MÄK VR-Forces 2.0-ngc Release Notes

This release note contains the following release-specific information for VR-Forces Release 2.0-ngc:

Document ID: VRF-2.0-ngc-3-011102

Systems Supported

VR-Forces 2.0-ngc is available for the platforms listed in Table 1. For toolkit users, application code must be built with the indicated compilers in order to link to VR-Forces libraries.

Table 1: Platforms supported

Platform	Compiler
SGI IRIX6.5	MipsPro7.2.1 C++ compiler (available from SGI)
PC with Red Hat Linux 6.0 (Linux 2.2.16-3)	GNU C++ (g++) distributed with Red Hat distribution. The command: `g++ -v` returns: `gcc version egcs-2.91.66 19990314/Linux (ecgs-1.1.2 release)`
PC with Red Hat Linux 7.1	gcc version 2.96
PC with Windows NT/2000	Microsoft Visual C++ 6.0, Service Pack 5

System Requirements for PCs

Minimum system requirements are:

- Microsoft Windows NT 4.0/2000
- Intel Pentium compatible CPU, 300 Mhz or better
- RAM: 128MB. 256-512 recommended. VR-Forces loads a copy of the terrain database for each front-end and back-end loaded.
- Hard disk space - Approximately 160 MB for complete installation of application and toolkit. Approximately 90 MB for application alone. Most of this is for the terrain databases shipped with VR-Forces. The class reference documentation for the toolkit uses 59 MB. You may need considerably more space depending on the databases you plan to use.
- Two or more multiples of the amount of RAM for virtual memory. Running VR-Forces in HLA may require increased amounts of virtual memory.

Linux Support

VR-Forces will run on Red Hat Linux 6.0 and 7.1. To recompile VR-Forces on Linux 7.1, you need to get the Linux 7.1 version of VR-Link and you need to use the Linux compiler's backward compatibility flag. Versions of VR-Forces built against Linux 7.1 are not compatible with the DMSO RTI NG.

Online Help

The online help is in HTML format and uses the default browser for your computer. For all online help features to work, you must enable Java and Javascript in your browser.

Network Compatibility

HLA only

VR-Forces 2.0-ngc was built against VR-Link 3.7-ngc and is compliant with:

- RPR-FOM 0.5, 0.7, 0.8, 1.0, and 2.0
- MÄK RTI 1.3.5-ngc
- DMSO RTI NG v1, v2, v3.0, v3.1, v3.2, and 4.0.

VR-Forces 2.0-ngc is compatible with applications that use earlier versions of VR-Link if they support versions of the RPR FOM listed above.

DIS only

VR-Forces 2.0-ngc supports DIS 2.0.4, IEEE 1278.1, 2.1.4, and IEEE 1278.1a, and can therefore interoperate with DIS applications of any of these versions.

RPR FOM Versions Supported

VR-Forces 2.0-ngc has built-in support for versions 0.5, 0.7, 0.8, 1.0, and 2.0 of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, VR-Forces 2.0-ngc uses RPR FOM 1.0.

If you are building an application with the VR-Forces toolkit and you want to specify a version of the RPR FOM through code, pass the version number (0.5, 0.7, 0.8, or 2.0) to the *DtRprFomMapper* constructor and pass the resulting object to the *DtExercise-Conn* constructor. Also, make sure you are using a federation execution name that corresponds to the right FED file. For example:

```
DtExerciseConn conn("VR-Link08", "MyApp", new DtRprFomMapper(0.8));
```

RTI Support

The MÄK RTI is included on all MÄK product CDs. It is link-compatible with the DMSO RTI NG. VR-Forces 2.0-ngc has been tested with DMSO RTI NG v1, v2, v3.0, v3.2, and 4.0.

If you use the MÄK RTI, be sure that VR-Forces can find your FED file, and optional *rid.mtl* file, by putting them in the directory from which you are running, or by setting the environment variable RTI_CONFIG to the directory that contains them.

RTI Library Names

VR-Forces 2.0-ngc expects library names that match those of DMSO RTI NG v3.2. If you run VR-Forces 2.0-ngc with DMSO RTI NG v3.2, both the UNIX and PC versions work without any modification. If you want to run VR-Forces 2.0-ngc with DMSO RTI NG v1 or v2, you must make the following changes:

- For UNIX, you must use the libraries *libRTI-NGv2.so* or *libRTI-NGv1.so*, which are supplied by MÄK in the *./lib* directory. These libraries correspond to DMSO RTI NG v1 and v2. To use VR-Forces 2.0-ngc with RTI NG v1 or v2, copy the appropriate library to the DMSO RTI NG *lib* directory and rename it *libRTI-NG.so*. (You might want to rename the v3.x *libRTI-NG.so* library first.)
- For PCs, rename the libraries that come with RTI NG v1 or v2 to the names used by the RTI NG v3.2 libraries (*libRTI-NG.lib* and *libfedtime.lib*).

New Features

This release of VR-Forces has the following new features:

- A new API architecture that provides the ability to run multiple, independently addressable front-ends and back-ends. These changes have the following effects:
  - New executable names and startup options
  - New project files
  - New examples
  - New log files
  - The ability to configure a scenario across multiple back-ends from a single GUI
  - New scenario files
- Improved database support, including the following:
  - Improved support for DTED databases
  - Improved support for terrain database records
  - Support for extent records
  - New options for the tdbtool
- Additional entities and improvements to existing models:
  - Additional formations, improved formation handling, ability to configure your own formations.
  - Support for artillery entities and fire for effect tasks
  - Improved behavior for rotary-wing entities, including terrain avoidance, and new entity parameter database parameters
  - Support for joy-stick control of fixed-wing and rotary-wing entities
- Miscellaneous changes:
  - Online help.

New API Architecture

The new VR-Forces architecture separates simulation (back-end) and display (front-end) processes into separate executables that communicate through state update messages (for example, in DIS, PDUs). This feature allows you to spread the processing burden of simulation over multiple processes on one or more computers. It also allows you to spread the ability to create and manage scenarios among multiple users on multiple computers. With this feature, you can easily create exercises that support collaborative interaction amongst participants.

The VR-Forces processes identify each other by their site:application IDs. Each front-end and back-end must have a unique ID.

This section describes the changes from the user perspective and the mechanics of using VR-Forces. "Using the New API" on page 29 describes the new architecture for developers who use the VR-Forces toolkit.

New Executables and Startup Options

VR-Forces has separate executable files for the graphical user interface (GUI, or front-end) (*vrfGUI.exe*) and the simulation engine (or back-end) (*vrfSim.exe*). You can run just one front-end and one back-end on one computer, communicating with each other (combined mode), or you can run multiple front-ends and back-ends on as many computers as you want.

Running VR-Forces in Combined Mode

In combined mode, you run one front-end and one back-end on one computer.

! In combined mode, you cannot reliably interact with other VR-Forces executables. Therefore, do not start VR-Forces in combined mode as a simple way to start up a front-end and back-end on several computers.

To run in combined mode, use the new startup option `-z` as follows:

```
front-end -z back-end [options]
```

Using the executables supplied by MÄK, the command is:

```
vrfGUI.exe -z vrfSim.exe
```

This command is in the batch file *./bin/vrf.bat* (on PCs), which you can use to simplify starting up a front-end and back-end together.

