MÄK VR-Forces 3.7 Release Notes

This release note contains the following release-specific information for VR-Forces Release 3.7:

Document ID: VRF-3.7-3-040716

Systems Supported

VR-Forces 3.7 is available for the platforms listed in Table 1. For the availability of other platforms, contact your MÄK salesperson. For toolkit users, application code must be built with the indicated compilers in order to link to VR-Forces libraries.

Table 1: Platforms supported

Operating System	Compiler
SGI IRIX6.5	MipsPro7.3 C++ compiler (available from SGI)
Red Hat Linux 9.0	GNU C++ (GCC) 3.2.2
Windows 2000 SP2, XP	Microsoft Visual C++ 6.0, Service Pack 5 Microsoft .NET 7.1

If you are building VR-Forces with .NET on Windows 2000, you must install the latest version of the directX SDK (obtainable from Microsoft).

Patching the Microsoft Visual C++ 6.0 Compiler

VR-Forces is transitioning to use of the Standard Template Library, particularly for lists and map objects. Microsoft Visual C++ version 6.0 has a bug with the STL that is fixed with the supplied *xtree-msvc++patch* file. If you use MS Visual C++ version 6.0, you must install a patch to a standard header file. The patch comes from Dinkumware, Ltd (the company that developed the Standard C++ Library header files for Microsoft), but for convenience, we are distributing it with VR-Forces releases.

The patch fixes a bug that would cause applications to crash if they pass std::sets or std::maps from an application to a DLL or vice-versa, something that IEEE 1516 federates must do. For details about the patch go to http://www.dinkumware.com/vc_fixes.html.

To install the patch:

1. In the top level VR-Forces directory, is a file called *XTREE-msvc++patch*. Rename this file *XTREE*.
2. Make a back-up version of the original *XTREE*.
3. In the Visual Studio *include* directory (for example, *C:\Program Files\Microsoft Visual Studio\VC98\Include*), replace the original version of *XTREE* with the patched version.

Building on Linux or IRIX

The example Makefiles require a patched version of gmake 3.80, which has been included for your convenience in the *./mkbin* directory. This version of gmake is based on version 3.80, but includes a memory fix that is needed to use our makefiles. The source code for the modified gmake is freely available from *http://ftp.mak.com/out/gmake3.80-patch1.tar.gz*. This patch will be included in future versions of gmake.

Before you compile the examples, go to the top level of your installation and create a symbolic link 'VR-Link' to your VR-Link installation and another called 'RTI' to your RTI installation.

Running VR-Forces on SGI IRIX

If you want to run VR-Forces on SGI IRIX, you must set the color depth to 24-bit.

True Type fonts are not supported on SGI IRIX. Use bitmaps for entity icons.

Disk Space Requirements

A full installation of VR-Forces requires approximately 540 MB of disk space. Terrain databases use approximately 106 MB and documentation (primarily the class reference) uses 200 MB.

MÄK Product Compatibility

To build VR-Forces applications, you must link with VR-Link 3.9.1. If you are building for HLA and want to link with the MÄK RTI, use MÄK RTI 2.x.

MÄK VR-Forces 3.7 is a parallel release to MÄK Plan View Display 2.7. It is not compatible with previous releases of the Plan View Display.

Qt Release Compatibility

If you want to do development using the VR-ForcesGUI API, you need to use Qt, a cross-platform GUI toolkit. The VR-Forces GUI was built with Qt release 3.3.1. A VR-Forces developer's license includes a Qt development license. MÄK provides you with a Qt license key. However, you must download the correct version of Qt from *www.trolltech.com*.

Third-Party Library Requirements

DtTerrainProfileWidget is based, in part, on the work of the Qwt project (*http://qwt.sf.net*). To compile examples or user projects, you must download the QWT distribution listed on their home page. Set the MAK_QWTDIR environment variable to point to the location of the QWT library.

Network Compatibility

HLA only

VR-Forces 3.7 was built against VR-Link 3.9.1 and is compliant with:

- RPR-FOM 1.0 and a subset of 2.0 (draft 6 and 17)
- MÄK RTI 2.2
- DMSO RTI NG 6.

If you need to run the Linux version of VR-Forces with the DMSO RTI, contact MÄK technical support.

VR-Forces 3.7 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 3.7 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.

Backwards Compatibility

The format of interface messages used to communicate between VR-Forces front-ends and back-ends has changed. You cannot run previous versions of VR-Forces with the current version in the same exercise.

The format of the object parameters database has changed. VR-Forces can read previous versions of the object parameters database, but you are encouraged to upgrade any modified object parameter databases you have created to the new format. See "Updating Legacy Object Parameters Database Files," on page 9, for information about how to upgrade.

There have been many improvements and bug fixes to the terrain database file format. While terrain databases created with previous versions of VR-Forces may continue to work, if you have converted any databases to GDB format, it is strongly recommended that you import them again.

RPR FOM Versions Supported

VR-Forces 3.7 has built-in support for versions 1.0 and 2.0 (draft 6 and 17) of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, VR-Forces 3.7 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 (for example, 2.0006) 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-Link20006", "MyApp", new DtRprFomMapper(2.0006));
```

In order to support RPR FOM 1.0, we have added the following extensions (which are supported in later versions of the RPR FOM):

- EnvironmentProcess
- GriddedData
- EmbeddedSystem.IFF
- EmbeddedSystem.IFF.NatoIFF
- EmbeddedSystem.IFF.NatoIFF.NatoIFFTransponder
- EmbeddedSystem.IFF.NatoIFF.NatoIFFInterrogator
- EmbeddedSystem.IFF.SovietIFF
- EmbeddedSystem.IFF.SovietIFF.SovietIFFTransponder
- EmbeddedSystem.IFF.SovietIFF.SovietIFFInterrogator
- EmbeddedSystem.IFF.RRB
- BaseEntity.AggregateEntity
- ObjectRoot.BaseEntity.PhysicalEntity.Lifeform.

For both RPR FOM 1.0 and 2.0 VR-Forces 3.7 relies on the LgrControl and View-Control custom MÄK extensions.

New Features and Updates

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

- The object parameters database API and file format have changed
- Ground entities now avoid obstacles and features
- New and updated interface features
  - While loops, Stealth control, and new triggers for plans
  - Ability to load symbol map files at runtime
  - Animated, representational, entity icons
  - Ability to select features and feature segments
  - Ability to create graphical objects based on feature data
  - Ability to show directional arrows on lines
  - Ability to view the status of entity resources
  - Enhanced Move To task
  - New Move To Location task
  - Filtered object lists for tasks and set data requests
  - Ability to print the map image or save it to a file
  - Reorganized entity information dialog box
  - Continuous navigation when pressing Navigation toolbar buttons
  - Reorganization of the main menu
- New special effects:
  - Display of electromagnetic emissions
  - Animation of munitions fire and detonation
  - Sound effects
- Changes to how entity parameters are saved to the order of battle file
- New examples
  - How to convert an object parameters database to the new format
  - How to create a condition
  - How to create an object parameters database dynamically
  - How to create new symbols and model data
- New configuration parameters
- New startup option (load bounding volume configuration file at startup)

- API Changes
  - Collision detection and avoidance has been rewritten
  - New Move To Destination controller and reorganization of the Move To controller class hierarchy
  - Changes to the Terrain API
  - GUI API changes
  - Changes to the VR-Forces API
  - Miscellaneous API changes
  - New build system to support customization of examples
- New online help browser (see "Documentation Updates," on page 40.)

The Object Parameters Database API and File Format have Changed

The object parameters database file format and API has changed, as follows:

- The object parameters database file format has changed to make editing and maintaining the database easier.
- Object parameters database entries are now loaded on demand, resulting in less memory usage (unused parameter entries are no longer automatically loaded).
- The object parameters database classes have been refactored to remove the restriction that the object parameters database can only be read in from file in *DtReaderWriter* format (the format used by *vrfSim.opd*). The interface for loading and saving object parameters database information is now independent of the *DtReaderWriter* format files.

By default VR-Forces continues to use *DtReaderWriter* as the mechanism for serializing parameter data.

New Object Parameters Database Format

The object parameters database has been broken up into multiple files, referenced by a top-level index file. The top-level index file (*vrfSim.opd*) contains a list of object parameter entry files (**.ope*), keyed to object type. Each object parameter entry file contains the *DtReaderWriter* representation of a *DtVrfObjectParameters* object. Individual object parameter entry files are self-contained. They contain the complete information about an object.

Parameters specified in *.ope* files do not inherit from more generic object types (as they did in previous releases.) All parameters for an object must be specified in its *.ope* file. Legacy format object parameters databases use the inheritance mechanism as before.

VR-Forces 3.7 includes an object parameters database in the new format (*vrfSim.opd*) and in the legacy single-file format (*vrfSim_pre3.7.opd*). VR-Forces can read both types of object parameters database file. The legacy file format is included to help you migrate customized object parameters database files to the new format.

Individual parameter entries are loaded on demand. When VR-Forces first creates a particular kind of entity, aggregate, or object, it looks up the filename for the object parameter entry file in the object parameters database and loads it. Any subsequent lookups simply return the entry directly. The object parameters database loads into memory only those parameters necessary for the simulation.

Old format (pre-version 3.7) object parameters database files load all parameters into memory.

! Future versions of VR-Forces will not include an object parameters database in the single file format. MÄK strongly encourages customers to migrate their object parameters databases to the new format.

Updating Legacy Object Parameters Database Files

If you have a previous version of VR-Forces and have not edited your object parameters database, then you can run your scenarios with the new default object parameters database, *vrfSim.opd*. If you have edited the default object parameters database, then you should migrate it to the new format using one of the following methods:

- Convert the object parameters database using the VR-Forces GUI (only in combined mode.)
- Use the *convertParamDb* utility.
- Edit the new *vrfSim.opd* file and associated object parameter entry files (*.ope*) by hand.

Converting Object Parameters Databases through the GUI

This option is easiest, but it does not preserve comments when converting from the old format to the new format.

! If your object parameters database refers to new kinds of classes, such as new kinds of component descriptors or parameter objects, you must recompile the back-end with your extensions before converting your legacy object parameters database file.

To convert an object parameters database through the VR-Forces GUI:

1. Create a new scenario, or load an existing scenario in combined mode. VR-Forces automatically detects that the object parameters database is in an older format and prompts you to save it in the new format.
2. Click Yes.

Using the convertParamDb Utility to Convert Object Parameters Databases

You can use the *convertParamDb* utility to convert-single file object parameters databases into the new format. This tool tries to preserve comments when converting the legacy file into the new format. However, the format of comments may be altered somewhat during the transformation. In particular, comments that start on the same line as a symbol will appear on the line following that symbol in the new format file.

! If your object parameters database contains references to any new derived types of component descriptors, resources, or parameter classes, you must rebuild *convertParamDb* before you convert your legacy object parameters database file. Edit *main.cxx* and register creator functions for your new class types before loading the parameter file. For example:

