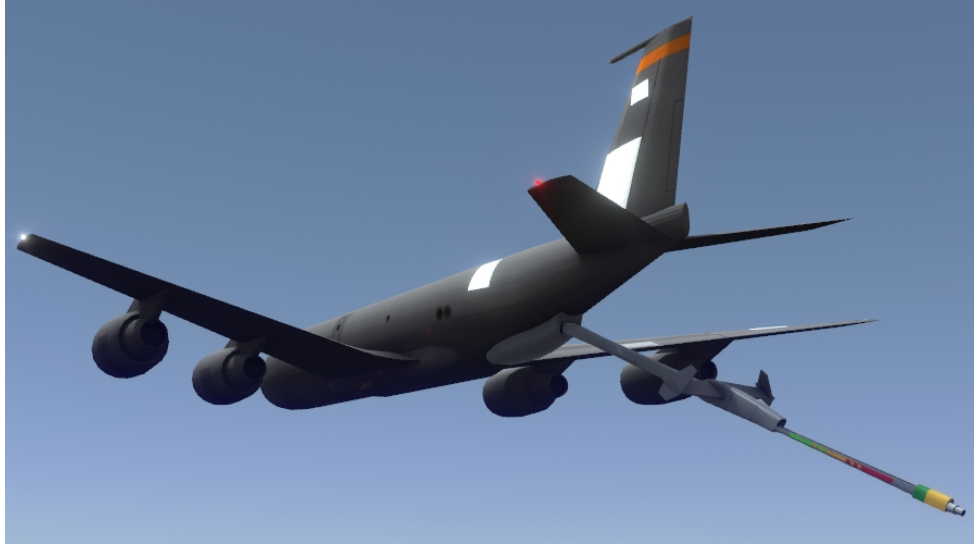
1. Select the entity.
2. Choose **Task > Cargo Door > Close Cargo Door**.

### 7.7.5 Deploy Refueling Boom

The Deploy Refueling Boom task causes an aircraft that has a refueling boom articulated part to extend it (Figure 38). Use Stow Refueling Boom (see "7.7.6 Stow Refueling Boom" on the next page) to retract the boom.



- This task only deploys the boom. See "3.7. Establish Refueling Station" on page 43 for the refueling task.
- To use the task, the entity must have the appropriate refueling system. To visualize the task, the model must be configured with the articulated part.

**Figure 38. KC-135 Stratotanker with refueling boom deployed**

► **To assign a Deploy Refueling Boom task:**

1. Select the entity.
2. Choose **Task > Refueling Parts > Deploy Refueling Boom**. The boom extends.

### 7.7.6 Stow Refueling Boom

The Stow Refueling Boom task causes an aircraft that has a refueling boom articulated part to stow it. Use Deploy Refueling Boom (see "7.7.5 Deploy Refueling Boom" on the previous page) to deploy a refueling boom.

► **To assign a Stow Refueling Boom task:**

1. Select the entity.
2. Choose **Task > Refueling Parts > Stow Refueling Boom**. The boom contracts.

### 7.7.7 Deploy Port or Starboard Refueling Drogue

The Deploy Port Refueling Drogue and Deploy Starboard Refueling Drogue tasks cause an aircraft that has a port-or starboard-side refueling drogue articulated part to extend the specified drogue (Figure 39). Use Stow Port Refueling Drogue or Stow Starboard Refueling Drogue (see "7.7.8 Stow Port or Starboard Refueling Drogue" on the facing page) to retract the drogue.



- These tasks only extend a drogue. See "3.7. Establish Refueling Station" on page 43 for the refueling task.
- To use the task, the entity must have the appropriate refueling system. To visualize the task, the model must be configured with those articulated parts.

**Figure 39. KC-130 Hercules Tanker with port and starboard drogues deployed**

► **To assign a deploy refueling drogue task:**

1. Select the entity.
2. Choose **Task > Refueling Parts > Deploy Port Refueling Drogue** or **Deploy Starboard Refueling Drogue**. The drogue extends.

### 7.7.8 Stow Port or Starboard Refueling Drogue

The Stow Port Refueling Drogue and Stow Starboard Refueling Drogue tasks cause an aircraft that has a port-or starboard-side refueling drogue articulated part to stow it. Use Deploy Port Refueling Drogue and Deploy Starboard Refueling Drogue (see "7.7.7 Deploy Port or Starboard Refueling Drogue" on the previous page) to deploy a refueling drogue.

► **To assign a stow refueling drogue task:**

1. Select the entity.
2. Choose **Task > Refueling Parts > Stow Port Refueling Drogue** or **Stow Starboard Refueling Drogue**. The drogue retracts.

### 7.7.9 Extend Towed Array Decoy

The Extend Towed Array Decoy task causes an aircraft that is configured with a towed array decoy articulated part to deploy it. Use Retract Towed Array Decoy (see "7.7.10 Retract Towed Array Decoy" on the next page) to retract the decoy.

**i** To use the task, the entity must have the Towed Array Decoy system. To visualize the task, the model must be configured with the articulated part.

► **To assign an extend towed array decoy task:**

1. Select the entity.
2. Choose **Task > Other > Extend Towed Array Decoy**. The array extends.

### 7.7.10 Retract Towed Array Decoy

The Retract Towed Array Decoy task causes an aircraft that is configured with a towed array decoy articulated part to retract it. Use Extend Towed Array Decoy (see "7.7.9 Extend Towed Array Decoy" on the previous page) to deploy the decoy.

► **To assign a retract towed array decoy task:**

1. Select the entity.
2. Choose **Task > Other > Retract Towed Array Decoy**. The array extends.

### 7.7.11 Raise Periscope

The Raise Periscope task raises the periscope of a subsurface entity. The periscope is an articulated part that is visible in the Stealth observer mode.

► **To assign a Raise Periscope task:**

1. Select the entity.
2. Choose **Task > Other > Raise Periscope**.

### 7.7.12 Lower Periscope

The Lower Periscope task lowers the periscope of a subsurface entity. The periscope is an articulated part that is visible in the Stealth view.

► **To assign a Lower Periscope task:**

1. Select the entity.
2. Choose **Task > Other > Lower Periscope**.

## 7.8. Embedded Entity Tasks

You can deploy and recover embedded entities. You can also task an embedded entity to conduct a sector search. For details about embedded entities, see "Embedded Entities".

### 7.8.1 Deploying Embedded Entities

To add an embedded entity to a simulation, you deploy it from the parent entity. You can deploy an embedded entity explicitly or implicitly (by assigning a task to it). An embedded entity's label displays its name and the name of its parent.

 There is a delay of a few seconds from the time a deployment is ordered and the time it is executed.

► **To assign a Deploy task:**

1. Select the parent entity.
2. Choose **Task > Embedded Entity > Deploy**. The Deploy parameters display.
3. Select the entity you want to deploy.

4. Click OK. The embedded entity deploys next to the parent entity in the relative position in its configuration.

### 7.8.2 Recovering Deployed Entities

If you want an embedded entity that has been deployed to return to its parent, you can have the parent recover it. If you do not recover it, the deployed entity continues to function in the scenario just like any other entity.

► **To assign a Recover task:**

1. Select the parent entity.
2. Choose **Task > Embedded Entity > Recover**. The Recover Embedded Entity parameters display.
3. Select the entity you want to recover.
4. Click OK. The parent sends a message to the deployed entity to return. It returns and is embedded again.

### 7.8.3 Sector Search Operation

The Sector Search Operation task causes an embedded entity to fly a sector search and rescue pattern. In addition to the basic search and rescue parameters, you have these options:

- If there are no undeployed embedded entities available, divert an already deployed entity to do the search.
- Force the parent entity to wait until the search is complete and the embedded entity is recovered until it continues with its plan or allow the parent entity to continue with its plan while the search is underway.

► **To assign a Sector Search Operation task:**

1. Choose an entity that has embedded entities.
2. Choose **Task > Sector Search Operation**. The Sector Search Operation parameters display.
3. Select the Allow Diversion check box if you want a deployed entity to be diverted to this task if necessary.
4. Select the Recover When Finished check box if you want the parent entity to wait until it recovers the deployed entity before it continues with its plan.
5. Specify the starting point of the search.
6. In the Leg box, specify the length of each leg of the search.
7. In the Altitude box, specify the altitude the search aircraft should fly at.
8. In the Air Speed box, specify the speed of the search aircraft.
9. Click OK.

### 7.8.4 Deploying an Entity by Assigning a Task

You can assign a task to a deployable entity, that is, an embedded entity that has not been deployed yet. This implicitly deploys the entity so that it can execute the task. You can assign a task to an embedded entity in the parent's plan or you can directly assign a task to the embedded entity. The Sector Search example scripted task is an example of how to task a parent entity to deploy an embedded entity.

 There is a delay of a few seconds from the time a deployment is ordered and the time it is executed.

► **To assign a task to a deployable entity in a parent's plan:**

1. Select the parent entity.
2. Choose **Objects > Plan**. The Plan window opens.
3. Choose **Embedded Entities > entity > Task (or Set) > task (or set)**, where *entity* is one of the entities that the parent can deploy and *task (or set)* is one of the tasks or sets on the Task or Set menu. The parameters for the selected task or set display.

 If an entity does not have embedded entities, the Embedded Entities menu is not available.

4. Fill out the dialog box like you would for any task or set.
5. Click OK. The task is added to the plan. When you run the scenario, the embedded entity deploys and is assigned the task or set.

When you deploy an entity from the Objects menu, you can continue to set its state or assign tasks using this method or you can select the entity itself and issue tasks and sets from the Task and Set menus. The logic of assigning tasks from the Objects menu's Embedded Entities submenu rather than directly, is that the parent entity issues the tasks to the deployed entity.

► **To deploy an embedded entity by assigning it a task:**

1. Select the parent entity.
2. Choose **Objects > Embedded Entities > entity > Task (or Set) > task (or set)**, where *entity* is one of the entities that the parent can deploy and *task (or set)* is one of the tasks or sets on the Task or Set menu. An appropriate dialog box for the task or set opens.
3. Fill out the dialog box like you would for any task or set.
4. The entity is deployed. If you assigned it a task, it begins to execute its task. If you set its state, it is deployed with the updated state setting.

## 7.9. Deploy Sonobuoy

The Deploy Sonobuoy task lets an aircraft deploy a sonobuoy for taking sonar readings. A sonobuoy is deployed as an entity. This task is available to entities that have a Sonobuoy Deployer system configured.



**i** RPR FOM 1.0 does not support active sonar. You will not be able to create sonobuoys if they have an active sonar system.

► **To assign a Deploy Sonobuoy task:**

1. Select the entity.
2. Choose **Task > Deploy Sonobuoy**. The Deploy Sonobuoy parameters display.
3. Select the type of sonar to use – Active or Passive.
4. Specify the depth at which to deploy the sonobuoy.
5. Click OK.

## 7.10. Deploy Sonobuoys Along Route

The Deploy Sonobuoys Along Route task lets an aircraft deploy sonobuoys as it moves along a route. A sonobuoy is deployed as an entity. This task is available to entities that have a Sonobuoy Deployer system configured.

**i** RPR FOM 1.0 does not support active sonar. You will not be able to create sonobuoys if they have an active sonar system.

► **To assign a Deploy Sonobuoys Along Route task:**

1. Select the entity.
2. Choose **Task > Deploy Sonobuoys Along Route**. The Deploy Sonobuoys Along Route parameters display.
3. Select the route to use.
4. Specify the distance between each sonobuoy.
5. Select the type of sonar to use – Active or Passive.
6. Specify the depth at which to deploy the sonobuoys.
7. Click OK.

## 7.11. Sonar Dip

Rotary-wing entities that have sonar can dip the sonar into water at a specified depth. Dipped sonar is not visualized in the Stealth view. The sonar is just simulated at the specified depth. Entities cannot dip sonar if they are moving faster than a set speed.

► **To assign a Sonar Dip task:**

1. Select the entity.
2. Choose **Task > Sonar Dip**. The Sonar Dip parameters display.
3. Specify the location at which to dip the sonar.
4. Specify how long the sonar should stay dipped.
5. Select the type of sonar to dip – Active or Passive.

 RPR FOM 1.0 does not support active sonar.

6. Specify the depth for the sonar.
7. Click OK.

## 7.12. Release Next Entity From Queue

The Release Next Entity From Queue task is sent to a queue tactical graphic and releases the entity at the head of the queue to move to the next task in its plan.

You can assign this task directly, as described in this procedure, or from the **Commands > Send Task** command in a plan. If you are trying to create a scenario that involves crowd queueing behaviors, then using a plan to control the queue is the more likely approach.

 When an entity is released from the queue, the Wait in Queue task ends. If the entity does not have another task in its plan, it remains in its location. The next entity in the line moves to the head of the queue. This might result in some visual anomalies.

► **To assign a Release Next Entity From Queue task:**

1. Select the queue tactical graphic.
2. Choose **Task > Release Next Entity From Queue**.

## 7.13. User Task

User tasks are tasks that have been added to VR-Forces using the VR-Forces toolkit. To assign a user task, you must know the name of the task and the required values for up to four parameters.

 The User Task command is not available on the Task menu unless a developer adds a controller that accepts a user task.

► **To assign a user task:**

1. Select the simulation object that you want to execute the task.
2. Choose **Task > Other > User Task**. The User Task parameters display.
3. Type the name of the task.
4. Type any required parameters.
5. Click OK.

## 7.14. Reactive Tasks

Reactive tasks are not listed individually because most of them run automatically; typically, no user action is required to execute them.

You can disable or enable reactive tasks as needed; see "2.5. Reactive Tasks" on page 19 for instructions.

Table 5 lists some of the reactive tasks provided with VR-Forces.

**Table 5. Reactive tasks included with VR-Forces (partial list)**

| Reactive Task                   | Description                                                                                                                                                                                                                                                     |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Automatic Headlights            | Turn headlights and taillights on and off based on local lighting conditions. This is disabled by default. You must enable this reactive task to have the lights come on automatically.                                                                         |
| Automatic Jamming               | Causes an entity to automatically respond to being tracked by radar.                                                                                                                                                                                            |
| Automatic Look                  | Causes humans to randomly look at other humans.                                                                                                                                                                                                                 |
| Auto-Enable Show Terrain        | Turns on Show Terrain task for object upon creation.                                                                                                                                                                                                            |
| Autonomous Fire reactive tasks  | Looks for valid targets for any of the weapon systems that are enabled to fire autonomously. Prioritizes those targets and sends a fire task for the top priority one.                                                                                          |
| Avoid Other Ships               | Considers ship type: Aircraft carrier is higher than fishing boat (assumes nets are out), which is higher than sailboat, which is higher than other. Will avoid surfaced submarines.                                                                            |
| Bingo Fuel Monitor              | Monitors fuel on embedded entities and requests a recovery when bingo fuel level is reached.                                                                                                                                                                    |
| Evade Missile                   | Causes fixed-wing entities to try to evade missiles.                                                                                                                                                                                                            |
| Flee From threat reactive tasks | Entity will run from explosions, the source of gun fire, entities killed nearby, and objects that match the configured entity types.<br>Flee from Fire Events is disabled by default, but you can enable it for entities in the Simulation Object Editor (SOE). |
| IADS Launcher Control           | Automates the launcher of an Integrated Air Defense system. Communicates with a control station running the IADS Command Station script.                                                                                                                        |
| IFF Code Changer                | Automatically changes the IFF code every half hour based on the time of day.                                                                                                                                                                                    |
| Make Way for Emergency          | If an emergency vehicle that has its lights on approaches a ground vehicle that is driving on a road, the vehicle pulls over to the side of the road.                                                                                                           |
| Suppressed Crouch               | Forces entity to crouch when suppressed. Returns to previous posture when no longer suppressed. Does not otherwise interrupt movement.                                                                                                                          |
| Suppressed Prone                | Forces the entity to stop and fall prone when suppressed to level 2 or 3. Resumes movement at old posture when no longer suppressed.                                                                                                                            |
| Visit Interest Points           | Causes civilians to visit nearby civilian visit points.                                                                                                                                                                                                         |

## 7.15. Background Processes

*EntityLevel.sms* has these background processes:

- Delete Created Pedestrians. Deletes pedestrians when a pedestrian area is deleted.
- Flare Fade. Gradually fades out illumination flares.
- Queue Manager. Manages queuing for the Wait in Queue task.



## 8. Tasks for Aggregate-Level Scenarios

This chapter describes tasks that apply to aggregate-level scenarios.

|                                                  |     |
|--------------------------------------------------|-----|
| 8.1. Introduction to Aggregate-Level Tasks ..... | 163 |
| 8.2. Engagement Tasks .....                      | 163 |
| 8.2.1 Air-to-Air Attack .....                    | 163 |
| 8.2.2 Air-to-Ground Air Base Attack .....        | 163 |
| 8.2.3 Air-to-Ground Attack .....                 | 164 |
| 8.2.4 Attack by Fire .....                       | 164 |
| 8.2.5 Attack with Anti-Air Missile .....         | 165 |
| 8.2.6 Attack with Anti-Ship Missile .....        | 165 |
| 8.2.7 Attack with Land-Attack Missile .....      | 165 |
| 8.2.8 Attack with Torpedo .....                  | 166 |
| 8.2.9 Automatic Air Defense .....                | 166 |
| 8.2.10 Biological Attack .....                   | 166 |
| 8.2.11 Bomb Location .....                       | 167 |
| 8.2.12 Chemical Attack .....                     | 167 |
| 8.2.13 Indirect Fire .....                       | 167 |
| 8.2.14 Nuclear Attack .....                      | 168 |
| 8.3. Combat Engineering Tasks .....              | 168 |
| 8.3.1 Breach Obstacles .....                     | 169 |
| 8.3.2 Construct Abatis .....                     | 170 |
| 8.3.3 Construct Anti-Tank Ditch .....            | 170 |
| 8.3.4 Construct Barbed Wire .....                | 170 |
| 8.3.5 Construct Barricade .....                  | 171 |
| 8.3.6 Construct Berm .....                       | 171 |
| 8.3.7 Construct Booby Trap .....                 | 172 |
| 8.3.8 Construct Bridge .....                     | 172 |
| 8.3.9 Construct Dry Gap .....                    | 173 |
| 8.3.10 Construct Fortified Area .....            | 173 |
| 8.3.11 Construct Fortified Line .....            | 174 |

|                                                         |     |
|---------------------------------------------------------|-----|
| 8.3.12 Construct Minefield .....                        | 174 |
| 8.3.13 Construct Strong Point .....                     | 175 |
| 8.3.14 Destroy Bridge .....                             | 175 |
| 8.3.15 Destroy Ditch .....                              | 176 |
| 8.3.16 Destroy Explosive .....                          | 176 |
| 8.3.17 Destroy Fortification .....                      | 176 |
| 8.3.18 Destroy Obstacle .....                           | 176 |
| 8.3.19 Improve Booby Trap .....                         | 177 |
| 8.3.20 Improve Breach .....                             | 177 |
| 8.3.21 Improve Bridge .....                             | 177 |
| 8.3.22 Improve Ditch .....                              | 178 |
| 8.3.23 Improve Fortification .....                      | 178 |
| 8.3.24 Improve Obstacle .....                           | 178 |
| 8.4. Air Base Tasks .....                               | 179 |
| 8.4.1 Launch Flight .....                               | 179 |
| 8.4.2 Launch QRA Aircraft .....                         | 180 |
| 8.4.3 Prepare QRA .....                                 | 180 |
| 8.5. Movement Tasks for Aggregate-Level Scenarios ..... | 181 |
| 8.5.1 Change MOPP Level .....                           | 181 |
| 8.5.2 Change Posture .....                              | 182 |
| 8.5.3 Escort Air Unit .....                             | 182 |
| 8.5.4 Halt Movement .....                               | 182 |
| 8.5.5 Land at Air Base .....                            | 183 |
| 8.5.6 Move Along Route .....                            | 183 |
| 8.5.7 Move Along Route Retrograde .....                 | 183 |
| 8.5.8 Move To .....                                     | 183 |
| 8.5.9 Move to Retrograde .....                          | 183 |
| 8.5.10 Patrol Route .....                               | 183 |
| 8.5.11 Patrol Between .....                             | 183 |
| 8.5.12 Turn To Heading .....                            | 183 |
| 8.5.13 Wait for Escort .....                            | 184 |
| 8.5.14 Wait Tasks .....                                 | 184 |
| 8.6. Embarkation Tasks .....                            | 184 |
| 8.6.1 Disembark .....                                   | 184 |
| 8.6.2 Disembark All .....                               | 184 |
| 8.6.3 Embark .....                                      | 184 |
| 8.7. Radio Tasks .....                                  | 185 |
| 8.7.1 Send Radio Set .....                              | 185 |
| 8.7.2 Send Radio Task .....                             | 185 |
| 8.7.3 Send Text Message .....                           | 185 |
| 8.8. Other Tasks .....                                  | 185 |

|                                    |     |
|------------------------------------|-----|
| 8.8.1 Activate Jammer .....        | 185 |
| 8.8.2 Deactivate Jammer .....      | 185 |
| 8.8.3 Drop Fuel Tanks .....        | 185 |
| 8.8.4 FASCAM .....                 | 185 |
| 8.8.5 Refuel from Tanker .....     | 186 |
| 8.8.6 Refueling Mission .....      | 186 |
| 8.8.7 Send NBC Report .....        | 187 |
| 8.8.8 Send Obstacle Report .....   | 187 |
| 8.8.9 User Task .....              | 187 |
| 8.9. Background Process List ..... | 188 |

## 8.1. Introduction to Aggregate-Level Tasks

This chapter describes tasks that simulation objects in aggregate-level scenarios can run. Some tasks are identical to tasks supported by simulation objects in entity-level scenarios.

## 8.2. Engagement Tasks

The following topics describe tasks in the engagement category that apply to aggregate-level scenarios.

### 8.2.1 Air-to-Air Attack

The Air-to-Air Attack task instructs an air unit to attack another air unit.

► **To assign an air-to-air attack:**

1. Select an air unit.
2. Choose **Task > Engagement > Air-to-Air Attack**, or click the Air-to-Air Attack button (). The Air-to-Air Attack parameters display.
3. Select the target unit.
4. Optionally, in the Maximum Engagement Distance box, specify the maximum distance from which the unit can attack the target. If it is set to 0 (zero), the maximum range is the maximum range of the weapon.
5. Click OK.

### 8.2.2 Air-to-Ground Air Base Attack

The Air-to-Ground Air Base Attack task causes a unit to move along an ingress route, move to a target point, make HE attack(s) against an air base, and then move along the egress route. The unit attacks with each different HE ammunition type in succession. For each ammunition type, the unit expends all of its ammunition. There is a 30-second delay between attacks with different ammunition types as the unit prepares to attack again.

► **To order an air-to-ground air base attack:**

1. Select an aircraft unit.
2. Choose **Task > Engagement > Air-to-Ground Air Base Attack**, or click the Air-to-Ground Air Base Attack button (  ). The Air-to-Ground Air Base Attack parameters display.
3. In the Target Air Base window, select the air base to attack.
4. Optionally, in the Part to Target list, select a specific type of target at the air base (buildings, apron, runway, or no specific part).
5. In the Ingress Route window, select the route the unit should follow to approach the air base.
6. In the Egress Route window, select the route the unit should follow to leave the air base.
7. Click OK.

### 8.2.3 Air-to-Ground Attack

The Air-to-Ground Attack task requires an attack (ingress) route, an exit (egress) route, and a target point.

When the aircraft reaches the target point, it attacks with each type of HE ammunition it has. It expends all ammunition of each type. There is a 30-second delay between attacks with each ammunition type.

► **To assign an Air-to-Ground Attack task:**

1. Select the unit that will attack.
2. Choose **Task > Engagement > Air-to-Ground Attack**, or click the Air-to-Ground Attack button (  ). The Air-to-Ground Attack parameters display.
3. Select a target point.
4. Select an ingress route.
5. Select an egress route.

 This is a multiple list dialog box. For details about how to control which list has focus, see "2.1.7 Specifying Parameters for Tasks" on page 11.

6. Click OK.

### 8.2.4 Attack by Fire

The Attack By Fire task is for use by units created by aggregating multiple units. When assigned, the units move to the specified location. Armor and mechanized units are placed towards the front of the unit. Artillery is placed in the rear. Other units are placed between the forward and rear simulation objects. When the unit is in position, artillery begins indirect fire on the target and the other units engage any enemy simulation objects that are in range.



► **To assign an Attack By Fire task:**

1. Select the unit.
2. Choose **Task > Engagement > Attack By Fire**. The Attack By Fire parameters display.
3. Specify the location of the target entity.
4. Specify a location for the attacking unit.
5. Click OK.

### 8.2.5 Attack with Anti-Air Missile

This task is available to any simulation object that has anti-air missiles.

► **To assign an Attack with Anti-Air Missile task:**

1. Select the simulation object that will attack.
2. Choose **Task > Engagement > Attack with Anti-Air Missile**, or click the Attack with Anti-Air Missile button (). The Attack with Anti-Air Missile parameters display.
3. Select a target.
4. Specify the number of missiles to fire.
5. Click OK.

### 8.2.6 Attack with Anti-Ship Missile

This task is available to any simulation object that has anti-ship missiles.

► **To assign an Attack with Anti-Ship Missile task:**

1. Select the simulation object that will attack.
2. Choose **Task > Engagement > Attack with Anti-Ship Missile**, or click the Attack with Anti-Ship Missile button (). The Attack with Anti-ship Missile parameters display.
3. Select a target.
4. Specify the number of missiles to fire.
5. Click OK.

### 8.2.7 Attack with Land-Attack Missile

This task is available to entities that have land-attack missiles, such as the Arleigh Burke DDG.

► **To assign an Attack with Land-Attack Missile task:**

1. Select the simulation object that will attack.
2. Choose **Task > Engagement > Attack with Land-Attack Missile**, or click the Attack with Land-Attack Missile button (). The Attack with Land-Attack Missile parameters display.
3. Select a target.

4. Specify the number of missiles to fire.
5. Click OK.

### 8.2.8 Attack with Torpedo

This task is available to entities that have torpedoes, such as the Arleigh Burke DDG.

► **To assign an Attack with Torpedo task:**

1. Select the entity that will attack.
2. Choose **Task > Engagement > Attack with Torpedo**, or click the Attack with Torpedo button (). The Attack with Torpedo parameters display.
3. Select a target.
4. Specify the number of torpedoes to fire.
5. Click OK.

### 8.2.9 Automatic Air Defense

When a simulation object has the Automatic Air Defense task, it automatically fires at enemy aircraft that enter the specified area. A simulation object must have an anti-air weapon, such as air-to-air missile to execute this task.

► **To assign an Automatic Air Defense task:**

1. Select a simulation object that has anti-air capability.
2. Choose **Task > Engagement > Automatic Air Defense**, or click the Automatic Air Defense button (). The Automatic Air Defense parameters display.
3. Select the area in which the simulation object should automatically fire anti-air weapons.
4. Click OK.

### 8.2.10 Biological Attack

The Biological Attack task lets a simulation object create an area contaminated with biological agents.

When you assign a Biological Attack task, you specify the center of the area in which it will take place. The size of the area and the allowed range from the simulation object is specified in the biological weapon's weapon system. You can edit these values in the Simulation Object Editor.

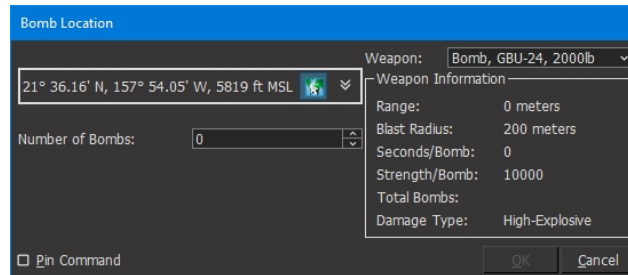
► **To assign a Biological Attack task:**

1. Select the simulation object that will make the attack.
2. Choose **Task > Engagement > Biological Attack**. The Biological Attack parameters display.
3. Click a location for the center of the area of attack or specify the coordinates.
4. In the Agent Type list, select the type of biological agent.
5. Click OK.

### 8.2.11 Bomb Location

You can specify that a simulation object bomb a location. When you choose a weapon, the Bomb Location dialog box displays information about the chosen weapon (Figure 40).

**Figure 40. Bomb Location dialog box**



► **To assign a Bomb Location task:**

1. Select the simulation object that will attack.
2. Choose **Task > Engagement > Bomb Location**. The Bomb Location parameters display.
3. Click the location to bomb or enter the coordinates. You can click on the terrain or expand the location group box by clicking the arrow next to the coordinates and altitude to enter a specific location.
4. Specify the number of bombs to drop.
5. In the Weapon list, select the type of bomb to drop.
6. Click OK.

### 8.2.12 Chemical Attack

The Chemical Attack task lets a unit create an area contaminated with chemical agents.

When you assign a Chemical Attack task, you specify the center of the area in which it will take place. The size of the area and the allowed range from the unit is specified in the chemical weapon's weapon system. You can edit this value in the Simulation Object Editor.

► **To assign a Chemical Attack task:**

1. Select the unit that will make the attack.
2. Choose **Task > Engagement > Chemical Attack**. The Chemical Attack parameters display.
3. Click a location for the center of the area of attack or specify the coordinates.
4. In the Agent Type list, select the type of chemical agent.
5. Click OK.

### 8.2.13 Indirect Fire

The Indirect Fire task allows a simulation object to fire on a point. The indirect fire scatters bursts within a sheaf radius, which is a circle around the target point.

If multiple units are within the radius, they are all attacked. Attack strength is applied fully to each attacked simulation object. It is not divided among them. For example, if there are four units in the sheaf radius and the attack strength is 100, an attack strength of 100 is applied to each of the units.

If the blast effect radius of the munitions being fired is less than the sheaf radius, the attack strength is reduced by the ratio of the areas of the blast circle to the sheaf circle. For example, if the sheaf radius is 100 and the blast effect radius is 50, the ratio of the areas is 1:4, so the attack strength would be 25% of the full value.

► **To specify indirect fire for a unit:**

1. Select the unit.
2. Choose **Task > Engagement > Indirect Fire**, or click the Indirect Fire button (). The Indirect Fire parameters display.
3. Select the point on which you want to center the indirect fire.
4. In the Weapon list, select the weapon you want to use. Its characteristics are displayed.
5. In the Number of Attacks box, specify how many attacks to make.
6. In the Sheaf Radius box, specify the radius, in meters, around which to scatter indirect fire. This value must be equal to or less than the Max Sheaf Radius value shown in the Weapon Information group box for the selected weapon.
7. Click OK.

### 8.2.14 Nuclear Attack

The Nuclear Attack task lets a simulation object create an area contaminated with nuclear radiation.

When you assign a Nuclear Attack task, you specify the center of the area in which it will take place. The size of the area and the allowed range from the simulation object is specified in the nuclear weapon's weapon system. You can edit this value in the Simulation Object Editor.

► **To execute a nuclear attack:**

1. Select the simulation object that will make the attack.
2. Choose **Task > Engagement > Nuclear Attack**. The Nuclear Attack parameters display.
3. Click a location for the center of the attack area or specify the coordinates.
4. In the Agent Type list, select the type of radiation.
5. Click OK.

## 8.3. Combat Engineering Tasks

The following topics describe tasks in the combat engineering category that apply to aggregate-level scenarios.

### 8.3.1 Breach Obstacles

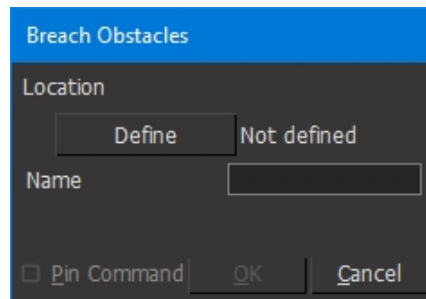
The Breach Obstacles task lets an engineering unit breach an obstacle so that units can pass through it. Depending on the breaching system a unit has, it may be able to breach fortifications, minefields, or ditches.

A breach line is specified with two points.

► **To assign a Breach Obstacles task:**

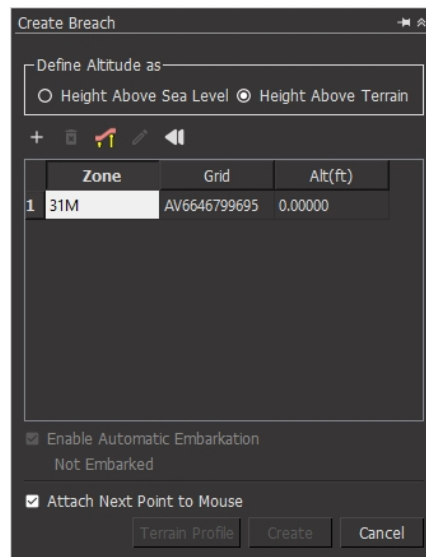
1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Breach Obstacles**, or click the Breach Obstacles button (🛡️🔧). The Breach Obstacles parameters display (Figure 41).

**Figure 41. Breach Obstacles dialog box**



3. Click Define. The cursor changes to draw mode and the Create Breach dialog box opens or becomes available (Figure 42).

**Figure 42. Create Obstacles dialog box**



4. Draw a two-point line to specify where to create the breach. After the second click the line disappears. This is because it is not being created now. The Breach Obstacles dialog box button now says Edit and the text next to it says Defined.
5. Click OK.

### 8.3.2 Construct Abatis

Combat engineering units can construct abatises. An abatis is specified as a point.

 You can create abatises directly from the Create Object Palette. This task is used when you want a unit to construct this obstacle.

► **To assign a Construct Abatis task:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Abatis**, or click the Construct Abatis button (). The Construct Abatis parameters display (similar to Figure 41 on the previous page ). The Location is specified as "Not defined".
3. Click Define. The Create Abatis dialog box opens (similar to Figure 42 on the previous page).
4. Click where you want to locate the abatis.
5. In the Create Abatis dialog box, click OK.
6. Optionally, specify a name for the abatis.
7. In the Construct Abatis dialog box, click OK.

### 8.3.3 Construct Anti-Tank Ditch

Combat engineering units can construct anti-tank ditches. An anti-tank ditch is a linear object. This task takes place over time.

 You can create anti-tank ditches directly from the Create Object Palette. This task is used when you want a unit to construct this obstacle.