On PCs, the `-C` startup option starts a console for the front-end only. Each back-end automatically displays a console window. If you specify the -C option, the output of the console window is automatically saved to the file *vrfGUI.log*, in the same directory as the executable. The back-end always creates a separate log file for its console window output, *vrfSim.log*, in the same directory.

Starting Multiple Front-Ends and Back-Ends

If you start up back-ends and front-ends independently, you need to specify a unique site ID and application number for each one. You assign the site ID with the `-s` startup option and the application number with the `-a` startup option, for example:

```
vrfSim.exe -s 2 -a 3
vrfSim.exe -s 3 -a 4
vrfGUI.exe -s 3 -a 2
vrfGUI.exe -s 2 -a 2
```

The default site and application numbers for the front-end are site 1, application 3. The default site and application number for the back-end are site 1, application 4.

The site and application number are required for HLA and DIS.

DIS only

Application numbers must be in the range 3000-4000.

Working with Multiple GUIs

If you run VR-Forces with multiple GUIs, it behaves as follows:

- If you load a database in one GUI, it loads in all the GUIs
- If you load a scenario in one GUI, it loads in all the GUIs
- Any object that you create in any of the GUIs is visible in the others and can be managed from them
- When you close a scenario in one GUI, it closes in all of them
- While the scenario is running, objects can be created or destroyed from any of the GUIs.

Because actions in one GUI affect all other GUIs, if you work in an environment that has multiple VR-Forces users, who are not participating in the same exercise, you should configure your exercise to use a UDP port other than the default. In DIS, use the `-P` startup option to specify the port. In HLA, if you are using the MÄK RTI, change the **RTI_udpPort** parameter in the *rid.mtl* file. If you are using the DMSO RTI, set the **RtiExecutiveMulticastDiscoveryEndpoint** parameter in the *rti.rid* file.

Specifying the Back-End to Use for Entities and Objects

If you run VR-Forces with multiple back-ends, you can specify the back-end on which an entity or object will be simulated. Each entity and object that you create is simulated on the back-end listed in the Back-End list on the Information Bar when the object is created.

> To change the back-end on which new entities and objects are simulated, select a back-end from the Back-End list.

When you save a scenario, VR-Forces saves the relationship between an object and a back end in the *scenario_name.omp* file. You cannot dynamically change the back-end on which an existing entity or object is simulated. However, you can edit the object/back-end associations in the *.omp* file.

When you load a scenario, it uses the information in the *.omp* file to assign the creation of entities and objects to particular simulation back-ends. If you load a scenario in combined mode, the *.omp* file is ignored, and the creation of all entities and objects specified in the order of battle file is assigned to the single back-end.

Memory Requirements for Multiple Back-ends

VR-Forces loads a copy of the terrain database for each front-end and back-end that is running. If you use a large database with multiple front-ends and back-ends, we recommend that you run no more than one front-end and back-end per computer to avoid performance problems due to low memory.

Startup Options

VR-Forces has the following new startup options:

- `-v ver` sets the DIS protocol version contained in outgoing PDUs, where `ver` is one of the following:
  - 4 - DIS 2.0.4
  - 5 - IEEE 1278.1
  - 6 - DIS 2.1.4 (IEEE 1278.1a)
- `-z` indicates that you are running VR-Forces in combined mode.

Object Parameters Databases

To support the new VR-Forces architecture, the entity parameters database has been replaced with two new object parameters databases: *./config/vrfGui.opd* and *./data/parameters/vrfSim.opd*. These databases have been expanded to include control object parameters as well as entity parameters. These new parameters allow VR-Forces to understand the objects created across multiple GUIs and back-ends. Inheritance of parameters, as described in *VR-Forces User's Guide* continues to apply to the organization of the parameters database.

If you add a new entity to VR-Forces, you must add entries to the *vrf.ent* file (to put it on the Create Entity menu), the *vrfSim.opd* file, and the *vrfGui.opd* file. The *vrfGui.opd* parameter database contains an entry for each entity type in *vrfSim.opd*. The echelon-level and category strings in the entries in the two files must be identical. The differences in the entries are because the objects created in the GUI are not configured to perform simulation the way they are in the simulation back-end.

The *vrfGuil.opd* File

The *vrfGui.opd* file contains the basic information about each object type that the front-end needs to create and name objects and communicate with the back-ends. If you add new entity types to VR-Forces, you need to add an entry to *vrfGui.opd* for the new entity. The following is a typical entry in *vrfGui.opd*:

```
;;
;; Entry for generic US attack helicopter platform
;;
(US-generic-attack-helicopter-platform
   (parameter-type "vrf-base-object-param")
   (object-type  1 (1 2 225 20 -1 -1 -1))
   (category "USAtRW")
   (bounding-volume
      (local-bvol  8.8400000 1.6400000 2.22000000)
      (offset  0.000000 0.000000 0.750000)
   )
)
```

The **entity-type** parameter has been replaced by the **object-type** parameter, which extends the DIS/RPR entity type 7 byte enumeration (see IST-CR-99-04, "Enumeration and Bit Encoded Values for Use with Protocols for Distributed Interactive Simulation Applications"). A new field has been added to the enumeration (the leading '1' before the (1 2 225 20 -1 -1 -1) in the example above). This field is referred to as the super type (representing the super-set of possible entity-types). It distinguishes between individual, aggregate, and non-entity objects. Values for this new field, along with extensions to the existing entity type enumerations, are in *objTypeEnums.h*.

Table 2 lists values for the **object-type** enumeration:

Table 2: Object type field values

Enumeration	Meaning
0	Other
1	Individual (platform entities, individual control objects)
2	Aggregate
3	Pseudo Aggregate
4	True Aggregate

The *vrfGui.opd* file has object-type entries similar to the following:

```
(M1A2
   (parameter-type "ground-vehicle-param")
   (object-type  1 (1 1 225 1 1 3 -1))
   (…)
)
(pseudo-aggregate-entity
   (parameter-type "moving-object-param")
   (object-type  3 (11 -1 -1 -1 -1 -1 -1))
   (…)
)
(phase-line-parameters
   (parameter-type "vrf-base-object-param")
   (object-type  1 (17 -1 -1 1 -1 -1 -1))
   (…)
)
```

The *vrfSim.opd* File

The *vrfSim.opd* file is comparable in format and function to the *default.epd* file used in VR-Forces 1.x, except that it now includes objects as well as entities. Except as noted later in these release notes, entities use the same parameters as documented in *VR-Forces User's Guide*. If you add new entities to VR-Forces, you need to add an appropriate parameter block in *vrfSim.opd*. Use an existing parameter block as a template for the new entity. Refer to *VR-Forces User's Guide* for details.

For example, the M1A2 entry in *vrfSim.opd* is:

```
(M1A2
   (parameter-type "ground-vehicle-param")
   (object-type  1 (1 1 225 1 1 3 -1))
   (echelon-level "")
   (category "M1A2")
   (…)
)
```

The entry for the M1A2 in *vrfGui.opd*:

```
(M1A2
   (parameter-type "vrf-base-object-param")
   (object-type  1 (1 1 225 1 1 3 -1))
   (echelon-level "")
   (category "M1A2")
   (bounding-volume
      (local-bvol  4.000000 1.800000 1.440000)
      (offset  0.000000 0.000000 -1.440000)
   )
)
```

The **object-type**, **echelon-level**, and **category** values are the same. The **parameter-type** value is different.