```
DtDefaultResourceFactory myResourceFactory;
DtDefaultComponentDescriptorFactory myComponentDescriptorFactory;
DtDefaultParameterFactory myParameterFactory;

DtSimResource::setFactory(&resourceFactory);
DtComponentDescriptor::setFactory(&componentDescriptorFactory);
DtVrfObjectParameters::setFactory(&parameterFactory);

// Add creator funtions to factories for new customer created types in the
// object parameters database here.
//
// e.g.
// parameterFactory.addCreatorFcn(DtSpaceVehicleParamType,
//    DtSpaceEntityParameters::creator);

DtMtlParameterDb parameterDb;
parameterDb.init();

// Load old-style object parameters database file (pre-version 3.7)
```

```
if(!parameterDb.load(oldFormatParamDbFile))
. . .
```

To use the *convertParamDb* utility, run it from the *./bin* directory as follows:

```
convertParameterDb old_parameter_DB_file new_parameter_DB_file
```

Edit the Object Parameters Database Files Directly

If your changes to the object parameters database are not too extensive you may want to edit the new object parameters database files by hand. Open the *.opd* or *.ope* files in a text editor and edit them. This option offers you the most control over the format of your comments.

Parameters in individual object parameter entry (*.ope*) files are not inherited from more generic types, so be sure to specify all parameters that you want to change for an object in its *.ope* file.

New Object Parameters Database API

DtVrfParameterDb is the base object parameters database class. *DtVrfParameterDb* holds a collection of *DtVrfObjectParameters* entries, keyed on *DtObjectType*. It has member functions for adding, removing, and looking up *DtVrfObjectParameters* entries. It also includes an interface for load() and save(), but with an empty implementation. Developers can create derived types of object parameters database classes that know how to load and save themselves from file in whatever format they choose.

VR-Forces has one kind of derived *DtVrfParameterDatabase* type, *DtMtlParameterDb*. *DtMtlParameterDb* extends the base class with the ability to load from and save to *DtReaderWriter* (MTL) format. The *DtMtlParameterDb* class knows how to read and write object parameters database files in this format and in the older, single file format.

The *DtCgf* class is responsible for creating the object parameters database object. By default, it creates an instance of type *DtMtlParameterDb*. Developers can override this behavior and have VR-Forces create some other kind of derived *DtVrfObjectParametersDb*. To do this, register a static creator function for the new kind of *DtVrfObjectParametersDb* with the *DtVrf DtDefaultVrfCreator* class before initializing the *DtCgf*, for example:

```
DtCgf cgf;
. . .
DtDefaultVrfCreator creator;

// Specify the base parameter database type, DtVrfParameterDb. This
// is the simplest kind of parameter database. It does not serialize
// itself to file in MTL format.
creator.setParameterDatabaseCreatorFcn(DtVrfParameterDb::create);

// Initialize the CGF, passing in our altered creator.
cgf->init(exerciseConn, &creator);
```

The new example createParamDbDynamically demonstrates how to use a different type of object parameters database with VR-Forces.

Ground Entities Avoid Obstacles and Features

Ground entities now avoid obstacles drawn on the map. This behavior applies only to obstacles created with the Create → Obstacle menu option. It does not apply to graphical objects drawn on overlays. When an entity detects an obstacle it avoids the obstacle's bounding volume, which is conceptually the smallest rectangle that bounds all of the points in the obstacle.

Ground entities can also avoid features, such as buildings and plants. By default, entities avoid other entities, buildings, water, and vegetation. (LCACs do not avoid vegetation by default.) See the entry for the M9 Ace in the object parameters database for an example of detailed obstruction avoidance configuration.

Notes

- To specify that ground entities should not move through the terrain (for example, through a wall or through a hill), add the following line to the automotive actuator declaration (powertrain):

  ```
  (check-terrain-collisions True)
  ```

  For performance reasons, this defaults to False, so that entities that do not care about terrain collisions do not perform the check.
- If any entity is tasked to move to a point that is inside an obstruction, or very close to one, it might never reach the point, but it will continue to try to reach the point as long as the task is in effect. This behavior may also apply to entities following routes that have vertices inside an obstruction.
- If a terrain has geometric features as well as point vector features, the feature avoidance may not work because the points are placed on the terrain on “top” of the existing geometry, which means they are on top of the geometric feature. A new version of Makland, Makland_OnlyVectorFeatures.gdb, has been provided which does not have point vector features, and as such works with feature avoidance.
- Path following and obstacle avoidance may not work well together. Entities moving along a road may seek to avoid obstacles, such as trees or light posts, that are placed close to the road, even though they would not hit the obstacles as long as they stay on the road. To prevent this problem from occurring, the ground collision avoidance controller has been given a lower priority than the Move To Path controller.

Configuring Obstruction Avoidance

Obstruction avoidance, formerly obstacle avoidance, has been enhanced in this release. Ground entities can be configured to avoid specific entity types, vector features, or both.

The more vector feature types and entity types specified for avoidance, the more memory and processing power required.

The default parameters for obstruction avoidance have been altered to make them more realistic for avoidance behavior. The **stopping-distance-factor** parameter has been reduced to allow for more aggressive movement and the lateral clearance has been reduced to allow entities to move near each other more easily, especially lifeform entities. If entities are displaying erratic behavior, try raising the values, lowering the speed of the entities, or both.

Configuring Object Types

To add or remove types of objects that an entity type should avoid, add or remove the appropriate object type from the obstruction sensor entry in the object parameter entry file. For example, the following object types are configured for ground entities:

```
(object-type
  (object-type  1 (18 -1 -1 2 -1 -1 -1)) ;; vrfObstacles object type
  (object-type  1 (1 -1 -1 -1 -1 -1 -1)) ;; DtPlatform entity kind
  (object-type  1 (3 -1 -1 -1 -1 -1 -1)) ;; DtLifeForm entity kind
)
```

Configuring Vector Feature Types

Configuration of avoidance of vector features relies on feature specifications in GDB databases. To configure a vector feature type for obstruction avoidance, modify the appropriate **vector-type** parameter in the **vector-obstruction-type** parameter block for the obstruction sensor in an entity's object parameters database entry.

The vector obstruction types are used to create *DtSpecifications*, which are then used by the vector network intersection metrics in determining if obstructions are the requisite types or not. Any attribute specified for avoidance must be in the vector feature for it to be considered an obstruction. Because the vector feature may contain additional properties, simple vector obstruction types can map to a large number of features.

For example, to specify all types of buildings, add a **vector-type** parameter with the single attribute:

```
(vector-obstruction-type
  (vector-type
    (string-attributes
      (DtFeatureGroup "MAK000")
    )
  )
)
```

If you only want to avoid a specific feature, create a more detailed vector-type. For example, the following will cause only standard lights to be considered obstructions.