► **To assign a Construct Anti-Tank Ditch task:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Anti-Tank Ditch**, or click the Construct Anti-Tank Ditch button (). The Construct Anti-Tank Ditch parameters display (similar to Figure 41). The Location is specified as "Not defined".
3. Click Define. The Create Anti-Tank Ditch dialog box opens and the cursor changes to draw mode.
4. Click to specify the vertexes of the line as you would when creating any linear object, or add points in the dialog box and click Create.
5. Optionally, specify a name for the ditch.
6. Click OK.

### 8.3.4 Construct Barbed Wire

Combat engineering units can construct barbed wire obstacles. Barbed wire is specified as a line.

**i** You can create barbed wire directly from the Create Object Palette. This task is used when you want a unit to construct this obstacle.

► **To construct barbed wire:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Barbed Wire**, or click the Construct Barbed Wire button (). The Construct Barbed Wire parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Barbed Wire dialog box opens.
4. Click to specify the vertexes of the line as you would when creating any linear object.
5. Optionally, specify a name for the barbed wire.
6. In the Construct Barbed Wire dialog box, click OK.

### 8.3.5 Construct Barricade

Combat engineering units can construct barricades. A barricade is specified as a line.

**i** You can create barricades directly from the Create Object Palette. This task is used when you want a unit to construct this obstacle.

► **To construct a barricade:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Barricade**, or click the Construct Barricade button (). The Construct Barricade parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Barricade dialog box opens.
4. Click to specify the vertexes of the line as you would when creating any linear object.
5. Optionally, specify a name for the barricade.
6. In the Construct Barricade dialog box, click OK.

### 8.3.6 Construct Berm

Combat engineering units can construct berms. A berm is specified as a line.

**i** You can create berms directly from the Create Object Palette. This task is used when you want a unit to construct this obstacle.

► **To construct a berm:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Berm**, or click the Construct Berm button (). The Construct Berm parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Berm dialog box opens.
4. Click to specify the vertexes of the line as you would when creating any linear object.
5. Optionally, specify a name for the berm.
6. In the Construct Berm dialog box, click OK.

### 8.3.7 Construct Booby Trap

Combat engineering entities can construct booby traps. A booby trap is a location where explosives detonate if triggered by the presence of a simulation object. Booby traps cause some level of attrition on the affected simulation object. To create a booby trap, a unit must have explosive resources.

 You can create booby traps directly from the Create Object Palette. This task is used when you want a unit to construct this object.

► **To construct a booby trap:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Booby Trap**, or click the Construct Booby Trap button (). The Construct Booby Trap parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Booby Trap dialog box opens.
4. Click where you want to locate the booby trap or type the coordinates in the dialog box and click OK.
5. Optionally, specify a name for the booby trap.
6. Click OK.

### 8.3.8 Construct Bridge

Combat engineering units can construct bridges. A bridge is specified as a two-point line.

 You can create bridges directly from the Create Object Palette. This task is used when you want a unit to construct this object.



► **To construct a bridge:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Bridge**, or click the Construct Bridge button (). The Construct Bridge parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Bridge dialog box opens and the cursor changes to draw mode.
4. Click on the terrain to specify the start and end vertices of the bridge. The Create Bridge dialog box closes.
5. Optionally, specify a name for the bridge.
6. Click OK.

### 8.3.9 Construct Dry Gap

Combat engineering entities can construct dry gaps. A dry gap is a type of ditch. (Use Improve Ditch and Destroy Ditch to continue building a dry gap or destroy it.)

 You can create dry gaps directly from the Create Object Palette. Use this task when you want a unit to construct this object.

► **To construct a dry gap:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Dry Gap**, or click the Construct Dry Gap button (). The Construct Dry Gap parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Dry Gap dialog box opens and the cursor changes to draw mode.
4. Click to specify the vertices of the dry gap.
5. Double-click to finish the dry gap, or click Create in the Create Dry Gap dialog box.
6. Optionally, specify a name for the dry gap.
7. Click OK.

### 8.3.10 Construct Fortified Area

Combat engineering entities can construct fortified areas. A fortified area is defined as a box. You specify the southwest corner of the area. Construction of a fortified area takes place over time.

 You can create fortifications directly from the Create Object Palette. This task is used when you want a unit to construct this object.

► **To construct a fortified area:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Fortified Area**, or click the Construct Fortified Area button (). The Construct Fortified Area parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Fortified Area dialog box opens and the cursor changes to show an area.
4. Click where you want to locate the southwest corner of the area or type the coordinates in the dialog box and click Create.
5. Optionally, specify a name for the area.
6. Click OK.

### 8.3.11 Construct Fortified Line

Combat engineering entities can construct fortified lines.

 You can create fortifications directly from the Create Object Palette. This task is used when you want a unit to construct this object.

► **To construct a fortified line:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Fortified Line**, or click the Construct Fortified Line button (). The Construct Fortified Line parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Fortified Line dialog box opens and the cursor changes to draw mode.
4. Click to specify the vertexes of the line as you would when creating any linear object, or add points in the dialog box and click Create.
5. Optionally, specify a name for the line.
6. Click OK.

### 8.3.12 Construct Minefield

Combat engineering entities can construct volcano minefields. Minefields are fixed size rectangles.

 You can create minefields directly from the Create Object Palette. This task is used when you want a unit to construct this obstacle.

► **To construct a minefield:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Minefield**, or click the Construct Minefield button (). The Construct Minefield parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Minefield (Volcano) dialog box opens and the cursor changes to show a minefield.
4. Click where you want to locate the southwest corner of the minefield or type the coordinates in the Create Minefield (Volcano) dialog box and click Create.
5. Optionally, specify a name for the minefield.
6. In the Construct Minefield dialog box, click OK.

### 8.3.13 Construct Strong Point

Combat engineering entities can construct strong points. A strong point is defined as a box. You specify the southwest corner.

 You can create strong points directly from the Create Object Palette. This task is used when you want a unit to construct this object.

► **To construct a strong point:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Construct Strong Point**, or click the Construct Strong Point button (). The Construct Strong Point parameters display (similar to Figure 42 on page 169). The Location is specified as "Not defined".
3. Click Define. The Create Strong Point dialog box opens and the cursor changes to show a box.
4. Click where you want to locate the southwest corner of the box or type the coordinates in the dialog box and click Create.
5. Optionally, specify a name for the strong point.
6. Click OK.

### 8.3.14 Destroy Bridge

When you task an engineering unit to destroy a bridge, it consumes its Explosives resource. As the unit attacks the bridge, the bridge's completeness percentage decreases. When the completion percentage reaches zero, the bridge is removed from the scenario.

► **To destroy a bridge:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Destroy Bridge**, or click the Destroy Bridge button (). The Destroy Bridge parameters display.

3. Select the bridge you want to destroy.
4. Click OK.

### 8.3.15 Destroy Ditch

When you task an engineering unit to destroy a ditch or dry gap, it consumes its Explosives resource. As the unit attacks the ditch, the ditch's completeness percentage decreases. When the completion percentage reaches zero, the ditch is removed from the scenario.

► **To destroy a ditch:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Destroy Ditch**, or click the Destroy Ditch button (). The Destroy Ditch parameters display.
3. Select the ditch you want to destroy.
4. Click OK.

### 8.3.16 Destroy Explosive

The Destroy Explosive task lets a unit destroy a booby trap or unexploded ordnance.

► **To destroy explosives:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Destroy Explosive**, or click the Destroy Explosive button (). The Destroy Explosive parameters display.
3. Select the booby trap or unexploded ordnance you want to destroy.
4. Click OK.

### 8.3.17 Destroy Fortification

When you task an engineering unit to destroy a fortification, it consumes its Explosives resource. As the unit attacks the fortification, the fortification's completeness percentage decreases. When the completion percentage reaches zero, the fortification is removed from the scenario.

► **To destroy a fortification:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Destroy Fortification**, or click the Destroy Fortification button (). The Destroy Fortification parameters display.
3. Select the fortification you want to destroy.
4. Click OK.

### 8.3.18 Destroy Obstacle

The Destroy Obstacle task lets a unit destroy obstacles (such as berms, barricades, and barbed wire) that do not have a specific destroy task, such as Destroy Fortification or Destroy Explosive.

► **To destroy an obstacle:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Destroy Obstacle**, or click the Destroy Obstacle button (). The Destroy Obstacle parameters display.
3. Select the obstacle you want to destroy.
4. Click OK.

### 8.3.19 Improve Booby Trap

The Improve Booby Trap task lets a combat engineering unit improve an incomplete booby trap. When you improve a booby trap, it is improved to as complete a strength level as the unit can bring it to. This task can be assigned to a unit to assist another unit that is creating a booby trap.

► **To improve a booby trap:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Improve Booby Trap**. The Improve Booby Trap parameters display.
3. Select the booby trap that you want to improve.
4. Click OK.

### 8.3.20 Improve Breach

The Improve Breach task lets a combat engineering unit improve an incomplete breach. When you improve a breach, it is improved to as complete a strength level as the unit is able to do. You can assign this task to a unit to assist another unit that is breaching an obstacle.

► **To improve a breach:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Improve Breach**. The Improve Breach parameters display.
3. Select the breach you want to improve.
4. Click OK.

### 8.3.21 Improve Bridge

The Improve Bridge task lets a combat engineering unit improve an incomplete bridge. When you improve a bridge, it is improved to as complete a strength level as the unit can bring it to. This task can be assigned to a unit to assist another unit that is creating a bridge.

► **To improve a bridge:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Improve Bridge**. The Improve Bridge parameters display.
3. Select the bridge you want to improve.
4. Click OK.

### 8.3.22 Improve Ditch

The Improve Ditch task lets a combat engineering unit improve an incomplete anti-tank ditch or dry gap. When you improve a ditch, it is improved to as complete a strength level as the unit can bring it to. This task can be assigned to a unit to assist another unit that is creating a ditch.

► **To improve a ditch:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Improve Ditch**. The Improve Ditch parameters display.
3. Select the ditch that you want to improve.
4. Click OK.

### 8.3.23 Improve Fortification

The Improve Fortification task lets a unit improve an incomplete fortification area, line, or strong point. When you improve a fortification, it is improved to as complete a strength level as the unit can bring it to. This task can be assigned to a unit to assist another unit that is creating a fortification.

► **To improve a fortification:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Improve Fortification**. The Improve Fortification parameters display.
3. Select the fortification that you want to improve.
4. Click OK.

### 8.3.24 Improve Obstacle

The Improve Obstacle task lets a unit improve an incomplete obstacle (such as berms, barricades, and barbed wire) that does not have a dedicated improve task, such as Improve Breach or Improve Bridge. When you improve an obstacle it is improved to as complete a strength level as the unit can bring it to. This task can be assigned to a unit to assist another unit that is constructing an obstacle.

► **To improve an obstacle:**

1. Select a combat engineering unit.
2. Choose **Task > Combat Engineering > Improve Obstacle**. The Improve Obstacle parameters display.
3. Select the obstacle that you want to improve.
4. Click OK.

## 8.4. Air Base Tasks

The following topics describe tasks for aggregate-level air bases.

### 8.4.1 Launch Flight

An air base uses the Launch Flight task to configure and launch a flight of aircraft. A flight can be launched to execute a single unit mission or it can be one of several flights being launched as part of a multi-unit mission. If you want to set up a multi-unit mission, you must create a unit independently of the air base to act as the mission superior. For information about missions, see "Air Missions".

When you assign this task, aircraft of the indicated type are taken from the available aircraft at the base. The air base determines the munitions and fuel required by the flight given the loadout and provides them from its stores. Air crews are supplied from the available air crews. If the air base does not have enough of any of these resources, the task fails and the flight is not scheduled to launch.

When the flight is launched, the aircraft and crews are removed from the air base.

A Launch Flight task takes values for these parameters:

- Call sign.
- Aircraft type.
- Number of aircraft in the flight.
- Loadout.
- Mission superior.
- Mission type. The mission can be one of:
  - BAI. Battlefield air interdiction.
  - CAP. Combat air patrol.
  - CAS. Close air support.
  - Escort.
  - Intercept.
  - OCA. Offensive counter air (bombing air bases).
  - Recon. Reconnaissance.
  - SEAD. Suppression of enemy air defense.

- Sweep. Sweep enemy aircraft from the air space where you are going to attack.
- Tanker.

The mission type is informational. It does not influence the behavior of the flight.

- Launch time. This can be immediate or at a specified date and time.
- Wait for launch. Global plans only. If selected, the global plan does not move to the next task until the flight is launched.

► **To launch a flight:**

1. Select an air base.
2. Choose **Task > Launch Flight**, or click the Launch Flight button (). The Launch Flight parameters display.
3. Specify a call sign for the flight. It can be any text.
4. Select the aircraft type. It must be a type of aircraft that is already assigned to the air base. The list of aircraft types may include aircraft that are not assigned.
5. Specify the number of aircraft in the flight.
6. The Unit Type is not editable. It is taken from the aircraft type.
7. Select a loadout from the Loadout list.
8. Optionally, specify a mission superior if this flight is subordinate to another one.
9. In the Mission Type list, select the type of mission this is.
10. Select Launch as Soon as Aircraft are Available or specify a date and time for the launch.
11. Optionally, for global plans only, specify that the global plan not proceed to the next task until the flight is launched (Wait for Launch check box).

### 8.4.2 Launch QRA Aircraft

The Launch QRA Aircraft task creates an air unit and launches the specified number of aircraft that are on Quick Reaction Alert (QRA) status.

► **To launch a QRA unit:**

1. Select an air base.
2. Choose **Task > Launch QRA Aircraft**, or click the Launch QRA Aircraft button. The Launch QRA Aircraft parameters display.
3. Specify a call sign for the unit that is to be created.
4. Specify the number of aircraft to include in the unit.
5. Click OK.

### 8.4.3 Prepare QRA

The Prepare QRA task assigns aircraft to quick reaction alert (QRA) status. You specify the type of aircraft, loadout, when they should be prepared, and their response time. Once the aircraft are prepared they are available to be launched.



► **To prepare aircraft for QRA status:**

1. Select an air base.
2. Choose **Task > Prepare QRA**, or click the Prepare QRA button (). The Prepare Aircraft for QRA Status parameters display.
3. In the Aircraft Type list, select an aircraft type.
4. In the Number list, select the number of aircraft.
5. In the Loadout list, select a loadout.
6. If you want aircraft to be prepared as soon as they are available, select the Prepare As Soon As Aircraft Are Available check box. If you want them prepared at a specific time, clear the check box and in the Start Time box, specify the time when you want the preparation process to start.
7. Select a Response Time option.
8. Click OK.

## 8.5. Movement Tasks for Aggregate-Level Scenarios

The following topics describe tasks in the movement category that apply to aggregate-level scenarios.

### 8.5.1 Change MOPP Level

Units automatically raise their mission oriented protective posture (MOPP) level when they encounter a contamination area. However, you can manually change a unit's MOPP level.

Supported MOPP levels are:

- MOPP Level 0 — Protective mask carried. Suit, gloves, and boots accessible/available.
- MOPP Level 1 — Suit worn. Mask, gloves and boots carried.
- MOPP Level 2 — Suit and boots worn. Gloves and mask carried.
- MOPP Level 3 — Suit, boots and mask worn. Gloves carried.
- MOPP Level 4 — All protection worn.

Changes to MOPP level do not take place immediately. (To change the MOPP level immediately, see "MOPP Level" for details on the set data request.) Increasing the MOPP level degrades a simulation object's speed and combat effectiveness. Increasing the MOPP level does not remove existing contamination, but prevents most NBC contamination from harming the simulation object.

► **To assign a Change MOPP Level task:**

1. Select the unit.
2. Choose **Task > Movement > Change MOPP Level**, or click the Change MOPP Level button (). The Change MOPP Level parameters display.

3. Specify a new MOPP level.
4. Click OK.

### 8.5.2 Change Posture

You can set a simulation object's posture to:

- Travel
- Reconnaissance
- Hasty attack
- Deliberate attack
- Hasty defense
- Deliberate defense

Changes to posture take place over time. To change posture immediately, use Set Posture.

► **To assign a Change Posture task:**

1. Select the simulation object.
2. Choose **Task > Movement > Change Posture**, or click the Change Posture button. The Change Posture parameters display.
3. Specify a new posture.
4. Click OK.

### 8.5.3 Escort Air Unit

The Escort Air Unit task causes an aircraft unit to move to a location and escort the specified unit.

► **To assign an escort air unit task:**

1. Select an air unit.
2. Choose **Task > Movement > Escort Air Unit**, or click the Escort Air Unit button (). The Escort Air Unit parameters display.
3. Select a point as the location to meet.
4. Select the unit to escort.
5. Click OK.

### 8.5.4 Halt Movement

The Halt Movement task causes a simulation object to stop moving.

► **To halt a simulation object's movement:**

1. Select the simulation object.
2. Choose **Task > Movement > Halt Movement**, or click the Halt Movement button (). Movement stops immediately.

### 8.5.5 Land at Air Base

The Land at Air Base task causes an air unit to request an air base to let it land. The air unit tells the air base how many aircraft it has and how long a runway it requires.

If the request is accepted, the unit flies to the air base and requests final permission to land. The air base absorbs the resources of the unit and deletes it from the scenario. For complete details about how landing works, see "Landing Air Units".

► **To land an aircraft unit at an air base:**

1. Select an aircraft unit.
2. Choose **Task > Movement > Land at Air Base**, or click the Land at Air Base button (  ). The Land at Air Base parameters display.
3. Select the air base you want the unit to land at.
4. Click OK.

### 8.5.6 Move Along Route

For details, please see "3.20. Move Along Route" on page 52.

### 8.5.7 Move Along Route Retrograde

Move Along Route Retrograde is the same as Move Along Route, except that the simulation object does not change its heading to the direction of movement. It maintains the heading at the start of movement or a heading necessary for defensive operations. See "3.20. Move Along Route" on page 52 for the procedure for this task.

### 8.5.8 Move To

For details, please see "3.26. Move To" on page 58.

### 8.5.9 Move to Retrograde

Move to Retrograde is the same as Move To, except that the simulation object does not change its heading to the direction of movement. It maintains the heading at the start of movement or a heading necessary for defensive operations. Please see "3.26. Move To" on page 58 for the procedure for this task.

### 8.5.10 Patrol Route

For details, see "3.33. Patrol Route" on page 62.

### 8.5.11 Patrol Between

For details, please see "3.32. Patrol Between" on page 61.

### 8.5.12 Turn To Heading

The Turn to heading task for aggregate-level scenarios causes entities to turn to a new heading. It happens instantly, similar to the Heading set data request.

► **To assign a Turn to Heading task:**

1. Select the entity.
2. Choose **Task > Movement > Turn to Heading**. The Turn to Heading parameters display.
3. Type a heading.
4. In the Relative To list, select one of these options:
  - True North. Turn to the specified absolute heading (true north).
  - Host Object (if embarked). If embarked, turn to the specified heading relative to the host simulation object.
  - Current Heading. Turn the specified number of degrees relative to the current heading.
  - Magnetic North. Turn to the specified heading relative to magnetic north.
5. Click OK.

### 8.5.13 Wait for Escort

You can instruct an aircraft unit to move to a waypoint and wait for an escort.

► **To assign a Wait for Escort task:**

1. Select an aircraft unit.
2. Choose **Task > Movement > Wait for Escort**, or click the Wait for Escort button (). The Wait for Escort parameters display.
3. Select the waypoint that the unit should move to.
4. Click OK.

### 8.5.14 Wait Tasks

You can specify unconditional wait, wait for a duration, or wait until a simulation time. For details, please see "7.6. Wait Tasks" on page 149.

## 8.6. Embarkation Tasks

The following topics describe tasks in the embarkation category that apply to aggregate-level scenarios.

### 8.6.1 Disembark

For details, please see "7.1.1 Disembark" on page 133.

### 8.6.2 Disembark All

For details, please see "7.1.2 Disembark All" on page 133.

### 8.6.3 Embark

For details, please see "7.1.3 Embark" on page 133.

## 8.7. Radio Tasks

This section lists the tasks that allow you to use the VR-Forces radio network to send messages, tasks, and set data requests.

### 8.7.1 Send Radio Set

For details, please see "7.5.1 Send Radio Set" on page 147.

### 8.7.2 Send Radio Task

For details, please see "7.5.2 Send Radio Task" on page 148.

### 8.7.3 Send Text Message

For details, see "7.5.3 Send Text Message" on page 148.

## 8.8. Other Tasks

The following topics describe other tasks that apply to aggregate-level scenarios.

### 8.8.1 Activate Jammer

The Activate Jammer task turns on jamming for a simulation object equipped with a jamming emitter. This task has the same effect as Set Emitter - On.

- To activate a simulation object's jammer, choose **Task > Jamming > Activate Jammer**, or click the Activate Jammer button ().

### 8.8.2 Deactivate Jammer

The Deactivate Jammer task turns off jamming for a simulation object equipped with a jamming emitter. This task has the same effect as Set Emitter - Off.

- To deactivate a simulation object's jammer, choose **Task > Jamming > Deactivate Jammer**, or click the Deactivate Jammer button ().

### 8.8.3 Drop Fuel Tanks

The Drop Fuel Tanks task causes an air unit to drop its fuel tanks.

- **To drop an air unit's fuel tanks:**
  1. Select an air unit.
  2. Choose **Task > Drop Fuel Tanks**, or click the Drop Fuel Tanks button ().

### 8.8.4 FASCAM

Entities with a FASCAM weapon system, such as artillery, can fire FASCAM (family of scatterable mines).

► **To fire FASCAM:**

1. Select a simulation object that can execute this task.
2. Choose **Task > FASCAM**, or click the FASCAM button (). The FASCAM parameters display (similar to Figure 41). The Location is specified as "Not defined".
3. Click Define. The Create Minefield (ADAM-RAAM) dialog box opens and the cursor changes to show a minefield.
4. Click where you want to locate the southwest corner of the minefield or type the coordinates in the Create Minefield (ADAM-RAAM) dialog box and click Create.
5. Optionally, specify a heading.
6. In the FASCAM dialog box, click OK.

### 8.8.5 Refuel from Tanker

The Refuel from Tanker task causes an aircraft unit to request refueling from a fuel tanker. A message is sent to the tanker indicating the amount of fuel required and the type of connection. The tasked aircraft then performs a series of steps:

1. The aircraft unit flies to a standby position near the tanker and follows the tanker.
2. The aircraft unit follows the tanker at range 0 and refuels.
3. The aircraft unit moves away from the tanker and follows at a small distance.

Each of these steps is performed in response to a message from the tanker indicating that the tanker is ready. During this time the receiving aircraft unit also gives the tanker information about the number of aircraft in the unit and the maximum transfer rate for the aircraft.

If the receiving aircraft unit is an echelon containing subordinate units, then each subordinate coordinates with the tanker independently.

► **To refuel an aircraft unit:**

1. Select the unit.
2. Choose **Task > Combat Service Support > Refuel from Tanker** or click the Refuel from Tanker button (). The Refuel from Tanker parameters display.
3. Select the tanker you want to refuel from.
4. Click OK.

### 8.8.6 Refueling Mission

The Refueling Mission task causes a tanker aircraft to follow a route or move to a waypoint and fly a holding pattern. It awaits requests from other aircraft to refuel; see "8.8.5 Refuel from Tanker" above.

► **To issue a refueling mission:**

1. Select an aircraft that has refueling capability.
2. Choose **Task > Combat Service Support > Refueling Mission**, or click the Refueling Mission button (). The Refueling Mission parameters display.
3. Specify the altitude at which to conduct the refueling mission.
4. Specify how much fuel to reserve for the tanker's own use. If the fuel level drops to this amount, the tanker will stop refueling other aircraft.
5. Specify a route to follow or specify the parameters for a race track holding pattern.
6. Click OK.

### 8.8.7 Send NBC Report

Sending an NBC report is the equivalent of sending a spot report when an NBC area is found. (An NBC area is a nuclear, biological, or chemical contamination area.) NBC reports are sent automatically if a simulation object has an NBC sensor. However, this task allows you to send a report manually. For more on NBC areas, see "Contamination Areas".

► **To send an NBC report:**

1. Select a simulation object.
2. Choose **Task > Send NBC Report**. The Send NBC Report parameters display.
3. Select the contamination area for which to send a report.
4. Specify the Engagement Result. This is any text that you want to accompany the report.
5. Click OK.

### 8.8.8 Send Obstacle Report

Sending an obstacle report is the equivalent of sending a spot report when an obstacle area is found. Obstacle reports are sent automatically if a simulation object has an engineering object sensor. However, this task allows you to send a report manually.

► **To send an obstacle report:**

1. Select a simulation object.
2. Choose **Task > Send Obstacle Report**. The Send Obstacle Report parameters display.
3. Select the obstacle for which to send a report.
4. Specify the Engagement Result. This is any text that you want to accompany the report.
5. Click OK.

### 8.8.9 User Task

For details, please see "7.13. User Task" on page 158.

## 8.9. Background Process List

Background processes are not described in detail because they run automatically; no user action is required to execute them.

Table 6 lists the background tasks provided. These tasks are only available for units in aggregate-level SMSs.

**Table 6. Background processes**

| Background Process               | Description                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Attack Enemies                   | Determines when to begin and stop attacking enemies.                                                                                                                                                                                                                                                                                                                                        |
| Attrit Air Base                  | Handles air-to-ground attacks against an air base.                                                                                                                                                                                                                                                                                                                                          |
| Calculate Maximum Speed Modifier | Calculates a modifier that is applied to the maximum possible speed of a unit to modify its movement rate.                                                                                                                                                                                                                                                                                  |
| Make EW Attack                   | If a jammer system is turned on, attack all non-friendly simulation objects within range of the jammer. The simulation object attacks continuously.                                                                                                                                                                                                                                         |
| Manage Air Base                  | Launches scheduled flights and updates resources for an air base.                                                                                                                                                                                                                                                                                                                           |
| Perform Attrition                | Determines when a unit is being attacked and modifies its health based on the current attrition rate.                                                                                                                                                                                                                                                                                       |
| Provide Supplies                 | Continually checks to see if conditions allow providing supplies (not in combat, not ordered to stop providing supplies).<br>Checks to see what friendly units are in range, and sends each new unit an influence to let it know that this unit is available.<br>When a unit goes out of range, or this unit is ordered to stop providing supplies, it sends a stop influence to that unit. |
| Receive Construction             | Allows an engineering object to receive construction from other simulation objects.                                                                                                                                                                                                                                                                                                         |
| Resize Footprints                | Updates the size of a simulation object's physical and sensor footprints.                                                                                                                                                                                                                                                                                                                   |
| Update Combat Effectiveness      | Updates the combat effectiveness state of a simulation object.                                                                                                                                                                                                                                                                                                                              |
| Update EW Degradation            | Processes jamming interactions and computes the level of degradation in communications and radar from these attacks.                                                                                                                                                                                                                                                                        |
| Use Supplies                     | Uses supplies based on the simulation object's current activities.                                                                                                                                                                                                                                                                                                                          |





## 9. Creating Scripted Tasks and Sets

VR-Forces lets you create new tasks and set data requests by writing scripts. You do not need to know C++ or have a developers license to write these scripts. This chapter explains how to create and edit script metadata and how to organize and enable scripts. "Chapter 10. Writing Scripts" on page 229 explains how to write script code. "Chapter 12. Using Qt Designer for Script Dialog Boxes" on page 297 explains how to use Qt Designer to lay out dialog boxes.

|                                                                             |     |
|-----------------------------------------------------------------------------|-----|
| 9.1. Introduction to Scripts .....                                          | 191 |
| 9.1.1 Scripted Tasks .....                                                  | 192 |
| 9.1.2 Scripted Sets .....                                                   | 192 |
| 9.1.3 Reactive Task Scripts .....                                           | 192 |
| 9.1.4 Background Process Scripts .....                                      | 192 |
| 9.1.5 Script Metadata .....                                                 | 193 |
| 9.1.6 The Lua Scripting Language .....                                      | 193 |
| 9.2. Creating a New Script .....                                            | 193 |
| 9.2.1 Specifying the Script ID .....                                        | 198 |
| 9.2.2 Specifying a Menu Icon .....                                          | 199 |
| 9.2.3 Specifying an Extended Name .....                                     | 199 |
| 9.2.4 Specifying a Short Description .....                                  | 200 |
| 9.2.5 Specifying a Description .....                                        | 200 |
| 9.2.6 Specifying the Parameters for a Script .....                          | 200 |
| 9.2.7 Specifying a Menu Location .....                                      | 210 |
| 9.2.8 Specifying Action Categories .....                                    | 211 |
| 9.2.9 Configuring a Script's Availability to a Simulation Object Type ..... | 211 |
| 9.2.10 Specifying Equivalent Scripts for a Toolbar Button .....             | 215 |
| 9.2.11 Specifying the Programming Language for a Script .....               | 216 |
| 9.2.12 Configuring Scripted Tasks to Run Concurrently .....                 | 217 |
| 9.2.13 Creating Scripted Sets .....                                         | 217 |
| 9.2.14 Creating Reactive Tasks .....                                        | 218 |
| 9.2.15 Creating Background Processes .....                                  | 219 |

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| 9.3. Saving Scripts .....                                       | 220 |
| 9.4. Editing a Script .....                                     | 221 |
| 9.5. Filtering the List of Scripts .....                        | 221 |
| 9.6. Organizing Scripts into Folders .....                      | 223 |
| 9.6.1 Adding a Folder .....                                     | 223 |
| 9.6.2 Renaming a Folder .....                                   | 223 |
| 9.6.3 Deleting a Folder .....                                   | 223 |
| 9.6.4 Adding Scripts to a Folder .....                          | 223 |
| 9.6.5 Removing a Script from a Folder .....                     | 223 |
| 9.7. Exporting and Importing Scripts .....                      | 223 |
| 9.7.1 Importing a Script Package .....                          | 224 |
| 9.8. Copying a Script .....                                     | 225 |
| 9.9. Deleting a Script .....                                    | 225 |
| 9.10. Creating a System Script .....                            | 225 |
| 9.10.1 Including System Scripts on the Task and Set Menus ..... | 226 |
| 9.11. Specifying a Script Editor .....                          | 227 |
| 9.12. Editing Lua Files .....                                   | 228 |

## 9.1. Introduction to Scripts

The script capability in VR-Forces allows you to create new tasks and set data requests for simulation objects without using the VR-Forces Toolkit and C++ API. Scripts use the Lua programming language. The actions that simulation objects can carry out are still limited to the basic C++ tasks provided by VR-Forces. However, since Lua is a full-featured scripting language, it is far more powerful than the language used to create plans and you can be much more creative in how you use the basic tasks. In addition to functions for tasks and sets, Lua scripts can make many types of calls to the simulation engine to get simulation object status or other simulation data. Scripts can also set some simulation data.

Scripts are classified as system scripts or as scenario scripts. System scripts are saved as part of a simulation model set (SMS) and are available to all scenarios that use that SMS. Scenario scripts are available only in the scenario for which the script was created. However, you can save a scenario script as a system script if you want to and, just as with other data used by VR-Forces, there are ways of copying scripts from one VR-Forces installation to another.

VR-Forces supports the following types of scripts:

- Task. Scripted tasks can be listed on the Task menu and used as independent tasks or in plans just like C++ tasks.
- Reactive task. A reactive task monitors the simulation and activates when the conditions set in the script become true. Reactive tasks are enabled or disabled for simulation objects through the GUI.
- Background process. A background process runs continuously from the time a simulation object is created. There is no user interface for assigning background processes to simulation objects. Their use is based entirely on the scripting process.
- Set data request. Set data request scripts change simulation object state without interrupting the current task.

From a Lua script you can:

- Create and delete simulation objects and tactical graphics.
- Set weather and environment state.
- Get simulation object state, such as:
  - Aggregation
  - Location
  - Embarkation
  - Type
  - Force
  - Damage state
  - Velocity
- Monitor and set simulation object resources.

- Get simulation object sensor contacts.
- Task other simulation objects.
- Subtask self.
- Send Set Data Request messages.
- Test terrain:
  - Get terrain height.
  - Check line of sight.
  - Find feature data.

VR-Forces includes ZeroBrane Studio, a lightweight IDE, as the default script editor. You can configure VR-Forces to use a different editor if you want to. For details, see "9.11. Specifying a Script Editor" on page 227.

### 9.1.1 Scripted Tasks

Scripted tasks can be added to the Task menu and are treated just like the tasks that are built into VR-Forces.

When you create a scripted task, you can specify input parameters. VR-Forces can automatically build a dialog box for users to specify the parameter values or you can use Qt Designer to lay out the dialog box. For more information, see "Chapter 12. Using Qt Designer for Script Dialog Boxes" on page 297.

### 9.1.2 Scripted Sets

Scripted sets can be added to the Set menu and are treated just like the set data requests that are built into VR-Forces.

When you create a scripted set, you can specify input parameters. VR-Forces can automatically build a dialog box for users to specify the parameter values or you can use Qt Designer to lay out the dialog box. For more information, see "Chapter 12. Using Qt Designer for Script Dialog Boxes" on page 297.

A scripted set cannot issue tasks to the entity that executes it or to any other entity.

### 9.1.3 Reactive Task Scripts

Reactive tasks monitor a simulation and activate when the conditions set in the script become true. Reactive tasks are not added to the Task menu. In most respects the process for creating and writing reactive tasks is the same as creating and writing all scripts. Assume that all details about scripts apply to reactive tasks unless noted otherwise. For conceptual details about reactive tasks, see "2.5. Reactive Tasks" on page 19.

### 9.1.4 Background Process Scripts

The background process is a script that runs in the background for the simulation object types for which it was written. Since a background process has no user interface, it does not have any input parameters. When you create a background process, you specify the simulation object types for which it is available and you write the script.

### 9.1.5 Script Metadata

A script has two major components, the script code, and the script metadata. The metadata provides all of the information that VR-Forces needs to store the script, add it to a menu, and generate a dialog box.

### 9.1.6 The Lua Scripting Language

Lua is a lightweight scripting language used by VR-Forces to implement the script feature. VR-Forces includes a set of Lua functions that map the built-in tasks and set data requests as well as the ability to obtain simulation object state information. You use these functions as the starting point for building new tasks. For details about the Lua language, please visit the Lua website (<https://www.Lua.org>).