New Log Files

In parallel with the two new executables, VR-Forces has new log files, *vrfGui.log* and *vrfSim.log*. They log the messages for *vrfGui.exe* and *vrfSim.exe* respectively.

New Scenario Files

The overlay file (*.ovl*) is no longer used, except for reading VR-Forces 1.x scenarios. Control objects are now listed in the order of battle file (*.oob*). If you load a VR-Forces 1.x scenario and then save the scenario, the new scenario file will no longer reference the overlay file. The object map file (*.omp*) maps objects to back-ends.

The Object Map File

The object map file maps objects to back-ends. It designates which simulation back-end is responsible for creating each object in the simulation. The following sample entries from an object map file map a control object named “Waypoint 4” to the back-end identified as site ID 2, application number 2, and an entity named “2 M1A2, 1 Force” to the back-end identified as site ID 1, application number 3.

```
(map-entry
   (address  2 2)
   (name  "Waypoint 4")
)
(map-entry
   (address  1 3)
   (name  "2 M1A2, 1 Force")
)
```

You can refer to the object map file if you forget the site:application numbers for the back-ends required for a particular scenario. To load a scenario, you need to have back-ends with the same site:application numbers as those listed in the object map file to run the scenario again. The scenario will not load unless all back-ends specified in the object map file are present on the network.

If you load the scenario from the GUI in combined mode, the object map file is ignored. All objects specified in the order of battle file are created in the single back-end.

Loading Scenarios

When VR-Forces loads a scenario, it must distribute information about objects to all participating VR-Forces processes. It must also send plan information to the back-ends simulating the various entities. Depending on the number of objects in your scenario, this process may take some time. Therefore, we recommend that you not start a scenario until it appears that message traffic in the console window has finished. If you run the scenario while plans are still being read in, there could be some delay in the start of entity task completion.

Reading VR-Forces 1.x Scenarios

VR-Forces 2.0-ngc can read scenarios created in VR-Forces 1.x, but you must edit the scenario file first.

> To make a VR-Forces 1.x file readable, change the **Parameter-Database** argument in the *.scn* file to point to *vrfsim.opd* instead of *default.epd*.

If you make changes to a legacy scenario and save the changes, VR-Forces saves the scenario in the 2.0-ngc format. Any control objects that were originally stored in the 1.x .ovl file are saved to the 2.0-ngc *.oob* file, along with any entities in the original *.oob* file. The new *.scn* file will no longer reference the old *.ovl* file.

New Entities and Improvements to Existing Models

VR-Forces 2.0-ngc has the following improvements for entities:

- Additional formations, improved formation handling, ability to configure your own formations
- Support for artillery entities and fire for effect tasks
- Improved behavior for rotary-wing entities, including terrain avoidance, and new parameters
- Support for joy-stick control of fixed-wing and rotary-wing entities
- New entities based on existing models.

Improved Support for Formations

This release adds wedge and vee formations and generally improves the behavior of entities when they move in formation. Formations are now defined in formation files in the directory *./data/parameters/formation*. Formation files have the extension *.frm*. In addition to the named formations provided by MÄK, VR-Forces supports user-defined formations.

The vee and wedge formations provided with VR-Forces place the aggregate leader at the end of one of the arms of the formation. If an aggregate has fewer than seven or eight members, it will not be symmetrical.

Configuring Formations

The object parameters database lists the formations available to aggregate entities in the **pseudo-aggregate formation controller** block. Each formation is identified by its name and the relative path to the formation file, for example:

```
(pseudo-aggregate-formation-controller
   (component-descriptor-type "formation-controller-descriptor")
   (component-type  "pseudo-aggregate-formation-controller")
   (attached-to-list
   )
   (formation-list
      (column-formation
         (formation Column)
         (formation-file
            (filename "..\data\parameters\formation\column.frm")
      )
      (line-formation
         (formation Line)
         (formation-file
            (filename "..\data\parameters\formation\line.frm")
         )
      )
...
```

In addition to listing formations, the **pseudo-aggregate formation controller** block has two parameters, **formation-tolerance** and **control-speed**. These two parameters work together to control how VR-Forces responds when entities drift out of formation.

The **formation-tolerance** parameter defines the number of meters that an entity can be away from its assigned place in the formation before it is considered to be out-of-formation. The **control-speed** parameter is set to True or False. If it is set to True, VR-Forces slows the speed of the aggregate to allow members who are out-of-formation to catch up. The aggregate slows down if the number of entities that are out-of-formation (based on the **formation-tolerance**) exceeds thresholds of 20%, 40%, and 60%, getting progressively slower as higher thresholds are passed. If **control-speed** is set to False, aggregate entities do not slow down when members are out-of-formation.

Formation Files

A formation file specifies the name of the formation and an entry for each entity in the default formation definition. Each entry specifies a **promotion-id**, a **leader-promotion-id**, and a **position-offset**, where:

- The **promotion-id** specifies the order in which entities are promoted to aggregate leader if the leader is incapacitated.
- The **leader-promotion-id** specifies the aggregate member from whom the entry's position will be offset.
- The **position-offset** specifies how many meters behind, to the right, and above, the entry will follow its leader.

The following is a portion of the formation file for a column formation:

```
(column-formation
   (entry-0
      (promotion-id  0)
      (leader-promotion-id  -1)
      (position-offset  0.000000 0.000000 0.000000)
   )
   (entry-1
      (promotion-id  1)
      (leader-promotion-id  0)
      (position-offset  -25.000000 0.000000 0.000000)
   )
   (entry-2
      (promotion-id  2)
      (leader-promotion-id  1)
      (position-offset  -25.000000 0.000000 0.000000)
   )
   (entry-3
      (promotion-id  3)
      (leader-promotion-id  2)
      (position-offset  -25.000000 0.000000 0.000000)
   )
```

In a column, each entity follows the entity in front of it by 25 meters. Therefore, the **leader-promotion-id** for each entry is the **promotion-id** of the entity in front of it, and each entry has a **position-offset** of -25 meters.

The number of entries in a formation file is finite. However, it is possible to place more entities into an aggregate than there are entries in a formation file. If this happens, VR-Forces increments the **promotion-id** of the last entry in the formation file and assigns the new entity the same **position-offset** as the last entry in the formation file. For a simple formation like a column, this works well. If you are using a more complex formation, this mechanism might create unintended relationships among entities.

Creating A User-Defined Formation

You can create formations other than those provided by VR-Forces. A user-defined formation must have the name UserFormation*x*, where *x* is a number from one to as many formations as you define. Using the sample user formation supplied with VR-Forces as your template, place as many entries as you want in the formation file and specify their offsets.

To specify a user-defined formation in the VR-Forces GUI, enter the formation name in the Formation field of the Set Formation dialog box. User-defined formations are not included in the list of formations.

We recommend that you configure your formations so that entities follow the leader, rather than the next closest member of the formation (as is done in column formation.) This ensures that formations have the best chance of maintaining their formation as they maneuver.