```
(vector-obstruction-type
  ;; Light Standard (street light) detailed spec
  ;; illustrates how to create a detailed obstruction type
  (vector-type
    (integer-attributes
      (DtFidCode 537)
      (DtFeatureType 0)
      (DtSmc 1)
      (DtSedrisCodes 0)
      (DtOrientation 0)
      (DtHeight 3)
    )
    (string-attributes
      (DtFeatureName "537 Light Standard")
      (DtFeatureGroupName "Power")
      (DtFeatureGroup "MAK050")
      (DtDescription "US-Light Standard/Light Support  UK-Light
        Standard/Light Support/Lamp Post")
      (ECC "AL110")
    )
    (real-attributes
      (GDB_Width 0.0)
      (GDB_Length 2.0)
    )
  )
)
```

New Plan Features

This release of VR-Forces has added the following new features for plan development:

- While loops: As long as the condition is met, the While block continues to get executed.
- Stealth control: You can attach the MÄK Stealth to, and detach it from, an entity from a plan.
- Time-based triggers: You can create a condition that tests whether simulation time is greater than or less than the specified time.

See *MÄK VR-Forces User's Guide* for details.

Loading Symbol Map Files at Runtime

You can now change the symbol map file used for entity icons at any time. Choose File → Load Symbol Map and select the file you want to use.

Animated Entity Icons

The new symbol map file, *./config/symbolmap-images.mtl* has entity icons that are representational of the entity type and include animations, such as moving rotors for helicopters, as illustrated in Figure 1.

Helicopter Dismounted infantry M1A2 tank

Figure 1. Animated icon images

Selecting Features

Some terrain databases contain feature data, such as roads, buildings, trees, and so on. You can select features and feature segments, and display information about them. You can also create routes and points based on feature data.

Among the databases provided with VR-Forces, the *makland* and *berkeley* databases contain feature data.

See *MÄK VR-Forces User's Guide* for details.

Creating Objects from Features

You may want entities to be able to move to or along the points and lines represented by terrain features. However, except for path following and obstruction avoidance, entities do not know about terrain features and cannot interact directly with them. To work around this problem, you can create routes from line features and points from point features.

When you create a route from a line feature, the vertices of the route are placed at the intersection of each feature segment. You can create a waypoint at the location of a point feature, or create an entity. (Some VR-Forces users model environmental features as entities to facilitate interaction with entities.)

See *MÄK VR-Forces User's Guide* for details.

Displaying Entity Resources

You can now view the status of resources in the Entity Information dialog box and as an entity label.

The Entity Information dialog box lists the entity's current amount of resources, its starting amount, and the percentage remaining (Figure 2).

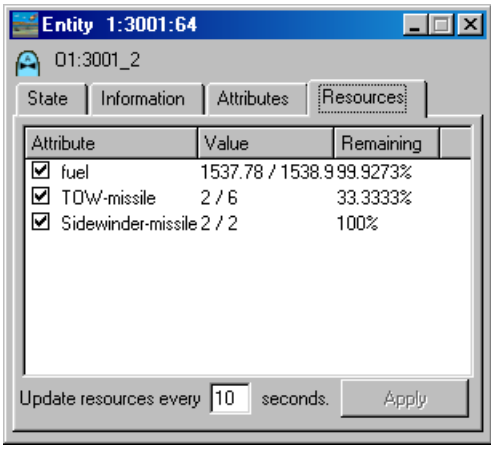

Figure 2. Entity resources

You can display bar graphs as part of an entity's label that show the approximate status of resources. Entity labels use colored bars to display resource status. The status is represented by the length of the bar and the color (Figure 3), as follows:

- Green = 40 - 100% of resource is available
- Yellow = 10 - 40% of resource is available
- Red = Less than 10% of resource is available
- Black rectangle = resource is fully depleted.

Figure 3. Entity resource labels

The display in Figure 3 corresponds to the resource display in Figure 2.

See *MÄK VR-Forces User's Guide* for details.

Enhanced Move To and Patrol Between Tasks

The Move To Waypoint and Patrol Between Waypoints tasks have been enhanced. The new Move To and Patrol Between tasks let you task entities to move to or between waypoints, overlay points, and other entities.

New Move To Location Task

The Move To Location task lets you task an entity to move to a location on the terrain. See *MÄK VR-Forces User's Guide* for details.

Filtered Object Lists for Tasks and Set Data Requests

All dialog boxes that require selection of an entity let you filter for entities only, aggregates only, and for each type, friendly and opposing forces.

The Move To and Patrol Between dialog boxes let you filter for waypoints, overlay point objects, and entities. Only published objects are displayed in the object list.

Printing the Map Image or Saving it to A File

VR-Forces supports the following new options for capturing images of the map area:

- Print the image
- Save the image to a file
- Print the objects (no map data).

See *MÄK VR-Forces User's Guide* for details.

Reorganized Entity Information Dialog Box

The Entity Information dialog box, Aggregate Information dialog box, and Stealth Information dialog box have been redesigned. They are smaller and have information divided up into tabbed panels that display identifying information, state information, attributes, and resource status.

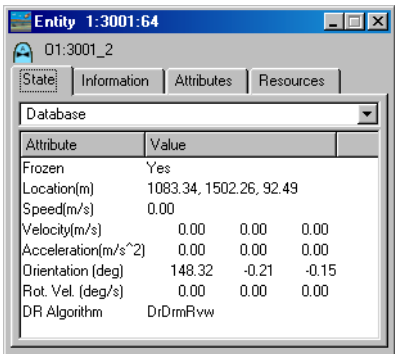

Figure 4. Entity Information dialog box

Continuous Navigation When Pressing Navigation Toolbar Buttons

The Navigation tab of the Terrain View Options dialog box has a new check box labeled Continuous Navigation. If you enable this feature, when you click and hold a button on the Navigation toolbar, VR-Forces continues to move or zoom the display until you release the button.

Reorganization of the Main Menu

Menu options for opening terrain profiles, editing objects, opening information dialog boxes, and deleting objects, have been moved to the Object menu. This menu also has options for setting the selection mode. The Options menu now has options to toggle special effects on and off.

Display of Directional Arrows on Lines

In previous releases of VR-Forces you had to deduce the direction of a line from the placement of its label. In this release, you can specify display of a directional arrow on routes and lines.

To display a route's arrow:

1. Choose Options → Display. The Display Options dialog box opens.
2. Select the Symbols tab.
3. In the Icon Labels group box, check the Route Direction Arrow check box.
4. Click Apply.

New Special Effects

VR-Forces supports the following special effects:

- Display of emitter beams
- Animation of munitions fire and detonation
- Sound effects (Windows only).

You can toggle these options on and off from the Options menu or the Symbols tab of the Display Options dialog box.

If there are no speakers attached to your PC when you start VR-Forces, and if you then attach speakers so that you can hear sound effects, you must restart VR-Forces to hear the sound effects.

See "Using Special Effects," in *MÄK VR-Forces User's Guide* for details.

Saving Default Entity Parameters to the Order of Battle File

In previous releases of VR-Forces (since checkpointing was implemented), when a scenario was saved, most entity parameters were written to the order of battle file, even if their values had not changed from the default values in the object parameters database. If a VR-Forces user subsequently changed the value specified in the object parameters database, it was not picked up by entities that had previously been saved because they got their initial values from the order of battle file. This behavior has changed for VR-Forces 3.7.

In VR-Forces 3.7, when an entity in a scenario is saved to the order of battle file, parameters that are unchanged from the initial values in the object parameters database entry are written out in the form:

```
(parameter-name USE-DEFAULT)
```

The USE-DEFAULT keyword indicates that the next time this file is loaded (as part of a scenario) it should be initialized from data in the entity's object parameters database entry. The practical effect of the USE-DEFAULT keyword in OOB files is that if you modify the object parameters database, then the next time you load a scenario, those changes can affect the entities and objects in the scenario.

You can edit the OOB file manually to override the value obtained from the object parameters database.

This change makes it possible to experiment with changes to entity parameters and view the effects of the changes to existing scenarios, without modifying the order of battle files.

New Examples

VR-Forces has the following new example applications:

convertParamDb

This example application converts an old format (pre-version 3.7) object parameters database file to the new format. If you have extended VR-Forces by modifying the object parameters database (to add entries for new kinds of entities or to modify existing entries), then you can use this utility to convert your modified file into the new format. See "Using the convertParamDb Utility to Convert Object Parameters Databases," on page 10 for more information.

createParamDbDynamically

This example demonstrates how to create and populate an object parameters database with entries dynamically, without reading it in from a file. It also shows how to create a type of object parameters database other than the type normally created by VR-Forces.

condition

The condition example (*./examples/condition*) explains how to add a new condition to the VR-Forces plan editor.

showMissileImpact

The Show Missile Impact example (*./examples/showMissileImpact*) shows how to add new symbols and model data to the GUI. This example was written for the MÄK Plan View Display, but it is equally applicable to VR-Forces.

New Configuration (MTL) Parameters

The *vrfSim.mtl* file has the following new configuration parameters:

- **VRF_defaultMainLoopSleepInterval** – sets the sleep interval for DtCgf::tick(). By adjusting how long your application sleeps before it ticks the cgf, you can tune the performance of the main application loop.
- **VRF_visibilityTestOrder** – specifies the order in which the isVisible() check processes visibility tests. For more information, see "Checking Line-of-Sight," on page 35.
- **VRF_publishSimulationTimeForStealth** – specifies whether or not to publish the simulation time to the network so that the MÄK Stealth can set its own simulation clock to work in conjunction with the simulation time. For more information, see "Publishing the Simulation Time to the MÄK Stealth," on page 21.
- **VRF_exerciseId** – allows you to set the DIS exercise ID in *vrfSim.mtl*.

Publishing the Simulation Time to the MÄK Stealth

The **VRF_publishSimulationTimeForStealth** parameter lets you publish the simulation time. The MÄK Stealth does not currently make any use of this information. However, a Stealth customer could use the Vega-Prime API to modify the Stealth and use this information. If you capture a VR-Forces simulation on a data logger recording, the information will be available for analysis. The syntax for the parameter is:

```
VRF_publishSimulationTimeForStealth mode
```

where *mode* is one of the following:

- 0 = do not publish simulation time
- 1 = publish simulation time.

Default: 0.

Loading An Alternative Bounding Volume Configuration File

You can load an alternative bounding volume configuration file at startup with the **-b** startup option, as follows:

```
vrfGui -b file
```

This option is valid only with the front-end.

Collision Detection and Avoidance Has Been Rewritten

The collision avoidance code has been rewritten. The collision detection and avoidance behavior has generally been improved. Entities now have an obstruction sensor, which can be configured to detect different types of entities to avoid. A new collision avoidance controller listens to the sensor and tries to avoid any potential collisions the sensor detects. This is managed programmatically by the Obstruction Manager.

In the object parameters database, you can specify which entities and vector features to avoid.

Component Changes

The collision controller has been split up into a much more extensible set of components.

- *DtObstructionSensor* (*obstructionSens.h*) – The primary responsibility of the obstruction sensor is to calculate the collision bounding volume of the entity, and to initialize the *DtObstructionManager* and query it for potential collisions. Any potential collisions detected are reported to the collision avoidance controller through ports.
- *DtObstructionManager* (*obstructionManager.h*) – The Obstruction Manager's responsibility is to determine if the specified collision bounding volume intersects (collides) with any of the user-specified obstructions. Any potential collisions detected are returned to the *DtObstructionSensor*.

- *DtGroundCollisionAvoidanceController* (*gndColAvoid.h*) – The ground collision avoidance controller is responsible for avoiding potential collisions detected by the obstruction sensor.

API Additions

The following objects have been added to the Terrain API:

- *DtRwObjectTypeList* (*rwObjectTypeList.h*) – Used by the *DtObstructionSensor* to allow you to specify an arbitrary number of *DtRwObjectTypes* to treat as obstructions.
- *DtRwVectorObstructionTypeList* (*rwVectorObstructionTypeList.h*)
- *DtRwVectorObstructionType* (*rwVectorObstructionType.h*)
- *DtRwStringAttribute* (*rwStringAttribute.h*)
- *DtStringAttribute* (*stringAttribute.h*)
- *DtRwIntegerAttribute* (*rwIntegerAttribute.h*)
- *DtIntegerAttribute* (*integerAttribute.h*)
- *DtRwRealAttribute* (*rwRealAttribute.h*)
- *DtRealAttribute* (*realAttribute.h*).

  These classes are all used by the *DtObstructionSensorDescriptor* to allow users to specify an arbitrary number of *DtRwVectorObstructionTypes*, which are basically human readable *DtSpecifications*. The *DtRwVectorObstructionTypes* are transformed into *DtVectorObstructionIDs* that contain the specification attributes specified in the *DtRwVectorObstructionType*. Any vector features with a matching specification (the attributes of the vector feature may be a superset of the attributes specified in the *DtRwVectorObstructionType*) are treated as obstructions to be avoided by the associated entities.

Updates to the Move-To Controller Component Hierarchy

Two new controllers have been added to the Move-To controller class hierarchy:

- *DtGroundMoveToDestinationControllerComponent*
- *DtGroundMoveToLocationControllerComponent.*

The *DtMoveToDestinationControllerComponent* (*gndMvToDestinationCntlr.h*) is now the base class of the hierarchy, with both the *DtMoveToControllerComponent* and the *DtGroundMoveToLocationControllerComponent* deriving from it. The MoveToDestination controller provides all of the functionality necessary for ground entities to move to a specified destination. The destination is stored in the ground entity's process state repository, but is also cached locally. The derived controllers, including not only the MoveTo (which is responsible for moving to a control point) and MoveToLocation, but also the child classes of the MoveTo, are now only responsible for properly updating the location, in world coordinates, stored in the process state repository.

For example, when processing a MoveToLocation task, the MoveToLocation controller simply calculates the location in world coordinates, sets it in the process state repository, and then defers to the MoveToDestination controller for the actual movement to the specified location.

Similarly, when moving to a specified control point (waypoint), the MoveTo controller simply ensures that the location in the process state repository matches the location of the control point. If the location of the control point moves, or if the control point no longer exists, then the controller must update the location in the PSR or clear the task as appropriate.

Note that whenever the location of the destination has changed, the child controllers must invalidate the destination vector so that it is recalculated next time it is requested. This can be accomplished by calling down to the MoveTo controller's processDestinationChanged() function, or overriding the function and calling invalidateDestinationVector() directly.

Changes to the *DtGroundMoveToControllerComponent* API

There are several important changes to the MoveTo controller API. Primarily, the following function names have changed:

- processWaypoingChanged() is now processDestinationChanged().
- setupControlPoint() is now setupDestination().

Maintaining backwards compatibility was not possible with these changes.

The MoveToDestination controller has modified somewhat the concept of the goal vector, now the destination vector. The destination vector is cached as before, but now when the destination vector is no longer valid, such as when the location of the destination changes, the controllers call the member function invalidateDestinationVector(). This ensures that the next time the destination vector is requested, it is recalculated. In derived controllers, ensure that when performing or responding to any action that invalidates the destination vector, invalidateDestinationVector() is called. The easiest way to do this is to override processDestinationChanged() to perform any controller specific functionality, and then call up to the parent's version of the processDestinationChanged() function.

Migration to the New MoveTo API

If you have derived a controller from any of the MoveTo controller components, you should not have to do much work to update your controllers. Ensure that you are calling the correct functions, setupDestination() and processDestinationChanged() as appropriate. And ensure that when overriding processDestinationChanged(), that you either call up to the parent's version of the function, or call invalidateDestinationVector() directly.

Member functions that used the word goal in their names, have had it replaced with destination, for example, atGoal(), passedGoal(), nearGoal(), and so on, have all been changed to atDestination(), passedDestination(), nearDestination(), and so on.

Changes to the Terrain API

The GDB file format has been updated to accommodate bug fixes and the vector network additions. You are strongly encouraged to re-import (to GDB format) terrain databases from the original source data, especially terrains containing vector data.

The tdbTool now lets you specify the notification level for logging. In order to change the level of output logging, use the –n option with a value between 0 and 4. See "Using the Terrain Converter Tool," on page 3, in *MÄK VR-Forces Configuration Guide* for details.

While the default value for planarizing is true, planarizing is required only for importing DFD and DFAD data. If the DFD and DFAD data is not planarized, results are unpredictable. Shape data does not have to be planarized.

Planarizing a vector network now only detects intersections between linear features (DtFeatureType equals 1) and only if the linear features are of MÄK group MAK010 or MAK030, which should be water and roads. As a result, only rivers and road intersections should be detected and fixed.

Deprecated Terrain API Classes

DtGdbVertexList class is deprecated and should no longer be used. Use std::list<DtVertex> where appropriate.

Updates to the *DtVectorNetwork*

The topology concept has been removed from *DtVectorNetwork*. The elements of the vector network are stored only once, and only in the forward direction. Forward is defined as the direction along the edge from the start point to the next point in the edge. The forward direction represents the direction of traversal for a one-way street, and can be either respected or ignored when traversing the points contained in an edge.

When creating a *DtNetworkEdge* programmatically, there is no longer any need to create an edge in the reverse direction. A single edge should be created.

Additionally, *DtNetworkEdges* now present a point interface, instead of a segment interface. Developers create *DtNetworkEdges* by adding points to them. An edge must have at least two points. Technically, the edges are still represented internally by segments created when points are added. These segments are owned by the *DtNetworkEdge* objects, and not by the *DtVectorNetwork*. However, developers are strongly encouraged to use the point interface, and not access the segments directly as they are now considered an implementation detail and may change in the future.

To iterate over the points of an edge, ask the edge for a *DtNetworkEdgeTraverser* object. When asking an edge to create an edge traverser, you must specify whether or not to respect the directionality of the edge. If you ignore the directionality, both a forward and backward traverser may be created. However, if you respect the directionality, then for a unidirectional street, only a forward edge traverser can be created.

Once a traverser is created for an edge, you must ask for a *DtNetworkEdgePointIter* over that edge. The point iterator acts as expected, and allows you to iterate over the points contained in the edge. The *DtNetworkEdgePointIter* abstracts away the direction of traversal. Thus, when moving both forwards and backwards along an edge, incrementing the *DtNetworkEdgePointIter* returns an iterator pointing to the next point as appropriate.

These changes were made to implement selection of terrain features and to present a much more flexible, robust, and simpler interface to the *DtVectorNetwork* and the *DtNetworkEdge*. See the updated terrain examples for detailed examples of the new vector network API in use, including feature creation and intersection selection.

Clarification of Vector Network Intersection

When the vector network is intersected, the returned features do not necessarily intersect the specified chord if the interest range is larger than the size of the feature as defined by its attributes. A more appropriate conceptualization of vector network intersection is to think of the intersection as selecting the features of the vector network that meet the specified criteria, much like a SQL SELECT statement selects data from a database. An intersection call to the vector network, with a vector record, is conceptually similar to selecting all records from a database where the records' attributes are equal to the vector record.

As a result, to perform an actual intersection with vector features, the features must be selected from the vector network, and then an actual intersection test must be done, using the properties of the features that define an intersectable object, presumably a volume (width, length, height).

Updates to Metrics Used to Calculate Intersections

The *DtRangeNetworkNodeMetric* and *DtRangeNetworkSegmentMetric* classes now respect the **EvalInZ** parameter. If **EvalInZ** is false, they ignore the Z value in their calculations. If **EvalInZ** is true, they evaluate Z when calculating distance.

Updates to the Display of Vector Point Features

Point features that have both length and width are now displayed as solid rectangles. This applies only to DFD data. DFAD point features do not have width attributes.

GUI API Changes

This section describes changes to the GUI API. Some of these changes require you to update existing code.

Ability to Select Terrain Feature Objects

Symbol selection has been changed to support selection of terrain feature objects. There are now two types of selectors for object selection: *DtObjectSelector* (which takes the place of the existing *DtSymbolSelector*) and *DtTerrainSelector* (which allows the selection of terrain features). If you are currently subclassing *DtSymbolSelector*, you must now subclass the *DtObjectSelector*.

If you have overridden the selectSymbols() member function, the prototype has changed to now include the state of the mouse and the state of the keyboard. The symbol selectors are now responsible for checking these flags in order to decide whether or not to merge the new selection into the existing selection. The merge() member function is responsible for this operation and will only be called if the Ctrl key is being pressed.

The main member functions for determining whether or not a symbol is valid for a particular selection mode have not changed. If you have added new selection modes or are filtering out objects based on an existing selection mode, once you change your subclass to *DtObjectSelector* you should not have to change any additional code.

The Entity Information Dialog Boxes have been Redesigned

The entity information dialog boxes have been completely rewritten to support the display of resources and to support a smaller, more concise display of information. The dialog boxes are organized as a set of tabbed pages. The pages that are supported for each dialog box types are:

- *DtEntityInfoDialog*
  - *DtEntityStateInfoPage* – displays information about speed, location, velocity, and so on.
  - *DtBaseEntityInfoPage* – displays entity ID, force ID, entity type and guise.
  - *DtEntityAttributePage* – displays information about entity attributes.
  - *DtResourceInfoPage* – displays resources for the entity and allows for modifications of resource update and display.

- *DtAggregateInfoDialog*
  - *DtEntityStateInfoPage* – displays information about speed, location, velocity, and so on.
  - *DtAggregateInfoPage* – displays information about number of subordinates, dimensions, and so on.
  - *DtAggregateSubordinateInfoPage* – displays information about subordinates of this aggregate.
  - *DtResourceInfoPage* – displays resources for the entity and allows for modifications of resource update and display.

Migrating Code to the New Entity Information API

If you have modified the aggregate or entity information dialog box, you must update your code to match the new API as follows:

1. Decide which page you want to put your information on.
2. Subclass the specific object for that page and create it in the dialog box you are overriding. Use the correct member function for the dialog box type (Table 2) and return a new instance of your page.

Table 2: Entity Information dialog box objects and member functions

Dialog Box Class	Object	Member Function
DtEntityInfoDialog	DtEntityStateInfoPage	createEntityStatePage()
	DtBaseEntityInfoPage	createEntityInfoPage()
	DtEntityAttributePage	createEntityAttributePage()
	DtResourceInfoPage	createResourceInfoPage()
DtAggregateInfoDialog	DtEntityStateInfoPage	createEntityStatePage()
	DtAggregateInfoPage	createAggregateInfoPage()
	DtAggregateSubordinateInfoPage	createAggregateSubordinateInfoPage()
	DtResourceInfoPage	createResourceInfoPage()

3. In your subclass of the page, override initPageContents() to add your items to the display. To add an item, call the addInfoItem() member function, providing a unique ID and the title of the item. The item is added to the list box that is displayed on the page. The first column displays the title; the second displays the value of the item.

4. To update the value of the item you added, override the updatePageContents() member function. The model key and the state repository of the item being updated are passed into this member function. To update a particular item, call infoItem() to retrieve the QListViewItem created when you added your item. Then pass the unique ID you used when creating the item. Once you have the item, set its value by calling the Qt member function setText(1, *value*) on it.

Adding A New Page to An Information Dialog Box

If you want to create a new page to display items, you must subclass *DtEntityInfoPage* and implement the following member functions:

- init()
- initPageContents() – adds new page items
- createPageContents() – creates the actual page to display information
- title() – specifies the title of the tab on the page
- updatePageContents() – updates the page's contents.

The init() member function should look like this:

```
// The BaseInfoPageUi contains a page that has a list box associated with
// it. If you want to create a new page, this is where the contents should
// go)

QWidget* DtAggregateInfoPage::createPageContents()
{
  myInfoPage = new BaseInfoPageUi(this);

  return myInfoPage;
}

bool DtAggregateInfoPage::init()
{
  if (!DtEntityInfoPage::init())
  {
    return false;
  }

  myInfoView = myInfoPage->InfoBox;
  myInfoView->setSorting(-1);

  return initPageContents();
}
```

The updatePageContents() member function is called only if your page is the current page being displayed.

Creating Graphical Objects

The DtCreateControlObjectHandler::create() member function has been changed to:

```
virtual void create(QPtrList<DtPoint>& points, QString name,
  QString label, QColor color, int forceType, int appearance,
  bool publish, DtArchivableObject* userData, bool createDone = true);
```

The create() member function now includes the points that are used to create the object and a new optional flag at the end of the createDone() member function. If this flag is true (the default), the create dialog box is removed after the object is created. If this flag is false, the dialog box stays open. This flag was added to support the creation of multiple routes from a terrain selection when turning a selected feature into a route. This change also means that the signals that are emitted from the *DtLineConfigWidget* and *DtPointConfigWidget* have been changed to match the new create() member function.

Setting Resources

The set resources widget (*DtSetResourcesWidget*) now queries the back-end for the resources of the selected entity, and displays those resources and the current values. If you have overridden the set resources widget to show a specific resource, you no longer need to override it.

Filtering Pick Lists

The task, set, and condition dialog boxes that require a user to pick an entity or object, such as Move To or EntityInArea, now support filtering of the pick lists. For example, you can filter for all entities, or only friendly entities, and so on. The Move To tasks have been generalized to include all point objects and entities.

These task and set widgets are now subclassed from a new class, *DtSelectionWidget*. This widget is a generic class that allows filtering of entity types. The classes affected by this change are:

- *DtMoveToWaypointWidget* has been replaced by *DtMoveToWidget*
- *DtPatrolBetweenWaypointsWidget* has been replaced by *DtPatrolObjectsWidget*
- *DtEntityInAreaWidget*
- *DtFireEffectEntityWidget*
- *DtFollowEntityWidget*
- *DtMoveAlongRouteWidget*
- *DtMoveToWidget*
- *DtPatrolAlongRouteWidget*
- *DtPatrolObjectsWidget*
- *DtSetTargetWidget*

If you have subclassed any of these widgets to accomplish the following results, you must update your code, as follows:

- Modified a widget to add additional information for a task or set data request.

  There is now a layout for each of the affected widgets called SelectionWidgetUiLayout. If you have added additional information to a specific layout of one of these widgets, you can make those modifications in the new init() member function prototype (init(DtTMMapArea, int iNumListBoxes)).

- If you have added additional entity types that can be selected for a task or set data request, you can now add them via selection filters. Override the initFilters() member function. Call the parent member function to set up the base selection filters, then add your own filters with the addFilter() member function. The entity type can be any entity type (specific or wildcard) you want to select. If you want a more complex filter, the following filters are available and you can add them to the filter list by calling addFilter(DtFilter*):
  - *DtObjectFilter* - Select an object of a specific type.
  - *DtNotObjectFilter* - Select any object except this object type.
  - *DtKindLessThanFilter* - Select any object whose kind() is less than the kind specified in the object filter (and matching all other conditions).
  - *DtMatchAnyFilter* - Matches any of the supplied entities keys added via the addKey() member function.
  - *DtOrFilter* - This filter contains one or more filters such that the first filter that evaluates to true will allow that item to be selected.
  - *DtAndFilter* - This filter contains one or more filters such that all filters must evaluate to true for the particular type in order for this item to be selected.

Creating Filtered Lists

The *DtFilteredListView* class manages filtered lists. You can use this class in your own dialog boxes to allow users to select from a list of filtered entity items. This class also contains a combo box, which, if more than one filter is present, allows the user to select from any of the filters to show a set (or subset) of selectable items. For example, here is the filter code for the *DtMoveToWidget*:

```
bool DtMoveToWidget::initFilters(unsigned int iListBox)
{
  addFilter(trUtf8("Waypoints"),
    DtEntityMapperKey(-1, -1, DtPointObjectKind, -1, -1,
    DtPointObjectCategoryControlPoint, -1, -1, -1));

  DtOrFilter allFilter(trUtf8("All"));