## 9.2. Creating a New Script

To create a new script, you specify the metadata for the script and write the script code. A script's metadata specifies the script's name, where it will be placed on a menu, and its parameters, if any. Specifying metadata may be an iterative process, particularly the parameters, which may change as you write the code for the script. However, the minimum first step in creating a script is to specify a name for the script and add it to the list of scripts. Once you do this, you can edit the metadata and the code as needed until you have completed the script to your satisfaction.

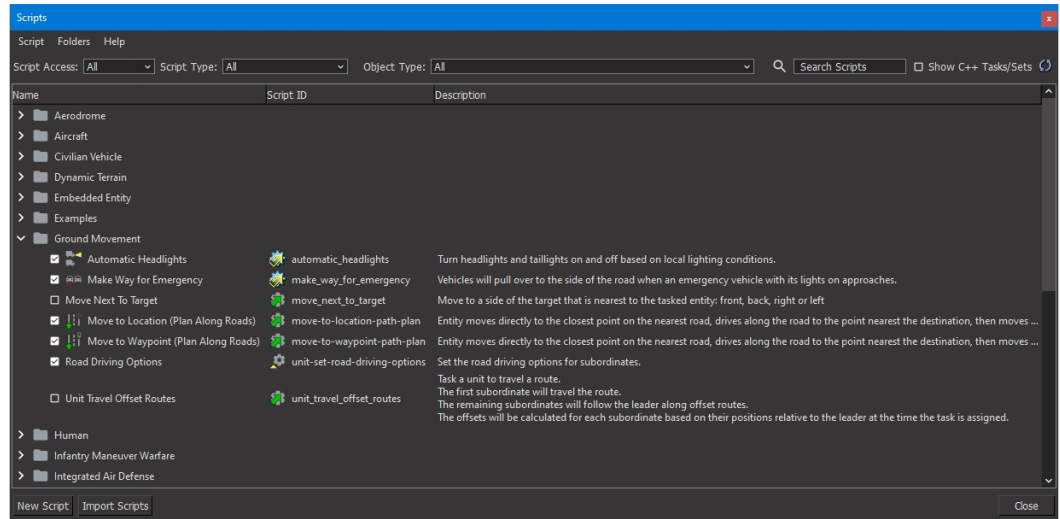
All new scripts are created as scenario scripts. If you want to create a new system script, you save the scenario script as a system script. For details, see "9.10. Creating a System Script" on page 225.

 To create a script, you must have a scenario open. This is true even if you plan to create a system script rather than a scenario script.

► **To create a new script:**

1. Open a scenario. (If you want to create a system script, you must create it in the context of a scenario. Then you will promote it to a system task.)
2. Choose **Simulation > Scripts**. The Scripts dialog box opens (Figure 43).

Figure 43. Scripts dialog box



3. If you have a folder hierarchy and want to create the script in a particular folder, select the folder. (For more information, see "9.6. Organizing Scripts into Folders" on page 223.)
4. Choose **Script > New Script** or click the New Script button. The New Script dialog box opens (Figure 44).

Figure 44. New Scripted Task dialog box

**New Script**

Script ID: <auto generate>

Script Type: Task Language: Lua

Name: Use Qt Designer

Extended Name: Parameters

Short Description: Parameter Type Visible

☐ Description:

Version: 1 ☒ Make Available Based on Entity Type

Action Categories: ☐ Depth control ☐ Lifeform Posture ☒ Movement ☐ Radio Communication ☒ Torpedo Weapon ☐ Weapon Valid for Object Types

☐ Can Be Executed More Than Once at a Time

Tool Tip: All

Menu Icon: ☐ Make Available Based on Behavior Set

Command Level: Primary

Magic Command: No

☐ Show On Top Level Right Click Menu

☐ One Click Support

☒ Show Task on Task Menu

Menu Location: Top Level Menu

☐ Show Task on Task Toolbar

Toolbar Location: At End of Toolbar ☐ Make Available to Global Plans

Edit Script Preview Add Cancel

5. In the Script Type list, select the type of script you are creating (Task, Reactive Task, Set, or Background Process). The dialog box displays the correct set of metadata for the type of script you chose.
6. Complete the fields in the dialog box. They are described in Table 7 and in following sections.
7. Optionally, click Preview to see the dialog box that the script will create.
8. Optionally, click Edit Script to begin writing the code for this script.
9. Click Add. The script is added to the list in the Scripts window.
10. To save the script task as part of the scenario, save the scenario.
11. Write the code for the script. For details, see "Chapter 10. Writing Scripts" on page 229.

**Table 7. Script metadata**

| <b>Input Field</b>                       | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Script ID                                | A unique name for the script. VR-Forces generates a default name, but you will probably want to change it to something meaningful. It is your responsibility to make sure that it is unique. See "9.2.1 Specifying the Script ID" on page 198 for details.                                                                                                                                                      |
| Script Type                              | <p>One of:</p> <ul style="list-style-type: none"> <li>• Task.</li> <li>• Reactive Task.</li> <li>• Background Process.</li> <li>• Set.</li> <li>• Task (Behavior).</li> </ul> <p>The fields on the New Script dialog box vary based on which type of task is being created. (Behavior tasks involve advanced development; see "Chapter 11. Writing Advanced Scripts with the Behavior Engine" on page 259.)</p> |
| Language                                 | Specify Lua if you are writing a Lua script. Specify C++ if you have written a new task controller and want to use the script interface to create the GUI portion of the task. (Does not apply to reactive tasks and background processes.) For more information, see "9.2.11 Specifying the Programming Language for a Script" on page 216.                                                                    |
| Name                                     | The text that will be added to the Task or Set menu for this script.                                                                                                                                                                                                                                                                                                                                            |
| Extended Name                            | An alternative name to use on context-insensitive task or set lists if more than one task or set has the same name. For details about extended names, see "9.2.3 Specifying an Extended Name" on page 199.                                                                                                                                                                                                      |
| Short Description                        | The text to use for this script in a Plan window.                                                                                                                                                                                                                                                                                                                                                               |
| Description                              | A description of what the script does. This text is displayed on the dialog box. Providing a description is optional. Also, optionally you can choose to include the description on the task or set dialog box.                                                                                                                                                                                                 |
| Version                                  | Script version. Available as a parameter to a Lua script. For use by script writers who want to have different versions of a script.                                                                                                                                                                                                                                                                            |
| Action Categories                        | For tasks, specifies the categories of tasks that this scripted task will interrupt. (In other words, action categories manage task concurrency. For details about concurrent tasks, see "2.1.4 Concurrent Task Execution" on page 9.) (Does not apply to background processes or sets.)                                                                                                                        |
| Can Be Executed More Than Once at a Time | Allows an entity to manage more than one instance of this task at a time. For details, see "9.2.12 Configuring Scripted Tasks to Run Concurrently" on page 217.                                                                                                                                                                                                                                                 |



**Table 7. Script metadata (continued)**

| <b>Input Field</b>                 | <b>Description</b>                                                                                                                                                                                                                                                     |
|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Tool Tip                           | Text for a tooltip when the script is selected on a menu. (Optional) (Does not apply to reactive tasks and background processes.)                                                                                                                                      |
| Menu Icon                          | The icon to display next to the name on a menu. See "9.2.2 Specifying a Menu Icon" on page 199 for details. (Optional) (Does not apply to reactive tasks and background processes.)                                                                                    |
| Parameters                         | The input parameters that the task or set requires. See "9.2.6 Specifying the Parameters for a Script" on page 200 for details. (Optional) (Does not apply to background processes.)                                                                                   |
| Command Level                      | For tasks and sets, you can specify at what level it displays in the Command Panel. Primary is for the most common ones, Secondary is for advanced, and Tertiary is for the least common. See "2.1.1 The Command Panel" on page 6 to learn how these options are used. |
| Magic Command                      | For tasks and sets, you can specify whether it is a magic command. Magic commands are options that are not available in reality, such as the Invulnerable and Undetectable set data requests.                                                                          |
| Show on Top Level Right Click Menu | Select to show the task or set on the top level of the shortcut menu. This setting can be in addition to showing the script on the Task or Set menu.                                                                                                                   |
| One Click Support                  | Select to allow the task or set to be assigned with a single click. You must assign the keyboard or mouse controls in the Key Mapping Editor as described in "Creating and Editing Key Mappings".                                                                      |
| Show Task on Task Menu             | Select to show the scripted task on the Task menu. Clear to hide the task. (You might want to write a script that will be used by other scripts, but which you do not want a user to assign directly to a simulation object.) (For task scripts only.)                 |
| Show Set on Set Menu               | Select to show the scripted set data request on the Set menu. Clear to hide the set. (You might want to write a script that will be used by other scripts, but which you do not want a user to assign directly to a simulation object.) (For set scripts only.)        |
| Menu Location                      | The location on the Task or Set menu where this script will be listed. See "9.2.7 Specifying a Menu Location" on page 210 for details. (Does not apply to reactive tasks and background processes.)                                                                    |
| Show Task on Task Toolbar          | Select to add this task to the Task Toolbar. The toolbar displays the name of the script. If you want to display an icon, select a Task menu icon for this task, as described in "9.2.2 Specifying a Menu Icon" on page 199. (Tasks only.)                             |
| Show Set on Set Toolbar            | Select to add this set data request to the Set Toolbar. The toolbar displays the name of the script. If you want to display an icon, select a menu icon for this task, as described in "9.2.2 Specifying a Menu Icon" on page 199. (Sets only.)                        |

**Table 7. Script metadata (continued)**

| <b>Input Field</b>                    | <b>Description</b>                                                                                                                                                                                                                                                                                             |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Make Available Based on Entity Type   | If selected, the script is available to the simulation object types in the Valid for Entity Types list.                                                                                                                                                                                                        |
| Valid for Object Types                | The simulation object types that can use this script. Editable only if Make Available Based on Entity Type is selected. This entry affects the context sensitivity of the Task or Set menu. See "9.2.9 Configuring a Script's Availability to a Simulation Object Type" on page 211 for details. Default: All. |
| Make Available Based on Behavior Sets | If selected, the script is available to the simulation object only if the selected behavior set is assigned to its force.                                                                                                                                                                                      |
| Behavior Sets                         | Select each behavior set that you want this script to be part of. You can also assign scripts to behavior sets in the Edit Behavior Sets dialog box. For details about behavior sets, see "2.6. Using Behavior Sets to Manage Scripts" on page 23.                                                             |
| Make Available to Global Plans        | If selected, the script is available on the Task or Set menu in global plans.                                                                                                                                                                                                                                  |
| Toolbar Location                      | The location on a toolbar for this script. You can assign more than one script to a toolbar button. For details, see "9.2.10 Specifying Equivalent Scripts for a Toolbar Button" on page 215.                                                                                                                  |
| Enabled by Default                    | Reactive tasks only. If selected, the reactive task is automatically enabled for all supported simulation object types.                                                                                                                                                                                        |
| Default Priority                      | Reactive tasks only. The priority for this task. For information about the effect of reactive task priority, see "2.5. Reactive Tasks" on page 19.                                                                                                                                                             |
| Compare Priorities                    | Reactive tasks only. Displays a list of the reactive tasks that are valid for this task's simulation object types and action categories. This helps you decide what priority you want to give to the reactive task you are creating or editing.                                                                |

### 9.2.1 Specifying the Script ID

Each script must have a unique ID. VR-Forces generates a default ID, but it does not contain meaningful text. Therefore, you may want to change the ID.



The script ID is used internally by VR-Forces to manage how a script is executed. If a script is used in a plan, it is referenced by its script ID. If you change the ID, you must ensure that it is unique.

► **To specify the script ID:**

1. In the New Script (or Edit Script) dialog box, click the button to the right of the Script ID. The New Script ID dialog box opens.
2. Type the ID you want to use. Internally, VR-Forces converts all script IDs to lower case. Therefore it is recommended that you use lower case when creating a script ID.
3. Click OK.

**Editing a Script ID**

Editing script IDs is not recommended because it can invalidate scripts that reference the changed script. However, you can change a script ID if you want to.

If you change the script ID for a scenario script, you are responsible for ensuring that scripts that reference it are changed accordingly.

If you change the script ID for a system script, VR-Forces generates an alias so that scripts that reference the old script ID will continue to work.

► **To edit a script ID:**

1. Open the script whose ID you want to change.
2. Click the button to the right of the Script ID. The New Script ID dialog box opens.
3. Type the ID you want to use. Internally, VR-Forces converts all script IDs to lower case. Therefore it is recommended that you use lower case when creating a script ID.
4. Click OK.

## 9.2.2 Specifying a Menu Icon

If you want to put an icon on the Task or Set menu next to the menu text, you can specify one. (Does not apply to reactive tasks and background tasks.)

► **To specify a menu icon:**

1. Click the Add button () next to the Menu Icon label. The Select Menu Icon dialog box opens.
2. Browse the directory structure and select the icon that you want to use.
3. Click Open.

- To remove a menu icon for a script, click the Remove button () next to the Menu Icon label.

## 9.2.3 Specifying an Extended Name

Sometimes a script needs to be implemented differently for different types of simulation objects. To accomplish this you can write different scripts for each type of simulation object, but give the task the same name. The scripts are distinguished by their script IDs and VR-Forces picks the correct script using context sensitivity. For example, you might have two scripts with the IDs `Move_To_Waypoint_Entity` and `Move_To_Waypoint_Unit`.

Both would have the name Move to Waypoint. When you wanted to give a Move to Waypoint task, you would select the simulation object and choose Move to Waypoint from the Task menu. VR-Forces would pick the correct version of the task.

The one problem with this system is that in cases where VR-Forces does not enforce context-sensitivity of task or set lists, you would end up with two tasks called Move to Waypoint and no way to know which was the correct one to use for the simulation object that you wanted to assign the task to. The extended name solves this problem. If you specify an extended name for a script, it is used to distinguish between scripts that use the same task or set name.

### 9.2.4 Specifying a Short Description

When you add a scripted task or set data request to a plan, the entry in the Plan window can be lengthy and cryptic. It often includes information, such as location values that are not easily understood or put to use by a reader. The Short Description field for a script lets you specify a short, meaningful description to be used in the Plan window.

The description can include variable information. To include variable data, use the syntax *\$variable\_name*, where *variable\_name* is one of the variables for the script. For example, the default listing for the Move to Waypoint (Plan Along Roads) task is:

```
move-to-waypoint-path-plan: destination=waypoint1
```

If you specify a short description of Move to \$destination (Plan Path), the listing in the Plan window would be:

```
Move to Waypoint1 (Plan Path)
```

In this case, the value of the destination parameter replaces the text \$destination in the description.

### 9.2.5 Specifying a Description

You can include a long description of the script. You can do this solely for the benefit of someone examining the metadata in the Edit Script dialog box or you can include the text of the description in the dialog box that is generated for the scripted task or set. This is a way of providing context-sensitive help to the person who must specify the parameter values for the task or set.

► **To specify a task or set description:**

1. In the Description box, type a description of the task or set.
2. If you want the description to appear on the generated dialog box, select the Description check box.

### 9.2.6 Specifying the Parameters for a Script

Scripted tasks and sets can take input parameters. When you specify parameters for a script, VR-Forces automatically builds a dialog box where you can enter the parameter values.

#### Script Parameter Types

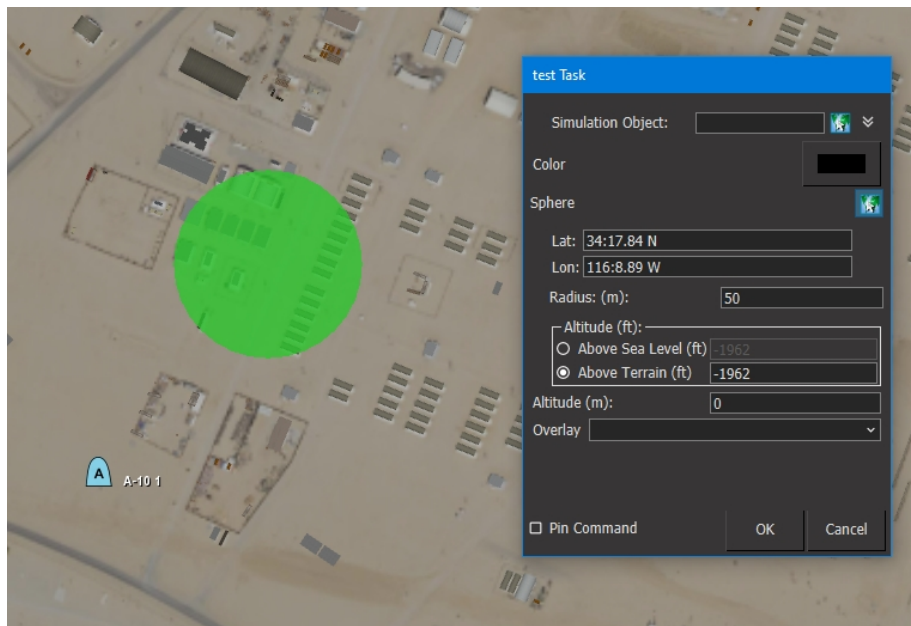
The parameter types that you can specify for a script are:

- Active Sonar Mode. Allows you to select the sonar mode to use.
- Aerodrome. Allows you to select an aerodrome.
- Aggregate Formation. Provides a list of formations for units.
- Air Speed. Allows you to specify the speed for an aircraft.
- Altitude. Allows you to specify the altitude.
- Altitude Label. Adds an altitude label independently of the altitude parameter.
- Altitude Rate Change. Allows you to specify the rate at which an aircraft changes altitude.
- Angle. Allows you to specify an angle.
- Animated Movement. Lets you assign an animated movement to an entity.
- Bomb Resource. Creates a list of bomb resources.
- Boolean. Creates a check box.
- Choice (Drop-Down List). Specifies choices presented in a drop-down list.
- Choice (Image Drop-Down List). Lets you specify a set of images to choose instead of text.
- Choice (Multiple Option Buttons). Specifies choices presented as a group of option (radio) buttons.
- Choice (Single Option Button). Specifies a choice presented as a single option (radio) buttons. Single option buttons can be associated through a button group, but placed at different locations on the dialog box.
- Color. Allows you to select a color.
- Country. Allows you to select a country.
- DI-Guy Animation. Allows you to select a DI-Guy animation.
- DI-Guy Appearance. Allows you to select a DI-Guy appearance.
- DI-Guy Gesture. Allows you to select a DI-Guy gesture.
- Date/Time. Specifies the date and time.
- Depth MSL. Specifies the depth below mean sea level.
- Direct Fire Weapon. Lets you select a direct fire weapon.
- Distance (Large). Specifies a large distance. This uses the Application Settings level's Distance measurement from the Display Units Settings.
- Distance (Small). Specifies a small distance. This uses the Application Settings level's Distance (Small) measurement from the Display Units Settings.
- Embarkation Slot. Allows you to select a slot an entity can embark on.
- Embedded Entity. Allows you to select an embedded entity to deploy.
- Filename. Allows you to specify a file name.
- Force. Specifies the force.
- Heading. Heading, in degrees.

- **Heading Label.** Adds a heading label independently of the heading parameter.
- **Hex.** Allows you to enter a hexadecimal.
- **Indirect Fire Weapon.** Specifies an indirect fire weapon.
- **Integer.** A whole number.
- **Integer Map.** Maps a keyword to an integer.
- **Jammer Selector.** Allows you to select a jammer for an entity. The jammer types list is built automatically based on the entity type.
- **Jamming Mode.** Lets you select a jamming mode. The jamming mode list is built automatically based on the entity type. If you add a Jamming Mode parameter, VR-Forces automatically adds an Emitter ID parameter to the dialog box.
- **Loadout.** Specify a loadout for an aircraft unit. (Aggregate-level only.)
- **Location (with altitude).** Vector description of a location, in geocentric, including altitude.
- **Location (without altitude).** Vector description of a location, in geocentric, altitude not included.
- **Location Reference.** Allows you to specify a location using either coordinates or a waypoint.
- **Munition Resource.** Creates a list of munitions.
- **Name Tag.** Specifies a name tag for a DI-Guy soldier.
- **Naval Mine Resource.** Specifies a naval mine resource.
- **New Tactical Graphic.** Allows you to define a tactical graphic.
- **Object Type.** Allows you to select one type of object.
- **Object Types.** Allows you to select multiple object types.
- **Observer Mode.** Specifies an observer mode, for example, Stealth, Plan View, or XR.
- **Observer Saved View.** Specifies a saved view.
- **Octal.** Allows you to enter an octal code.
- **Offset Location.** Specifies an offset to the side, behind, and above, as in the Follow Entity task. The default is x=10;y=20;z=30.
- **Overlay Parent.** Allows you to select an overlay.
- **Pitch.** Allows you to specify the pitch for a simulation object.
- **Radar Mode.** Allows you to specify a radar mode. The radar mode list is built automatically based on the entity type. If you add a Radar Mode parameter, VR-Forces automatically adds an Emitter ID parameter to the dialog box.
- **Radar Selector.** Allows you to select a type of radar. The radar list is built automatically based on the entity type.
- **Range Label.** Adds a range label.
- **Range Label (Small).**
- **Rank patch.** Specifies a rank patch for a DI-Guy soldier.

- Rate Label. Adds a rate label.
- Real. A floating point decimal number.
- Real Map. Maps a keyword to a real number.
- Resource. Creates a list of available resources.
- Runway Selector. Allows you to select an aerodrome's runway.
- SIS Pick Widget. Specifies a SISO document.
- Separator. Allows you to put a separator in the generated dialog box. This is just for formatting purposes.
- Simulation Object (Single). Allows you to select one simulation object in a list window.
- Simulation Objects (Multiple). Allows you to select multiple simulation objects in a list window.
- Speed. Any kind of rate, such as speed, climb rate, and so on.
- Sphere. Allows you to create a sphere for defining an area on the terrain (Figure 45).

**Figure 45. A sphere created as part of a scripted task**



- String. An alphanumeric string.
- String Map. Maps a keyword to a string.
- Subordinate Functions. Assign a subordinate function.
- Subordinates. Allows you to select one or more of a unit's subordinates.
- Time. Time in days:hours:minutes:seconds.
- Topographic Orientation. Specifies orientation in topographic coordinates.

- Turn Rate. Turn rate in radians/second.
- Turn Rate Label. Adds a turn rate label independently of the turn rate parameter.
- Unit Patches. Specifies a unit patch for a DI-Guy soldier.
- Visual Meta Data. Allows you to select markings for objects that support settings marking, such as a flag or license plate.
- Weapon. Specifies a weapon.

### Specifying Script Parameters

This section explains how to add parameters using the VR-Forces script creation interface and automatic dialog box creation. You can also add parameters using Qt Designer. For details, see "12.3. Adding Parameters When Using Qt Designer" on page 298.

**i** If a script does not have any input parameters, it executes as soon as it is assigned to a simulation object, like the Wait command does.

#### ► To specify a parameter for a script:

1. Click the Add button (+) above the Parameters list. The Add Parameter dialog box opens (Figure 46).

**Figure 46. Add Parameter dialog box**

2. In the Parameter Name box specify a name for the parameter. This is the name used in scripts to get the parameter value.
3. Select the type of parameter from the Type list. If appropriate, the dialog box redisplayes to request the details required for the parameter type you selected.
4. Specify the details for the parameter. Table 8 explains how to fill out the properties for the various parameter types.



Table 8. Scripted task parameter properties

| Parameter property             | Description                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parameter Name                 | The name to be used by the Lua script for this parameter. For example, if you give a parameter the name <b>taskDestination</b> , in the Lua code for the script, the parameter will be accessed as <b>taskParameters.taskDestination</b> .                                                                                                                                   |
| Label                          | This is the text used on the dialog box that is generated for the task. If you leave this field blank, the parameter name is used.                                                                                                                                                                                                                                           |
| Button Group                   | For Choice (Single Option Button) parameters, specifies the button group that the option buttons are a part of.                                                                                                                                                                                                                                                              |
| Tool Tip                       | Text for a tool tip. (Optional)                                                                                                                                                                                                                                                                                                                                              |
| Indent Level                   | The level of indentation for this parameter. The indentation level is multiplied by the Indent setting (in pixels) on the Application Settings dialog box, Script Settings page. This allows you to apply some formatting to the dialog box. For details about setting the indentation distance, see "Setting the Indentation Distance for Script Dialog Boxes" on page 209. |
| Required                       | For Simulation Object lists only, specifies that this parameter must be set before the OK button is enabled. If the check box is cleared, the requirement is not in effect.                                                                                                                                                                                                  |
| Visible check box              | Specifies that the parameter be included on the script dialog box. (You might want a parameter to be available in the code, but not have it set by a user when the script is assigned.)                                                                                                                                                                                      |
| Default value or state         | Many, but not all, parameters have an optional default value.                                                                                                                                                                                                                                                                                                                |
| Initialize from State Property | This is available for some parameters. Select the check box if you want to indicate that it is associated with a state property. If you do, you can enter the state property name. In the scripted task dialog, users will see the values for that state properties.                                                                                                         |
| Range Bottom                   | The lowest acceptable value for this parameter, if applicable.                                                                                                                                                                                                                                                                                                               |
| Range Top                      | The highest acceptable value for this parameter, if applicable.                                                                                                                                                                                                                                                                                                              |

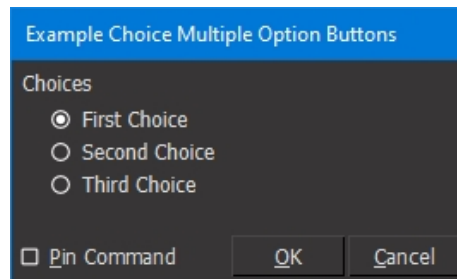
**Table 8. Scripted task parameter properties (continued)**

| Parameter property                  | Description                                                                                                                                                                                                                    |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Default value for boolean parameter | The Boolean parameter type lets you create a check box. You can specify that the default be selected or cleared.                                                                                                               |
| Affects Controls                    | If selected, some fields are populated with the actual simulation object state values. For details about the Affects Controls option for the simulation object parameter, see "Using the Affects Controls Option" on page 208. |
| Choice                              | Adds choice option buttons. For details about adding choices to a Choice parameter, see "Specifying Choices for the Choice Parameters" on the facing page.                                                                     |
| Filter category                     | Lets you add categories for filtering object lists. For details about adding categories to a filtered list, see "Specifying Categories for a Simulation Object" on page 208.                                                   |

5. Click Add.

### Choosing an Option Button Type

Scripted tasks can use two types of option buttons – multiple or single. The multiple option button choice is a typical option list, as illustrated in (Figure 47). In a multiple option list, the choices are mutually exclusive and are automatically associated with each other. You cannot enable or disable other input parameters based on the selection of one of these options.

**Figure 47. Choice (Multiple Option Buttons)**

A single button option puts one option button on the dialog box. You can create multiple single button options and place them on the dialog box wherever you want. This allows you to associate other parameter inputs with the option button using the dialog box layout (Figure 48). To ensure that these options are mutually exclusive, you associate them by specifying a button group.

**Figure 48. Choice (Single Option Button)**

Heading (deg): 360 True

Speed (kts): 583

Altitude (ft): 16404

Turn Direction: Right

Pattern Type

☒ Time

Time per Leg: 0 00:01:00

☐ Distance

Length (NM): 6

Width (NM): 1

☐ Pin Command OK Cancel

You can create multiple sets of options and multiple buttons groups if you need to.

Although you can lay out single option buttons and associated parameters using the default GUI builder for scripts, to associate parameters with the options, you must use Qt Designer. For details about laying out dialog boxes using Qt Designer, see "Chapter 12. Using Qt Designer for Script Dialog Boxes" on page 297.

### Specifying Choices for the Choice Parameters

The Choice (Drop-Down List), Choice (Image Drop-Down List), and Choice (Multiple Option Buttons) parameters require a list of choices, which are added to the dialog box as option buttons or as a list. For the image drop down list, you specify images. For the other two, you specify text. You can add choices, edit them, and delete them.

#### ► To add a text choice:

1. Click the Add Parameter button (+) on the Choice Box Values line. The New Choice dialog box opens.
2. Type the text for the choice.
3. Click OK.
4. Repeat this procedure for each choice that you want to have available. (The last choice added becomes the default choice.)

#### ► To add an image choice:

1. Click the Add Parameter button (+) on the Choice Box Images line. The Select Image dialog box opens.
2. Type the path for an image or click the Edit button and select an image in the file browser. The image displays in the dialog box.

3. Click OK. The image is added to the list next to Choice Box Images.
4. Repeat this procedure for each image that you want to have available. (The last choice added becomes the default choice.)

### **Specifying Categories for a Simulation Object**

The Simulation Object parameter lets you build a list of categories on which to filter the display of simulation object types in a dialog box.

#### **► To add a category to the simulation object list:**

1. Click the Add Parameter button () on the Filters line. The Choose Filter dialog box opens.
2. Select a filter from the Filters list.
3. Click OK.
4. Repeat the procedure to add as many filters as you want.

### ***Editing the Category List for a Simulation Object Parameter***

You can edit the list of filter categories for a Simulation Object parameter. The edit operation is essentially a replacement of the current category with a new one. You can achieve the same effect by deleting the current category and adding a new one.

#### **► To edit the list of filter categories in a Simulation Object parameter:**

1. Select the parameter you want to edit in the Parameters list on the New Scripted Task dialog box.
2. Click the Edit button (). The Script Variable dialog box opens.
3. In the Filters list, select the filter that you want to edit.
4. Click the Edit button (). The Choose Filter dialog box opens.
5. Select the filter category that you want to use.
6. Click OK. The original category is removed and the new one is added.

#### **► To delete a category from a Simulation Object parameter:**

1. Select the parameter you want to edit in the Parameters list on the New Scripted Task dialog box.
2. Click the Edit button (). The Script Variable dialog box opens.
3. In the Filters list, select the filter that you want to delete.
4. Click the Delete button (). The category is removed from the list.

### **Using the Affects Controls Option**

The Simulation Object (Single) and Simulation Objects (Multiple) parameters have a check box labeled Affects Controls. If you select this check box, when you select a simulation object in the task dialog box, the values for the following parameter types update to the correct values for the selected simulation object:

- DI-Guy Animation
- DI-Guy Appearance
- Embedded Entity
- Emitter
- Force
- Formation
- Location
- Overlay
- Weapon

If you use the Simulation Objects (Multiple) parameter, it only affects controls if you select just one simulation object.

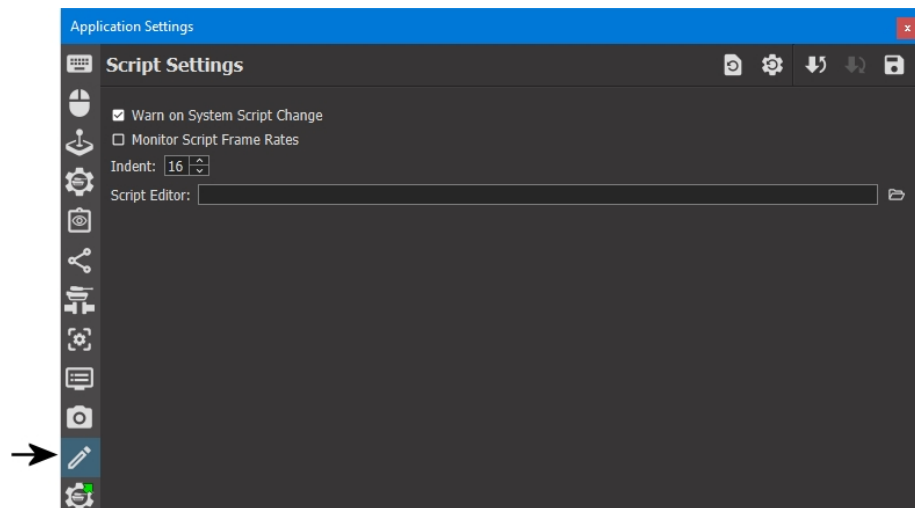
### Setting the Indentation Distance for Script Dialog Boxes

When you add parameters to a script dialog box, you can specify an indent level. The indent level is the level of indentation, not the distance of indentation. The distance that the parameter is indented on the dialog box is the indent level multiplied by the Indent value (in the Application Settings dialog box, Script Settings page).

► **To set the indentation distance for script dialog boxes:**

1. Choose **Settings > Application**. The Application Settings dialog box opens.
2. Select the Script Settings page (Figure 49).

**Figure 49. Script Settings page**



3. Change the value (in pixels) in the Indent box.
4. Click Close.

**i** You can also open the Script Settings page from the Scripts menu in the Scripts dialog box.

### 9.2.7 Specifying a Menu Location

Scripted tasks are added to the Task menu unless you specify that they not be shown (by clearing the Show Task on Task Menu check box). Scripted set data requests are added to the Set menu. The default location is the top level of the menu. However, you will probably want to organize your tasks and sets using the submenus provided by VR-Forces or in newly created submenus. The New Script dialog box displays the menu under which the script is placed. It does not show the full path from the top level menu.

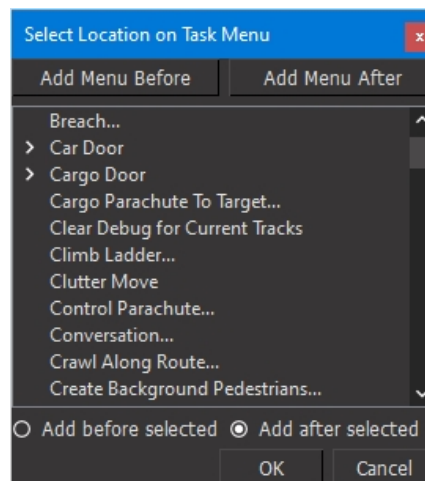
You can also add a scripted task or set to the top level shortcut menu. The script can be listed in both places.

- When you add a script to a menu, it does not appear on the menu until you reload the scenario.
- By default, VR-Forces alphabetizes all Task and Set menu commands. Therefore even if you specify a specific location on a menu, the task or set will be listed in alphabetical order unless you disable alphabetization. For information about alphabetizing Task and Set menu commands, see "Alphabetizing the Task and Set Menus".

► **To specify a menu location for a script:**

1. On the Menu Location line, click the Edit button (✎). For task scripts, the Select Location on Task Menu dialog box opens (Figure 43 on page 194). For set scripts, the Select Location on Set Menu dialog box opens. The two dialog boxes are functionally identical.