Fire-for-Effect Task

This release adds three tasks for use with artillery entities:

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

To assign a fire-for-effect task:

1. Select the entity to which you want to assign the task.
2. Choose Entities → Retask → Fire-for-Effect on Entity (or Location or Target). The appropriate dialog box opens.
3. Identify the target as follows:
   - For FFE Entity, select an entity in the list, or select it in the map.
   - For FFE Location, click a location in the terrain, or enter the X and Y coordinates.
   - For FFE Target, select a waypoint from the list, or select it in the map.
4. Specify the number of rounds.
5. Optionally, specify the height above the terrain.
6. Click OK.

Notes:

- The Fire-for-Effect tasks are listed on the Retask menu for all entities; however, they are only meaningful when assigned to artillery entities.
- When you choose Fire-for-Effect on Entity, VR-Forces notes the location of the entity and fires on that spot. If the targeted entity moves, the artillery does not change the location at which it is firing.

Artillery entities have maximum and minimum ranges. However, these ranges are not indicated visually on the map. If you have a console window open while you are running VR-Forces, it prints messages when you select targets that are outside of the entity's range.

Improved Rotary-Wing Entities and New Parameters

Table 2-1 lists new and revised rotary-wing pilot parameters for the move-to task. Table 2-2 lists new and revised rotary-wing pilot parameters for following routes.

Table 2-1: Rotary-wing move-to pilot parameters

Parameter	Description
default-aggressiveness	The acceleration level used by the helicopter in maneuvers is scaled by this value, which is restricted to lie in the rage 0 to 1. Default: 0.5
pd-radius	The radius within which bank-to-turn logic is not invoked. Default: 800.0
transition-radius	The radius used to trigger completion of tasks requiring movement to a waypoint. Default: 50.0
contour-flight-velocity-threshold	The desired-speed threshold below which the terrain-avoidance logic is eliminated. Default: 5.0
contour-flight-position-threshold	The desired-position threshold within which the terrain-avoidance logic is eliminated. Default: 10.0
height-feedback-gain	The maximum gain parameter for terrain avoidance. The terrain-avoidance velocity is found by multiplying an actual gain by the distance between the helicopter and the terrain avoidance height. The actual gain is found by ramping from zero to the maximum gain as the helicopter moves from the terrain-avoidance height to the surface. Default: 3.0

Table 2-1: Rotary-wing move-to pilot parameters

Parameter	Description
height-versus-speed	The table function used to describe the terrain-avoidance height versus speed. Default: {{0.0,30.0}, {25.0,30.0}, {50.0,50.0}, {75.0,75.0}, {100.0,110.0}}
is-align-with-targets	A flag indicating whether or not the helicopter aligns with targets when kinematic constraints allow it. Default: True
max-angular-vel	The maximum angular velocity the pilot will attempt. Default: 6.0
max-angular-acc	The maximum angular acceleration the pilot will attempt. Default: 20.0
vel-to-align-desired	The desired-velocity threshold beyond which the helicopter pilot aligns the helicopter body with the desired velocity within kinematic constraints. Default: 5.0
vel-to-align-actual	The actual-velocity threshold above which the pilot aligns the helicopter body with the actual velocity. This represents the top-level kinematic constraint on the pilot and overrides all other alignment algorithms when it is triggered. Default: 15.0
radius-trigger-vel	The velocity threshold beyond which bank-to-turn logic is invoked. Default: 10.0
radius-fraction	The radius-fraction parameter that is used in the bank-to-turn logic. The larger this value, the tighter the turn. Default: 0.85
position-gain	The position gain for the position controller. Default: 0.09
velocity-gain	The velocity gain for the position controller. Default: 0.36
orientation-gain	The orientation gain for the orientation controller. Default: 0.75
body-rate-gain	The rate gain for the orientation controller. Default: 4.32

Table 2-1: Rotary-wing move-to pilot parameters

Parameter	Description
noise-level	The noise-level parameter for the helicopter-controls error model. The noise-level parameter sets the magnitude of the power spectral density (PSD) of the white noise approximation on each control in percent (for collective, 0.1 percent). That is, as the time step goes to zero, the PSD of the error process goes to the noise-level value. (Open loop, a process with a PSD of 1 would integrate over time T to a random variable with variance T). Default: 0.08
landing-fuel-level	The fuel level below which the pilot attempts to land. Default: 20.0

Table 2-2: Rotary-wing patrol route pilot parameters

Parameter	Description
route-transition-radius	If the helicopter begins moving away from a route point within this radius, it transitions to the next route point. Default: 150.0 meters
route-x-gain	The x-direction gain for route-position feedback. Default: 0.05
route-y-gain	The y-direction gain for route-position feedback. Default: 0.05
route-z-gain	The z-direction gain for route-position feedback. Default: 0.02
route-lookahead-factor	The lookahead factor for route following. The lookahead factor ranges from 0.0–1.0, with 0.0 giving sharp turns at waypoints and 1.0 giving very rounded turns at waypoints. Default: 0.5

Loiter Controller

DtRotaryWingLoiterControllerComponent is derived from *DtSimpleRotaryWingPilot*. It implements a controller for loitering at a point. It does not use waypoints or routes; it loiters at the desired position set by the last controller to be used. If the position selector was set to zero by the previous pilot model, it loiters at its current position.

Rotary-Wing Pilot Model and Terrain Avoidance

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. So, a helicopter could crash if the terrain rises suddenly. The rest of this section describes helicopter movement algorithms in detail.

All the VR-Forces helicopter pilot models set the controls based on some task or desire. For example, the move-to pilot model sets controls to fly to a point. As illustrated in Figure 1, each of these pilot models is composed of two functional pieces: a guidance module and an autopilot module. The guidance module determines desired motion and the autopilot module sets the helicopter controls based on the guidance module's desired motion. As a parent class of the full pilot models, *DtSimpleRotaryWingPilot* implements the autopilot part of this pair.

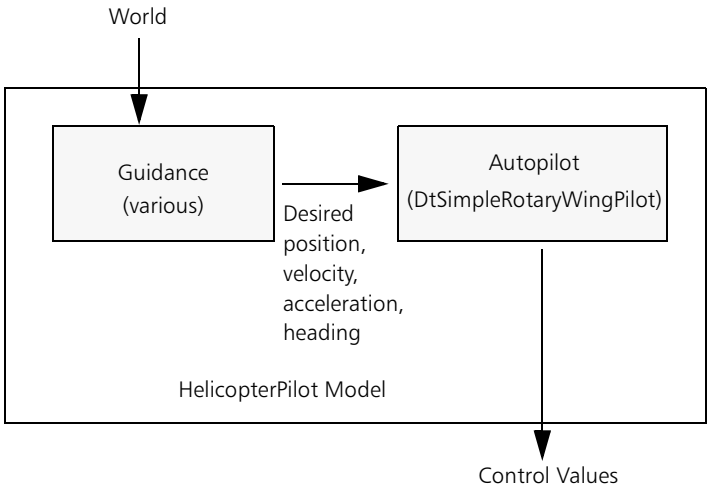

Figure 1. The helicopter model

The input to the autopilot is a set of values representing the desired motion of the helicopter. This set of values includes the following: vector position, vector velocity, vector acceleration, and scalar heading. A scalar selector value is also used with each of these four values. Generally, a selector is either zero or one to represent turning on or off that type of control, though other values may be used to scale the selection. These guidance values and selectors are part of the entity state to allow the loiter and collision avoidance pilot models (or any pilot model the developer may create) to use the information to improve their performance.

The algorithm used in the *DtSimpleRotaryWingPilot* class is built on a foundation of two proportional-plus-derivative (PD) controllers. These controllers are described below. The gains are configurable through the descriptor class.

The first controller finds a desired acceleration $\vec{a}$ by summing three terms as follows:

$$\vec{a} = k_v((s_v\vec{v}_d - \vec{v}_a) + k_p s_p \Delta\vec{p}) + s_d\vec{a}_d$$

Equation 1

where:

- k_p and k_v are user-configurable gains
- s_p, s_v, and s_a are selectors
- $\Delta\vec{p}$ is the vector from the current to the desired position
- $\vec{v}_d$ is the desired velocity
- $\vec{v}_a$ is the actual velocity
- $\vec{a}_d$ is the input desired acceleration.

The desired velocity is capped at the maximum value, and the output acceleration is capped at its maximum value.

For input to the second PD controller, the output acceleration from the equation (1) is used to calculate a desired orientation to achieve that acceleration. The orientation so calculated is that giving alignment of the rotor axis with the desired acceleration including gravity (that is, the "felt" acceleration, not the "seen" acceleration) while aligning the heading with the control value if appropriate (if the heading selector is nonzero and the velocity is below a user-configurable threshold) or the helicopter's target if appropriate.

The second PD controller uses this orientation as input. It calculates an angular acceleration that is proportional to the desired change in orientation and proportional to the current angular velocity as follows:

$$\vec{\alpha} = k_\omega(\vec{\omega} + k_\Theta \Delta\Theta\hat{\rho})$$

Equation 2

where:

- k_Θ and k_ω are gains.
- $\hat{\rho}$ is the approximate axis of rotation between the desired and the actual orientation. (The approximation is easy to calculate and becomes more accurate for small Θ.)
- Θ is the angle between the desired and the actual orientation about the axis of rotation.
- $\vec{\omega}$ is the angular velocity.

Once the desired linear and angular velocity are known using equations (1) and (2), the collective is set to give a least-squares approximation to the desired acceleration, and the cyclic and pedal controls are set to give the desired angular acceleration. Random errors are added to these values to simulate the inaccuracy of real controls.

Several modifications are made to the foundation created by the above two PD controllers. These include 1) bank-to-turn logic, 2) terrain avoidance, and 3) body alignment with velocity. The bank-to-turn logic modifies the desired velocity vector by rotating the current velocity only partially toward the desired velocity when the helicopter speed exceeds a configurable threshold. The terrain avoidance logic inserts a desired upward velocity when the helicopter comes within a prescribed range (H in Figure 2) of the terrain. The body alignment logic aligns the helicopter body with the velocity vector—by setting the desired orientation described above to be consistent with the desired alignment—when the velocity magnitude exceeds a configurable threshold.

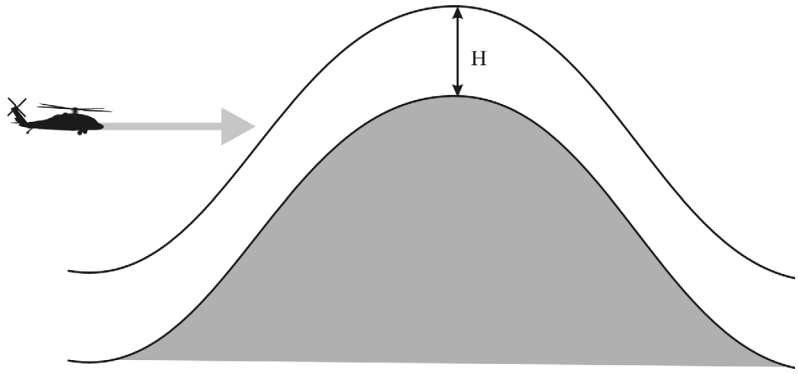

Figure 2. Terrain avoidance

Terrain avoidance will not prevent a helicopter from crashing in all circumstances.

Support for Joysticks

VR-Forces 2.0-ngc supports use of joysticks to control fixed-wing and rotary-wing entities. The default joystick parameters are specified in the file *./data/parameters/joyParam.dat*. You can override the default values for specific entities by specifying parameters in the object parameters database.

VR-Forces supports most 3 axis, 1 throttle joysticks that adhere to the following conditions:

- On Windows 95/98/2000, any device that has drivers that work with DirectInput 5 or greater. This includes devices that use the Universal Serial Bus. Windows NT does not support DirectInput 5 or greater.
- On Windows 95/98/2000/NT, any device that has drivers for the operating system, regardless of whether DirectInput is installed.

Enabling and Disabling Use of Joysticks

To enable joystick use:

1. Select the entity for which you want to enable joysticks.
2. On the popup menu, choose Enable Joystick.

To disable joystick use, follow the same procedure, but choose Disable Joystick.

Configuring Joysticks

The *joyParam.dat* file specifies the default values for joystick parameters. The parameters are:

- **joy-device-type** - Possible values are:
  - `pre-direct-input-joydevice` - used on Windows for joysticks that do not use DirectInput.
  - `direct-input-joydevice` - used for direct-input joysticks on Windows.
  - `linux-joydevice` - used for joysticks on Linux.
- **joy-trans-sensitivity** and **joy-rot-sensitivity**- affect the translation and rotation displacement required to make VR-Forces respond. The value should be between 0 and 1.0, with 0 being not sensitive at all, and 1.0 being the most sensitive it can be.
- **joy-trans-gain** and **joy-rot-gain** - affect the velocity of motion for a given displacement. The value is a multiplier, so the default value of 1.0 has no effect on the gain.
- **joy-trans-invert-x** (y, z) and **joy-rot-invert-x** (y,z) - allow you to invert axes on an individual basis.

The **joystick-control** parameter in the object parameters database indicates whether or not use of joysticks is enabled for an entity or entity type.

You can override the default values for joystick parameters for specific fixed-wing and rotary-wing entity types in the object parameters database. Change the values in the joystick parameter block for the specific entity.

```
(joystick
   (component-descriptor-type "base-joy-controller-descriptor")
   (component-type "fixed-wing-joy-flight-controller")
   (attached-to-list
      (flight-kinematics
         (port-interface "fixed-wing-input-port")
      )
   )
   ;; override defaults here
)
```

The rotary wing controller has some extra values that you can set.

```
(joystick
   (component-descriptor-type "rotary-wing-joy-controller-descriptor")
   (component-type "rotary-wing-joy-flight-controller")
   (attached-to-list
      (flight-kinematics
         (port-interface "rotary-wing-input-port")
      )
   )

   ;; controller rise time
   ;; default: 1.5
   ;; rotary-wing-joy-controller-descriptor only
   (controller-rise-time 1.6)

   ;; pitch feedback constant
   ;; default: 1.0
   ;; rotary-wing-joy-controller-descriptor only
   (pitch-control-feedback-constant 1.0)

   ;; roll feedback constant
   ;; default: 2.2
   ;; rotary-wing-joy-controller-descriptor only
   (roll-control-feedback-constant 2.265)

   ;; yaw feedback constant
   ;; default: 2
   ;; rotary-wing-joy-controller-descriptor only
   (yaw-control-feedback-constant 2.1)
)
```

New Entities Based on Existing Models

VR-Forces allows you to create the following new entities:

- Aavc7a1 landing vehicle
- M113 armored utility vehicle
- M88 medium recovery vehicle
- M9 Ace
- M58 MICLIC
- ZSU-23-4 Shilka
- MT-LB multipurpose armored vehicle
- BTR-80 armored personnel carrier
- RAH-66 Comanche
- VTUAV Cypher
- VTUAV Dragon Warrior
- LCAC landing craft air cushion
- LSD-49 Harper's Ferry class
- Guided missile destroyer (blue)
- Aircraft carrier (blue)
- Guided missile frigate (red)
- SSBN (blue and red)
- Dismounted infantry
- Civilian (male and female).