**Figure 50. Select Location on Task Menu dialog box**



2. To add the script to an existing submenu:
  - a. Expand the submenu on which you want to put the script.
  - b. Select the menu item that you want the new script to be next to.
  - c. To put the new script above the selected menu item, select the Add Before Selected option. To put the new script below the selected menu item, select the Add After Selected option.
  - d. Click OK.
3. To create a new submenu for the script:
  - a. Select the submenu next to where you want to add the new submenu.
  - b. To add the new menu before the selected submenu, click Add Menu Before. To add the new menu after the selected submenu, click Add Menu After. The New Menu Name dialog box opens.
  - c. Type a name for the new submenu.
  - d. Click OK.
  - e. Add the script to the new submenu using the procedure in step 2.

### 9.2.8 Specifying Action Categories

Some VR-Forces tasks can run at the same time as other tasks, while some are mutually exclusive. This is called task concurrency. (For details about concurrent tasks, see "2.1.4 Concurrent Task Execution" on page 9.) The C++ tasks provided with VR-Forces are organized in groups to help manage concurrency. They are configured in `./data/simulationModelSets/<model_set>/vrfSim/taskRules/default-task-rules.tsk`.

In the New Script dialog box, these groups are represented as action categories. The action categories displayed in the New Script dialog box are configured in `./data/simulationModelSets/<model_set>/vrfSim/taskRules/actionCategories.tsk`. (For details about configuring action categories, see "2.11.1 Configuring Action Categories" on page 36.)

When you create or edit a scripted task, you can specify which groups of tasks the new task can run concurrently with and which it will interrupt. By default, the Movement category is selected, which means that if you do not change the setting, the new task will interrupt movement tasks.

- To specify which tasks a scripted task will interrupt, in the Action Categories list, select the categories to interrupt. For example, if the new task involves movement, you probably want it to interrupt other movement tasks, so you would select the Movement category check box. If it is a new type of message task, you probably want to let it run concurrently with movement tasks, so you would clear the Movement category check box.

### 9.2.9 Configuring a Script's Availability to a Simulation Object Type

Scripts can be available to a simulation object based on these criteria:

- The script is available for certain simulation object types. For details, see "Specifying the Valid Simulation Object Types for a Script" below.
- The script is part of a behavior set. For details, see "Specifying Availability by Behavior Set" on the facing page.
- A task or set is specified in a system definition. (For example, the Naval Mine Sweep task is available to entities with a Naval Mine Sweep system.) This state is not shown in the Edit Script dialog box. For details, see "Specifying a Scripted Task or Set in a System Definition" on the facing page.

The default setting for new scripts is that they are available to all simulation objects.

Availability by simulation object type and by behavior set is configured in the New Script and Edit Script dialog boxes. Table 9 shows the interaction of these settings.

**Table 9. Scripted task availability**

| Available by Entity Type            | Available by Behavior Set           | Availability                                                                                                                                                 |
|-------------------------------------|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | The script is available only to those simulation object types listed.                                                                                        |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | The script is available to all simulation objects, but only if one of the selected behavior sets is applied to the simulation object's force.                |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | The script is available to the specified simulation object types, but only if one of the selected behavior sets is applied to the simulation object's force. |

### Specifying the Valid Simulation Object Types for a Script

You can restrict a script to specific simulation object types. If you do this, the script will not appear on the Task or Set menu when you select a simulation object for which it is not valid. Just as with built-in VR-Forces tasks and sets, there are cases, such as for global plans, where VR-Forces cannot determine the simulation object context and the script will be available on the menu even though it is not appropriate to the simulation object that you selected. In those cases it is up to you to know that a task or set is valid.

#### ► To specify the valid entity types for a script:

1. Select the Make Available Based on Entity Type check box.
2. Click the Add button (🛎) on the Valid for Entity Types line. The Object Type dialog box opens (Figure 51). You can build an enumeration by selecting options from the enumeration field lists or you can type in the enumeration directly.



Figure 51. Object Type dialog box

3. 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.

4. If appropriate, add additional enumerations for each simulation object type that is valid for the script.

### Specifying Availability by Behavior Set

You can restrict a script to specific behavior sets. If you do this, the task will not appear on a menu when you select a simulation object whose force does not have the behavior set assigned to it.

#### ► To specify that a script is available by behavior set:

1. Select the Make Available Based on Behavior Set check box.
2. Select each behavior set for which you want this script to be available.

**i** You must create the behavior sets before you can assign scripts to them. All available behavior sets are automatically listed in the dialog box window.

### Specifying a Scripted Task or Set in a System Definition

Some scripted tasks and sets are available to simulation objects because they are specified in a system that the simulation object uses. They are not assigned to simulation object types as part of the script metadata.

**i** This method is used for many of the system scripted tasks supplied with VR-Forces.

Scripted tasks and sets are specified in a system definition by their script IDs in the **script-ids** parameter. (In aggregate-level SMSs, some scripted tasks for combat engineering objects are specified in the **create-script-id**, **improve-script-id**, and **destroy-script-id** parameters.) Script IDs can be specified in a script-enable-controller and in controllers derived from it.

The following example is from *mech-aggregate-movement.sysdef* in *AggregateLevelBase.sms*. Any simulation object with this system definition can use the `Maximum_Speed_Modifier`, `Halt_Movement`, `Move_To_Location_Plan_Path`, and `Move_To_Waypoint_Plan_Path` scripted tasks. The variables described in the script-variables block are accessible to the script through the `vrf:getScriptAttribute()` function.

```
(script-enable-controller
  (component-descriptor-type "script-enable-controller-descriptor")
  ...
  (script-ids "Maximum_Speed_Modifier" "Halt_Movement"
    "Move_To_Location_Plan_Path" "Move_To_Waypoint_Plan_Path"
    "halt_movement_before_obstacles" "Flee_when_Routed")
  (script-variables
    (maximum-speed-modifier-variables
      (script-id "Maximum_Speed_Modifier")
      (variables
        (DtRWReal Rain-Modifier-By-Intensity
          $rain-modifier-by-intensity)
        (DtRWReal Snow-Modifier-By-Intensity
          $snow-modifier-by-intensity)
        (DtRWReal Visibility-Degrades-When-Below
          $visibility-degrades-when-below)
        (DtRWReal Visibility-Can-Degrade-Speed-Up-To
          $visibility-can-degrade-speed-up-to)
      )
    )
    (move-to-location-variables
      (script-id "Move_To_Location_Plan_Path")
      (variables
        (DtRWString Obstacle-Feature-Query "MAK_OBSTACLE")
        (DtRWString Path-Feature-Query "MAK_ROAD")
      )
    )
    ...
  )
)
```

The following example is from the *aggregate-5-inch-gun.sysdef* in *AggregateBaseLevel.sms*. It adds the **script-ids** parameter to the **aggregate-indirect-fire-controller**, which is derived from the **script-enable-controller**. Any simulation object with this system can use the Indirect Fire and Limited Munition Attack scripted tasks.

```
(weapon-indirect-fire
  (systems )
  (sensors )
  (controllers
    (aggregate-indirect-fire-controller
      (component-descriptor-type "aggregate-indirect-fire-
        controller-descriptor")
      (component-type "aggregate-indirect-fire-controller")
      (min-tick-period -1.000000)
      (min-tick-period-variance -1.000000)
```

```

        (process-state-repository-name "aggregate-indirect-fire-
controller-process-state-repository-default")
        (process-state-repository-type "aggregate-indirect-fire-
controller-process-state-repository")
        (is-enabled True)
        (script-ids "Indirect_Fire" "Limited_Munition_Attack")
        (weapon-name "5 inch shell")
        ...
    )
)

```

The following example is from *obstacle-builder.sysdef* and shows the **create-script-id** and **improve-script-id** parameters:

```

(creatable-objects
  (affectable-object
    (object-type 1 (16 1 0 62 3 0 0))
    (time-to-complete $time-to-create-abatis)
    (range $creation-range-abatis)
    (create-script-id "Construct_Abatis")
    (improve-script-id "Improve_Obstacle")
  )
  ...
)

```

### 9.2.10 Specifying Equivalent Scripts for a Toolbar Button

There may be occasions when you want to have tasks or sets with the same name that work differently depending on the simulation object you give them to. For example you might have a movement task that works one way for an individual entity and another way for a unit. You can ensure that when you select a simulation object it gets the correct version of the script by specifying the simulation object type that the script is available to. To simplify the Task or Set toolbars, you can map these different scripts to the same toolbar button. This is done by specifying that one script is equivalent to another script.

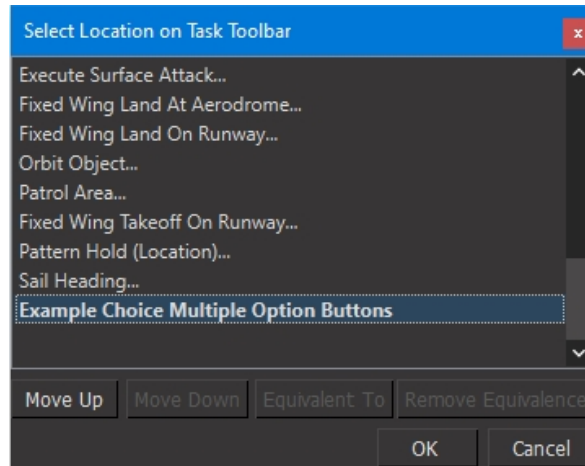
If you map multiple scripts to the same toolbar button, when you select a simulation object, the toolbar button is enabled and you can give the simulation object the task or set. VR-Forces chooses the correct version of the script. If you select multiple simulation objects that would have different versions of the script, the toolbar button becomes disabled.

#### ► To specify an equivalent script for a Task or Set Toolbar button:

1. Create a new script.
2. On the New Script dialog box (or Edit Script dialog box), select Show Task on Task Toolbar (or Show Set on Set Toolbar).
3. Click the Toolbar Location edit button . The Select Location on Task (Set) Toolbar dialog box opens.
4. Select the task or set that you want this script to be equivalent to. The Equivalent To button becomes active.

- Click the Equivalent To button. Text is added to the script's entry in the list indicating the script it is equivalent to (Figure 52).

**Figure 52. Select Location on Task Toolbar dialog box**



**i** Do not move the new script to a different location on the toolbar. Moving it breaks the equivalency.

- Click OK.
- Save the script.

### 9.2.11 Specifying the Programming Language for a Script

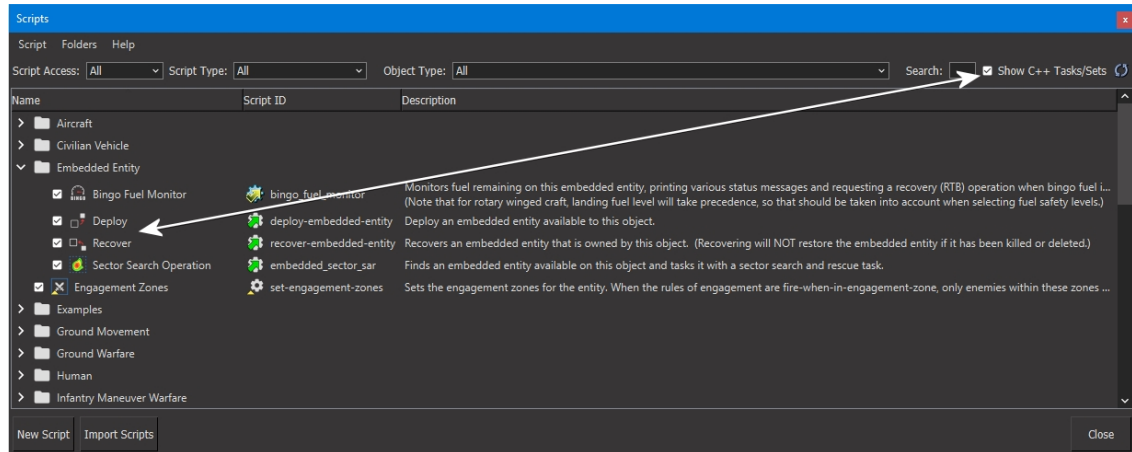
The Language list in the New Script dialog box has two options, Lua and C++. In most cases you will choose Lua and write a script to implement the task or set. However, since the script interface automatically generates dialog boxes and updates menus, it can be a convenient way to add the GUI component for new tasks or sets that developers have written in C++ with the VR-Forces Toolkit. Developers who take this approach do not have to rebuild the VR-Forces GUI or write a plug-in to add their new task or set to the menu and code a dialog box.

**i** The Language option only applies to Task and Set type scripted tasks. Reactive tasks and background processes are always written in Lua.

If a developer adds a new C++ task controller using this approach, it looks for the script ID of the scripted task. For example, the Deploy and Recover tasks for embedded entities use the scripted task interface to create their dialog boxes. The addTask example in the VR-Forces Toolkit demonstrates this method of adding tasks. For more information, please see the *VR-Forces Developers Guide*.

You can display C++ tasks and sets in the Scripts dialog box by selecting the Show C++ Tasks check box (Figure 53).

Figure 53. C++ tasks



### 9.2.12 Configuring Scripted Tasks to Run Concurrently

You can specify that an entity be able to run more than one instance of a task at the same time. The concurrent task feature is intended for tasks that require an entity to manage two non-conflicting activities, such as when an air base launches a flight in an aggregate-level scenario. If an air base receives two launch flight tasks in close succession and concurrent tasks are allowed, both tasks would complete. If this option is not in effect, the second task would cause the air base to stop executing the first instance of the task.

**i** This setting overrides the action category settings. It is your responsibility to make sure that you do not allow concurrent task execution for tasks that would not make sense to run concurrently.

- To specify that a task can be run concurrently, select the Can Be Executed More Than Once at a Time check box on the Edit Scripts dialog box.

### 9.2.13 Creating Scripted Sets

The metadata for a scripted set is identical to that of a scripted task except that a scripted set does not have action categories.

► **To create a scripted set:**

1. Choose **Simulation > Scripts**. The Scripts dialog box opens (Figure 43 on page 194).
2. If you have a folder hierarchy and want to create the script in a particular folder, select the folder. (For more information, see "9.6. Organizing Scripts into Folders" on page 223.)
3. Choose **Script > New Script** or click the New Script button. The New Script dialog box opens (Figure 44 on page 195).
4. In the Script Type list, select Set. The New Script dialog box redisplay to show the metadata fields for a scripted set (Figure 54 on the next page).

Figure 54. New Script dialog box, scripted set

Script ID: <auto generate>

Script Type: Set

Language: Lua

Name:

Extended Name:

Short Description:

☐ Description:

☐ Use Qt Designer

Parameters

| Parameter | Type | Visible |
|-----------|------|---------|
|           |      |         |

Version: 1

Tool Tip:

Menu Icon:

Command Level: Primary

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

☒ Make Available Based on Entity Type

Valid for Object Types

All

☐ Make Available Based on Behavior Set

☐ Make Available to Global Plans

Edit Script Preview Add Cancel

5. Complete the fields in the dialog box. Fields that are common to all scripts are described in Table 7 on page 196.

### 9.2.14 Creating Reactive Tasks

The process for creating a reactive task is largely the same as for creating other scripts. This procedure focuses on the differences.

► **To create a reactive task:**

1. Choose **Simulation > Scripts**. The Scripts dialog box opens (Figure 43 on page 194).
2. If you have a folder hierarchy and want to create the script in a particular folder, select the folder. (For more information, see "9.6. Organizing Scripts into Folders" on page 223.)
3. Choose **Script > New Script** or click the New Script button. The New Script dialog box opens (Figure 44 on page 195).
4. In the Script Type list, select Reactive Task. The New Script dialog box redisplay to show the metadata fields for a reactive task (Figure 55).

**Figure 55. New Script dialog box, reactive task**

Script ID: <auto generate>

Script Type: Reactive Task

Language: Lua

☐ Use Qt Designer

Name:

Extended Name:

Short Description:

☐ Description:

Version: 1

Action Categories:

- ☐ Depth control
- ☐ Lifeform Posture
- ☒ Movement
- ☐ Radio Communication
- ☐ Weapon

☒ Enabled by Default

Default Priority: 5

Compare Priorities

| Parameter | Type | Visible |
|-----------|------|---------|
|           |      |         |

☒ Make Available Based on Entity Type

Valid for Object Types

All

☐ Make Available Based on Behavior Set

☐ Make Available to Global Plans

Edit Script Preview Add Cancel

- Complete the fields in the dialog box. Fields that are common to all scripts are described in Table 7 on page 196.
- To make this reactive task enabled by default, select the Enabled By Default check box.
- In the Default Priority box, type the priority for this task. The lower the number, the higher the priority. To help you set the priority, you can view a list of all reactive tasks that are valid for the simulation object types of this task so that you can consider their relative importance and set an appropriate priority.
- Optionally, click Compare Priorities to see a list of reactive tasks and their priorities.
- Click Add.

### 9.2.15 Creating Background Processes

The procedure for creating a background process is essentially the same as for other script types. The main difference is that there are fewer fields to complete in the New Script dialog box. For details about writing a background process script, see "10.11. Background Processes" on page 257.

► To create a background process:

1. Choose **Simulation > Scripts**. The Scripts dialog box opens (Figure 43 on page 194).
2. If you have a folder hierarchy and want to create the script in a particular folder, select the folder. (For more information, see "9.6. Organizing Scripts into Folders" on page 223.)
3. Choose **Script > New Script** or click the New Script button. The New Script dialog box opens (Figure 56).
4. In the Script Type list, select Background Process. The New Script dialog box redisplayes to show the metadata fields for a background process (Figure 44 on page 195).
5. Complete the fields in the dialog box. Fields that are common to all scripts are described in Table 7 on page 196.

**Figure 56. New Script dialog box, background process**

The screenshot shows the 'New Script' dialog box with the following fields and options:

- Script ID:** <auto generate>
- Script Type:** Background Process
- Language:** Lua
- Name:** (empty text field)
- Extended Name:** (empty text field)
- Short Description:** (empty text field)
- Description:** (checkbox, unchecked)
- Make Available Based on Entity Type:** (checkbox, checked)
- Valid for Object Types:** (list box containing 'All')
- Make Available Based on Behavior Set:** (checkbox, unchecked)
- Version:** 1
- Buttons:** Edit Script, Add, Cancel

6. Click Add.

## 9.3. Saving Scripts

When you create or edit a script, its listing in the Scripts window is in italics, prefaced by an asterisk. When you save the scenario, or promote a script to a system script, the script is saved and the italics and asterisk are removed.



It is likely that you will test scripts by running a scenario and you will want to edit the script as you find problems. If you create or edit a script while a scenario is running and you decide to rewind the scenario without saving it (which would be normal behavior for rewinding a scenario), you are prompted to preserve the changes to the script after the scenario rewinds. The script is still not saved and you will, therefore, receive the standard "scenario modified" prompt when you start the scenario again. You do not have to save the scenario if the only thing that changed is the script. You can continue to run the scenario and rewind while you are working on the script. Eventually you must save the scenario if you want to save your changes to the script.

Metadata for system scripts is saved to a file in the format

`./data/simulationModelSets/<model_set>/scripts/script_name.xml`. The script name is always saved as lower case.

- To save a scenario script, save the scenario.
- To save a system script, after updating the script in the Edit Script dialog box, click Update.

## 9.4. Editing a Script

You can edit scripts.

 If you edit a system script, the changes affect all scenarios that use the script. You are responsible for ensuring that any changes have the intended results for all scenarios that are affected.

### ► To edit the metadata for a scenario script:

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Select the script that you want to edit.

 In the Script ID column, the icon indicates the type of script: Task () , Reactive Task () , Set () , or Background Process () .

3. Choose **Script > Edit Script Meta Data**, or double-click the script name. The Edit Script dialog box opens. (If you are editing a system script, you are prompted to confirm that you want to edit it.)
4. Edit the metadata as desired.
5. Click Update.

## 9.5. Filtering the List of Scripts

You can filter the tasks listed in the Scripts dialog box by selecting options from the lists below the menu bar:

- Script Access: Show system scripts, scenario scripts, or all scripts.
- Script Type: Show scripted tasks, reactive tasks, sets, background processes, or all scripts.
- Entity Type: Show scripts that apply to the selected simulation object type.

 This filter is based on the Valid for Entity Types list for each script. Some scripts do not have any simulation object types listed. These scripts, such as Drop Naval Depth Charge, become available to simulation objects based on the systems that they have configured. Therefore, this list may be somewhat misleading in terms of the overall applicability of scripts to the different platforms.

- Show C++ Tasks: Toggles the display of tasks and sets that are written in C++, but whose GUIs are created using the script mechanism.

## 9.6. Organizing Scripts into Folders

You can create folders and organize scripts by folder.

### 9.6.1 Adding a Folder

► To add a folder:

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Choose **Folders > Add Folder**. The Folder Name dialog box opens.
3. Type a name for the folder.
4. Click OK.

### 9.6.2 Renaming a Folder

► To rename a folder:

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Select the folder that you want to rename.
3. Choose **Folders > Rename Folder**. The Folder Name dialog box opens.
4. Type a new name for the folder.
5. Click OK.

### 9.6.3 Deleting a Folder

You cannot delete a folder that has scripts in it.

► To delete a folder:

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Select the folder that you want to delete.
3. Choose **Folders > Delete Folder**.

### 9.6.4 Adding Scripts to a Folder

► To add a script to a folder, drag it onto the folder.

### 9.6.5 Removing a Script from a Folder

► To remove a script from a folder, drag it away from the folder.

## 9.7. Exporting and Importing Scripts

Scenario scripts are saved in a scenario's SPT file, not as individual scripts. If you want to make a scenario script available to other scenarios, but you do not want to save it as a system script, you can export the script from the scenario in which you created it and import it into any other scenario. You can export multiple scripts to one file as a script "package".

**i** You cannot add or remove scripts from a script package. To change the contents you would have to create a new package.

If scripts are organized in folders, when you save them the folder structure is also saved.

► **To export scripts:**

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Select the scripts that you want to export. (The window supports typical methods for selecting multiple entries in a list.)
3. Choose **Script > Export Script Package**. The Choose Script File dialog box opens. By default it opens in `./userData/taskScripts`.
4. Type a name for the scripts file.
5. Click Save. The scripts are saved to a file with the extension `.spt`. The file contains the metadata and the script code.

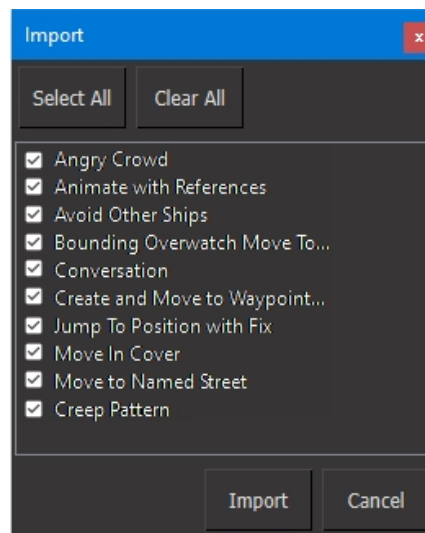
### 9.7.1 Importing a Script Package

If you import a script package that has multiple scripts, you can choose which scripts to import. Scripts are imported with the folder hierarchy that they were saved in.

► **To import scenario scripts:**

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Choose **Script > Import Script Package** or click Import Scripts. The Choose Script File dialog box opens.
3. Select the script file that you want to import.
4. Click Open. The Import dialog box opens (Figure 57). It lists the scripts saved in the script file.

**Figure 57. Import dialog box**



5. Select the scripts that you want to import.
6. Click Import.

## 9.8. Copying a Script

If you want to create a new script that is similar to an existing one, you can copy an existing script and then edit it for its new purpose.

### ► To copy a script:

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Select the script that you want to copy.
3. Choose **Script > Duplicate**. The New Script dialog box opens. If you copied a scenario script, the duplicate script is assigned a new script ID. If you copied a system script, the script ID is not changed. You will have to change the script ID manually.

 If you do not change the script ID for a copied system script, the system script is removed from the list of scripts for this scenario and the new script is added as a scenario script. This lets you experiment with changes to a system script without actually changing it. The original system script will still be available to other scenarios.

4. Edit the script.
5. Click Duplicate. The script is added to the list.

## 9.9. Deleting a Script

If you delete a script that is assigned to a simulation object in a scenario, a task assignment, whether as an independent task or as part of a plan, will fail. If a scripted task is used as a subtask in another scripted task, that scripted task will fail. Scripted sets also fail. VR-Forces does not warn you if either of these cases exist. It is your responsibility to consider the effects of deleting a script.

### ► To delete a script:

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Select the script that you want to delete.
3. Choose **Script > Delete Script**.

## 9.10. Creating a System Script

To create a system script, you create a scenario script and then save it as a system script. System scripts are saved in `./data/simulationModelSets/<model_set>/scripts`. A system script has two files. The metadata is saved in a file ending with the extension `.xml`. The code is saved in a file with the extension `.lua`.

► To save a scenario script as a system script:

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. Select the script that you want to save as a system script.
3. Choose **Script > Promote to System Script**. You are prompted to confirm the change.
4. Click Yes.

9.10.1 Including System Scripts on the Task and Set Menus

Most of the system scripted tasks and sets included with VR-Forces are listed on the Task and Set menus. The scripts in the Examples folder and a few others are not listed. You can specify whether particular scripted tasks or sets are listed. You might want to do this to reduce clutter on the menus. Or you might want to limit the tasks available to players in a particular deployment of VR-Forces.

A system script's inclusion on a menu is controlled by these settings:

- The Show Task on Task Menu (or Show Set on Set Menu) check box on the New Script and Edit Script dialog boxes.
- The check box next to the script's entry on the Scripts dialog box (Figure 58).
- Its membership in a behavior set. The effect of behavior sets on script availability is additive to the configuration discussed in this section. For details about using behavior sets, see "2.6. Using Behavior Sets to Manage Scripts" on page 23.
- Crowd tasks file names must be included in the list of script IDs in *./data/simulationModelSets/EntityLevel/vrfSim/systems/movement/pedestrian-crowd-movement.sysdef*. The script files must be saved in *./data/simulationModelSets/EntityLevel/scripts*.

Figure 58. Show script on menu check box

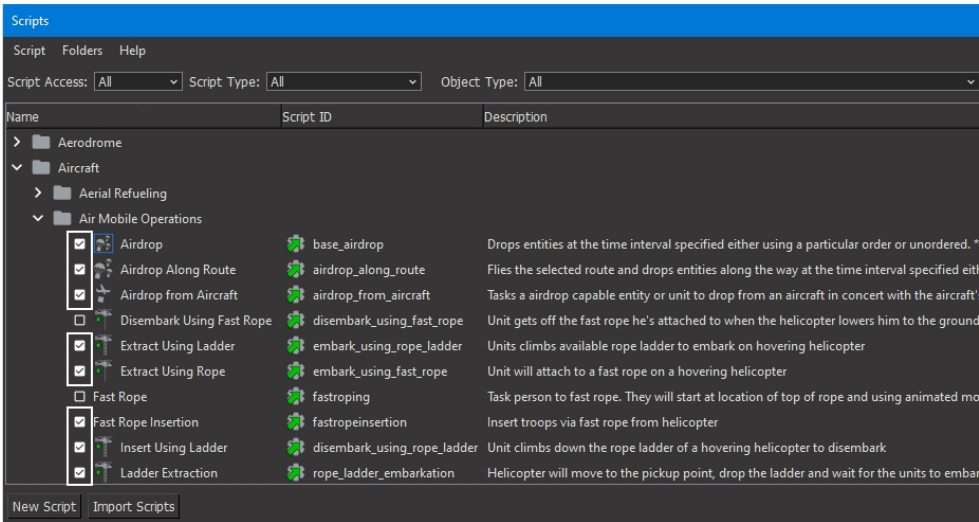


Table 10 shows when a scripted task is listed on a menu based on the configuration check boxes selected.

Table 10. System scripted task inclusion on Task menu

| Show Script on Menu check box       | System script list check box        | Task included on menu |
|-------------------------------------|-------------------------------------|-----------------------|
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | Yes                   |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | No, for this scenario |
| <input type="checkbox"/>            | <input type="checkbox"/>            | No                    |

 You cannot select the check box in the Scripts dialog box if the check box in the New Script or Edit Script dialog box is not selected. If you change the setting of the check box in the Edit Script dialog box, that change is saved in the script when you click Update.

If you unselect the check box in the Scripts dialog box, the task or set is immediately hidden from the menu. If you reload the scenario without saving first, however, it is restored to the menu.

► **To specify the availability of a system script on a menu:**

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43 on page 194).
2. In the list of system scripts, select the script whose availability you want to change. Use Table 10 to determine which check box configuration you need for your desired menu status.
3. Choose **Script > Edit Script Meta Data**. The System Script Editing prompt opens. (If you have disabled this prompt, the Edit Script dialog box opens. Skip to step 5.)
4. In the System Script Editing prompt, click Yes. The Edit Script dialog box opens.
5. Select or clear the Show Task on Task Menu check box (or the equivalent for sets).
6. Click Update.
7. In the list of scripts, select the check box next to the system script you are editing to make it available. Clear the check box to make it unavailable.
8. If this is for a crowd task, edit `./data/simulationModelSets/EntityLevel/vrfSim/systems/movement/pedestrian-crowd-movement.sysdef` and add the task to the **script-ids** list.

## 9.11. Specifying a Script Editor

ZeroBrane Studio is included with VR-Forces. VR-Forces generates a configuration file to support code completion and syntax highlighting features for all of the VR-Forces Lua API functions. However, you can use any text or script editor that you want.

► **To specify the editor to use for scripts:**

1. Choose **Settings > Application**. The Application Settings dialog box opens.
2. Select the Script Settings page (Figure 49 on page 209).
3. Click the Browse button () next to the Script Editor box. The Choose Script Editor dialog box opens.

4. Select the application that you want to use as the editor.
5. Click Open.

 You can also open the Script Settings page from the Scripts menu in the Scripts dialog box.

## 9.12. Editing Lua Files

To edit the code for scenario scripts you need to work through the Script GUI because the scripts are embedded in the scenario scripts file rather than being saved as individual Lua files. System scripts are saved as Lua files, but even in this case it is better to edit them in the context of the entire script process rather than by editing the Lua file. However, VR-Forces includes some libraries of Lua functions that are independent of any of the scripts. They are available to be included in your scripts. If you want to edit the functions in these libraries or add to them, you need to edit the individual Lua files. The function libraries are in `./userData/scripts`. You can open them in any editor, or access them through the Scripts dialog box.

► **To edit a Lua file from the Scripts window:**

1. Choose **Simulation > Scripts**. The Scripts window opens (Figure 43).
2. Choose **Script > Edit External Script**. A file chooser dialog box opens.
3. Select the script that you want to edit. The file opens in a text editor.
4. Edit the file.
5. Save the file.





## 10. Writing Scripts

This chapter explains how to write Lua scripts for scripted tasks and sets.

|                                                          |     |
|----------------------------------------------------------|-----|
| 10.1. The VR-Forces Lua Interface .....                  | 230 |
| 10.1.1 Lua Classes .....                                 | 230 |
| 10.1.2 Lua Modules .....                                 | 232 |
| 10.2. Script Loading and Execution .....                 | 233 |
| 10.2.1 Script Entry Points .....                         | 233 |
| 10.2.2 The Script Execution Sequence .....               | 235 |
| 10.2.3 Limitations for Checkpointing Scripts .....       | 239 |
| 10.2.4 Editing Scripts While a Scenario is Running ..... | 241 |
| 10.3. Tasks and Subtasks .....                           | 241 |
| 10.3.1 Monitoring the Status of Tasks and Subtasks ..... | 242 |
| 10.3.2 Stopping Tasks .....                              | 243 |
| 10.3.3 Task Parameters .....                             | 243 |
| 10.4. Geometry .....                                     | 247 |
| 10.4.1 Location3D .....                                  | 247 |
| 10.4.2 Vector3D .....                                    | 247 |
| 10.4.3 VectorOffset3D .....                              | 249 |
| 10.4.4 VectorGeoc3D .....                                | 249 |
| 10.5. Coding Messages for Translation .....              | 250 |
| 10.6. Error Detection .....                              | 250 |
| 10.7. Interactive Scripts .....                          | 251 |
| 10.8. A Basic Scripted Task .....                        | 251 |
| 10.8.1 Create and Move to Waypoint Metadata .....        | 252 |
| 10.8.2 The Create and Move to Waypoint Lua Script .....  | 254 |
| 10.9. A Simple Scripted Set .....                        | 255 |
| 10.10. A Simple Reactive Task .....                      | 256 |
| 10.11. Background Processes .....                        | 257 |

## 10.1. The VR-Forces Lua Interface

The VR-Forces Lua interface provides predefined elements that you can use to create scripts. These elements include object classes that represent the simulation, simulation objects, and geometric objects, plus Lua modules that provide tools for common tasks.

The classes and functions in the Lua interface are documented in the VR-Forces Lua API Documentation. This is a set of HTML files similar to the VR-Forces Toolkit documentation. It is accessible from the Documentation folder in the Windows Start menu or in `./doc/luadoc/index.html`.

### 10.1.1 Lua Classes

The Lua interface defines classes for common objects and functions. The classes in the interface are not fully featured classes as in the C++ language, but they do abstract the representation of information associated with an object and collect a set of methods related to the object. In Lua, to call a method `fn` on object `obj`, write:

```
obj : fn()
```

Several of the classes defined in the interface do not have constructors, because their objects are pre-defined or created only as return values to other functions. The classes and pre-defined objects in the interface are:

- *ComponentSystem*. A component system is a capability that is assigned to a simulation object in the Simulation Object Editor. Systems represent movement and damage capabilities, and represent individual weapons and sensors. In scripts, system objects are obtained from the ownship simulation object ("this") with a call to `getSystem()`, passing the name of the system (as used in the Simulation Object Editor) as an argument. The script can then access various attributes of the system through `getAttribute` calls.
- *FeatureSet*. This class represents a set of features obtained from a query to the terrain database. A *FeatureSet* is created with a *vrf* query function. *FeatureSet* methods allow a script to examine the features to determine their attribute values, vertex locations, and so on.
- *Location3D*, *Vector3D*, *VectorGeoc3D*, *VectorOffset3D*. These classes represent geometric objects. For more information, see "10.4. Geometry" on page 247.
- *ScriptAnimationModel*. *ScriptAnimationModel* refers to an animation model that can be used by a scripted task to animate an entity's movement. This class is serialized to the simulation and might contain movement data if supplied by the developer.
- *SensorGimbal*. *SensorGimbal* is a class of Lua object that represents a sensor that rotates on a gimbal.
- *SimObject*. *SimObject* refers to a single object in the simulation. It may be an entity, waypoint, route, or any other simulation object. These objects may be VR-Forces simulated objects, or remote objects received over DIS or HLA by VR-Forces. There is no constructor for *SimObjects* in the interface, but there are *vrf* functions that create objects, that get objects by name, that get nearby objects, and so on. *SimObject* methods are provided for getting information about the object such as

location, force type, and embarkation status.

- *SwingingArtPart*. *SwingingArtPart* is a class of Lua object that represents a rotating articulated part.
- *TranslatableArtPart*. *TranslatableArtPart* is a class of Lua object that represents a translating articulated part.
- *TranslatableStream*. *TranslatableStream* is a text stream class that can be translated by VR-Forces for display in the GUI as text that is translated to the current language the user is displaying (as long as those translations exist.)
- *this*. *this* is an object of class *SimObject* that is defined for all Lua scripts. It represents the simulation object running the script, also known as "ownship". It is not possible to construct a *this* object in a script. In addition to being able to use all of the *SimObject* methods, a Lua script can change the location and heading of *this*, manipulate its resources, and examine the contacts detected by the simulation object's sensors.
- *vrf*. *vrf* is defined as a global object for all Lua scripts. It is an instance of a class that represents the Lua interface; other instances cannot be created in a script. The *vrf* object provides functions that control and access information in the VR-Forces simulation.

**i** In Lua, variables assigned to objects actually just hold a reference to an object, so that if one variable is assigned to another they will both hold a reference to the same object. In this example, `firstObj` and `secondObj` refer to the same object.

```
firstObj = getObjectByName("APC 1")
secondObj = firstObj
```

In most cases this is what a Lua programmer wants. However, for the geometric objects, a Lua programmer may sometimes want to make a copy of an object. A simple assignment does not make a copy in Lua, so the geometric objects have `makeCopy()` methods to provide copies.

## Reserved Words

Table 11 lists reserved words (in addition to those reserved by Lua), which should not be used except for their intended purpose.

**Table 11. Reserved words**

| Name                         | Module                    | Function         |
|------------------------------|---------------------------|------------------|
| <code>vrf</code>             | <code>vrf</code>          | Module singleton |
| <code>this</code>            | <code>SimObject</code>    | Ownship name     |
| <code>ScriptVrfObject</code> | <code>SimObject</code>    | Class name       |
| <code>Location3D</code>      | <code>Location3D</code>   | Class name       |
| <code>Vector3D</code>        | <code>Vector3D</code>     | Class name       |
| <code>VectorGeoc3D</code>    | <code>VectorGeoc3D</code> | Class name       |

Table 11. Reserved words (continued)

| Name                 | Module               | Function                  |
|----------------------|----------------------|---------------------------|
| VectorOffset3D       | VectorOffset3D       | Class name                |
| ScriptFeatureSet     | FeatureSet           | Class name                |
| ScriptAnimationModel | ScriptAnimationModel | Global function           |
| vrfprint             | vrfutil              | Global function           |
| printDebug           | vrfutil              | Global function           |
| printVerbose         | vrfutil              | Global function           |
| printInfo            | vrfutil              | Global function           |
| printWarn            | vrfutil              | Global function           |
| printError           | vrfutil              | Global function           |
| EntityType           | vrfutil              | Global table of functions |
| bhaveLoaded          | vrfutil              | Global function           |
| new                  | fsm                  | Function name             |

### 10.1.2 Lua Modules

The Lua interface includes Lua modules for use in Lua scripts. You can access a module `stuff` in a Lua script by writing:

```
require "stuff"
```

The Lua documentation includes detailed information about each module. The modules available in the interface include:

- *aggregateUtils* is a collection of tables and functions that provide useful capabilities and definitions related to aggregates.
- *airportDataUtils* includes functions that compute information of interest about airports and runways.
- *behaviorEngine* implements a behavior engine aimed at supporting unit behaviors, or "commands." These commands are invoked and executed the same as all other Lua task scripts.
- *displayGraphUtil* has functions to assist with generating graph commands in the dot language and displaying the graph.
- *fileUtils* contains utility functions for reading data from text files.
- *formationUtils* includes functions used for coordinating tank platoons.
- *fsm* defines a class that implements finite state machine logic. There is one function in this module, `new()`, which creates an *fsm* object.
- *msgUtil* is a collection of functions that maintain queues of object or text messages that the entity receives.
- *munitionLauncherUtils* contains some functions that support munition-launching scripts such as `Delayed_homing` and `Launch_ASW_Missile`.

- *parseUtil* includes functions that assist with parsing of strings, especially from received messages.
- *pathUtil* is a collection of utility functions that help with path planning and ground-clamping points on a route.
- *spatialUtil* includes functions that compute things in space using points, vectors, and so on.
- *swgs\_test* contains utility functions for SWGS performance test scenarios.
- *vrfutil* is not technically a Lua module. (It is not defined with a module function.) It is simply a set of global utility functions. The Lua script template that opens when a new script is created contains:

```
require "vrfutil"
```

so that these functions are available to the script by default.

The *.lua* module files are located in *./makLua*. You can add user-created modules to this directory and then import them into scripts with a `require` statement.



Lua also searches the *./userData/scripts* directory, therefore you can also save files there.

## 10.2. Script Loading and Execution

This section describes script entry point functions and the execution sequence for scripted tasks and sets. It also describes how VR-Forces checkpoints scripted tasks and executes them when a checkpointed scenario is run.

### 10.2.1 Script Entry Points

The Lua API includes functions that we call "entry points" to a scripted task. When you create a new script, they are included in the default script template. Unlike the Lua functions that you might write as part of a script, you do not have to call entry point functions from within the script. They are called by the simulation engine. (However, if an entry point function is not defined in a script, it is not called.)

- `init()`. Use `init()` to set up the initial state of your scripted task's environment. The `init()` function is the first function called when a scripted task is executed. It is called when a task is assigned to a simulation object, even if the scenario is paused. This function is called only once for an instance of a scripted task or background process when it starts. If a scenario is loaded with a scripted task or background process already started, the `init()` function is not called again because of the scenario load or rewind.
- `tick()`. The `tick()` function is the heart of every scripted task. It is called at every tick of the simulation engine once the simulation object has completed its initialization. Most of a scripted task's logic should be written into the `tick()` function as it is the only entry point function consistently called in the script.

- `suspend()`. The `suspend()` function executes when a reactive task is triggered for a simulation object and the reactive task cannot run concurrently with the current task. The typical behavior, as embodied in the scripted task template, is to stop all tasks and subtasks.
- `resume()`. The `resume()` function executes when a reactive task completes. The typical behavior is to reinitialize the suspended task.

**i** As you can see from their descriptions, the `suspend()` and `resume()` functions do not exactly suspend and resume tasks as those terms are commonly understood. In most cases (this is task dependent), VR-Forces does not preserve the state of a simulation object's tasks when they are suspended, it stops the tasks. Nor does it resume the previous task from the point at which it was suspended, it restarts the task from the current state of the simulation object. However, VR-Forces always saves the lua state of scripted tasks, so when a scripted task resumes, its variables may need to be reset in `init()` or `checkInit()`.

- `saveState()`. If a script is running, the `saveState()` function is called just before a scenario is about to be saved. VR-Forces automatically saves the state of script execution. (For details, see "10.2.3 Limitations for Checkpointing Scripts" on page 239.) This function lets you do any additional processing to save the state of your scripted task. Most scripts do not use this function.
- `loadState()`. If a script was running when a scenario was saved, the `loadState()` function is called just after a scenario is loaded or rewound. VR-Forces automatically restores the state of a scripted task that was running when a scenario was saved. This function lets you do any additional processing required to restore the state of the script. If your script uses the `saveState()` function, you probably want to include complementary processing in `loadState()`. Most scripts do not use this function.

**i** If your `loadState()` function assumes the presence of other simulation objects, you must take into account that other simulation objects might not have loaded yet when the function is called.

- `shutdown()`. The `shutdown()` function is called when the task is ending for any reason. You will not typically need to write any code in this function, however it is available if there is a case in which you want to do anything before the task ends.
- `receiveObjectMessage()`. The `receiveObjectMessage()` function is called when the simulation object executing this task receives an object message. An object message, in Lua, is a table with key-value pairs. The keys are strings, and the values may be numbers, strings, Booleans, tables, simulation objects, locations, vectors, or offset vectors. The function provides some data to determine what type of object message it is and which simulation object the message came from. Developers may write C++ code to send and receive these object messages, as well.

This function takes three arguments:

- **messageType**. A text string identifying the message that matches the registered name of the object message type. The types are pre-defined in VR-Forces.
- **message**. A table of parameters that is specific to the messageType.
- **sender**. The simObject that sent the message.

Refer to the `sendObjectMessage()` function in the Lua documentation for messageType-specific parameters. A message of type "generic-table" may contain any keys and values; a script processing this type of message must examine the fields of the table to see what is in it.

- `receiveTextMessage()`. The `receiveTextMessage()` function is called when the simulation object executing this task receives a text message. The function provides some data to determine what the text message is and which simulation object the text message came from. This function takes two arguments, the message and the sender. The message is a text string and the sender is the simObject that sent the message.
- `entityDestroyed()`. The `entityDestroyed()` function is called once when the entity executing the script is destroyed.
- `receiveInfluence(influencer, influenceName, influenceParams)`. The `receiveInfluence()` function is called in aggregate-level scenarios when a simulation object receives an Influence interaction from another simulation object. Influence interactions represent different ways an object might affect another. These include engagement, jamming, providing supplies, improving engineering objects, or destroying engineering objects.

The `receiveInfluence()` function has the following parameters:

- **influencer**. The simulation object that is performing this influence action.
- **influenceName**. The name of the influence being performed. It matches one of the subclasses of the Influence interaction FOM class.
- **influenceParameters**. A table of parameters of the influence. These parameters correspond to the parameters available to the Influence interaction FOM subclass, as specified by **influenceName**. Different subclasses have different parameters.

The following script entry points are only used in reactive tasks:

- `check()`. The `check()` function defines the condition under which the reactive task will trigger. It keeps checking the condition until it becomes true, at which point it starts the reactive task.
- `checkInit()`. The `checkInit()` function initializes the reactive task when it is enabled. It sets the initial conditions for the `check()` function. `checkInit()` runs when a reactive task is first enabled and runs again after the reactive task completes so that it can begin checking the condition again.

### 10.2.2 The Script Execution Sequence

A Lua script has several distinct sections, including the entry points described in "10.2.1 Script Entry Points" on page 233 and the global commands in the script. When a scripted task is assigned to a simulation object, the following actions happen, even when the

scenario is paused:

1. The script environment is initialized. This step defines the *vrf* and *this* objects and all of the class functions in the Lua API.
2. The script text is loaded. All Lua statements in the global scope are executed. Normally, the script defines (assigns initial values to) global variables here.
3. The `init()` function is called.

This sequence is followed whenever a scripted task is assigned, including these situations:

- A scripted task is assigned while the scenario is running.
- A simulation object has a script assigned at the beginning of a scenario and the scenario is loaded or rewound.
- You modify the script or script metadata while the script is running.

If the simulation is running, after the `init()` function is called VR-Forces begins calling the `tick()` function at the period specified by the `setTickPeriod()` function (default: 0.5 seconds).

Suppose a script "test" contains the code:

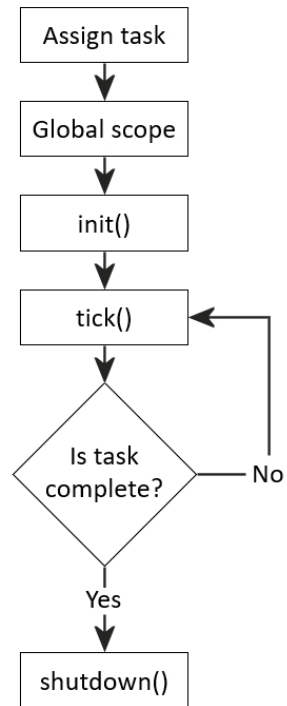
```
print("Global statements")
t = {}
function init()
  print("Init function")
  vrf:setTickPeriod(0.5)
  t[1] = 0
end
function tick()
  table.insert(t, vrf:getExerciseTime())
  print("Tick ", #t)
end
```

If the script is assigned to a simulation object and then the scenario runs for a short time, the console output would be:

```
Global statements
Init function
DtCgfDispatcher::controlScenario: run
Tick 2
Tick 3
Tick 4
DtCgfDispatcher::controlScenario: pause
```

Figure 59 illustrates the control flow for a scripted task.

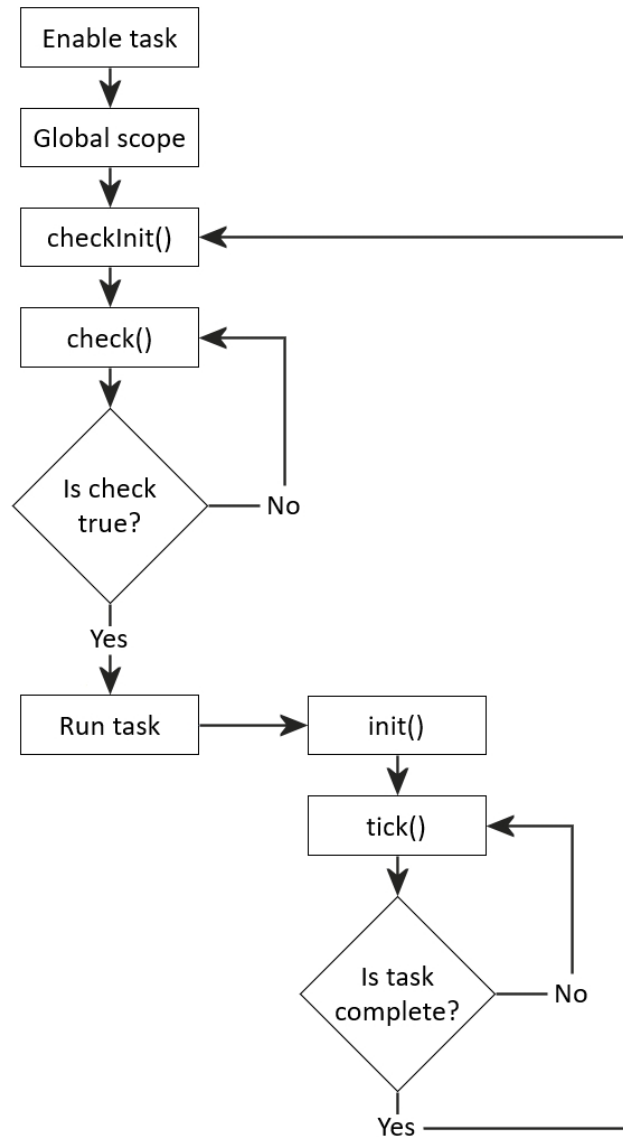


**Figure 59. Scripted task control flow**

## The Script Execution Sequence for Reactive Tasks

Figure 60 illustrates the script execution sequence for reactive tasks. It is distinguished from that for scripted tasks by the addition of the checking process.

**Figure 60. Reactive task control flow**



## The Script Execution Sequence for Saved Scenarios

When a scenario is saved (checkpointed), VR-Forces first runs `saveState()` and then saves the Lua state in the scenario file. (Checkpointing a scripted task has some limitations. For details, see "10.2.3 Limitations for Checkpointing Scripts" on the facing page. For general details about checkpointing, see "Saving a Scenario".) When a saved scenario is loaded, the following steps are taken to restore the scripted task:

1. The script environment is initialized. This step defines the *vrf* and *this* objects and all of the class functions in the Lua API.

2. The script text is loaded. All Lua statements in the global scope are executed. Global variables defined and initialized in this part of the script are re-initialized.
3. VR-Forces restores the values of global variables to the saved values. These values replace the initial values assigned in step 2.
4. The `loadState()` function is called.

 The `init()` function is not run when a scenario is reloaded.

Assume that the "test" scripted task described in the previous section is saved at the point shown in the sample code (`t = 4`). When the scenario is run from the checkpoint, the console output will be:

```
DtCgfDispatcher::controlScenario: run
Tick 5
Tick 6
DtCgfDispatcher::controlScenario: pause
```

### 10.2.3 Limitations for Checkpointing Scripts

When VR-Forces saves (or checkpoints) a scenario, it saves the following aspects of the Lua state in the checkpoint:

- The values of global variables that contain strings, numbers, and objects (of the classes listed in "10.1.1 Lua Classes" on page 230).
- Tables, but only table values that contain strings, numbers, objects, and tables. Tables cannot contain values that refer to the same table, either directly or indirectly through another table.

The following aspects of the Lua state are not saved in a checkpoint:

- Variables containing functions or threads.
- Variables declared in the global scope with the keyword **local** preceding the declaration.
- Table entries that do not use numbers or strings as keys.

Lua scripts that contain elements that are not saved can still be restored after a checkpoint if the global statements in the script reset the elements. For example, a variable containing a function is not saved in the scenario file, but if this variable is initialized in the global Lua statements, it is reset when those statements are loaded. However, if the value of this variable was changed during script execution (in the `init()` or `tick()` functions), then its value will not be restored correctly.

Tables that reference themselves in the table will not save properly.

To demonstrate how scripted tasks are saved, assume the following script:

```
print("Global statements")
a = 0
t = {foo = 0, bar = 1}
f = function () print("hello") end
```

```
function init()
  print("Init function")
  vrf:setTickPeriod(0.5)
  a = 100 -- change a
  t.foo= 10 -- change t.foo
  t.bar = nil -- remove t.bar
  f = function () print("world") end -- change f
end
function tick()
  print ("Tick. a = ",a, "t.foo = ", t.foo, "t.bar = ", t.bar)
  f()
  print()
end
```

The output the first time the scripted task runs is:

```
Global statements
Init function
DtCgfDispatcher::controlScenario: run
Tick. a = 100      t.foo = 10      t.bar = nil
world
```

When the scenario is saved and then reloaded, and the simulation is run again, the following is printed:

```
DtCgfDispatcher::controlScenario: run
Tick. a = 100      t.foo = 10      t.bar = 1
hello
```

A comparison of the outputs reveals that variable **a** and table entry **t.foo** were restored to the correct values. However, table entry **t.bar** and function **f** were re-initialized in the global statements, but were not restored to the values they had when the scenario was saved.

### Saving the State of Global Variables

Global variables are saved when a scenario is checkpointed in one of the following ways:

- If the checkpoint mode is `AllGlobals`, all global variables defined are saved.
- If the checkpoint mode is `CheckpointStateOnly`, the script only saves variables that are part of the `checkpointState` table. If you remove this value, it defaults to the behavior of saving all globals.

If you want to change the mode, call `setCheckpointMode(AllGlobals)` to save all globals or `setCheckpointMode(CheckpointStateOnly)` to save only those variables in the `checkpointState` table. They are reinitialized when a checkpointed scenario is loaded.

```
vrf:setCheckpointMode(CheckpointStateOnly)
```

The following code, from *vrfutil.lua*, defines some global values:

```
-- Global booleans to help define which checkpoint state to use
AllGlobals = false
CheckpointStateOnly = true
```

For more information about `vrf:setCheckpointMode`, see the VR-Forces Lua documentation.

### 10.2.4 Editing Scripts While a Scenario is Running

You can add, remove, and edit scripts during scenario creation or while a scenario is running. If a script is not being used by any simulation objects, editing or deleting it has no effect on the scenario. If you change a script that is being executed by a simulation object, when you save the change the script may be stopped, restarted, or continue to execute depending on what has changed.

Table 12 lists the effect on scripts and scripted tasks running as subtasks if you edit the code or metadata:

**Table 12. Side effects of editing scripts**

| Change to script                                                                  | Result                                                                                                                                              |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Change code or any metadata except the script ID. Entity type is still supported. | The script restarts. Changes to the Action Category take effect when the scenario restarts.                                                         |
| Entity types supported changes.                                                   | The script stops for simulation objects that no longer support it.                                                                                  |
| Script ID changes.                                                                | The script continues to execute. Any script that uses this scripted task as a subtask will fail until its code is updated to use the new script ID. |

If you changed the simulation object types that can use the script, then it is added to or removed from the menu for the appropriate simulation object types. (This change takes place immediately.)

## 10.3. Tasks and Subtasks

The primary way that a scripted task carries out actions is through starting subtasks for the simulation object running it or sending tasks to other simulation objects. All C++ tasks and other scripted tasks are available for sending as subtasks or tasks.

If a scripted task needs to give a task to the simulation object that is running it, it does so as a subtask. (The simulation object's current task is the scripted task itself, so any action that it needs to take is a subtask to the scripted task.) Subtasks are started using the `vrf:startSubtask()` function.

If a scripted task needs to give a task to a simulation object other than the one running it, it uses the `vrf:sendTask()` function.

A `startSubtask()` or `sendTask()` function specifies the task name and task parameters, if any. The name must be the name of one of the C++ tasks, as listed in the Task page of the Lua API Documentation, or the script ID of another scripted task.

The task parameters are passed as a Lua table, with parameter names as the indices and the parameter values as the values. The valid parameters for C++ tasks are in the Lua API documentation. For scripted tasks, the parameter names are the parameter names defined for that task. You can view them by opening a scripted task in the Edit Script dialog box. Any parameters that are not specified when starting the task use their default values.

This example subtask gives a Move to Waypoint task to the simulation object running the scripted task:

```
vrf:startSubtask("move-to", {destination = "waypoint 1"})
```

**i** If a task does not have any parameters, `startSubtask()` must represent the parameters as `{}`. For example:

```
vrf:startSubtask("wait", {})
```

This example task sends a Patrol Between Waypoints task to the simulation object named "someEntity":

```
vrf:sendTask(someEntity, "patrol-two-points", {first_control_point="waypoint 1", second_control_point="waypoint 2"})
```

If a task starts with an invalid name or invalid parameters, a runtime error occurs in the script.

Whenever a task or subtask starts, a task ID integer is returned. This task ID is used to monitor or stop the task.

**i** If a task is started on a simulation object that cannot execute it, the task will enter the Complete (failed) state shortly after the start call is made. This may not happen until a subsequent tick of the scripted task.

### 10.3.1 Monitoring the Status of Tasks and Subtasks

Once a task is started, the scripted task that started it can monitor it using its task ID. A task can be in one of three states:

- Running. The task was started, but has not completed or been canceled.
- Complete (success). The task has completed, and called `endTask(true)` to end itself.
- Complete (failure):
  - The task has ended itself by calling `endTask(false)`.
  - The task failed to start because the simulation object was incapable of executing this task.
- Canceled:

- The task was explicitly canceled using `stopTask()`.
- The task was replaced by a conflicting task.
- The user issued a Skip Task.

i The success or failure of task completion is defined by the task being executed, and may not be consistent between task types.

Subtasks are monitored with the functions:

- **isSubtaskComplete(taskID)**
- **subtaskResult(taskID)**
- **isSubtaskRunning(taskID)**
- **isSubtaskCanceled(taskID).**

Tasks are monitored with the functions:

- **isTaskComplete(taskID)**
- **taskResult(taskID)**
- **isTaskRunning(taskID)**
- **isTaskCanceled(taskID).**

### 10.3.2 Stopping Tasks

Any task or subtask that has been started from a scripted task can be stopped by that task before it has completed. This is done using **stopTask(taskID)** or **stopSubtask(taskID)**.

### 10.3.3 Task Parameters

Task parameters are the parameters that are gathered from the user, or set by a parent task, when a task is started. They are passed into the new task when it starts. A script gets the value of a parameter with the syntax **taskParameters.parameterName**, as in this example:

```
waypoint = vrf:createwaypoint
({location=taskParameters.waypointLocation})
```

For scripted tasks, the task parameters are available in a Lua table named `taskParameters` available in the global scope. The table contains items named using the parameter names specified in the parameters section of the Scripted Task dialog box when the task was created. The types of these parameters depend on the parameter types selected by the user in this dialog box.

Table 13 lists the parameter types you can specify when you create a scripted task and the corresponding Lua types in the `taskParameters` table.

**Table 13. Task parameters and Lua types**

| <b>Parameter Type</b>            | <b>Lua Type</b>                                          |
|----------------------------------|----------------------------------------------------------|
| Active Sonar Mode                |                                                          |
| Aggregate Formation              | String.                                                  |
| Altitude Label                   | View only. No Lua type.                                  |
| Altitude MSL                     | Number (meters).                                         |
| Altitude Rate Change             | Double (meters/second).                                  |
| Animated Movement                | String.                                                  |
| Bomb Resource                    | String (resource name).                                  |
| Boolean                          | Boolean. Must be true or false (lowercase).              |
| Choice (Drop-Down List)          | Number (Index of the option chosen).                     |
| Choice (Image Drop-Down List)    | Number (Index of the option chosen).                     |
| Choice (Multiple Option Buttons) | Number (Index of the option chosen).                     |
| Choice (Single Option Button)    | Number (Index of the option chosen).                     |
| Color                            | Color: rgba (name red, green, blue, alpha from 0 - 255). |
| Date/Time                        | Integer (seconds in UTC).                                |
| Depth MSL                        | Double (meters).                                         |
| DI-Guy Animation                 | String.                                                  |
| DI-Guy Appearance                | String.                                                  |
| DI-Guy Gesture                   | String (gesture name).                                   |
| Direct Fire Weapon               | String (weapon name).                                    |
| Distance                         | Number (meters).                                         |



**Table 13. Task parameters and Lua types (continued)**

| <b>Parameter Type</b>       | <b>Lua Type</b>                                                                                                                                       |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| Embarkation Slot            | Slot: <name>.slot, <name>.chooseAny (where slot is an integer and chooseAny is a boolean which means choose any other if that slot is not available.) |
| Embedded Entity             | String (embedded entity name).                                                                                                                        |
| Emitter                     | Number (Index of the emitter system).                                                                                                                 |
| Entity Type                 | String (entity type as x:x:x:x:x:x).                                                                                                                  |
| Entity Types                | Table (list) of entity types.                                                                                                                         |
| Filename                    | String (pathname).                                                                                                                                    |
| Simulation Object Type      | String (DIS type enumeration, colon separated).                                                                                                       |
| Force                       | String.                                                                                                                                               |
| Heading                     | Number (radians).                                                                                                                                     |
| Heading Label               | View only. No Lua type.                                                                                                                               |
| Indirect Fire Weapon        | String (name of weapon).                                                                                                                              |
| Integer                     | Number.                                                                                                                                               |
| Integer Map                 | Table of key of string and value of integer.                                                                                                          |
| Jammer Selector             | Integer (jammer selected, like Radar Mode).                                                                                                           |
| Jamming Mode                | Integer (mode selected, like Radar Mode).                                                                                                             |
| Loadout                     | String.                                                                                                                                               |
| Location (with altitude)    | Location3D.                                                                                                                                           |
| Location (without altitude) | Location3D.                                                                                                                                           |
| Munition Resource           | String (resource name).                                                                                                                               |
| Naval Mine Resource         | String (resource name).                                                                                                                               |
| New Tactical Graphic        | A table pf Dt:pcatopm3D Lua objects.                                                                                                                  |
| Observer Mode               | String.                                                                                                                                               |

**Table 13. Task parameters and Lua types (continued)**

| <b>Parameter Type</b>               | <b>Lua Type</b>                                                                        |
|-------------------------------------|----------------------------------------------------------------------------------------|
| Observer<br>Saved View              | A string that contains the XML output of an observer view record.                      |
| Offset Location                     | VectorOffset3D.                                                                        |
| Overlay Parent                      | String.                                                                                |
| Radar Mode                          | Integer.                                                                               |
| Radar Selector                      | Integer (radar selected, like Radar Mode).                                             |
| Range Label                         | View only. No Lua type.                                                                |
| Rate Label                          | View only. No Lua type.                                                                |
| Real                                | Number.                                                                                |
| Real Map                            | Map of key/value of key of string and value of double.                                 |
| Resource                            | String (resource name).                                                                |
| Separator                           | Not applicable.                                                                        |
| Simulation<br>Object (Single)       | SimObject.                                                                             |
| Simulation<br>Objects<br>(Multiple) | Table of SimObjects.                                                                   |
| Speed                               | Number (meters / second).                                                              |
| Sphere                              | Sphere: variable.center (Location3D), variable.radius (radius in meters).              |
| String                              | String.                                                                                |
| String Map                          | Map of key/value of key of string and value of string.                                 |
| Time                                | Number (seconds).                                                                      |
| Topographic<br>Orientation          | A Vector3D with bearing and inclination. Does not include roll.                        |
| Turn Rate                           | Number (radians / second).                                                             |
| Turn Rate<br>Label                  | View only. No Lua type.                                                                |
| Visual Meta<br>Data                 | String Map that contains key/value pairs of the marking and the value for the marking. |
| Weapon                              | String.                                                                                |

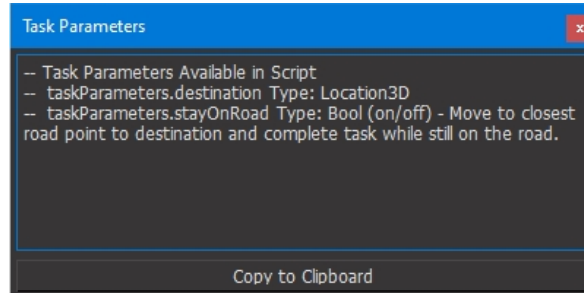
### Getting the Task Parameters for a Script

You can quickly get a list of the parameters available in a scripted task or set, their type, and a description, if one was created. You can copy the list to the clipboard so that you can paste it into your script.

► To get a list of the task parameters in a script:

1. Choose **Simulation > Scripts**. The Scripts dialog box opens.
2. Select the script whose parameters you want.
3. Choose **Script > Edit Script Meta Data**. If this is a system script, you might be prompted to continue. If so, click Yes. The Edit Script dialog box opens.
4. Click the Show Task Parameters Code button () . A window opens that lists the parameters in the script.

**Figure 61. Task parameters list**



5. Optionally, click Copy to Clipboard.

## 10.4. Geometry

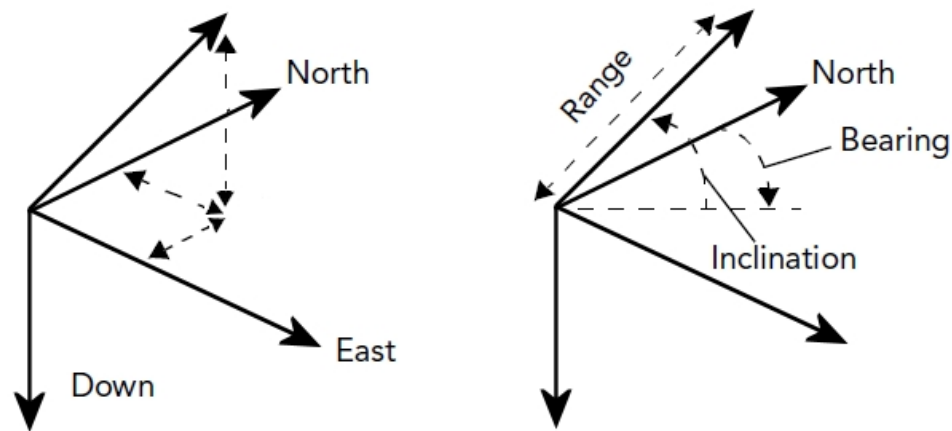
The Lua interface includes classes representing geometric objects. The classes represent locations and vectors in three dimensional space. Different classes are defined for vectors in different contexts so that functions that manipulate them know what their context is and can convert between contexts automatically. Therefore, a script does not have to, for example, create rotation matrices to perform coordinate transformations between vectors that are in an entity-body context and vectors that are in a world context.

### 10.4.1 *Location3D*

A *Location3D* represents a point in three-dimensional (3D) space. Scripts can access the three components of the location as latitude, longitude, and altitude. There is no other access to components in any other coordinate system. Methods are available to construct a *Location3D*, to create a vector from the difference between two locations, and to generate a new point by adding a vector to a location.

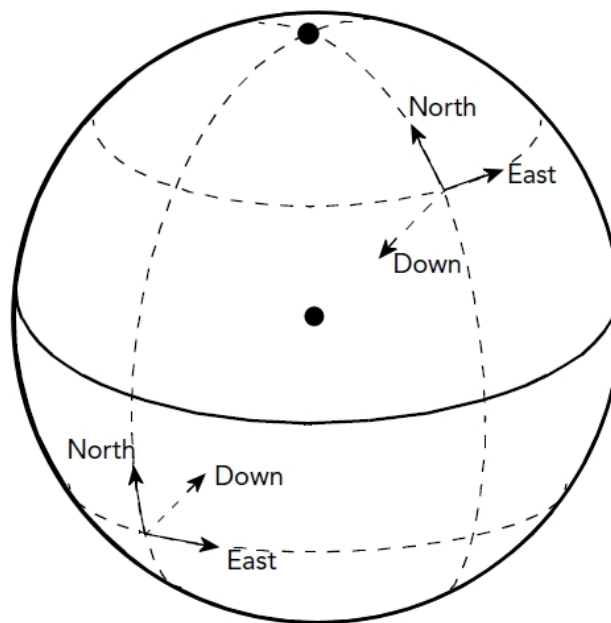
### 10.4.2 *Vector3D*

A *Vector3D* is a direction in a north-east-down coordinate system. Scripts can access these three components of a *Vector3D*, or access bearing-inclination-range (Figure 62). This is the type of vector that scripts will use most often to find directions between locations, to create new locations relative to other locations, and so on.

**Figure 62. Vector3D: North-East-Down, or Bearing-Inclination-Range**

When viewed from the perspective of a 3D Cartesian coordinate system that is fixed to the earth, there is no single north-east-down coordinate system. This is because "north" and "east" directions vary depending on where the observer is located on the earth. The *Vector3D* is therefore similar to a vector in topographic coordinates. At higher latitudes, "north" for two different simulation objects may not be the same direction (for example, two planes near the north pole, but at different longitudes, that are both flying "north," are not on parallel courses).