The surface entities are based on a generic ground vehicle model and the submarines are based on the helicopter models. Although these new entities do not model the behavior of their real-life counterparts, they use the correct entity enumerations, so if you view an exercise with a 3D viewer, such as the MÄK Stealth, the entities are represented by the correct 3D models. Also, you can change parameters for these entities to provide a closer approximation of actual size, speed, and behavior.

Improved Database Support

VR-Forces 2.0-ngc supports more types of databases, has improved support of DTED, and has enhanced the tdbtool.

Improved Support for DTED Terrain Database Records

DTED data is a grid of elevations. By default, VR-Forces shows a DTED database as projected onto UTM coordinates with white-on-green relief shading.

In a DTED terrain database record, you specify the subsampling rate, clipping latitude and longitude values, display parameters, and one or more DTED files. The syntax is as follows:

```
PvDtedMetafileRecord
{
   PostSpacing rate
   Projected flag
   LowColor red green blue
   HighColor red green blue
   LowColorHeight height
   HighColorHeight height
   ReliefShaded flag
   Origin latitude longitude
   MinMaxLatitude min_latitude max_latitude
   MinMaxLongitude min_longitude max_longitude
   SeaRectangleAdded flag

   File filename 1.dt0 (or .dt1, .dt2, .dt3, .dt4, or .dt5)
   File filename 2
   .
   .
   .
   File filename n
}
```

where:

- **PostSpacing** is an integer indicating the subsampling rate. The default value, 5, loads every fifth value along each dimension from the raw DTED data.
- **Projected** specifies whether or not the database should be projected onto UTM coordinates. The default value, True, projects the database. The value False means do not project the database.
- **LowColor** specifies the RGB values of the low color to be displayed. For relief shading, the low color is used for southwest facing polygons, and for height shading, the low color is used for polygons at the low color height and lower.
- **HighColor** specifies the RGB values of the high color to be displayed. For relief shading, the high color is used for northeast facing polygons, and for height shading, the high color is used for polygons at the high color height and above.
- **LowColorHeight** specifies the height, in meters, to use with the *LowColor* value in height shading. An interpolated value between the low color and the high color is used for polygons between the low-color height and the high-color height.

- **HighColorHeight** specifies the height, in meters, to use with the *HighColor* value in height shading. An interpolated value between the low color and the high color is used for polygons between the low-color height and the high-color height.
- **ReliefShaded** specifies whether or not to use relief shading. The default value, True, specifies relief shading. The value False specifies height shading.
- **Origin** specifies the latitude and longitude, in degrees, for the database origin. The default is to use the southeast corner of the first DTED file loaded.
- **MinMaxLatitude** allows you to clip the database from the north and south. No grid point with a latitude, in degrees, larger than the maximum or less than the minimum is loaded.
- **MinMaxLongitude** allows you to clip the database from the east and west. No grid point with a longitude, in degrees, larger than the maximum or less than the minimum is loaded.
- **SeaRectangleAdded** specifies whether or not to add a surface at sea level across the entire database. The default value, False, adds no additional surfaces to the database. This can be useful when DTED data is unavailable for an ocean area.
- **File** is a DTED filename. DTED files typically have extensions *.dt0* (for level 0 DTED), *.dt1* (for level 1 DTED), and so forth.

Improved Support for Paper Map Records

The following section updates "Paper Map Records," on page 3-21 in *VR-Forces User's Guide*.

Paper Map Records

The paper map record specifies the filename of the bitmap file, the coordinate system it uses, and the coordinates of the terrain database that it should overlay.

A paper map record uses an extent to specify a region of the world to view in Geodetic, UTM, or LCC coordinates. An extent is defined by two points, a southwest corner and a northeast corner to construct a "dummy" database that consists of a single, flat polygon. This extent can be used to provide underlying support for paper maps or to view entities in a particular section of the world in the absence of a database for that area.

Paper Map Bitmap Record

The bitmap record specifies the bitmap used to overlay the terrain database and its location. The syntax is:

```
PvPaperMapBmpRecord
{
   File filename.bmp
   CoordinateSystem {Geodetic | Utm | Lambert}
   SWCorner (X, Y)
   NECorner (X, Y)
}
```

where:

- **File** is the name of the bitmap to load. The file must use the bitmap format. (In other words, as used here, bitmap as not synonymous with raster file.)
- **CoordinateSystem** specifies the coordinate system that the **SWCorner** and **NECorner** extents are expressed in. The VR-Forces supports the following coordinate systems:
  - Geodetic
  - UTM
  - Lambert Conformal Conic (LCC, Lambert).

  UTM and LCC are only supported if the underlying terrain or extent is in the same coordinate system.
- **SWCorner** and **NECorner** specify the X and Y coordinates for the southwest corner and the northeast corner of the papermap. In the Geodetic coordinate system, the X and Y values define the corners in latitude and longitude (degrees). In the the UTM and LCC coordinate systems, the X and Y values define the corners in meters.

Support for Extent Records

You can specify extent records in the Geodetic, UTM, and LCC coordinate systems. The syntax for each type of extent record is as follows:

```
PvExtentRecord
{
   CoordinateSystem Geodetic
   SWCorner (latitude, longitude)
   NECorner (latitude, longitude)
}

PvExtentRecord
{
   CoordinateSystem Utm
   Origin (latitude, longitude, altitude)
   SWCorner (X, Y)
   NECorner (X, Y)
}

PvExtentRecord
{
   CoordinateSystem Lambert
   Lam0 longitude
   Phi0 latitude
   Phi1 latitude
   Phi2 latitude
   SWCorner (X, Y)
   NECorner (X, Y)
}
```

where:

- **CoordinateSystem** specifies the coordinate system in which the SWCorner and the NECorner are expressed.