Figure 63 shows two north-east-down coordinate systems with origins at different locations on the earth. The coordinate systems have different orientations with respect to a coordinate system fixed to the earth.

**Figure 63. Coordinate systems with different origins**

A *Vector3D* is not strictly a vector, in that it is not a straight line in a Cartesian coordinate system fixed to the earth. Just as the direction "east" changes (viewed in the Cartesian coordinate system) to follow the curvature of the earth, so does a *Vector3D*. This definition is different from topographic coordinates, which are Cartesian coordinates fixed to a point on the surface of the earth. Implications of this definition include:

- The heading of a *Vector3D* from one location to another is the heading of the shortest route around the earth (the great circle route) between the locations. The length is the great circle distance.
- The inclination of a *Vector3D* between two locations with the same altitude is 0.0, even when one location is over the horizon from the other.
- A *Vector3D* with length equal to the circumference of the earth, and inclination 0.0, when added to a location on the surface of the earth will produce a location coincident with the first location.

### 10.4.3 *VectorOffset3D*

A *VectorOffset3D* is a set of three values that define an offset from some reference direction defined by a *Vector3D*. The three values represent *right-forward-up*, where:

- *forward* is in the same direction as the reference.
- *right* is to the right in a horizontal plane. Horizontal means the plane defined by an inclination of zero; this definition implies that the "roll" component of the reference direction is assumed to be zero.
- *up* is perpendicular to *forward* and *right*, that is, *right* X *forward*.

A *VectorOffset3D* is appropriate to use, for example, to define the offsets of simulation object formation positions from a central location. Actual formation positions could be generated by transforming the offsets to *Vector3Ds* using the direction vector of the simulation object, and then adding the *Vector3Ds* to the simulation object location.

### 10.4.4 *VectorGeoc3D*

A *VectorGeoc3D* is a vector in a Cartesian coordinate system fixed to the earth (geocentric coordinates). The actual coordinate meanings and values are not accessible to the script, because they are considered to be abstract. A *VectorGeoc3D* can be generated only by taking the difference of two locations, or by transforming a *Vector3D* at a particular location.

A *VectorGeoc3D* between two locations is the line-of-sight vector between the locations. Therefore:

- A *VectorGeoc3D* between two points on opposite sides of the earth, at the same altitude, will go through the earth. (A *Vector3D*, by contrast, will go around the earth parallel to the surface.)
- The angle between two *VectorGeoc3D* vectors corresponding to north for two simulation objects near the north pole will be the difference in their longitudes. (The angle between their *Vector3Ds* for north, by contrast, will be zero.)

The Lua interface does not have any simulation functions that require *VectorGeoc3Ds*. However, they may be manipulated to compute angles and so forth.

## 10.5. Coding Messages for Translation

VR-Forces has a utility, `translationFileCreate`, that can extract the user-visible text in a script dialog box and make it available for translation. (For details, see "Localizing the Graphical User Interface".) If a script contains messages such as `printWarn`, `printInfo`, and so on, you can code these messages so that they are also extracted to the translation file.

To code messages for translation, use the syntax:

```
printwarn(vrf:trUtf8("Text message."))
```

You can include variable data in the message:

```
printwarn(vrf:trUtf8("Some text %1. Some more text %2"):arg
(variable1):arg(variable2))
```

where an argument can be a string or number.

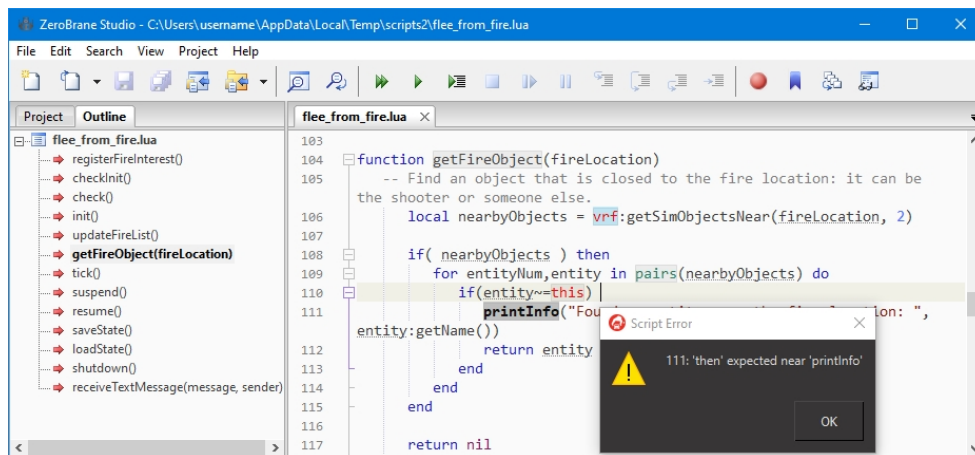
For more information about coding for translation, see the `!HowToPrint` page in the Lua documentation (`./doc/luadoc/index.html`).

## 10.6. Error Detection

The VR-Forces Lua implementation can detect syntax errors and runtime errors.

When you save a script, it is checked for syntax errors. If an error is found, it is reported in an error message. In Figure 64, the `if` statement in line 38 is missing the `then` keyword.

Figure 64. Lua syntax error

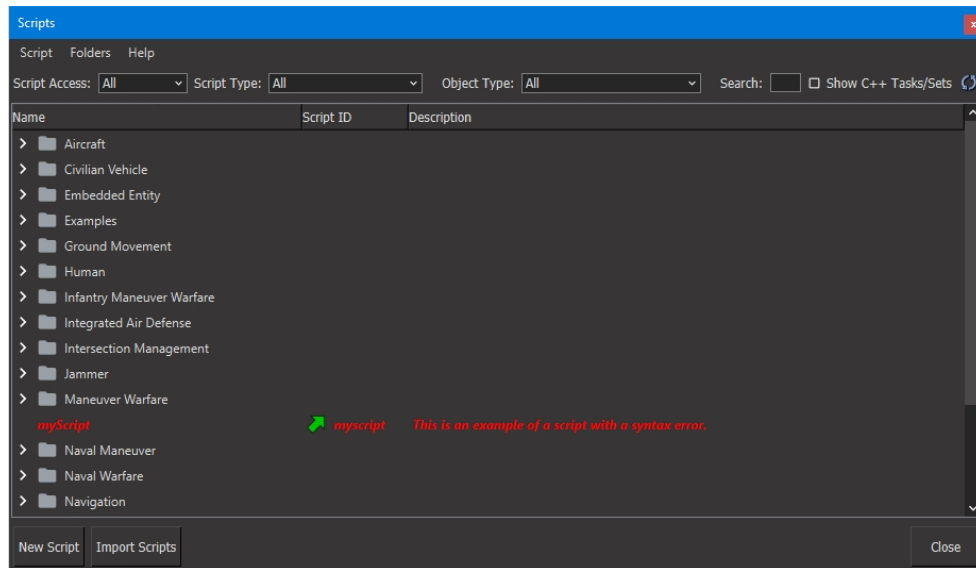


**i** Detection of a syntax error does not prevent the script from being saved. If you do not fix the error and try to use the script, the sim. engine generates an error.

When you run a script, the sim. engine can detect errors due to incorrect use of user functions and the Lua API functions, such as calling an object that does not exist, or trying to access feature data before it is paged in. When a runtime error is detected, the sim. engine generates an error message. When the sim. engine generates an error, the script

that caused the error is displayed in bold, red font in the Scripts dialog box (Figure 65). Therefore, you may want to keep the dialog box open while you are testing new or edited scripts.

**Figure 65. Lua error message**



► **To view an error message:**

1. In the Scripts dialog box, select the script that has generated an error.
2. Choose **Script > View Script Error**. An error message displays.

**i** Script errors are also sent to the console in the Information dialog box.

## 10.7. Interactive Scripts

You can write scripts that request input from the VR-Forces user. Script questions are sent to the simulation object's object console and to the Object Console Summary Panel. Users can answer questions from the Object Console Summary Panel.

To create interactivity in a script, use the `vrf:askUserQuestion()` Lua function. When a script sends a question to the console, the Answer Questions button becomes active. When the user clicks the button, a set of choices is displayed. The choices can be presented as buttons, as option buttons, or as a drop-down list.

The script can do whatever it wants with the user input.

## 10.8. A Basic Scripted Task

VR-Forces includes many scripted tasks, all of which can serve as examples of how to write one. The Create and Move to Waypoint scripted task is a basic example that shows you how to:

- Set up metadata.
- Include the VRF Utilities module.
- Initialize a subtask ID.
- Use the `init()` function.
- Create a waypoint using input from the task's dialog box.
- Send a set data request to a simulation object, using input from the task's dialog box.
- Use the `tick()` function.
- Send a subtask and get the subtask ID.
- Check for completion of the task.
- Stop the task.

The Lua code is comprehensively commented. This section supplements the comments.

### 10.8.1 Create and Move to Waypoint Metadata

Figure 66 shows the metadata for the Create and Move to Waypoint scripted task.

**Figure 66. Create and Move to Waypoint script**

Script ID: `create_and_move_to_waypoint`

Script Type: `Task` Language: `Lua`

Name: `Create and Move to Waypoint...`

Extended Name:

Short Description:

Create a waypoint at the chosen location and send the entity to it at the ordered speed.

☐ Description:

| Parameter     | Type                     | Visible |
|---------------|--------------------------|---------|
| waypointLo... | Location (with altitude) | Yes     |
| orderedSpeed  | Speed                    | Yes     |

Version: `1`

☒ Make Available Based on Entity Type

Valid for Object Types

Platform

Life form

Action Categories:

- ☐ Depth control
- ☐ Lifeform Posture
- ☒ Movement
- ☐ Radio Communication
- ☐ Weapon

☐ Can Be Executed More Than Once at a Time

Tool Tip: `Create a waypoint and move to it`

Menu Icon:

Command Level: `Primary`

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 Behavior Set

☐ Make Available to Global Plans

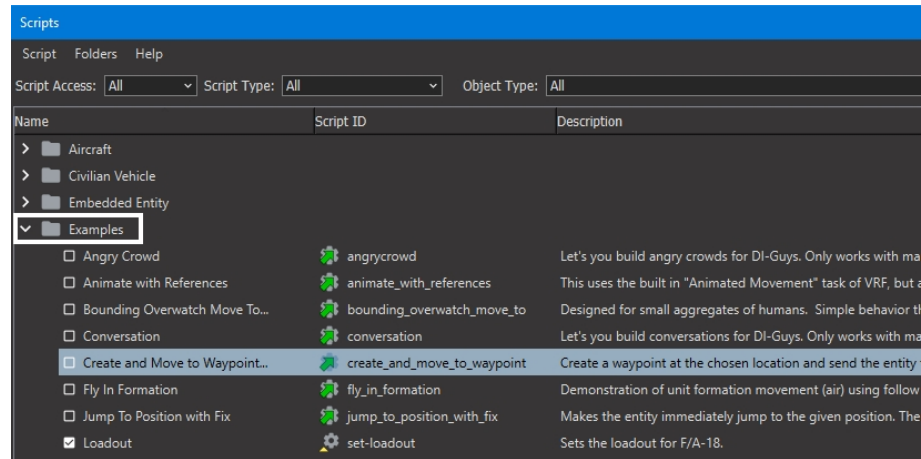
Edit Script Preview Update Cancel



► To view the metadata:

1. Choose **Simulation > Scripts**. The Scripts dialog box opens.
2. Expand the Examples folder (Figure 67).

Figure 67. Scripts dialog box



3. Select Create and Move to Waypoint.
4. Choose **Script > Edit Script Meta Data** or double-click the script name. The Edit Script dialog box opens (Figure 66).

Note how the Name column of the Create and Move to Waypoint entry in the Scripts dialog box matches the Menu Text in the Edit Script dialog box. The Script ID and Description fields are also drawn from the scripted tasks metadata.

Now let's look at the parameters. This scripted task has two parameters, waypointLocation and orderedSpeed. To see the definition of a parameter, select it and click the Edit button (✎). Figure 68 shows the dialog boxes that specify the parameters.

Figure 68. Parameter definitions

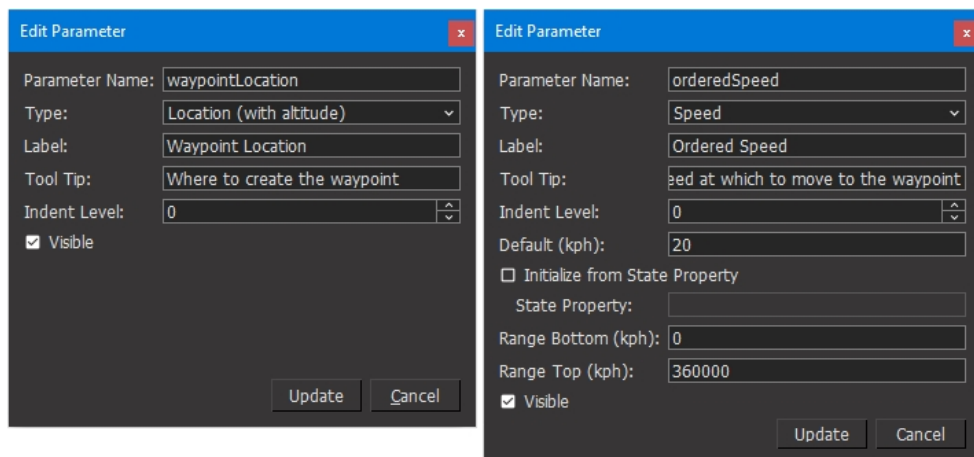
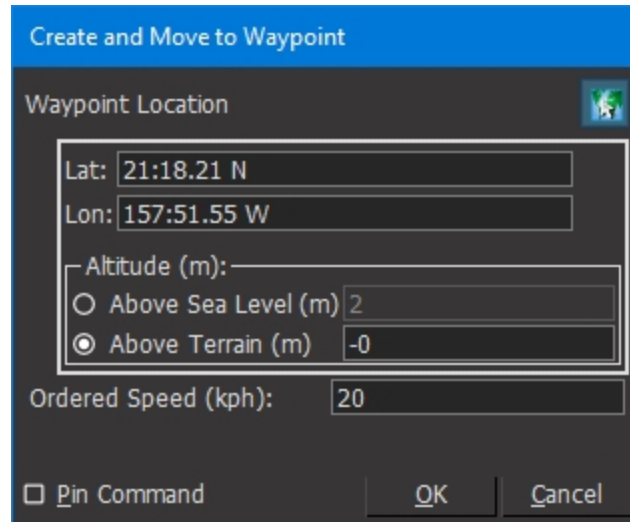


Figure 69 shows the dialog box that is displayed when you assign a simulation object this task. Note that the description of the task on the dialog box matches the description in the metadata. The parameter labels in the dialog box match the labels specified for the two parameters. The labels are not the same thing as the parameter name. The parameter name is the name used in the Lua script for this scripted task.

**Figure 69. Create and Move to Waypoint dialog box**



When this scripted task is assigned to a simulation object, the user will specify the waypoint and the ordered speed. Then the scripted task executes. Now we will look at the Lua script to see what happens.

### 10.8.2 The Create and Move to Waypoint Lua Script

The entire Lua script for this scripted task (without comments) is:

```

1  require "vrfutil"
2  moveToWaypointSubtaskId = -1
3  function init()
4      vrf:setTickPeriod(0.5)
5
6      waypoint = vrf:createWaypoint
7      ({location=taskParameters.waypointLocation,
8         force=this:getForceType()})
9
10     vrf:sendSetData(this, "set-speed",
11                      {speed=taskParameters.orderedSpeed})
12 end
13 function tick()
14     if moveToWaypointSubtaskId == -1 then
15         moveToWaypointSubtaskId = vrf:startSubtask("move-to",
16                                                     {control_point=waypoint})
17     end
18     if vrf:isSubtaskComplete(moveToWaypointSubtaskId) then
19         vrf:endTask(true)
20     end
21 end

```

```

19 |     end
20 | end
21 | function shutdown()
22 |     vrf:deleteObject(waypoint)
23 | end

```

The comments in the script describe the control flow. The essence is that, using the input from the task dialog box, the waypoint is created and the speed is set in `init()` (line 3). The first time the task is ticked, it sends the Move to Waypoint subtask. Thereafter, each time the task is ticked (line 12), it checks to see if the subtask is complete (line 13). When the subtask is complete (line 17), the task is ended and the waypoint is removed.

The functions in this script are all in the `vrf` class, which is described in "10.1.1 Lua Classes" on page 230. The `createWaypoint()` function (line 6) takes its location from the `waypointLocation` parameter that was defined in the scripted task metadata and whose value was provided in the Create and Move to Waypoint dialog box when the task was assigned. Similarly, the ordered speed is set with the `sendSetData()` function (line 9) and the value is taken from the `orderedSpeed` parameter for the task. The script gets the parameter values with the **`taskParameters.parameter`** syntax.

This script uses three of the entry point functions – `init()`, `tick()`, and `shutdown()`. If you were to checkpoint the scenario while the script was executing, the standard save actions would take place, but no script-specific actions would be taken, because there is no `saveState()` function in the script. Similarly, no special action would be taken when the scenario was loaded again because `loadState()` is not used.

For details about the parameters and return values of the functions, please see the VR-Forces Lua API documentation. Similarly, the tasks and set data requests that you can assign with the `startSubtask()` and `sendSetData()` functions are described on the Task and Set pages of the Lua API documentation. You should review the lists of functions to see what you can do in a scripted task.

## 10.9. A Simple Scripted Set

This example script shows how to create a scripted set. It takes two input values from a dialog box using `setParameters`. Then it uses the `sendSetData()` function to set the values for the entity. It needs an `endSet()` function to finish the process.

```

require "vrfutil"
function init()
    vrf:setTickWhilePaused(true)
    vrf:setTickPeriod(0.5)

end
function tick()
    vrf:sendSetData(this, "set-speed", {speed=setParameters.setSpeed})
    vrf:sendSetData(this, "set-force", {force=setParameters.setForce})
    vrf:endSet()
end

```

The following scripted task does exactly the same thing as the scripted set. It sets the speed **and** force of the selected simulation object. The difference is that since this is a scripted task, it interrupts the task that the simulation object is executing. Try these two scripts to see the difference.

```
require "vrfutil"
function init()
  vrf:setTickWhilePaused(true)
  vrf:setTickPeriod(0.5)

end
function tick()
  vrf:sendSetData(this, "set-speed",
{speed=taskParameters.setSpeed})
  vrf:sendSetData(this, "set-force",
{force=taskParameters.setForce})
  vrf:endTask()
end
```

The following scripted task does exactly the same thing as the scripted set. It sets the speed and force of the selected simulation object. The difference is that since this is a scripted task, it interrupts the task that the simulation object is executing. Try these two scripts to see the difference.

```
require "vrfutil"
function init()
  vrf:setTickWhilePaused(true)
  vrf:setTickPeriod(0.5)

end
function tick()
  vrf:sendSetData(this, "set-speed",
{speed=taskParameters.setSpeed})
  vrf:sendSetData(this, "set-force",
{force=taskParameters.setForce})
  vrf:endTask()
end
```

**i** As you can see, there is nothing in the scripts that identifies them as a scripted task or scripted set other than the `endTask()` and `endSet()` functions. VR-Forces determines the type of script from the `myScriptType` value in the metadata. Scripted task types are defined in `./include/vrfutil/scriptedTaskMetaData.h`.

## 10.10. A Simple Reactive Task

The difference between a scripted task and a reactive task is the use of the `check()` and `checkInit()` functions in the reactive task. The `check()` function tests a condition at whatever tick rate you specify. If the condition returns true, the task runs.

The simple reactive task in this section monitors the speed of a simulation object. When it exceeds 15 meters per second, the simulation object is destroyed. The script works as follows:

1. The `checkInit()` function sets the tick rate to 1.0 second.
2. In `check()`, VR-Forces gets the speed of the simulation object and prints it to the console.
3. It tests to see if the speed is greater than 15 meters per second. If the condition is met, the function returns true.
4. If the function returns true, the script executes `init()` and begins to tick().
5. In the `tick()` function, the simulation object is destroyed and the task is complete. Note that in the `executeSetData()` function, `set-destroy` does not have parameters, but the command must include empty braces.

```
function checkInit()
    vrf:setTickPeriod(1.0)
end

function check()
    currentSpeed = this:getSpeed()
    print("speed is:", currentSpeed)
    if currentSpeed > 15 then
        return true
    end

    return false
end

function init()
    vrf:setTickPeriod(0.5)
end

function tick()
    vrf:executeSetData("set-destroy", {})
    vrf:endTask(true)
end
```

## 10.11. Background Processes

Background processes are scripts that run continuously in the background. When you create a background process, you specify which simulation object types it is valid for. Thereafter, whenever you create those simulation object types the background process initializes as soon as the simulation object is created. The supported simulation object type can be as general or as specific as you want it to be, based on the entity type enumeration that you specify.

Background processes are used for simulation object actions that happen continuously in a scenario, such as consuming or receiving supplies, calculating the simulation object's footprint, and automatically attacking opposing forces.

Background processes have one important restriction - they cannot issue tasks to simulation objects or stop their tasks.

There is no indication in the GUI that a background process is running. The only way to know which background processes are available and which simulation objects they might be affecting is to examine them in the Scripts dialog box.

The following code is a very simple background process that checks a simulation object's speed every five seconds. If the speed exceeds three meters/second, it prints a message to the console. The script's metadata specifies that it applies to lifeforms. Therefore, this background process will run for each lifeform entity in the scenario.

```
require "vrfutil"
function init()
    vrf:setTickPeriod(5.0)
end

function tick()
    currentSpeed = this:getSpeed()
    if currentSpeed > 3 then
        print("Speed is:", currentSpeed, ". Going too fast.")
    else
        print ("Speed is OK.")
    end
end
```



## 11. Writing Advanced Scripts with the Behavior Engine

The VR-Forces behavior engine allows you to create behaviors for simulated forces.

|                                                                       |     |
|-----------------------------------------------------------------------|-----|
| 11.1. Overview of the VR-Forces Behavior Engine .....                 | 261 |
| 11.1.1 Behavior Trees .....                                           | 261 |
| 11.1.2 Unit-Specific Extensions to Standard Behavior Trees .....      | 261 |
| 11.1.3 Coordination of Subordinate Behaviors .....                    | 262 |
| 11.1.4 Other Behavior Representation Benefits of Behavior Trees ..... | 262 |
| 11.1.5 What Behavior Trees Are Not .....                              | 262 |
| 11.2. Behavior Node Definition .....                                  | 263 |
| 11.2.1 Common Node Attributes .....                                   | 263 |
| 11.2.2 Node Execution .....                                           | 264 |
| 11.3. Behavior Node Types .....                                       | 266 |
| 11.3.1 Task Node .....                                                | 266 |
| 11.3.2 Condition Node .....                                           | 266 |
| 11.3.3 Sequence Node .....                                            | 267 |
| 11.3.4 Loop Node .....                                                | 267 |
| 11.3.5 Selector Node .....                                            | 268 |
| 11.3.6 Parallel Node .....                                            | 270 |
| 11.3.7 Role Node .....                                                | 272 |
| 11.3.8 Sync Node .....                                                | 273 |
| 11.3.9 Send Node .....                                                | 275 |
| 11.3.10 Receive Node .....                                            | 275 |
| 11.3.11 Job Node .....                                                | 277 |
| 11.3.12 Query Node .....                                              | 277 |
| 11.4. Node Member Functions .....                                     | 278 |
| 11.5. Behavior Roles .....                                            | 279 |
| 11.5.1 Role Definition .....                                          | 279 |
| 11.5.2 Role Assignments .....                                         | 281 |
| 11.6. Behavior Engine API .....                                       | 281 |
| 11.6.1 Declaring a Behavior .....                                     | 281 |

|                                                                                               |     |
|-----------------------------------------------------------------------------------------------|-----|
| 11.6.2 Using State Variables .....                                                            | 282 |
| 11.6.3 Getting Role Information .....                                                         | 282 |
| 11.6.4 Variable Definitions .....                                                             | 282 |
| 11.6.5 Script Validation .....                                                                | 282 |
| 11.7. Behavior Integration into Lua Tasks .....                                               | 283 |
| 11.7.1 Script Type .....                                                                      | 283 |
| 11.7.2 Behavior Script Components .....                                                       | 284 |
| 11.7.3 Graphical Display of Behavior .....                                                    | 285 |
| 11.7.4 Editing with ZeroBrane Studio .....                                                    | 287 |
| 11.8. Programming Practice for Behaviors .....                                                | 288 |
| 11.8.1 Node Names .....                                                                       | 288 |
| 11.8.2 Perform Single Actions in Task Nodes .....                                             | 289 |
| 11.8.3 Avoid Excessive Computation in Node Functions .....                                    | 289 |
| 11.8.4 Put Task Parameters in State Variables .....                                           | 289 |
| 11.8.5 Minimize Global Communication Through State .....                                      | 289 |
| 11.8.6 Define Nodes in Variables and Use These Variables for Children of Other<br>Nodes ..... | 290 |
| 11.8.7 Put Static Options in doIf-Switched Nodes .....                                        | 294 |
| 11.8.8 Push Conditions out of Functions and into the Tree .....                               | 294 |



## 11.1. Overview of the VR-Forces Behavior Engine

The VR-Forces behavior engine is a Lua module that provides developers with structures, conventions, and tools to organize and simplify complex behavior scripts. Users are expected to be familiar with VR-Forces tasks and Lua scripting.

The VR-Forces behavior engine was developed primarily to support the creation of unit-level behaviors. VR-Forces already has a capable mechanism for creating entity-level behaviors: you can script tasks in Lua, using an extensible API to the simulation engine and packages such as `fsm.lua` to organize the flow of control. However, unit behavior is more complex than entity-level behavior in a fundamental way: units have subordinates. While entity behaviors are often composed from simpler entity level behaviors, unit behaviors are composed from simpler unit behaviors and individual subordinate behaviors. The subordinate behaviors typically execute in parallel, with subordinates performing different functions in coordination with one another. The behavior engine helps the developer create complex behaviors with coordinated parallel subordinate behaviors.

### 11.1.1 Behavior Trees

The VR-Forces behavior engine is based on the concept of behavior trees. Behavior tree engines are common in computer games, simulations, and even robots. Behavior trees provide several benefits to behavior design: they support decomposition of complex behaviors into simpler elements; they provide a visual representation of the decomposition; and they structure the flow of control with standard sub-behavior selection algorithms.

Behavior tree elements are called nodes. Nodes may have one or more children, which are arranged in a tree. The leaves of the tree are primitive actions. Actions in this engine may be VR-Forces tasks, asynchronous jobs, or immediate computations. Interior nodes in the tree may run their children in sequence, loop over their children repeatedly, run them in parallel, or use another standard execution control algorithm. These are all aspects of the VR-Forces behavior engine and common to most behavior tree systems.

### 11.1.2 Unit-Specific Extensions to Standard Behavior Trees

To support subordinate behaviors, the VR-Forces behavior engine uses extensions to standard behavior trees. "Roles" are parts of the unit behavior that are performed by only part of the unit—a subordinate. Subordinates are assigned to roles based on their type, their "function" in the unit, and other attributes that the developer defines. (Functions are set in the Simulation Object Editor. See "Defining Functions for Subordinates".) A part of the behavior tree reserved for a particular role is identified with a "role node". Multiple subordinates can be allowed to perform the same role, thereby creating multiple "instances" of the role node's sub tree. Typically, a unit behavior contains a parallel node whose children are role nodes, such that the subordinates performing different roles will do them in parallel.

The VR-Forces behavior engine also implements sync nodes. Sync nodes are like barriers that block execution until another sync node is executed elsewhere. These are used for synchronizing control flow in different parallel parts of the tree, particularly in parallel roles being executed by different subordinates.

### 11.1.3 Coordination of Subordinate Behaviors

Coordination between the tasks performed by subordinates is a key part of designing a unit behavior. Consider a platoon behavior to move to a support position and engage from alternating firing locations. This is a useful stand-alone behavior, and it is also useful as a component of a company (higher echelon) behavior. However, when used in a company behavior, it may be necessary to coordinate an event in the middle of the platoon behavior—for example, the moment the platoon is in position and ready to fire—with actions in the middle of another platoon's behavior. Therefore, the platoon supporting-fire behavior may be slightly different in the context of a company behavior than it is as a standalone platoon behavior.

There are several methods to achieve the coordination necessary between subordinate behaviors.

- First, explicit messages may be sent between the subordinate behaviors and the unit behavior. This approach requires explicit communication actions and a message receiving function. Messages may mimic real-world messages or be "simulation" messages. The latter can be used when in the real world there would be some other form of communication or synchronization. The VR-Forces behavior engine supports this approach with Send and Receive nodes.
- A second approach to coordination is through observation. For example, if a subordinate behavior must wait until another subordinate has completed some action, it may be possible to observe the completion of the action, and so no explicit communication is required. Pilots flying with a wingman can signal intended turns with appropriate motions, thus maintaining radio silence. This approach can be used in VR-Forces behaviors.
- A third approach to coordination is to incorporate the subordinate behavior into the unit behavior, such that the unit behavior has direct access to all of the nodes that control the subordinates. In this approach, coordination can use global variables; this is essentially what the Sync nodes are. This approach does not allow direct reuse of (stand-alone) subordinate behaviors with copying them; however, as noted above, subordinate behaviors may be different in the context of a higher unit behavior. Several behaviors that have been built using the behavior engine are designed in this way.

### 11.1.4 Other Behavior Representation Benefits of Behavior Trees

Behavior trees are typically used in games and simulations to define the behavior of a single agent. Behavior trees can switch between sub-behaviors when conditions change and the agent's current actions are no longer appropriate. The behavior assigned to an agent can persist indefinitely, switching between different tasks to satisfy one or more goals (such as a mission or self-preservation) over the course of the entire game or exercise. Behaviors created with the VR-Forces behavior engine can produce persistent, intelligent, and robust behavior for both units and individual entities.

### 11.1.5 What Behavior Trees Are Not

The aim of any behavior model is to describe what an agent should do next in the current situation. In the general case, such as modeling human-level general intelligence, an agent should be able to perform problem-solving processes such as planning, problem

recognition, problem-space search, and so on. For many virtual world applications, such as providing an environment of non-player characters for games or training systems, agent behavior can be described adequately with a simple ordering of steps and some simple decision making for branching the order. Behavior developers can effectively use behavior languages and models that abstract and simplify behaviors, even if these models do not support general problem solving.

Behavior trees govern the flow of control in a behavior. The decisions made in the tree are based on computations using information such as input parameters, unit status, locations and state of other entities, characteristics of terrain, and the environment. The behavior tree does not assist directly in any of these computations. Intelligent unit behavior may require terrain analysis, sensor fusion, threat assessment, spatial reasoning, and planning. The behavior engine does not do any of these things. Instead, the behavior engine in VR-Forces allows you to write Lua code and include it in any part of the behavior tree. Therefore, writing behavior is not an exercise in graphically composing a tree from pre-defined blocks, but a process of coding in Lua and organizing the code according to a structured architecture.

## 11.2. Behavior Node Definition

VR-Forces behaviors that use the behavior engine are Lua scripts that are organized into a behavior tree and executed with the help of the behavior engine Lua module. Behavior trees are composed of nodes, each of which is a chunk of Lua code. The behavior tree receives control every simulation cycle in a call to its tick function. The nodes perform actions and tick their child nodes. The tick functions return a value of success, fail, or running to the parent node. Nodes can contain other Lua functions as well, which perform initialization and exit tasks, start subtasks, check loop conditions, and so on. The tick function runs the Lua functions, selects children and ticks them, and processes results from children.

Because the tick function in each node executes according to a standard definition, the tick function is defined in the behavior engine. Developers do not write tick functions; instead, they indicate the children of a node, provide values for node attributes, and define functions to be called in the tick (if any of these things are relevant for the node). This node description is implemented as a Lua table. This is a simple example of a node description with attributes `nodeType` and `children`. It is a node represented as a Lua table:

```
aNode =
{
  nodeType = "sequence",
  children = {task1, task2}
}
```

where *task1* and *task2* are variables containing node descriptions.

### 11.2.1 Common Node Attributes

There is a set of node attributes that may be specified for all nodes. These attributes are described below. All of them are optional except for `nodeType` and sometimes `name`. All of the attributes that are functions take one argument, which is a node.

- **nodeType**. A string that defines the type of the node. Required for all nodes. The types available are described in "11.3. Behavior Node Types" on page 266.
- **name**. A string. This displays in the debugging graph output. (Required for sync nodes—see "11.3.8 Sync Node" on page 273.) It should very briefly describe what the node and the subtree below it do in the behavior.
- **dof**. A Lua function. Returns a Boolean. This function is executed after the behavior tree is created and subordinates have been assigned to roles, but before the tree is executed. The **dof** function takes a node as an argument; unlike the parameter passed to other functions in nodes, the **dof** function is passed its parent node.

If **dof** returns false, then the node is disabled—in effect, it is as if it was never created. The purpose of **dof** is not to cause the behavior to perform different actions in different situations that arise during the task; rather it is to make different versions of the behavior in different initial conditions. For example, a behavior might have a subtree related to fire support that is disabled if there is no forward observer, or a behavior might have a subtree of actions related to an optional task parameter that is disabled when the parameter is not present.

- **startWhen**. A Lua function. Returns a Boolean. The node (tick) is not executed until this function returns true. While it remains false, the node is in a "waiting" state. In this state the node returns "running". The execution of the behavior essentially pauses at the node until the **startWhen** function returns true.

A waiting node is treated in a special way by parent parallel nodes: it does not block the completion of the parent (see "11.3.6 Parallel Node" on page 270). If the **startWhen** never becomes true, the parent node can finish if its other children finish.

- **initFn**. A Lua function. No return value. This function is executed the first time the node is ticked.
- **exitFn**. A Lua function. No return value. This function is executed after the node finishes its tick for the last time (and thus has transitioned to a **success** or **fail** state).
- **endWhen**. A Lua function. Returns a Boolean. This function is executed at the beginning of every tick; if it returns true, then the tick function immediately returns with **success**.
- **failWhen**. A Lua function. Returns a Boolean. This function is executed at the beginning of every tick; if it returns true, then the tick function immediately returns with **fail**. This function is run before **endWhen**.