- **SWCorner** and **NECorner** specify the X and Y coordinates for the southwest corner and the northeast corner of the papermap. In the Geodetic coordinate system, the X and Y values define the corners in latitude and longitude (degrees). In the the UTM and LCC coordinate systems, the X and Y values define the corners in meters.
- **Origin** specifies the UTM origin for the coordinate system in latitude (degrees), longitude (degrees), and altitude (meters). (UTM only)
- **Lam0** specifies the central longitude (degrees) for the LCC projection. (LCC only)
- **Phi0** specifies the central latitude (degrees) for the LCC projection. (LCC only)
- **Phi1** specifies the lower latitude (degrees) for the LCC projection. (LCC only)
- **Phi2** specifies the upper latitude (degrees) for the LCC projection. (LCC only)

New Options for the tdbtool

The tdbtool has the following command-line options:

```
tdbtool {input-file |
   -m input-file1 ... input-filen |
   -k levels [-a min-lod] input-file}
   [-b br,lf
   -f input-format
   -l lod-dist
   -r tolerance
   -s x,y[,z]
   -t
   -o output-file]
```

where:

- `-a` (only for use with `-k`) sets the distance to use with external OpenFlight files for the low level-of-detail, paged out representation.
- `-b` balances the tree, with branching *`br`* and leafing *`lf`*, for example, `-b 4,4`.
- `-f` specifies the input file format where *`format`* can be `gdb`, `flt`, `c4b`, or `dtd` (the default is to guess from the filename extension).
- `-k` keeps a given number of levels of a OpenFlight file as external files. This converts the top-level OpenFlight file to a master file and preserves the external structure of the OpenFlight file. It also causes the external OpenFlight files to be converted into matching GDB files). The `-k` argument implies `-f` flt.
- `-l` sets the level-of-detail distance to use for OpenFlight files. Default: 0.
- `-m` creates a master file that loads the input files on demand.
- `-o` specifies the output filename (default: *out.gdb*).
- `-r` reduces the complexity of the terrain by combining vertices within a *`tolerance`* distance of one another.
- `-s` generates a spatial index grid with cell dimensions *`x`*, *`y`*, and (for 3-D grids) *`z`* measured in database units.

- `-t` converts all polygons to triangles.

Miscellaneous New Features

VR-Forces now has online help. It is HTML-based and runs in the default browser on your system.

> To open online help, choose Help → Contents.

Using the New API

This section provides an overview of the new architecture for toolkit users.

Overview of Architectural Changes

The VR-Forces architecture has undergone a number of significant changes.

- Entities and control objects now share common representation and management:
  - Entity and control objects have a common class hierarchy
  - The entity parameter class hierarchy has been expanded and generalized to represent both entity and control objects
  - The entity manager and control object managers have been replaced by a single object manager
- The thread message interface mechanism for communicating between GUI and simulation engine has been replaced by DIS/RPR data interactions. This mechanism is also used to communicate VR-Forces-specific information between VR-Forces applications, such as tasks, plans, and echelon organization information.
- Control objects are published on the network using the DIS/RPR Environmental Process interactions in VR-Link.

New Project Files

The sample project files create the release and debug executables *userVrfSim.exe*, *userVrfSimDebug.exe*, and a batch file, *userVrf.bat* that starts VR-Forces in combined mode. This naming convention ensures that you do not inadvertently overwrite the executables supplied by MÄK. The files are placed in the *./bin* directory or *./binRPR*, depending on the protocol you are building for.

The *./mkwin32* directory contains a workspace (*.dsw*) and project (*.dsp*) file. The project file contains release and debug configurations. The projects are:

- *userVrfSim5* - a project that builds the DIS version of the VR-Forces back-end.
- *userVrfSimRPR* - a project that builds the HLA version of the VR-Forces backend.

The sample project enables you to add additional features using the VR-Forces toolkit. It enables you to extend the existing VR-Forces simulation back-end. There is no corresponding project for rebuilding the VR-Forces GUI. You can also use the sample project as a starting point for building your own application that uses components of the VR-Forces toolkit. It effectively replaces the example backend project in version 1.3.

New Examples

VR-Forces 2.0-ngc includes new examples for how to add an actuator, an entity, a task, and a user task. Each example has its own subdirectory under the *./examples* directory. When you create an example, the project files build the front-end and back-end executables described in "New Executables and Startup Options" on page 6, but they do not create an equivalent to *userVrf.bat*.

Mapping Old Classes to New Classes

This section maps the most significant classes from VR-Forces 1.x to the new VR-Forces 2.0-ngc classes. Although VR-Forces 2.0-ngc provides some degree of backward compatibility, this section will help you understand the changes in the release and help you migrate applications to the new architecture.

Entities

Entities are the simulated objects that represent individual platforms and aggregates in VR-Forces. The class hierarchy for representing entities has been replaced by a less entity-specific class, *DtVrfObject*. The new class represents both entities and non-entity objects (control objects, such as routes, waypoints, obstacles).

Old class: *DtSimBaseEntity* and derived types (*baseEnt.h*)
New class: *DtVrfObject* (*vrfObject.h*)

Control Objects

Control objects are simulated objects such as routes, areas, and waypoints. The old classes for representing control objects have been replaced by the same class used to represent entities, *DtVrfObject*.

Old class: *DtControlObject* and derived types (*controlObj.h*)
New class: *DtVrfObject* (*vrfObject.h*)

Entity Manager

In VR-Forces 1.x, the Entity Manager managed all of the entity objects in the simulation. The Entity Manager has been replaced by the Object Manager, which manages all simulated objects (entities as well as non-entity objects, such as control objects).

Old class: *DtSimEntityManager* (*entMgr.h*)
New class: *DtVrfObjectManager* (*vrfObjMgr.h*)

Control Object Manager

In VR-Forces 1.x, the Control Object Manager managed all of the control objects in the simulation. It has been replaced by the Object Manager, which manages all of the simulated objects (entities and control objects).

Old class: *DtControlObjectManager* (*cntlMgr.h*)
New class: *DtVrfObjectManager* (*vrfObjMgr.h*)

Entity State Repositories

An entity state repository contains all of the state information for an entity. Entity state repositories have been expanded and generalized as object state repositories to contain the state information for all simulated objects.

Old class: *DtBaseEntityParameters* (*baseSR.h*)
New class: *DtVrfObjectBaseStateRepository* (*vrfObjBaseSR.H*)

Entity Parameters

Entity parameter classes contain the parameter data that is applied to an entity when it is created. When an entity is created, the parameter entry for that object is looked up in the parameters database, and a copy of that parameter entry is made for the entity. The entity parameter classes have been replaced with less entity-specific object parameter hierarchy.

Old class: *DtBaseEntityParameters* (*baseParams.h*)
New class: *DtVrfObjectBaseParameters* (*vrfObjBasePar.h*)

Entity Parameters Database

In VR-Forces 1.x, the entity parameters database stored parameter data for all entities simulated by VR-Forces. To reflect the fact that control object and entity data are now stored in the parameters database, VR-Forces 2.0-ngc uses an object parameters database.

Old class: *DtEntityParametersDB* (*entParamDb.h*)
New class: *DtObjectParametersDB* (*objParamDb.h*)

Simulation Creator

The simulation creator is responsible for creating all of the top-level objects (managers and factories) in VR-Forces.

Old class: *DtSimCreator* (*simCreator.h*)
New class: *DtVrfCreator* (*vrfCreator.h*)

Communication Between VR-Forces Processes

Communication between the VR-Forces GUI (front-end) and simulation engine (back-end) is now conveyed completely through DIS PDUs/RPR objects and interactions. In VR-Forces 1.x, the GUI and simulation engine were in separate threads. They communicated via the thread message interface mechanism and through the VR-Link exercise connection. Now the GUI and simulation engine are in separate processes, and all communication takes place through the VR-Link exercise connection. VR-Forces 2.0-ngc uses the same data interaction mechanism to communicate between local and remote VR-Forces applications.