### 11.2.2 Node Execution

The following is a pseudocode description of the tick function of a generic node:

```
node.state = "created" -- Set when node is created
...
function tick(node)
    result = node.state
    if node.state is in {"created", "success", "fail"} then
        node.initFn(node)
        node.state = "waiting"
    end
```

```
if node.state == "waiting" then
  if node.startWhen(node) then
    node.state = "running"
  else
    result = "running"
  end
end
if node.state == "running" then
  if node.failWhen(node) then
    node.state = "fail"
    result = "fail"
  else if node.endWhen(node) then
    node.state = "success"
    result = "success"
  end
  Process node according to type (see processNode function
  pseudocode for different nodes in section 3 below), ticking children
  as necessary. Set node.state.
  result = node.state
end
if result is in {"success", "fail"} then
  node.exitFn(node)
end
return result
end
```

Some notes on the pseudocode:

- After a node finishes (enters a success or fail state), if it is ticked again it will reinitialize. This can happen if the node is in a loop in the behavior tree.
- If startWhen is not defined, it is treated as if it returns true.
- If endWhen or failWhen is not defined, it is treated as if it returns false.

The above algorithm happens in one tick.

## 11.3. Behavior Node Types

This section covers the node types for use when scripting with the behavior engine.

### 11.3.1 Task Node

A task node is a leaf node in the behavior tree that runs VR-Forces tasks and performs other computations. Its `nodeType` is "task". It has one unique attribute:

- **task.** (required) A Lua function. Takes a node as an argument. Returns an integer. Return value meaning:
  - -1 - "fail"
  - 0 or nil - "success"
  - Positive - A VR-Forces task ID

If the task function returns nil, 0 or -1, the node tick function immediately sets the node state and exits.

Normally, the task function starts a VRF task and returns the VRF task ID. For example:

```
task = function(node)
  Compute some parameter values
  return vrf:startSubtask("...
end
```

When the task function starts a task, the node tick function returns "running" (unless the `startSubtask` call fails). Each subsequent tick, the tick function checks the task's status and sets the `node.state` to "running", "success", or "fail" depending on the status of the task.

You can use the task function to call `vrf:executeSetData()` (or `vrf:sendSetData()`). A `setData` function executes immediately, so the task function should return 0 or nothing.

You can also use the task function simply to compute something. Computations might produce data for the condition function in a subsequent condition node or parameters for a task node, for example. The results of the computation would be saved in state variables (see below) so that the subsequent node can access them. The task function would typically return nothing or 0.

### 11.3.2 Condition Node

A condition node is a leaf node with `nodeType` "condition". It has one unique attribute:

- **condition.** (required) A Lua function. Takes a node as an argument. Returns true or false.

If the condition function returns true, the node tick function sets the node state to **success**. Otherwise, it returns **fail**.

### 11.3.3 Sequence Node

A sequence node is a node that executes child nodes in sequence. Its nodeType is "sequence". It has one attribute beyond the common node attributes:

- **children.** (required) A list of node descriptions; that is, a Lua table whose elements are table descriptions of Lua nodes.

The tick function of a sequence node executes the tick functions of the nodes in its "children" list one at a time until they all succeed, or one returns **fail**. If one returns **running**, then the sequence node returns **running**. The children are executed in the order that they appear in the children list.

The sequence node tick function acts as follows:

```
node.index = 1 -- Set when node is created
...
function node.processNode()
    status = "success"
    while status == "success" and
        node.children[node.index] != nil do

        status = node.children[node.index].tick()
        node.index = node.index + 1
    end
    node.state = status
end
```

### 11.3.4 Loop Node

A loop node repeats its child until a condition fails or the child fails. The nodeType is "loop". The node has two attributes beyond the common attributes; both attributes are required:

- **doWhile.** (required) A function returning a Boolean. The node exits when this function becomes false.
- **child.** (required) A node description (table).

The loop node processNode function simply ticks the child and then tests doWhile. The algorithm is:

```
function node.processNode()
    status = node.child.tick()
    if status == "fail" then
        node.state = "fail"
    elseif status == "success" and dowhile == false then
        node.state = "success"
    else
        node.state = "running"
    end
end
```

The loop node returns **running** while its child is running. If the child returns **fail**, the loop returns **fail** immediately. The `doWhile` function is only checked when the child has finished and returns **success**.

The second time through a loop, the child starts in the **success** state; however, when a node in **success** state is ticked, it acts as if it is being ticked for the first time. Thus, the `initFn` is called again (see "11.2.2 Node Execution" on page 264).

### 11.3.5 Selector Node

A selector node is a node that attempts to execute its children until one succeeds or all fail. The `nodeType` is "selector". It has two attributes beyond the common attributes:

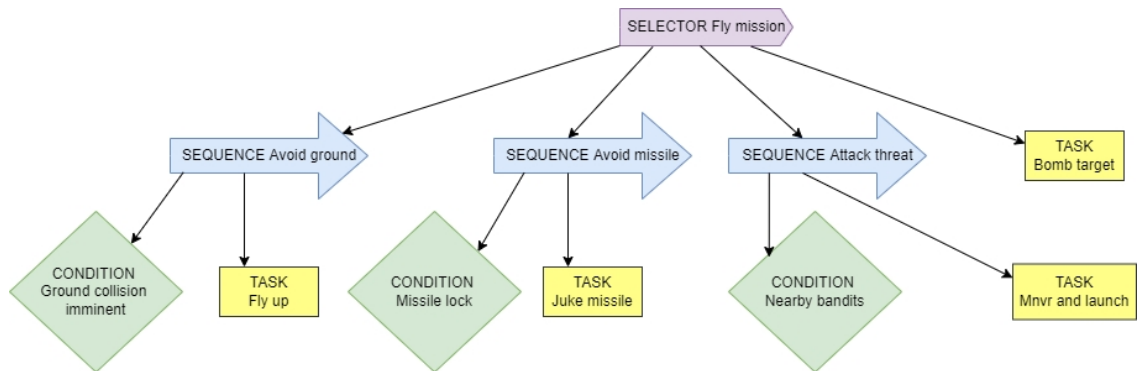
- `mode`. (optional) A string that is either "reactive" or "fallback". The default mode is "reactive".
- `children`. (required) A table of node descriptions.

The node has two modes of operation: reactive and fallback. These are similar algorithmically, but are appropriate for completely different applications.

#### Reactive Mode

In reactive mode, each tick the selector node ticks its first child node (the order of the child nodes is determined by the order of nodes in the "children" list). If the node returns **fail**, then the selector ticks the next node. This repeats until one child returns **success** or **running**; the selector node then returns the same result. If all nodes return **fail**, then the selector returns **fail**. Typically, the children are each sequences consisting of a condition followed by an action or subtree; the effect of the selector node in reactive mode is to select the highest-priority child node whose conditions are satisfied. This selection is made every tick, making the process very reactive. This design pattern is illustrated in Figure 70. Each tick, the selector node checks all of the nodes from left to right. When the condition for a node is true, the corresponding task is run. When none of the conditions are true and the Bomb Target task is finally complete, the selector node exits.

Figure 70. Example selector node in reactive mode





The pseudocode for the reactive mode selector tick function is:

```

node.index = 1 -- Set when node is created
...
function node.processNode()
  for childIndex = 1 to node.children.size() do
    status = node.children[childIndex].tick()
    if status is in {"success", "running"} then
      if node.index > childIndex then
        node.children[childIndex].stopNode()
        node.index = childIndex
        break
      end
    end
  end
  node.state = status
end

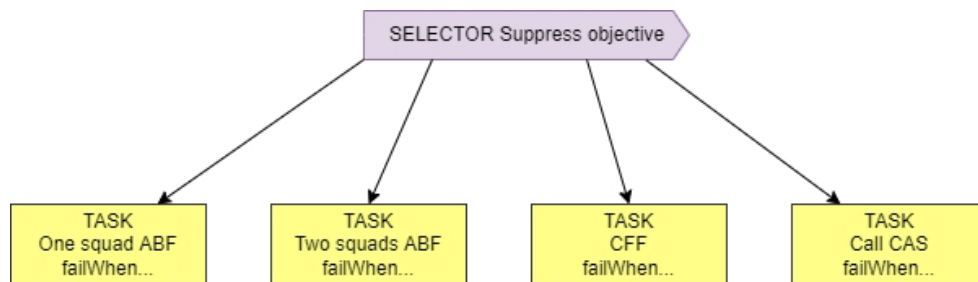
```

The `stopNode()` function in the code above stops VR-Forces tasks and jobs and cancel queries that are in a node or its children.

### Fallback Mode

In fallback mode, the children are again ticked until one of them returns **success** or **running** or all return **fail**. However, in each tick, the children are ticked beginning with the one that was **running** the last tick. The children of the selector node in this mode are not prioritized reactions, but methods for accomplishing an objective in order of preference. When the selector ticks a child, only if it fails does the selector try the next child in the list. The children are, in effect, fallback methods for one another. Figure 71 illustrates this design pattern. Each of the first children has a `failWhen` attribute that causes the node to return **fail** if the method does not successfully suppress the objective:

**Figure 71. Example selector node in fallback mode**



The pseudocode for the fallback mode selector tick function is:

```

node.index = 1 -- Set when node is created
...
function node.processNode()
  status = "fail"
  for childIndex = node.index to node.children.size() do
    status = node.children[childIndex].tick()
    if status is in {"success", "running"} then
      if node.index > childIndex then
        node.children[childIndex].stopNode()

```

```
        node.index = childIndex
      break
    end
  end
  node.state = status
end
```

### 11.3.6 Parallel Node

A parallel node is a node that ticks all its children every tick, in parallel. The `nodeType` is "parallel". There are three attributes beyond the common set:

- `children`. (required) A table of node descriptions.
- `successPolicy`. (optional) A string with the value "one" or "all"; the default is "one". This attribute specifies how many children must return **success** for the parallel node to return **success**.
- `letAllFinish`. (optional) A Boolean; the default is true. The return value of a parallel node may be determined before all children have returned **success** or **fail**; if this attribute is true, the parallel node continues to return **running** until all of the children have finished. For example, if a parallel node has a `successPolicy` of "one" and one child returns **success**, the value of the parallel node is **success**, regardless of whether the other children have finished or are still running. Setting this attribute to true makes the parallel node return **running** until all children have finished.

When a child node finishes, but the parallel node has not finished, the parallel node does not continue to tick that child.

If a parallel node has `letAllFinish = false`, and a child node result allows the parallel node to determine what its result is (for example, `successPolicy = "one"` and a node returns **success**), the parallel node will finish and return. However, if any of its children are still running, they will be stopped. Stopping a task means that the engine calls a function like `vrf:stopTask()`, stopping a job involves `job:cancel()`, and so on.

If a child node's `startWhen` function is false, it is not considered in the `letAllFinish` test. That is, if `letAllFinish` is true, and `startWhen` is false in the remaining running children, then the parallel node will finish (status changes from **running** to **success** or **fail**).

The processing of a parallel node in the tick function depends on the values of its attributes. The different cases are listed below.

#### **successPolicy = "one", letAllFinish = true**

```
node.running = node.children -- Set when node is created
node.result = "fail" -- Save the result until all finish
...
function node.processNode()
  for child in node.running do
    status = child.tick()
    if status is in {"success", "fail"} then
      Remove child from node.running
    end
    if status == "success" then
      node.result = "success"
    end
  end
end
```

```
    end
  end
  if node.running is empty then
    node.status = node.result
  else
    node.status = "running"
  end
end
end
```

**successPolicy = "all", letAllFinish = true**

node.running = node.children -- Set when node is created  
node.result = "success" -- Save the result until all finish

```
...
function node.processNode()
  for child in node.running do
    status = child.tick()
    if status is in {"success", "fail"} then
      Remove child from node.running
    end
    if status == "fail" then
      node.result = "fail"
    end
  end
  if node.running is empty then
    node.status = node.result
  else
    node.status = "running"
  end
end
end
```

**successPolicy = "one", letAllFinish = false**

node.running = node.children -- Set when node is created

```
...
function node.processNode()
  for child in node.running do
    status = child.tick()
    if status is in {"success", "fail"} then
      Remove child from node.running
    end
    if status == "success" then
      node.status = "success"
      break
    end
  end
end
```

```
    if node.running is empty and
        node.status == "running"
        -- Finished all children without a success

        node.status = "fail"
    end
end

successPolicy = "all", letAllFinish = false
node.running = node.children -- Set when node is created
...
function node.processNode()
    for child in node.running do
        status = child.tick()
        if status is in {"success:", "fail"} then
            Remove child from node.running
        end
        if status == "fail" then
            node.status = "fail"
            break
        end
    end
end
if node.running is empty and
    node.status == "running"
    -- Finished all children without a fail

    node.status = "success"
end
end
```

### 11.3.7 Role Node

A role node is a node that defines a portion of the behavior that will be performed by one or more subordinates of the unit performing the behavior. A unit behavior will often be composed of multiple roles such as "support", "assault" and "security" working together. Each of the component parts of the behavior is a child of a different role node.

The role node has nodeType "role". A role node has two required attributes:

- **role.** A string; the name of the role. This must match the roles defined for the behavior (see Behavior Roles below).
- **child.** A node description (table).

Several subordinates of the unit performing the behavior may be assigned to the same role (see Behavior Roles below). When this is the case, the behavior engine creates an instance of the child node for each of these subordinates. The role node executes each of these in parallel, much like a parallel node. The success policy of the role node is "one", and the let-all-finish policy is true.

If no subordinates are assigned to the role, the role node returns "success". (If in the behavior design it is desirable to ignore the role node when there are no subordinates assigned to the role, this condition can be tested in a doIf function in the role node.)

Roles may be marked as "critical" in the behavior; see "11.5.1 Role Definition" on page 279. When a role is critical, it must have at least one subordinate assigned to it, otherwise it fails. If the role node fails for this reason or because all of the instances of the child node fail, then the whole behavior fails.

Each instance subtree beneath a role node has its own private environment for saving state data. State data is described in "11.6.2 Using State Variables" on page 282 and "11.4. Node Member Functions" on page 278.

### 11.3.8 Sync Node

A sync node is a signaling feature for synchronizing control flow in parallel branches of the subtree. This mechanism involves a global state variable for recording the states of the sync nodes. The behavior engine manages the global variable and the state changes of the nodes.

The nodeType of a sync node is "sync". There are no special attributes for the sync node. In fact, none of the common attributes for nodes are allowed except for doIf. Thus initFn, startWhen, endWhen, failWhen, and exitFn are not allowed. The name attribute is required for a sync node. All sync nodes in the tree that are synchronized with each other must have the same name.

When control flow in the tree reaches a sync node, it is essentially blocked until control flow reaches all identically named sync nodes in all other parts of the tree. When a sync node is ticked, the behavior engine checks all other identically named sync nodes. If they have not been ticked, then the first node returns **running**. If all other nodes have been ticked, then the sync node returns **success**. All of the other nodes return **success** the next time they are ticked as well.

Sometimes a behavior subtree with sync nodes must be executed multiple times in a loop. After a sync node returns **success**, if it is ticked again it acts as if it is being ticked for the first time.

#### Sync Node Execution

Figure 72 is a diagram of the states that a sync node goes through. The node starts in the "inactive" state before it is ticked for the first time. When it is ticked, it enters the "waiting" state and returns **running** unless all other nodes with the same name are already in the "waiting" state; in that case, the node remains in the inactive state and returns **success**. The engine also changes the states of all other nodes with the same name to "free". A node in the "free" state becomes "inactive" and returns **success** the next time it is ticked. All of the nodes start "inactive" and change to "waiting" when they are ticked, until only one more node is inactive. When the last node is ticked, it remains "inactive" (to be ready for another synchronization event) but sets all the other nodes to "free". "Free" nodes change to "inactive" when they are ticked, and return **success**.

**Figure 72. Sync node states. References to other nodes (such as "other waiting nodes") indicate other nodes with the same name.**

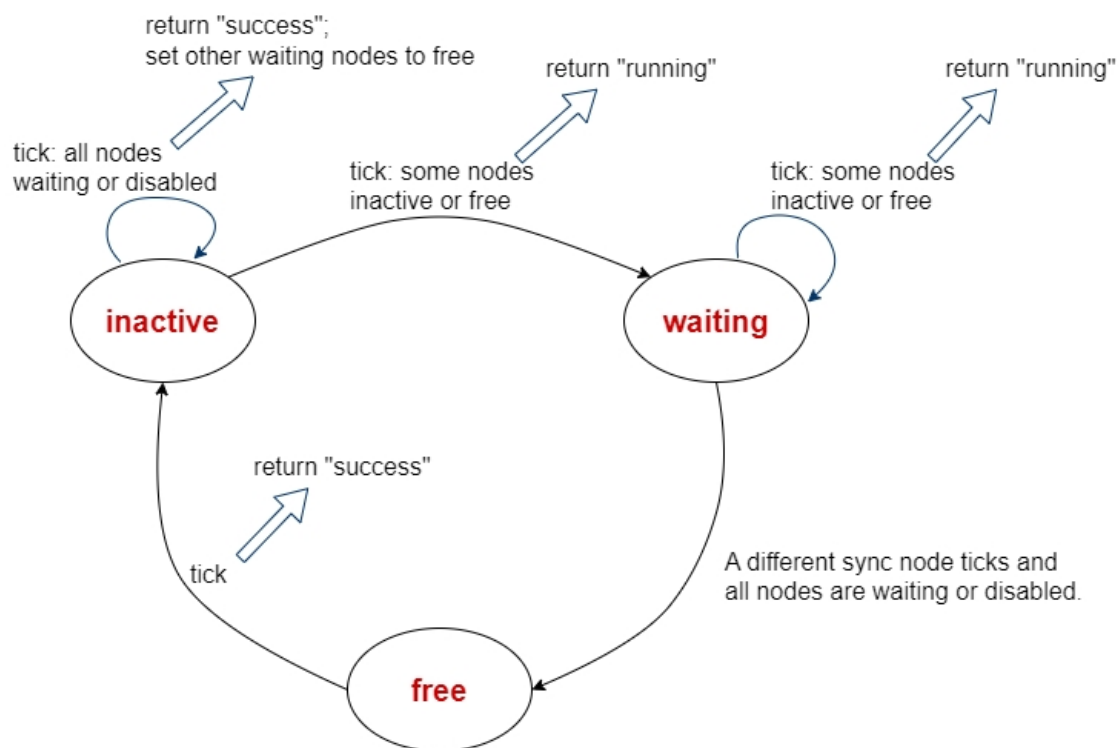
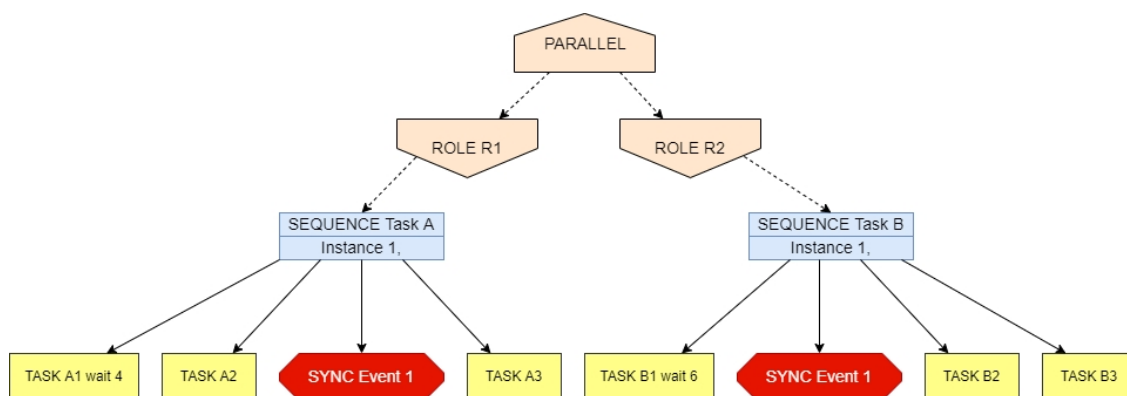


Figure 73 shows a simple application of sync nodes. In this example, task A3 in role R1 is intended to start at the same time as task B2 in role R2. The sequence in Task A is blocked with sync node Event 1 returning **running** until sequence Task B ticks sync node Event 1 (or vice versa).

**Figure 73. Example of using sync nodes to coordinate the start of tasks A3 and B2**



"11.8.6 Define Nodes in Variables and Use These Variables for Children of Other Nodes" on page 290 presents the code represented by this tree.

### Disabled Sync Nodes

There is an additional sync node state, "disabled". Sync nodes become disabled in certain conditions when a part of the behavior tree is not active (below). When a node is disabled, it is ignored for synchronization purposes; it is as if it is always in the "waiting" state. If, however, a disabled node is ticked again, it acts as if it was in the "inactive" state, going to the "waiting" state.

Sync nodes are disabled by the engine to attempt to prevent a behavior from blocking unintentionally when something goes wrong. Nodes are disabled if:

- A dolf function evaluates to false in a node, all sync nodes in the subtree under that node are disabled.
- A child of a sequence node returns **fail**, thus causing the sequence node to fail. Sync nodes are disabled in the subtrees of all children to the right of the failed child.
- A parallel node finishes while some of its children are still running (that is, `letAllFinish = false`), then the running children are stopped, and sync nodes in the subtrees under the children are disabled.

#### 11.3.9 Send Node

A send node is used for sending a message to another object. This message may be text, or it may be a structured message (Lua table). In effect, this node wraps the `vrf:sendTextMessage` and `vrf:sendObjectMessage` functions. It is intended to support "generic-table" object messages by default, as these are expected to be the basis for domain messages such as SITREPS.

The node type of a send node is "send". A send node has several attributes beyond the common attributes:

- `messageType`. (optional) A string that defines the VR-Forces message type to send. This is the name of an object message type, or it can be "text" to send an unstructured text message. If omitted, the message type is "generic-table".
- `sendTo`. This attribute specifies the recipient of the message. It can either be a `simObject`, or the string "unit" or "subordinates". If the value is "unit", the message is sent to this object's superior and all of the superior's subordinates (peer entities). If the value is "subordinates", then the message is sent to all of this object's subordinates.
- `messageFn`. A Lua function. The function takes on argument: a node. The function must return either a string (for text messages) or a table.

A send node finishes with a status of **success** the first time it is ticked.

#### 11.3.10 Receive Node

A receive node provides a means for a behavior to receive and process messages. It is a wrapper for the script entry points `receiveTextMessage` and `receiveObjectMessage`, and the function `vrf:registerForObjectMessage`. It makes use of the `msgUtil` Lua library to manage message queues.

The node type of a receive node is "receive". A receive node has the following attributes beyond the common attributes:

- **messageType.** (optional) A string defining the type of messages to receive. This is the name of an object message type, or the strings "text" or "all". If the value is "all", all types of object messages and text messages are received. Otherwise, only the given object messages are received. If omitted, the type is assumed to be "generic-table".
- **sender.** (optional) This attribute specifies who the sender of the message must be. This attribute may be a simObject, or it may be the strings "unit" or "subordinates". The meanings of these two strings are the same as in the `sendTo` attribute in `Send` nodes. If no sender is specified, then messages from all senders are received.
- **acceptFn.** A Lua function. This function takes one argument, a node. This function is called on each message received. The function should return either true, false, or nil. A value of true causes the node to finish with status **success**. A value of false causes the node to finish with status **fail**. A value of nil allows the node to keep running to process additional messages.
- **timeOut.** (optional) A number. If specified, `timeOut` indicates the maximum number of seconds that the node can remain running without receiving a message. If the `timeOut` is exceeded, then the node returns with status **fail**.

### Accessing the Message

The `acceptFn` takes the node as its argument. A special node function `node.result()` provides access to the message received. This function returns a table with the fields:

- **messageType.** A string defining the type of the message. This is an object message type, or "text".
- **message.** The message body, which is either a string or a table.
- **sender.** A simObject that is the sender of the message.

### Message Registration

Normally, when a script wishes to receive object messages, it must call `vrf:registerForObjectMessage` and specify the type of message to receive. However, when a receive node specifies a message type, the behavior engine makes the `registerForObjectMessage` call. The behavior script does not need to make this call.

If the message type is specified to be "all", the behavior engine does not call `registerForObjectMessage` for every possible object message type. In this case, the behavior script must make these calls for the messages of interest.

### Asynchronous Entry Points

Since the behavior engine uses the `msgUtil` package, the `receiveTextMessage` and `receiveObjectMessage` entry point functions are already defined, and they should not be defined in the behavior script. For more information, see the Lua documentation for `msgUtil`.



## Message Queues

The behavior engine runs the behavior tree synchronously, which is at regular intervals in a tick function. Behavior trees therefore process messages synchronously as well. The behavior engine makes use of the message queues in `msgUtil` to store incoming messages between ticks, and then receive nodes process them in order when they are ticked.

During each tick, the first (oldest) message is taken from the node's queue and the node's `acceptFn` is called. If the `acceptFn` returns true or false, the node's status is changed and it finishes processing. If the `acceptFn` returns nil, then the behavior engine gets the next message from the node's queue and again calls `acceptFn`. The queue is processed until there are no more messages or `acceptFn` returns true or false.

### 11.3.11 Job Node

A job node is similar to a task node in that it starts an activity in VR-Forces that completes in a later tick. A job node is used to start an asynchronous job. The `nodeType` is "job". This node has one attribute beyond the common attributes:

- `job`. A function that returns an `AsyncJob` object (defined in the VR-Forces Lua API).

The job function returns the result of a Lua API function that creates an `AsyncJob`. For example, `vrf:findCoverPoint(...)`. This node will return "running" until the `asyncJob` has finished. The job node should use the `exitFn` to retrieve the results of the job; the results can be saved in a state variable for use by other nodes. The `exitFn` uses a special node function `node.result()` which will return the result of the `asyncJob`. The form of this results depends on which Lua API function was invoked in the job function.

This is an example of a job node:

```
{
  nodeType = "job",
  name = "Get positions",
  job = function(node)
    return vrf: findCoverPoint(...)
  end,
  exitFn = function(node)
    setStateVar("coverPoints", node.result())
  end
}
```

### 11.3.12 Query Node

A query node allows some user interaction in the behavior, to the extent that user interaction is supported in Lua scripts. The `nodeType` is "query". A query node has two attributes beyond the common ones:

- **query**. This is a table with the following keyword elements:
  - **title**. A string that will display as the title of the dialog box.
  - **message**. A string that will display in the dialog box to prompt the user.
  - **options**. A table whose items are strings. These strings are the responses from which the user can select when responding to the query.
  - **displayStyle**. An integer specifying how the options will display:
    - 0 - Set of buttons
    - 1 - Radio buttons
    - 2 - Combo box
- **timeOut**. A number. (optional) The number of seconds that the behavior will wait for the user to respond to the query. If there is no response in this time, the query is canceled.

The options in the query attribute are only the arguments to the `vrf:askUserQuestion()` function.

The query node returns **running** when it is waiting for a response, and always returns **success** when it completes. The result of the choice can be found by calling the `node.result()` function (in the `exitFn`). `Result()` returns an integer corresponding to the user choice, with 0 as the first choice.

## 11.4. Node Member Functions

Each of the node attributes (described in "11.2.1 Common Node Attributes" on page 263 and "11.3. Behavior Node Types" on page 266) that is a function has one argument: a node. These functions are `dof`, `startWhen`, `initFn`, `endWhen`, `failWhen`, `exitFn`, `task`, `condition`, `doWhile`, and `job`. The node argument can provide context in the function. A node object provides several functions to provide this context:

- **subordinate()**. Returns a `simObject`. This is the subordinate that is executing the behavior node. This value is `nil` for the unit running the behavior from the top; however, underneath each role node, a subordinate will be assigned to run the subtree.
- **instanceN()**. Returns an integer  $\geq 1$ . If more than one subordinate has been assigned to a role, then each has a unique index assigned as well. This index will correspond to the order in which the subordinate has been assigned to the role (see role assignment, below). This index can be used to cause the behavior to be slightly different for each subordinate in a role; for example, by positioning the subordinates `instanceN() x spacing` along a line.
- **setRoleStateVar(variableName, value)**. Each instance of each role has a private environment in which to store state variables (in addition to the global environment; see "11.6.2 Using State Variables" on page 282). This function is used to store variables in the private state. `variableName` is a string identifying the variable; `value` is any Lua value (a string, number, object defined in the Lua API, or table).

Note that if this function is called from a place in the behavior that is not beneath a role node, the variable will be stored in the global environment.

- `getRoleStateVar(variableName)`. Returns the value of the state variable with `variableName` (which is a string).
- `result()`. Job and query nodes only. For jobs, returns the result of the job. For queries, returns an integer corresponding to the user choice, with 0 corresponding to the first choice.
- `status()`. The state of the function; a string ("created", "waiting", "running", "success", "fail"). Normally the status of nodes should not be needed in a behavior, except perhaps for printing debugging output. Status is used by the engine for implementing node logic.

## 11.5. Behavior Roles

Behaviors may include multiple roles in their definition. A role is a part of the unit behavior that is executed by a subordinate of the unit. Typically, multiple roles will be executed in parallel. Roles have no meaning for behaviors that are run on individual entities.

There are two parts to a role definition. The first part is a definition of the role name and specification of which subordinates can be assigned to each role. This part is written in the form of a table with keywords "types", "isCritical", "maxInstances", and "scoreFn". These are explained in "11.5.1 Role Definition" below. The second part is comprised of one or more role nodes in the behavior tree that have a "role" attribute that matches the role name in part one. Role nodes are described in "11.3.7 Role Node" on page 272.

If you write a behavior for an entity without subordinates, then the role definition table may be omitted from the behavior.

### 11.5.1 Role Definition

The role specification is given in a table in the script. The elements of the table are:

- **name.** (required) A string. This identifies the role. The "role" attribute of a role node must match this name exactly.
- **types.** (optional) A list (table) of entity type strings. Only subordinates with these types can be assigned to the role. Type strings may use "-1" in a field to indicate that any value matches. If the types attribute is missing, any type of subordinate is allowed to have the role.
- **isCritical.** (optional) Boolean. If this attribute is true, then the role is critical: at least one subordinate must be assigned to the role, and at least one instance of the role must return "success" for the behavior to succeed. If no subordinates can be assigned to the role, the behavior fails before starting. If a critical role node returns "fail", the entire behavior will fail at the end of the tick. If this element is not included in the definition table, "false" is assumed.
- **maxInstances.** (optional) An integer. The maximum number of subordinates that can be assigned to the role. If this element is not included in the definition table, the value is assumed to be 1.

- `scoreFn`. (optional) A function that takes a `simObject` and an integer as arguments and returns a number. The `simObject` passed is a subordinate, and the integer is an index into the list of subordinates currently in the unit. This function provides a score for a possible assignment of the `simObject` to the given role. Higher scoring subordinate-role pairs will be assigned before lower scoring ones. A score of 0 means that the subordinate should be assigned to the role only if it cannot be assigned to any other role. A score of  $< 0$  means that the subordinate will never be assigned to the role. If the `scoreFn` is missing from the definition table, a score of 0 is assumed.

This is an example of a role specification:

```
myRoles =
{
  {
    name = "first maneuver section",
    types = {"1:1:-1:1:-1:-1:-1", "1:1:-1:2:-1:-1:-1"},
    maxInstances = 2,
    isCritical = true,
    scoreFn = function(sub, index)
      local fn = subordinateFunction(sub)
      if fn == "PL" or fn == "PLWM" then
        return 1.0
      else
        return 0.0
      end
    end
  },
  {
    name = "second maneuver section",
    types = {"1:1:-1:1:-1:-1:-1", "1:1:-1:2:-1:-1:-1"},
    maxInstances = 2,
    isCritical = false,
    scoreFn = function(sub, index)
      local fn = subordinateFunction(sub)
      if fn == "PSGT" or fn == "PSWM" then
        return 1.0
      else
        return 0.0
      end
    end
  }
}
```

In this example, we assume that there is a function `subordinateFunction(sub)` that returns the function of the `simObject` `sub` in the unit. These `scoreFns` assign up to two vehicles to each of two maneuver sections, with preferences for certain roles in each section.

Score functions can use many factors to identify subordinates for a role. They can:

- Get the object type and look up parameters.
- Compute distances to control objects.

- Use index positions in the subordinate list.
- And so on.

### 11.5.2 Role Assignments

Subordinates are assigned to roles in the `init()` function of the script. The role assignments are permanent for the duration of the behavior. Each subordinate can only be assigned to one role. It is possible for a subordinate not to be assigned to any role.

Optimal assignment of subordinates to roles is a very complex problem in general, so the behavior engine does not try to find an optimal assignment. The assignment algorithm works as follows:

1. Compute the score for each subordinate for each role.
2. Go through the critical roles in list order and assign one subordinate to each; assign the highest scoring subordinate available. (If any critical role does not have a subordinate assigned, exit with failure.)
3. Go through the non-critical roles in list order and assign one subordinate to each; assign the highest scoring subordinate available, but do not assign anyone if the highest score is 0.
4. Find the highest score for any role among all remaining subordinates and make that assignment. If there are multiple scores with the same value, prefer an assignment to a role with fewer subordinates already assigned. Repeat until all subordinates have been assigned or all roles have been filled to `maxInstances`.

Throughout the process, constraints are maintained: assignments must satisfy the type constraint, roles can only have `maxInstances` subordinates assigned, and scores cannot be  $< 0$ .

## 11.6. Behavior Engine API

The behavior engine does not have a large API that scripts use to define or run behavior trees; most of the information is in the behavior node elements described in the previous sections. This section describes the function necessary to register a behavior tree, and the other functions provided by the behavior engine.

### 11.6.1 Declaring a Behavior

After the behavior and roles have been defined as tables, the tree is registered with the engine with the function:

```
function defineCommand(params)
```

where *params* is a table with two fields:

- `topNode`. (required) The table defining the behavior tree.
- `subordinateRoles`. (optional) The table defining the roles.

For example:

```
defineCommand({topNode = behavior,  
              subordinateRoles = roles})
```

In this example, the behavior tree is a table assigned to variable `behavior`, and the roles are in variable `roles`.