Backwards Compatibility

To ease the transition to the new architecture, this release provides some backwards compatibility features.

- Many of the old class header files are still provided. These files point to the corresponding new class header files.
- In many cases, typedef declarations map old class names to the new classes that replace them.
- Several key new classes inherit a backward compatible interface, making it easier to build existing code against the new libraries.
- VR-Forces 2.0-ngc does not support the VR-Forces 1.x entity parameters database and scenario file format. We provide an object parameters database and scenarios that use the new format.

! The backward compatible interface is not comprehensive, so depending on how your legacy applications access the API, you may need to transition to the new API right away.

Documentation Updates

This release has updated versions of the class hierarchy and header file documentation. *MÄK VR-Forces Developer's Guide* has not been updated to describe the changes to the API in this release. We expect to release an updated *Developer's Guide* shortly. Please contact your MÄK sales representative for information about its availability.

Updates to *MÄK VR-Forces User's Guide*

The following are miscellaneous updates to *MÄK VR-Forces User's Guide*:

- The description of the Frame Delta T value in the scenario file on page 10-33 refers to a **frame-delta-time** parameter. The correct name for this parameter is **frame-time**.
- When you create an entity, it has a full set of resources, as specified in the object parameters database. You do not need to assign resources to a new entity.
- The layout and controls on the Settings dialog box have changed slightly from those pictured in the manual, but are functionally the same.
- Table 10-3, on page 10-11, lists the incorrect names for two parameters:
  - underFireTime should be **under-fire-time**.
  - underFireDistance should be **under-fire-distance**.
- On page 9-19 of *VR-Forces User's Guide*, the description of the Any parameter for the EntDestroyed condition should say that it returns true if any member of an aggregate is destroyed.

Definition of Remote and Local Entities

This section is to clarify the use of the terms remote entity and local entity as described in section 7.1, "Local Entities Compared to Remote Entities" in VR-Forces User's Guide. From the point of view of a particular VR-Forces application, a local entity or object is any entity or object that is simulated in that process. In that same application, a remote entity or object is an entity or object that is received over the network from an external application, such as another VR-Forces simulation back-end, or a manned simulator. A given VR-Forces application participating in a networked exercise may have a mixture of local and remote entities and objects.

Remote entities may be further broken down in 2 categories: non-VR-Forces remote entities and VR-Forces-simulated remote entities. Non-VR-Forces entities have the DIS/RPR entity state associated with them. The VR-Forces remote entities share additional state information over the network, including a string name (which may contain echelon ID information, if the entity is organized), and a string label.

Intervisibility and Target Detection

When it calculates intervisibility, VR-Forces draws a line from the center of one entity to the center of the other entity. If the line does not intersect the terrain database, the entities can "see" each other.

The target detection sensor is a little more sophisticated. It first checks the line from the location of the target detection sensor (it is offset from the position of the entity by the "detect-offset" vector) to the top of the target entity. This is done so that entities that are just barely visible over a hill (or above the surface of the water) can be targeted. The target detection sensor also checks to make sure that no other entities are in the way (that is, if the line between it and the target passes through the bounding volume of another entity).

Rules of Engagement

The fire-when-fired-upon rule works as follows:

The entity considers itself under fire in one of two situations:

- An "Entity Impact" detonation interaction that targeted this entity has recently (within **under-fire-time** seconds (default 120)) been received.
- Any detonation has happened within **under-fire-distance** (default 1000 meters) of this entity.

The **under-fire-time** and **under-fire-distance** parameters are top-level entity parameters (like **max-speed**), but they are not in the standard *vrfSim.opd* file.

There is no attempt to return fire specifically at the attacking entity. If an entity thinks that it is under fire, it will fire at any enemies that it can target.

Updates to MÄK VR-Forces Developer's Guide

- In the Notes for Section 12.2.1 (page12-2) of *VR-Forces Developer's Guide*, there is a reference to "Chapter 11, Using Plug-ins With VR-Forces". That chapter is in *VR-Forces User's Guide*, not the Developer's Guide.

Bug Fixes

Force-level pseudo-aggregates no longer publish themselves over the network. This eliminates the bug in the previous release in which 'phantom' top-level aggregate entities would appear on the network.

Known Problems

VR-Forces Release 2.0-ngc has the following known problems:

- The Go To Next and Go To Previous options on the Map Navigation menu don't work with aggregates. If you have a mixture of aggregates and individual entities, using these menu options works fine if there is a series of contiguously listed individual entities. Once the selection moves to an aggregate (11:1:0 ...) the entity list window loses a selection and the view in the Simulation View window doesn't change.
- Aggregates composed of fixed-wing entities do not carry out tasks as an aggregate. This is because the Follow Entity task is not implemented for fixed-wing entities.
- When you create a When statement, it may have the syntax "do Replace" in the statement until you save the plan. This does not affect the plan.
- When you create a When or If statement that tests for the condition of an aggregate being destroyed, the condition never evaluates to true, even if all of the members in the aggregate get destroyed.
- If you are using VR-Forces on a multi-processor computer, set the affinity to single-processor. VR-Forces does not currently take advantage of multiple processors.
- The header file comments for the DtMissileEntity::place() function are incorrect. The header file indicates that the version of the place() function that takes a DtVector& location argument positions the missile on the ground, keeping its current heading. The place() function actually positions the missile at the (X, Y) coordinates given by the location, and a Z coordinate of 1000 meters. All versions of the place() function reposition the missile in Z to 1000 meters, regardless of whether or not they have the location specified as an explicit argument.

- When you aggregate a set of aggregates into a higher-echelon aggregate using the "Aggregate As" menu item, make sure that the aggregates being combined have been collapsed to their highest level. Failure to do so will aggregate the entities at the level they are currently displayed. For example, suppose that you want to aggregate two teams of two tanks each into a platoon. If at the time of the "Aggregate As" operation, one of the teams is expanded, displaying its constituent tanks, while the other team is collapsed, displaying a single team icon, the resulting platoon will be composed of two individual tanks and a team, instead of two teams.
- Fire interactions are ignored by the under-fire test for tanks.
- Phase line labels are displayed in the middle of the phase line rather than the beginning. To determine which side of a phase line is "left" for the purpose of the EntLeftOfLine condition, you will need to remember where you started each phase line.
- The missile icon may flash briefly in the lower left-hand corner of the display when a missile is fired.
- Moving in reverse does not work for ground vehicles. If you set speed to a negative value, the entity does not move.
- A wait task entered as an initial statement in a plan may appear to take too long. This is due to the fact that it takes time for the plans to be delivered to the appropriate back-end for execution.
- Surface and lifeform entities are not configured to fire weapons or be fired upon. You can customize the parameter database entries to enable this.
- Repeated attempts to toggle the intervisibility fan will result in multiple intervisibility fans displayed on top of one another. It will not turn them off.
- The waypoint icon is currently being displayed centered on its location. According to MIL-STD-2525B it should be positioned with the lower tip of the positioned at the waypoint's location.

Any previously reported problems that are not in this list have been fixed.