### 11.6.2 Using State Variables

The Lua functions in the behavior nodes (as well as Lua code outside the nodes) can save data into a global environment. (In many behavior tree implementations, this environment is called a "blackboard".) Global state can be used to pass information between nodes of a tree (although this reduces the modularity of the tree) and for saving state for a scenario save and reload. In behaviors using the behavior engine, state is saved and accessed using two functions:

- `setStateVar(name, value)`. Where `name` is a string, the name of the state variable, and `value` is any value (including tables). Does not return a value.
- `getStateVar(name)`. Where `name` is a string, the name of the state variable. Returns the value.

Inside the engine, the `checkpointState` global variable is used to store the behavior state; however, this variable should not be accessed directly in the script. The behavior engine takes care of setting the checkpoint state mode.

### 11.6.3 Getting Role Information

There are several functions in the behavior engine API related to subordinates and their assignments to roles.

- `roleForSubordinate(subordinate)`. Argument `subordinate` is a `simObject`. This function returns a string which is the name of the role assigned to the subordinate.
- `subordinatesAssignedToRole(name)`. Argument `name` is a string, the name of a role. This function returns a list (table) of `simObjects` that are assigned to the given role.
- `isRoleCritical(name)`. Argument `name` is a string, the name of a role. This function returns a Boolean indicating whether the given role is critical (that is, the `isCritical` element in the definition is true).

### 11.6.4 Variable Definitions

The behavior engine defines the following global variables for use in the behavior script:

- `_subordinates`. This is a list (table) of `simObjects`, which are the subordinates of the unit. These are the direct subordinates, not individuals in the unit. The list contains only capable subordinates that are capable of moving and shooting.

### 11.6.5 Script Validation

Although this is not part of the behavior engine API, script validation is another function that is performed by the engine. When VR-Forces loads the script, the engine reads the description tables provided in the **`defineCommand`** function and checks the script for errors. Each node description is checked to make sure that all required attributes are

present, and that all attributes present are valid attributes. It also checks the type of each attribute provided. If there are any errors, the node name and a description of the error are printed to the Warn console.

## 11.7. Behavior Integration into Lua Tasks

The behavior engine is a set of Lua files that are used in the normal VR-Forces Lua scripting mechanism. Because a behavior is, underneath, only a Lua script, all of the Lua scripting features apply:

- It is created from the Script Manager dialog
- It uses script metadata, including name, script parameters, menu configuration, applicable entity types, and so forth.
- It is managed by the entity's Task Manager; it can be part of a plan.
- And so on.

This section describes the ways in which behaviors are different from normal Lua tasks.

### 11.7.1 Script Type

When creating a behavior that will use the behavior engine, select the "Task (Behavior)" script type in the New Script dialog box (Figure 74).

Figure 74. New Script dialog box

Script ID: <auto generate>

Script Type: Task (Behavior) Language: Lua

Name:  ☐ Use Qt Designer

Extended Name:

Short Description:

☐ Description:

Version: 1

Action Categories:

- ☐ Depth control
- ☐ Lifeform Posture
- ☒ Movement
- ☐ Radio Communication
- ☐ Weapon

☐ Can Be Executed More Than Once at a Time

Tool Tip:

Menu Icon:

Command Level: Primary

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

Parameters

| Parameter | Type | Visible |
|-----------|------|---------|
|           |      |         |

☒ Make Available Based on Entity Type

Valid for Object Types

All

☐ Make Available Based on Behavior Set

☐ Make Available to Global Plans

Edit Script Preview Add Cancel

When you select this option, the template for the task is tailored to a behavior engine behavior instead of a normal task script.

The New Script dialog sets the category of the task to "movement" by default. Most unit behaviors will involve movement. However, unit behaviors do many things, and it is more likely that multiple behaviors need to run on the unit at the same time, perhaps to task different subordinates of the unit. For this reason, it is often desirable to clear the movement box.

### 11.7.2 Behavior Script Components

A behavior tree script should normally start with comments describing what the behavior does, followed by two require statements:

```
require "vrfutil"
require "behaviorEngine"
```

Next, a behavior script typically has definitions of global functions, a table of role definitions, and tables of node definitions. Finally, there is a call to `defineCommand()`.



The behavior script should not include definitions of `init()`, `tick()`, `saveState()`, or `loadState()`. The behavior script should not call `vrf:setCheckpointMode()`.

Figure 75 below is an example of a complete behavior script that prints a "hello" from each of the unit subordinates.

### 11.7.3 Graphical Display of Behavior

The behavior engine includes an experimental feature that displays the behavior tree graphically. This feature uses an installation of parts of Graphviz—in particular, the dot layout program—in the `./bin64/Graphviz` directory. The behavior engine writes a file with the name `"be_<unit name><behavior name>.gv"` into the `./bin64` directory, then runs dot on that file to produce a file `"be_<unit name><behavior name>_buffer.html"`. The engine further creates a file `"be_<unit name><behavior name>.html"`, which contains JavaScript that displays the `..._buffer.html` file and refreshes it twice per second. The behavior engine starts an instance of the Microsoft Edge browser to execute this JavaScript code. When a behavior is run on a unit and the unit's Notify level is set to Debug, the behavior engine opens an Edge browser and displays the behavior. The behavior state is updated whenever the state changes, but no more often than every 0.5 second.

In the graphical display, node types are distinguished by shapes and color. The outline of a node indicates its status:

- gray - not started
- blue - running
- dashed green - success
- dashed red - fail

Graph edges are dashed to indicate they are parallel to another branch—that is, they come out of a parallel or role node). Edges are bold blue if they go into a running node; thus the current locus of control can be seen in the sequence of bold blue edges that goes from the top node of the tree to a leaf.

Inside the node graphics, the engine displays the `nodeType`, the node name, and indicators for the presence of `doIf`, `startWhen`, `endWhen`, and `failWhen` functions. Parallel nodes have indicators when the `successPolicy` is "one" or `lefAllFinish` is "false"; selector nodes include an indicator if the mode is "fallback"—that is, values other than the default.

Task parameters and state variables are displayed in tables in the graphical output. If a node is an instance under a role node, then its private environment is displayed in the node. The values of the variables in all these cases are only displayed if they are scalars or locations; also, only the first item in a table is displayed.

Figure 76 on page 287 is an example of the graphical output.

**Figure 75. Example of Complete Behavior Script**

```
-- Hello task

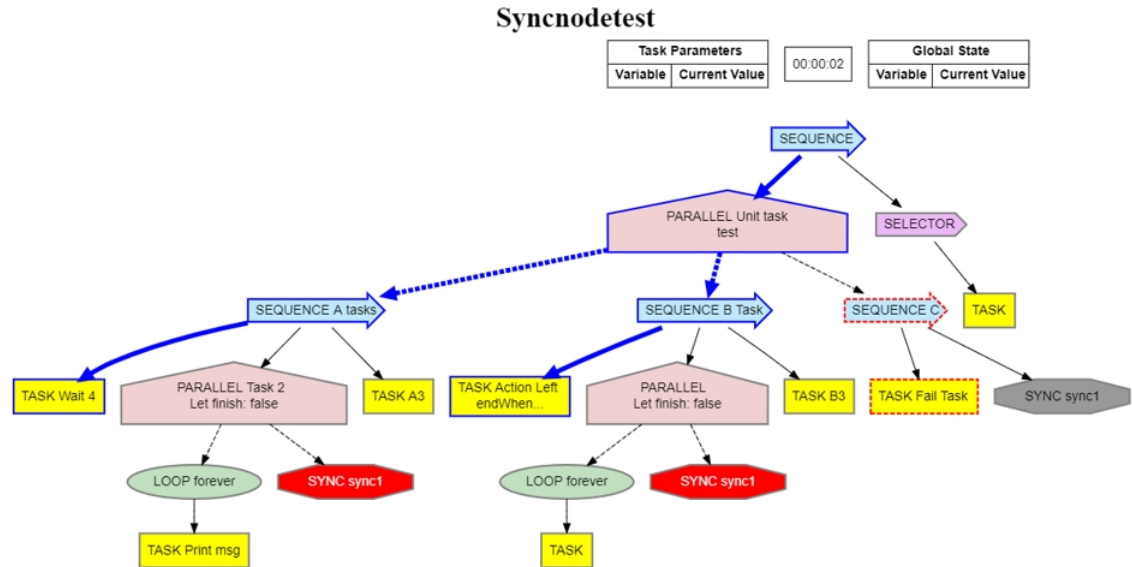
require "vrfutil"
require "behaviorEngine"
```

```
roles =
{
    {
        name = "All subs"
        ,maxInstances = #_subordinates
    }
}

-- Define behavior nodes.
ind_beh =
{
    nodeType = "task",
    name = "Print sub. name",
    task = function(node)
        printWarn("Hello from ",
            node.subordinate():getName())
        return 0
    end
}

behavior =
{
    nodeType = "role",
    role = "All subs",
    child = ind_beh
}

-- Register the roles and top node with the behavior engine.
defineCommand{
    subordinateRoles = roles,
    topNode = behavior
}
```

**Figure 76. Example of a graphical display behavior tree during execution**

### 11.7.4 Editing with ZeroBrane Studio

The ZeroBrane Lua IDE can be used to validate the behavior syntax and preview a behavior tree graphically before the behavior is run in VR-Forces.

► **To use ZeroBrane to validate the behavior syntax and preview the behavior tree:**

1. Open a Lua script from the Script Manager dialog.
2. Click the Execute button. This executes the script using the Lua interpreter built into the ZeroBrane IDE. The interpreter reports any Lua syntax errors in the script.

**Figure 77. Execute the script**

3. If there are references to VR-Forces-defined tables in the global part of the script that will be executed when the script is loaded, these will cause errors because ZeroBrane does not have the VR-Forces definitions loaded. For example, `vrf` and `this` are `nil`. Typically, in a behavior, functions and tables are only defined, not run, when the behavior script is loaded, so not many of these `nil` references will be executed. If some of them are run, they can be de-activated for the purposes of checking the script with the IDE with this code:

```
-- Causes error:
-- location = this:getLocation3D()
-- safe for checking script in editor:
if this ~= nil then
```

```
        location = this:getLocation3D()  
    end
```

If there are no Lua syntax errors in the script, and no references to undefined tables or functions, then the behavior engine checks the validity of the script (missing or incorrect node elements, and so on). If you do not have another installation of ZeroBrane, Lua, or Graphviz, the behavior engine creates the files to show the tree graphically, and launches an Edge instance (on Windows) to display it. The graphics files are saved in the project directory: *./userData/scripts*.

You can use the ZeroBrane editor to edit behavior scripts (or any Lua scripts) even without running VR-Forces. Simply select a file in the Windows File Explorer and open it using ZeroBrane, edit the script as desired, and then execute it in ZeroBrane. The Lua engine in ZeroBrane checks the Lua code for errors. Next, the behavior engine checks the node definitions for validity, and then a graph is generated.

## 11.8. Programming Practice for Behaviors

The behavior engine design for VR-Forces deliberately includes Lua code in the node definitions. This design decision means that a standard programming language is available for tackling what is often the hardest part of building behaviors—geometric calculations, situation awareness, threat assessment, and so forth—while a specialized engine handles flow of control.

The incorporation of general Lua code also means that behavior developers can write incomprehensible behaviors. On the other hand, if developers follow some guidelines and try to use the behavior engine for what it is intended, then the behaviors will be much easier to develop and maintain. In particular, the developers (and others who review the behavior) can understand much of the behavior just from the graphical display.

This section presents guidelines to help you write better behaviors.

### 11.8.1 Node Names

Node names display with every node in the graphical display. Names are not used for anything else, except in the case of sync nodes. Therefore, behavior developers should create node names that provide clear information to someone viewing the graph.

These are some guidelines for node names:

- Use multiple words to say very concisely what the node, or subtree under the node, is doing.
- Do not include the type of the node, because that is automatically printed. For example, task node names should not include "task".
- If a task node starts a VR-Forces task, use the task name.
- If the node includes a condition such as `doIf`, `startWhen`, `endWhen`, `failWhen`, or `doWhile`, try to include a brief description of the condition.

When the node names are read and displayed in the browser, they can be interpreted as HTML; therefore, do not use any special characters such as angle brackets, quotes, or anything else that might be interpreted.

### 11.8.2 Perform Single Actions in Task Nodes

The task function in a task node can be used to compute parameters, update state variables, issue Set commands, and issue subTask commands. In fact, all of these operations could be performed in the same task function. To make the behavior tree more visually understandable, however, different actions should be performed in different nodes rather than putting them all in one task function.

It is easy to test a condition in the task function and return -1 or 0 depending on the result. Instead of doing this, use a Condition node to make it clear that the purpose of the action is to test a condition.

### 11.8.3 Avoid Excessive Computation in Node Functions

All of the node functions—doIf, initFn, startWhen, task, and so on—are general Lua functions, and they may contain any amount of code. Normal coding practices should be applied: if a function is too large, factor it into pieces. Lower-level functions can be defined at the global level in the behavior script and referenced in the node function body. Predicate functions (returning a Boolean) should not in general cause side effects. (One exception to this is doWhile, which may need to increment a counting variable or something as part of the test, since there is no counting mechanism built into the node definition.) If the work accomplished by the computations is too much to concisely describe in the node name, then consider breaking the computations apart into multiple nodes.

### 11.8.4 Put Task Parameters in State Variables

The arguments to VR-Forces tasks that are started in task nodes can be computed directly in those nodes, if they are fairly simple. However, it may be useful to save copies of the arguments in state variables to ensure that they are shown in the graphical output.

### 11.8.5 Minimize Global Communication Through State

The Lua functions in the behavior tree nodes can set and test state variables. State variables can be used to communicate information between nodes. There are two reasons that this is undesirable to do this:

- The connection between nodes is hidden in the graph display, and difficult to see in the script itself. This makes the behavior more difficult to understand.
- One potential benefit of using behavior trees to represent behavior is that they can be very modular. For example, it is possible to add nodes to a sequence or parallel node without affecting the other child nodes. However, if subtrees have hidden connections in the form of state variable dependencies, some of the modularity is lost.

Uses of state variables that minimize these problems include:

- State variables that are set and used within a node. For example, a loop counter that is set and used in a loop node.
- State variables that pass information as locally as possible. For example, computing some information in a job node or a task node, and then using that information in the next node in a sequence.
- State variables that are derived from task parameters and other static information, that are computed in the first `initFn`, and that are not changed further in the script.

### 11.8.6 Define Nodes in Variables and Use These Variables for Children of Other Nodes

A behavior tree description is a nested table. This description can be written in the script with nested curly braces. However, this approach to writing scripts leads to long table definitions, which is poor practice (similar to writing long pages of procedural code). For example, the simple tree in Figure 73 on page 274 looks like this when written out.

```
top =
{
  nodeType = "parallel",
  children =
  {
    {
      nodeType = "role",
      role = "R1",
      child =
      {
        nodeType = "sequence",
        name = "Task A",
        children =
        {
          {
            nodeType = "task",
            name = "A1 wait 4",
            task = function (node)
            return
              vrf:sendTask(
                node.subordinate(),
                "wait-duration",
                {seconds_to_wait = 4})
            end
          },
          {
            nodeType = "task",
            name = "A2",
            task = function (node)
              return 0 end
          }
        }
      }
    }
  }
}
```

```
,{
  nodeType = "sync",
  name = "Event 1"
}
,{
  nodeType = "task",
  name = "A3",
  task = function (node)
    return 0 end
}
}
}
},{
  nodeType = "role",
  role = "R2",
  child =
  {
    nodeType = "sequence",
    name = "Task B",
    children =
    {
      {
        nodeType = "task",
        name = "B1 wait 6",
        task = function (node)
          return
            vrf:sendTask(
              node.subordinate(),
              "wait-duration",
              {seconds_to_wait = 6})
          end
        }
      },{
        nodeType = "sync",
        name = "Event 1"
      }
      ,{
        nodeType = "task",
        name = "B2",
        task = function (node)
          return 0 end
        }
      },{
        nodeType = "task",
        name = "B3",
        task = function (node)
          return 0 end
        }
    }
  }
}
```

```
    }  
  }  
}  
}
```

When written using variables to represent each node, all of the nesting is unnecessary, and the code looks like this:

```
sync1 =  
{  
  nodeType = "sync",  
  name = "Event 1"  
}  
A1 =  
{  
  nodeType = "task",  
  name = "A1 wait 4",  
  task = function (node)  
    return vrf:sendTask(  
      node.subordinate(),  
      "wait-duration",  
      {seconds_to_wait = 4})  
  end  
}  
A2 =  
{  
  nodeType = "task",  
  name = "A2",  
  task = function (node) return 0 end  
}  
A3 =  
{  
  nodeType = "task",  
  name = "A3",  
  task = function (node) return 0 end  
}  
SeqA = {  
  nodeType = "sequence",  
  name = "Task A",  
  children = {A1, A2, sync1, A3}  
}  
R1 =  
{  
  nodeType = "role",  
  role = "R1",  
  child = SeqA  
}
```



```
B1 =
{
  nodeType = "task",
  name = "B1 wait 6",
  task = function (node)
    return vrf:sendTask(
      node.subordinate(),
      "wait-duration",
      {seconds_to_wait = 6})
  end
}
B2 =
{
  nodeType = "task",
  name = "B2",
  task = function (node) return 0 end
}
B3 =
{
  nodeType = "task",
  name = "B3",
  task = function (node) return 0 end
}
SeqB =
{
  nodeType = "sequence",
  name = "Task B",
  children = {B1, sync1, B2, B3}
}
R2 =
{
  nodeType = "role",
  role = "R2",
  child = SeqB
}

top =
{
  nodeType = "parallel",
  children = {R1, R2}
}
```

The behavior illustrated here is simplistic, but it also shows opportunity for code reuse that can be used in more complex behaviors. Note that the sync node description is exactly the same in two places in the tree, so it is only described once. Other nodes are

highly similar; for similar nodes that are repeated in the tree, a creator function could be defined. For example, the task nodes A1 and B1 are the same except for their name and wait time. They could be defined this way:

```
function createWaitTask(name, time)
    local desc = {}
    desc.nodeType = "task"
    desc.name = name
    desc.task = function (node)
        return vrf:sendTask(node.subordinate(),
            "wait-duration", {seconds_to_wait = time})
    end
    return desc
end
A1 = createWaitTask("A1 wait 4", 4)
B1 = createWaitTask("B1 wait 6", 6)
```

### 11.8.7 Put Static Options in dolf-Switched Nodes

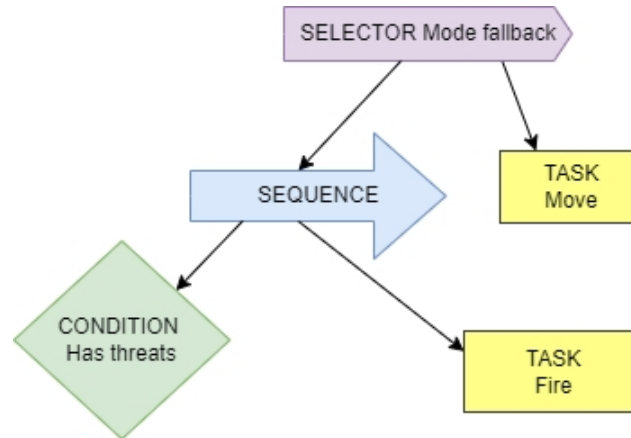
Make use of dolf functions to disable parts of the tree that are not relevant for the initial conditions. The dolf condition can test anything that is set before the start of the behavior: taskParameters, role assignments, instance numbers in roles, the unit type, unit or subordinate parameters, subordinate function, or anything derived from this type of information. As a behavior developer, you should avoid using condition nodes to select behavior branches at run time when the condition information is available at the beginning of the behavior; such selections make the runtime structure of the behavior needlessly complex. Instead, use dolf functions near the top of the tree to select behavior branches. With this approach, each branch handles one behavior condition and should be much simpler to design.

### 11.8.8 Push Conditions out of Functions and into the Tree

All flow of control in a behavior should ideally be defined in the behavior tree architecture. Although it is possible to put a complex, branching flow of control in the Lua code in the nodes, this hides the flow from the graph view and makes the behavior harder to understand. For example, a task node might have a task function written like this:

```
task = function(node)
    if getStateVar("threats") then
        return vrf:sendSubtask(node.subordinate(), "fire...")
    else
        return vrf:sendSubtask(node.subordinate(), "move...")
    end
end
```

This key branch in the logic should be implemented as a selector node in fallback mode, as shown in Figure 78.

**Figure 78. Explicit branching on conditions in the behavior tree**





## 12. Using Qt Designer for Script Dialog Boxes

You can let VR-Forces create dialog boxes for scripted tasks and sets using default layout rules or you can design the dialog boxes yourself using Qt Designer.

|                                                                           |     |
|---------------------------------------------------------------------------|-----|
| 12.1. Introduction to Using Qt Designer for Scripts .....                 | 298 |
| 12.2. Selecting Qt Designer for Dialog Box Layout .....                   | 298 |
| 12.3. Adding Parameters When Using Qt Designer .....                      | 298 |
| 12.3.1 Specifying Choices for the Choice Parameters in Qt Designer .....  | 301 |
| 12.3.2 Specifying Categories for a Simulation Object in Qt Designer ..... | 302 |
| 12.4. Adding Design Elements .....                                        | 303 |

## 12.1. Introduction to Using Qt Designer for Scripts

When you create a scripted task or set that requires input from a user, VR-Forces can automatically build a dialog box with fields for each parameter. These dialog boxes are built using a default layout scheme. This layout scheme is functional, but it may not always capture relationships between different parameters, particularly dependencies. Also, the resulting dialog boxes may not be as esthetically pleasing as well designed dialog boxes should be. As an alternative to automatic dialog box creation, you can use Qt Designer to lay out dialog boxes. This capability is integrated into the script creation process.

The VR-Forces graphical user interface is built using Qt, a cross-platform GUI SDK. Qt Designer is a utility that developers can use to quickly lay out GUI design elements. It is beyond the scope of this manual to fully explain how to use Qt Designer. This chapter provides enough information about it to help you start designing with the utility.

## 12.2. Selecting Qt Designer for Dialog Box Layout

By default, VR-Forces automatically builds dialog boxes for scripted tasks and sets.

- To specify use of Qt Designer for dialog box layout, in the New Script or Edit Script dialog box, select the Use Qt Designer check box.

## 12.3. Adding Parameters When Using Qt Designer

When you create or edit a script, you can add parameters in the New Script or Edit Script dialog box. If you select Qt Designer for dialog box layout, any parameters that you have already defined in the New Script dialog box are implemented in Qt Designer using the default dialog box layout (which you can change). However, once you select Qt Designer, you cannot add any more parameters in the New Script dialog box. You can only add parameters in Qt Designer.

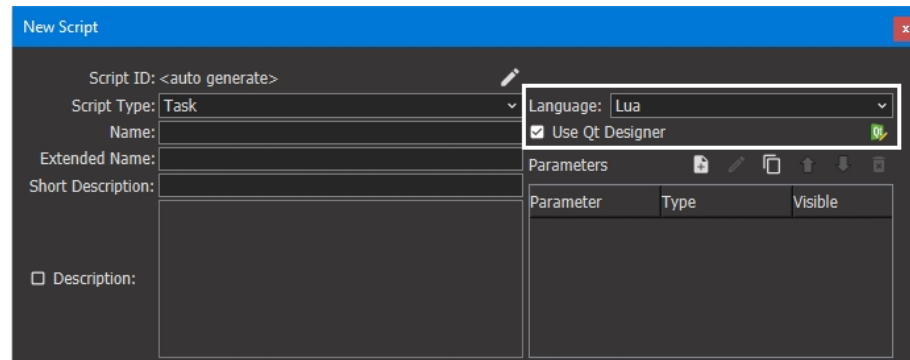
If you decide to use Qt Designer before you add parameters in the New Script dialog box, then all parameters are added using Qt Designer.

 If you lay out a dialog box using Qt Designer and then unselect the Use Qt Designer check box, all of the layout information that you added in Designer is lost, except that if you rearranged the order of parameters, those changes affect the order in which the parameters are listed in the Edit Script dialog box. Any parameters that you added in Designer are still valid and part of the script.

- **To add parameters in Qt Designer:**

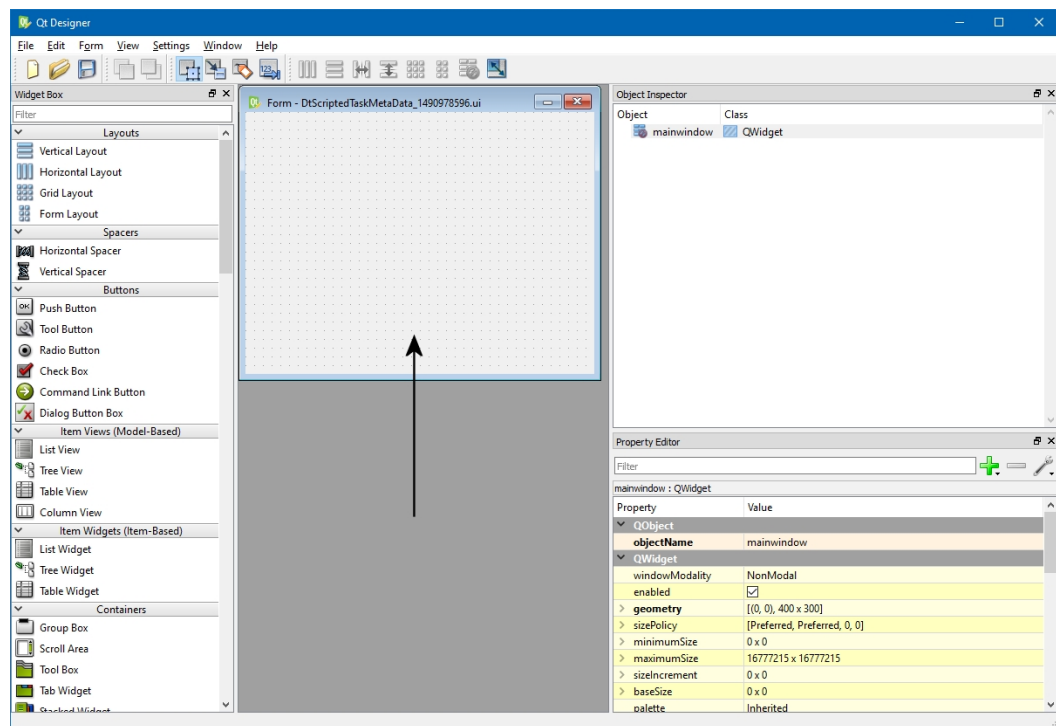
1. In the New Script or Edit Script dialog box, select the Use Qt Designer check box (Figure 79).

Figure 79. New Script dialog box



- Click the Qt button (  ) to the right of the Qt Designer check box. Qt Designer opens (Figure 80). If the script already has parameters, it displays a default dialog box layout for the script. If there are no parameters, it displays a blank canvas.

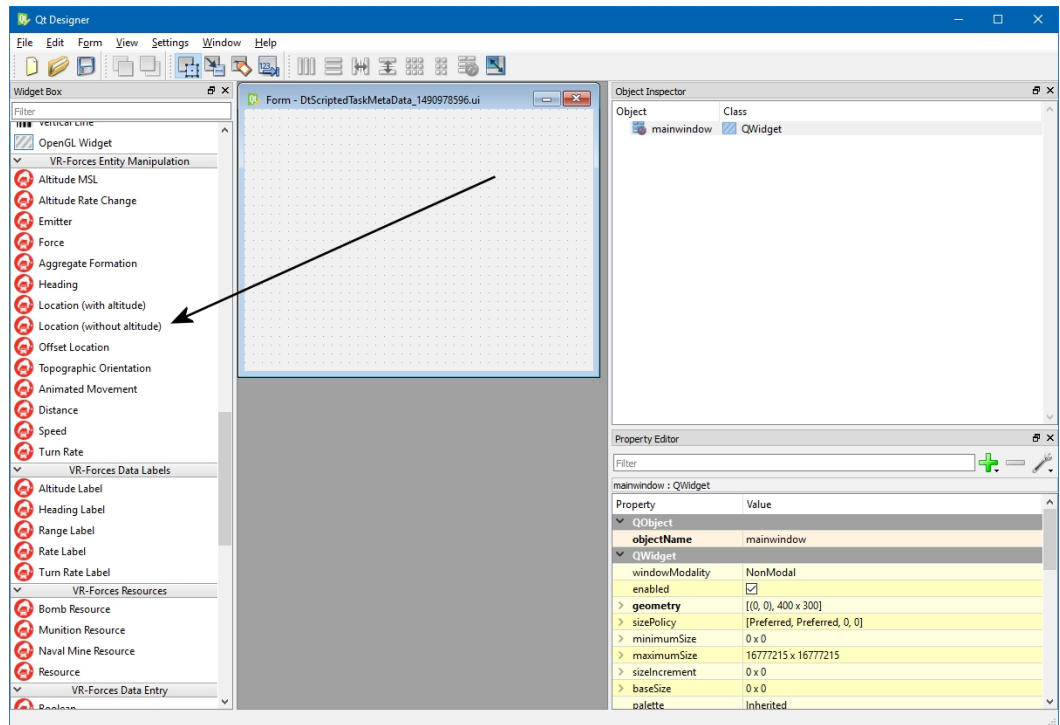
Figure 80. Qt Designer with blank canvas



 You must use the canvas that is created when you open Designer. If you close it and open a new one from the File menu, it might not work.

- Scroll through the widget box until you see the list of widgets for VR-Forces script parameters (Figure 81).

Figure 81. VR-Forces widgets



4. Drag a parameter onto the dialog box canvas.
5. In the Property Editor, specify the required and optional information for the parameter type, such as a label, tooltip, or default value.

All of the parameters have these properties:

- **objectName**. The name to be used by the Lua script for this parameter. For example, if you give a parameter the name **taskDestination**, in the Lua code for the script, the parameter will be accessed as **taskParameters.taskDestination**.
- **label**. This is the text used on the dialog box that is generated for the task. If you leave this field blank, the parameter name is used.
- **toolTip**. Optional text for a tool tip.
- **indent**. The number of pixels to indent the parameter on the task dialog box. This allows you to apply some formatting to the dialog box.
- **isVisible**. Specifies that the parameter be included on the script dialog box. (You might want a parameter to be available in the code, but not have it set by a user when the script is assigned.)

Many parameters let you specify a default value or state.

Some parameters let you specify a range for the acceptable values:

- **rangeBottom**. The lowest acceptable value for this parameter.
- **rangeTop**. The highest acceptable value for this parameter.



The Boolean parameter type lets you create a check box. You can specify that the default be checked or unchecked.

6. Add design elements such as dividers and labels as desired.

 Nothing in Qt Designer will prevent you from adding non-VR-Forces properties. However, you cannot use non-VR-Forces properties in scripts.

7. Choose a layout option on the Form menu. Doing so preserves your layout. If you do not select a layout option, VR-Forces might rearrange the layout when it displays the dialog box for the script.
8. In Qt Designer, choose **File > Save** or click the Save button. If this is a system script, Qt Designer saves a UI file with the layout information. If this is a scenario script, the layout information is saved in the scenario.
9. Close Qt Designer.

### 12.3.1 Specifying Choices for the Choice Parameters in Qt Designer

The Choice parameters require a list of choices, which are added to the dialog box as option buttons or as a list. You can add choices, edit them, and delete them.

Option buttons have a property called **buttonGroup**. Buttons within the same button group are mutually exclusive. If you add more than one group of option buttons to a script, you must edit the **buttonGroup** property to make sure that the different groups have a different value for the property.

#### ► To add a drop-down list of choices:

1. From the VR-Forces Data Entry section, drag the Choice (Drop-Down List) widget onto the canvas.
2. In the Property Editor, click the Value column for the **values** property. The text changes to Change String List.
3. Click Change String List. The Dialog - Qt Designer dialog box opens. Add a string for each choice you want to make available:
  - a. Click New. A text box is added to the list.
  - b. Type the string you want to be displayed.
  - c. Repeat this procedure for each string you want to be in the drop-down list.
  - d. Click OK. The dialog box closes. The strings you entered are listed in the Value column for the values property.

#### ► To add option button choices:

1. From the VR-Forces Data Entry section, drag the Choice (Multiple Option Buttons) widget onto the canvas. It creates a widget with one button that has the text Radio Button.
2. In the Property Editor, click the Value column for the **values** property. The text changes to Change String List.

3. Click Change String List. The Dialog - Qt Designer dialog box opens. Add a string for each choice you want to make available:
  - a. Double-click the text Radio Button to make it editable.
  - b. Change the text to something meaningful for this parameter.
  - c. Click New. A text box is added to the list.
  - d. Type the string you want to be displayed for the next option button.
  - e. Repeat steps c and d for each option button you want.
  - f. Click OK. The dialog box closes. The strings you entered are listed on the canvas and in the Value column for the values property.

### 12.3.2 Specifying Categories for a Simulation Object in Qt Designer

The Simulation Object parameter lets you build a list of categories on which to filter the display of simulation object types in a dialog box.

► **To add a category to the simulation object list:**

1. Double-click the Filter widget on the form. A dialog box opens. The top line reads Filters and it has several icons.
2. Click the Add button () on the Filters line. The Choose Filter dialog box opens.
3. Select a filter from the Filters list.
4. Click OK.
5. Repeat the procedure to add as many filters as you want.
6. Click OK.

#### Editing the Category List for a Simulation Object Parameter

You can edit the list of filter categories for a Simulation Object parameter. The edit operation is essentially a replacement of the current category with a new one. You can achieve the same effect by deleting the current category and adding a new one.

► **To edit the list of filter categories in a Simulation Object parameter:**

1. Double-click the Filter widget on the form. A dialog box opens. The top line reads Filters and it has several icons.
2. In the window, select the filter that you want to edit.
3. Click the Edit button () on the Filters line. The Choose Filter dialog box opens.
4. Select the new filter category.
5. Click OK. The original category is removed and the new one is added.
6. Click OK.

► **To delete a category from a Simulation Object parameter:**

1. Double-click the Filter widget on the form. A dialog box opens. The top line reads Filters and it has several icons.
2. In the window, select the filter that you want to delete.

3. Click the Delete button (). The category is removed from the list.
4. Click OK.

## 12.4. Adding Design Elements

You can add design elements such as dividers, group boxes, and indentation. You do so by dragging the appropriate widgets from the list onto the canvas and setting their properties. Please see the Qt Designer documentation for details about how to design dialog boxes.