  DtKindLessThanFilter lessThan(DtEntityMapperKey(-1, -1,
    DtLinearObjectKind, -1, -1, -1, -1, -1, -1));
  DtObjectFilter symbols(trUtf8("Overlay Object"),
    DtEntityMapperKey(-1, -1, 101, -1, -1, -1, -1, -1, -1));

  allFilter.addFilter(&lessThan);
  allFilter.addFilter(&symbols);

  addFilter(trUtf8("Overlay Object"),
```

```
    DtEntityMapperKey(-1, -1, DtPointObjectKind, -1, -1, -1, -1, -1,
      -1));

    addFilter(&symbols);

    addFilter(trUtf8("Friendly Objects"),
      DtEntityMapperKey(DtForceFriendly, -1, -1, -1, -1, -1, -1, -1, -1));

    addFilter(trUtf8("Opposing Objects"),
      DtEntityMapperKey(DtForceOpposing, -1, -1, -1, -1, -1, -1, -1, -1));

    addFilter(&allFilter);

    return true;
}
```

The **iListBox** parameter is the list box number that is being initialized (for example, the *DtPatrolObjectsWidget* has two list boxes, each of which can contain their own filter)

Other Changes to the GUI API

The following changes have been made to other objects:

- *DtSymbol* - A new member function has been added called scheduleDisplayComplete() that takes an argument specifying the number of milliseconds this symbol should be displayed. When this member function is called, a QTimer is created to run for the specified amount of time. When the timer is finished, the symbol emits the displayComplete() signal. You can then connect to this signal to perform some action on it. The detonation and fire symbols use this member function to determine when they should be removed from the display.
- *DtPvdSymbolGenerator* - There is no longer an event list to determine when fire and detonate interaction symbols are to be removed from display. The new scheduleDisplayComplete() member function is used to set up a timer for each symbol to determine when the symbol should be removed from the display. By default this is two seconds for non-animated symbols, or the length of time it takes to complete the animation for animated symbols. If you want to change the length of time these items are displayed, you can call the setInteractionTimeout() member function of the symbol mapper for all non-animated interaction symbols or install your own subclass of the symbol mapper and override the scheduleDisplayComplete() member function to schedule your own termination time for the symbol
- *DtPvdMapArea* - The **mySymbolSelector** member variable no longer exists for the map area. There are three new member variables:
  - **myCurrentSelector**
  - **myObjectSelector**
  - **myTerrainSelector**

The **myCurrentSelector** variable is set to either the object or terrain selectors (depending on which one is current). By default it is set to **myObjectSelector**.

- *DtTerrainBackground* - The draw member function now takes an optional painter argument on which to draw. If it receives a PaintDevice, it draws to it. If no drawer is supplied, all drawing is done to the **myPixmap** member variable. The QPainter argument was added so the background could be drawn to a printer.
- *DtSymbolLensArea* - All references to Quad trees have been removed to clean up the class, because quad trees were not used and not fully implemented.
- *DtPvdEntitySymbol* - A new member variable, **myEmitterSystems**, has been added to keep track of the emitter system symbols that might have been added to an entity. Emitter symbols have their own model data and symbol representation, so this member was added to tie the emitters to their owning entities.

VR-Forces API changes

This section describes changes to the VR-Forces (back-end) API.

Iterators

The various iterators VR-Forces uses have been changed to use templates. If you use these iterators, you do not need to make any changes to iterate through your lists. Iteration can now be done in a type safe manner. For example, if you have a *DtList* of a certain type of object, you can use the DtIteratorBase<class T> iterator to iterate over those objects in type-safe way as follows:

```
DtIteratorBase<MyObject> iterator(myList);
MyObject* objPtr = NULL;

for (objPtr = iterator.first(); objPtr ; objPtr = iterator.next())
{
}
```

Target Detection Sensor

DtTargetDetectionSensor and its use of the *DtVrfObjectStateRespository*::isVisible() member function have been refactored to make it easier to subclass.

The target detection sensor is one of the most complex and execution intensive models in the API. The API changes are designed to simplify the model, while making it faster to run and use. The isVisible() member function in the object state repository has been extended to easily allow certain visibility checks to be performed (or not performed) and easily add new types of visibility checks. The following sections describe how the target detection sensor works, the modes it can run in, visibility checks, and the API changes made.

How the Target Detection Sensor Works

The target detection sensor finds targets and places them in a list that other models can use to determine which of the targets the sensor knows about is usable during that tick. When determining which targets can be considered, the target detection sensor:

1. Checks to see if the entity is within range of the sensor.
2. Checks to see if the target is in the sector of responsibility. If it is, it checks the random detection number in the target detection file to see if the target is discovered.
3. If line-of-sight checking is enabled, checks line-of-sight to see if there is anything blocking the target. The three checks currently supported are Polygon, Entity, and Feature.
4. If the entity cannot be seen, but the entity was at one point detected, a random number is drawn to see if the target is removed from the target list. If it is not, it remains in the list even though it cannot be seen.
5. If there is a user target override (Set Target), add that to the list.

The member functions used to perform these steps are as follows:

```
tick()
    discoverTargets()
        fillInTargetList()
            if closest target mode calculateClosestTargets()
                acceptTarget()
                    entityInRange()
                    targetInSectorOfResponsibility()
                    checkLineOfSite()
                    checkEntityDetectForgetProbability()
            else checkEntityLists() (uses entity priority lists)
                acceptTarget()
                    entityInRange()
                    targetInSectorOfResponsibility()
                    checkLineOfSite()
                    checkEntityDetectForgetProbability()
```

Specifying the Number of Targets to Track

The updated target detection sensor allows you to specify how many of the closest targets to track over a period of time. This can enhance performance, because the isVisible() check (the computationally expensive part of this model) is made only be made for the specified number of closest entities. The following descriptor values have been added to configure this mode:

```
;; (use-closest-targets True) Toggle closest targets mode on or off.
;; (number-closest-targets 10) Specify the number of targets to check.
;; (update-closest-tick-period 10.0) Specify how long to track these
;; targets.
```

If you enable tracking of only the closest targets, the target detection sensor proceeds as follows:

1. At the start of the detection period, the sensor uses the prioritized entity lists to determine the range order of the targets in question.
2. It goes through the range ordered list until it finds the top *N* closest visible targets.
3. For each tick during the specified period of time, only the *N* closest targets are checked. If, before the end of the period, a target is destroyed, it is removed from the list and the next closest entity is added to the list.
4. When the **update-closest-tick-period** is over, the target detection sensor repeats the process.

If **use-closest-targets** is false, the sensor always checks the full prioritized entity list each time it is scheduled to run.

DtEntityTargetInformation Object

The *DtTargetDetectionSensor* now puts a more complex object on the target port list than a simple entity ID. This object is called *DtEntityTargetInformation* and contains the following items:

- The target name
- The target entity ID
- The original time of detection (the first time it showed up in the sensor)
- The last time of detection (the last time the sensor put this entity into the list)

If your code currently iterates through the target port list, you must cast the return value from *DtEntityIdentifier* to *DtEntityTargetInformation*, or you can change your iterator code to the following:

```
DtEntityTargetList::const_iterator iter(*targetList);

for (const DtEntityTargetInformation* info = iter.first(); info;
  info = iter.next())
{
  if (info->id() == currentTargetId)
  {
    targetFound = DtTrue;
    break;
```

```
}
```

If you do not update your target detection code, the back-end application will terminate abnormally when your target detection code is exercised.

If you override the functionality to add your own targets to the target list port, you must now add this new type of object to the port as follows:

```
DtEntityTargetInformation target(entity->objectIdentifier(),
  discoveryTime(simTime));

myTargetListPort->addEntity(target);
```

where simTime is the current simulation time.

Checking Line-of-Sight

If the target detection sensor is configured to do line-of-sight checking, it uses the isVisible() member function in *DtVrfObjectStateRepository*. This member function has been enhanced to allow you to configure which visibility tests to perform and the order in which they are performed. The *vrfSim.mtl* file has a new parameter defined as follows:

```
(setqb VRF_visibilityTestOrder "Polygon Entity Feature")
```

Currently, three modes (polygon, entity, and feature) are supported for visibility testing. The order in which they are listed in the parameter is the order in which the tests are performed. For performance reasons, if your database has a lot of polygons, but few entities, you may want to order the entity check first. If your simulation has a lot of entities, then you should keep the polygon check first.

Using static methods defined in *DtVrfObjectStateRepository*, you can easily add your own visibility tests to the model. These tests apply to all sensors and can be added as follows:

1. To add a new type of test:

   ```
   DtVrfObjectStateRepository::addVisibilityTest(<Test Identifier>,
     DtString("<Test Name>"), <Test Function>);
   ```

   The function callback prototype is:

   ```
   typedef bool (*DtVisibilityTest)(double rangeSquared,
     const DtEntityIdentifier& eyeKey, const DtVector& localEyePoint,
     const DtVector& localEndPoint, const DtVrfObjectStateRepository* sr);
   ```

   The return value of your function will be true if the object is blocked by some item and false if not.

2. To schedule the test, call:

   ```
   DtVrfObjectStateRepository::performTest(Test_Identifier)
   ```

   or you can modify the **VRF_visibilityTestOrder** parameter to include your new test name in the test order.

When you upgrade to the new target detection sensor, you should not have to change much (if any) code, because the main tick() method has not changed. The addition of new entry points, however, should allow you to organize your code to better use the sensor.

Miscellaneous API Changes

Table 3 lists classes that have been deprecated. In many cases, classes were removed to reduce the depth of certain inheritance hierarchies. Obsolete class names have been typedef'd to corresponding new class names to aid in the transition to new names and minimize initial impact on customer code. Users are encouraged to migrate existing code that references deprecated class types and header files to the new types and header files as soon as possible to ensure compatibility with future versions of VR-Forces.

Table 3: Deprecated classes

Old Class	New Class
Object Parameters Database Classes	
DtSimObjectParametersDB (*objParamDb.h*)	*DtVrfObjectParametersDB* (*objParamDb.h*)
DtSimObjectParametersList (*objParamList.h*)	*DtVrfObjectParametersList* (*objParamList.h*)
DtSimObjectParameterFactory (*paramFact.h*)	*DtVrfObjectParameterFactory* (*paramFact.h*)
DtSimObjectParameterFactoryEntry (*parFactEnt.h*)	*DtVrfObjectParameterFactoryEntry* (*parFactEnt.h*)
Parameter Classes	
DtSimObjectParameters (*simObjParams.h*)	*DtVrfObjectParameters* (*vrfObjParams.h*)
DtVrfObjectBaseParameters (*vrfObjBasePar.h*)	*DtVrfObjectParameters* (*vrfObjParams.h*)
Miscellaneous Classes	
DtSimObjectSubComponentTypes (*simObjSubCmpt.h*)	*DtVrfObjectSubComponentTypes* (*vrfObjSubCmpt.h*)
DtKeyedSimObjectIterator (*keyedObjIter.h*)	*DtBaseSimObjectIterator* (*keyedObjIter.h*)
DtKeyedVrfObjectIterator (*keyVrfObjIter.h*)	*DtBaseSimObjectIterator* (*keyedObjIter.h*)
State Repository Classes	
DtSimObjectBaseStateRepository (*baseObjSR.h*)	*DtSimObjectStateRepository* (*simObjSR.h*)

Table 3: Deprecated classes

Old Class	New Class
DtVrfObjectBaseStateRepository (*vrfObjBaseSR.h*)	*DtVrfObjectStateRepository* (*vrfObjSR.h*)
Vrf Object Classes	
DtSimObject (*simObject.h*)	*DtVrfObject* (*vrfObject.h*)
DtSimObjectData (*simObjectData.h*)	*DtVrfObject* (*vrfObject.h*)
Object Manager Classes	
DtSimObjectManager (*simObjectMgr.h*)	*DtVrfObjectManager* (*vrfObjMgr.h*)
DtSimObjectListInterface (*simObjListIF.h*)	*DtVrfObjectManager* (*vrfObjMgr.h*)

The following classes have been removed:

- *DtEntityParametersDBCompatibility1x* (*parDbCompat1x.h*)
- *DtEntityParametersListCompatibility1x* (*parLstCmpat1x.h*).

Table 4 lists deprecated header files.

Table 4: Deprecated header files

Old filename	New file name
simParamTypes.h	*vrfParamTypes.h*
simObjTypes.h	*vrfObjTypes.h*

- For performance and memory usage reasons, the copy of the parameter entry that used to be made when creating a state repository is no longer made when the state repository is initialized. The **myParameterEntry** member variable points to the original parameter entry in the object parameters database and should only be accessed by the `const DtVrfObjectParameters* parameterEntry() const` member function.

  If you want to get a copy of the parameter entry for modification, do not access the **myParameters** entry directly, because it will be NULL on access until a copy is created. This copy is created by calling the non const parameters() member function to return the **myParameters** entry. If a copy does not exist, one will be created by calling the createParameters() member function (at which point you can set the **myParameters** entry to your new parameter instance – this should be the only time you access the **myParameters** member variable directly). The parameters will then be initialized and setupParameterInheritance() will be called.

  If you try to access the **myParameters** variable directly, the program might crash at runtime because it might be NULL, so your code should be written (or modified) to only call the member function that will return the information you want:

`const DtVrfObjectParameters* parameterEntry() const` returns a const pointer to the original parameter entry
`const DtVrfObjectParameters* parameters() const` also returns a const pointer to the original parameter entry. This member function is provided for backwards compatibility only and could be deprecated in future releases.
`DtVrfObjectParamters* parameters()` returns a mutable copy of the original parameter entry. Changes to this entry will not effect the original pointer.

- Due to a performance enhancement change, the **myComponentDescriptor** member variable is now defined to be `const` and all access to this descriptor (including the componentDescriptor() member function) has been changed to be `const` access. If you have used this descriptor to modify values within the descriptor (either in the original descriptor or a derived descriptor class), this change will effect all entities that share this descriptor, because this descriptor now points to the original parameter database entry.

  If you currently modify member variables in a descriptor class, it is recommended that you make those member variables part of the component that uses them and modify them there. If you must modify the entries in the component descriptor, instead of using the supplied descriptor pointer, you must clone the descriptor entry and work with the clone. You must also delete the clone in your component destructor to avoid a memory leak.

- The placement of ground entities has been updated to allow for multiple terrain level support. When a ground entity is first placed on the map, it is placed on the highest possible terrain intersection. After that, it can respond to Set Altitude and Set Location requests that change its altitude. Therefore, for multiple terrain elevations, use Set Altitude and Set Location to change the elevation of an entity. For instance, use these functions to change the floor of a building where the entity is located.

  When ground entities move, they apply the same logic when calculating their next position. Thus, if the next calculated location resides in between two levels of terrain, and if the entity is configured to respect terrain intersections when moving and there is no terrain blocking their movement, the entity places itself at the closest elevation to the desired location.

- A parameter for calculating terrain intersections during movement has been added to the automotive actuator. The parameter, **calculate-terrain-intersections** defaults to false. When set to true, the actuator checks to see if any terrain exists between the current location and the next calculated location. If so, the entity moves only if the intersected terrain is passable. By default, any terrain with a slope greater than 80 degrees is not passable. To override this, derive a new actuator and override the terrainPassable() member function. See the class documentation for more information about the new behavior.

New Build System

The VR-Forces distribution has a new directory – *./scripts*. This directory contains files that allow you to modify example projects and your projects to easily add your own files without having to understand the format of the project files supplied with VR-Forces or to directly modify the system.

Each example or user directory, has the following project-related files:

- *gen.pro* (the Qt profile that is used to generate the *.dsp* or *.vcproj* files)
- *files-cc.txt* (a text file that lists the source files in the project)
- *files-h.txt* (a text file that lists the header files in the project)
- Possibly a *files-ui.txt* file (Qt designer files that define dialog boxes). This file is present if there are no dialog boxes in the application.

If you want to add dialog boxes to a project that does not have them, you must do the following before you generate your project files:

1. Create a file named *files-ui.txt* that lists the *.ui* files in your project
2. Add the following line to the *gen.pro* file:

   ```
   FORMS = $$system(type $${FILES_UI)
   ```

To generate a new project:

1. Open up a DOS command window and change your directory to the *./scripts* directory (for example, *C:\MAK\VRF3.7\SCRIPTS*)
2. Add your new source file names to *files-cc.txt*, your new header file names to *files-h.txt* and your new Qt UI files to *files-ui.txt.*
3. Run the *buildProject* program. The syntax for buildProject is:

   ```
   buildProject source_directory project_name
   ```

   For example, if you want to modify the *scenarioStatistics* example, run the program as:

   ```
   buildProject examples scenarioStatistics
   ```

   The only exception to this is the *conditionGui* example. To rebuild that project you must run the program as:

   ```
   buildProject examples\condition conditionGui
   ```

   The program creates six files – separate debug and release projects for DIS, HLA13 and HLA1516 projects. The projects named **D.dsp* (or **D.vcproj* if you are using .NET) are debug projects and only the debug configuration of that file is valid. The **R.dsp* (or **R.vcproj* if you are using .NET) are release projects and only the release configuration of that file is valid.
4. Create a new workspace.
5. Add the project files to the workspace that you want to build.

As you modify the *files-cc.txt*, *files-h.txt*, or *files-ui.txt*, remember to regenerate the project files in order to add those files to your project.

Documentation Updates

This section lists corrections to printed documentation and the status of electronic documentation provided with this release.

- Online help is now displayed in the Qt Assistant help browser, not the default web browser for your computer. The new browser opens more quickly and supports full text search, bookmarks, and the ability to increase the size of the text displayed in the window.
- The PDF version of *MÄK VR-Forces User's Guide* provided with this release has been updated to describe all new user features.
- The PDF version of *MÄK VR-Forces Configuration Guide* provided with this release has been updated with the changes to *tdbtool*, map file configuration, and the new object parameters database format. It does not include descriptions of new sensors and controllers used for collision avoidance or a description of how to configure obstruction avoidance. See these release notes for that information.
- The PDF version of *MÄK VR-Forces Developer's Guide* provided with this release has been updated with the API changes to the GUI API and Terrain API. Other changes to the APIs that are described in these release notes are not reflected in the manual.
- If you create a *DtSpecification* for an areal vector feature, that is, the specification has a FeatureType = 2 attribute, then the specification must also contain the following attributes:
  - DtXMin
  - DtXMax
  - DtYMin
  - DtYMax
  - DtZMin
  - DtZMax.

  Each of these attributes is a double. They represent the extent of the areal feature. If these attributes are not present, VR-Forces uses 0.0, which may result in unintended consequences.
- The **max-slope** parameter is defined incorrectly in *MÄK VR-Forces Configuration Guide*. The **max-slope** parameter specifies the maximum slope at which the entity can still have acceleration >= 0 in its direction of motion. Therefore, if a ground vehicle is heading up a hill, as soon as the slope of the hill is greater than the max-slope, it will no longer be able to accelerate up the slope. At best, without any friction or other source of negative acceleration, it will remain at its current speed. If the slope increases, it will start decelerating and even start rolling backwards. If the entity is moving down a hill, the opposite is true.

- The definition of the **max-taxiing-turn-speed** parameter in *MÄK VR-Forces Configuration Guide* may be unclear. It is the maximum speed at which a fixed-wing entity can taxi and still be able to turn. If an entity is taxiing at a higher speed and it has to turn for any reason, it must slow down.
- The -B startup options (for loading a batch file) works only with the back-end (vrfSim) in separate mode.
- Chapter 2, Working with Terrain Databases, in *MÄK VR-Forces Configuration Guide*, has a new section that describes the terrain databases supplied with VR-Forces.
- The code sample on page 2-8 of the printed version of *MÄK VR-Forces Developer's Guide* has an error. The line:

  ```
  creator.setObjectManagerCreator(MyObjManagercreate);
  ```

  should read:

  ```
  creator.setObjectManagerCreator(MyObjManager::create);
  ```
- The trackHistory example referenced on page 11-48 of *MÄK VR-Forces Developer's Guide* is no longer provided. Track histories are now part of the delivered application.
- The documentation does not make clear that if successive scenarios use the same object parameters database, VR-Forces does not reload the object parameters database before running each scenario. Therefore, if you are making changes to the object parameters database, you must close VR-Forces and restart it to put the changes into effect.
- If you increase the speed at which a scenario runs (by increasing the time multiplier), the frame rate is reduced and performance of models may degrade.
- The description of the Entity Has Target condition in *MÄK VR-Forces User's Guide* is incorrect. It suggests that the condition is true when the entity that owns the plan has acquired the entity specified in the condition. The correct description of the condition is that it is true when the entity specified in the condition has any target. For example, consider the entities in Figure 5. The condition in M1A1 1's plan is true when BMP 1 has a target, not when M1A1 1 targets BMP 1.

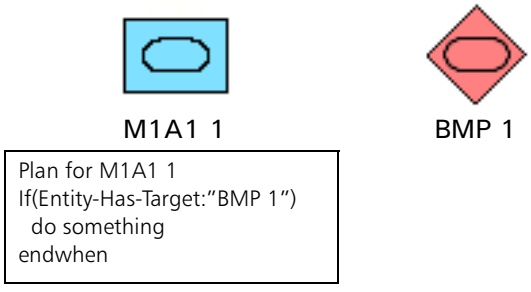

Figure 5. Entity Has Target condition

- The discussion of how to calculate the hit probability coefficient in Chapter 7 of *MÄK VR-Forces Configuration Guide*, has been clarified, as follows:
  - **coefficients** – a coefficient probability list is a list of coefficient probability entries. Each entry has a probability name and the coefficients of a standard polynomial that can be used with a given range to determine the specified probability. The order of the polynomial is determined by the number of coefficients provided. For example, an entry of:

    ```
    (probability -5.00000E-008 -300000E-012 1.0000)
    ```

    indicates that the formula to use for a given range *r* would be:

    $(- 5.00000E-008 * r^2) + (-300000E-012 * r) + 1.0$

    Values calculated will be constrained to be between 0.0 and 1.0. Zero values are factored out, for example:

    ```
    (probability -5.00000E-008 0.0000 1.0000)
    ```

    indicates that the formula to use for a given range *r* would be:

    $(- 5.00000E-008 * r^2) + (0.0 * r) + 1.0$

- Importation of OpenFlight terrain databases has the following restrictions:

  The following nodes are not imported or displayed:

  - Mesh Nodes
  - Switch Nodes
  - Light Points
  - Light Point System
  - DOF Nodes
  - Text Nodes
  - Sound.

  The following features are not imported:

  - Local Origin Offset (necessary CTS default)
  - Support Flat Earth Natively (We convert it to UTM, CTS/Creator default)
  - Billboards
  - UV map coordinates
  - Normals
  - LOD Centers
  - Geospecific RGB attributes.

The following face attributes are not supported:

- Multi texture
- Polygon Type Wireframe or Closed Wireframe
- Shade
- Line Style
- Transparency
- Material
- IR Color
- Feature ID
- SMC
- Cultural Footprint
- Roofline
- Terrain.

Bug Fixes

This release fixes the following problems:

- A bug in vector data clipping has been fixed. It's highly recommended that terrains that were clipped, planarized, or both, be reimported from their original format.
- Entities should now move correctly on different soil types. It is highly recommended that terrains be reimported from their source format to ensure proper behavior.
- Shapefiles containing elevation values for the data are now correctly imported.
- A fix has been added for incorrectly generated Soil Types. It will not catch all types of soil type issues, and may incorrectly map them, so regeneration from original data is highly recommended.
- OpenFlight files containing instances will now load correctly. Some OpenFlight files contained instances that were not properly interpreted. This has been fixed.
- A bug in the `-k` option for *tbdTool* has been fixed. Failure to load properly will also be reported.
- Paged GDB files are now imported and loaded correctly. Any paged Open Flight files imported in VR-Forces 3.5.1 or 3.6 should be regenerated from source data as they were incorrectly created.
- The AddSimpleFeatures example now uses the new Terrain API properly and has the proper specification attribute values.
- Fixed Wing Entities pitch is now properly clamped. The pitch will no longer be outside of -45 to 45 degrees.
- Entities will no longer move on inappropriately sloped terrain.

- The terrainHeight() function has been fixed to no longer return an empty intersection record.
- When intersecting the vector network, areal features were not intersected properly. This has been fixed and will improve correctness of selection process as well as performance.
- *DtNetworkEdges* were not being clipped properly, but only for linear features. Previously, this would create a degenerate edge. This has been fixed, although it is highly recommended that the GDB files be recreated from source data.
- When following another entity, but not tasked to do so (moving in same general direction and behind the entity), the secondary entity will no longer collide with the primary entity (followed).
- Entities will now correctly avoid collisions while performing another task, such as moving along a route. Previously, some entity/task combinations caused the entities to continue to move after they should have completed their task and stopped.
- Entities will no longer accelerate inappropriately while avoiding collisions.
- Clicking the New Folder button in any of the file selection dialog boxes no longer causes the GUI to appear to be hung.
- Surface entities now come to a stop when destroyed.
- Dismounted infantry entities now stop at the end of a route if opposing force DI are present near the end of the route.
- Dismounted infantry entities no longer snap to the wrong heading when loaded in from a scenario.
- Selected entities now show up correctly when moving from off screen on to the display. Previously, a map zoom or pan event was required to get them to appear when moving back onto the display.
- vrfGui now accepts a `-n` command line argument, enabling output of *DtDebug* statements in the front-end console.
- Some older GDB files that contained spatial indexing would not render correctly. These files now load correctly.
- Some older GDB files with spatial indexes would not load. These files will now load, however it is recommended that customers re-import older databases. The command to use with tdbtool to remove spatial indexing is:

  ```
  tdbtool -s 0,0,0
  ```

  To regenerate a database, use:

  ```
  tdbtool -s x,y,z
  ```
- Articulated parts (turrets, launchers) now stop moving when an entity is destroyed.
- Missiles and torpedoes can now fly outside of the terrain area. Previously they would detonate immediately upon leaving the terrain area.
- The mobility-kill and firepower-kill entries have been removed from the damage files because they do not have any effect on the damage actuator.
- If a surface entity strikes land, it will now stop.

- If an entity is given a Wait task while it is avoiding a collision, it stops and does not continue with collision avoidance.

Known Problems

VR-Forces Release 3.7 has the following known problems:

- If you are using VR-Forces in a DIS exercise, you may experience problems with dropped packets and entities may time out. To avoid this problem, you can use asynchronous I/O by starting with the `-I` parameter.
- Aggregates composed of fixed-wing entities do not carry out tasks as an aggregate.
- 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.
- Fixed-wing take-off, landing, and taxiing behaviors may be difficult to configure through the front-end when simulating on geocentric terrain databases. This is due to the fact that it uses a different projected terrain database in the front-end when simulating on a geocentric terrain database in the back-end. Correlation errors in terrain elevation between the two databases must be small (on the order of meters) when placing waypoints or routes on the ground to support fixed-wing ground behaviors. Either use high resolution terrain databases (minimizing the elevation error between the front-end and back-end databases) when configuring these behaviors, or restrict your scenarios to fixed-wing flying behaviors.
- Using the *tdbtool* 's balancing option (`-b br,lf`) to balance certain terrains may cause the generated *.gdb* file to display incorrectly. This has to do with a draw ordering bug that is not currently corrected for all terrains. The resulting GDB file will still give correct intersection information, so it will still be suitable for simulation purposes.
- Surface entities are not configured to fire weapons. You can customize the object parameters database entries to enable this.
- The Qt translation files (in *./translations*) for built-in Qt-level GUI items do not work properly. The translation files for VR-Forces work correctly.
- The Set Formation request does not work properly on aggregate entities that are subordinate members of larger aggregate units.
- If you are running VR-Forces on Linux with the DMSO RTI, you cannot run in combined mode. Start the GUI and back-end separately.

- The order in which you create plans for aggregate units and individual members of aggregate units affects how the plans are implemented. If you create a plan for an aggregate member and then create a plan for the aggregate, the aggregate plan overrides the individual plan. If you create a plan for an aggregate and then create a plan for a member of the aggregate, the member plan is executed.
- Aggregates that have members simulated on multiple back-ends do not always act correctly. Remote subordinates do not break formation when they are independently tasked and do not report tasks for assumption if they get destroyed.
- If you run a MÄK Logger recording of a VR-Forces simulation and the site:application number for the back-end is the same as the site:application number of the back-end used to create the recording, VR-Forces may crash. To avoid this problem make sure that you run VR-Forces with different site:application numbers, or just run the front-end and use it as a plan view display to view the recording.
- The component descriptions in Appendix C of *MÄK VR-Forces Developer's Guide* have not been updated to reflect changes made in recent releases of VR-Forces. In particular references to port interfaces are obsolete and the component descriptions do not reflect the use of process state repositories. Fixed-wing and rotary-wing components may be significantly incorrect.
- Using large filled polygons, such as minefields, in overlays can degrade performance.
- You cannot load vector data with geocentric databases. Vector data works only with geodetic or UTM databases.
- You cannot give a helicopter a Set Altitude request that places it at less than the starting height value.
- Nested aggregates do not always follow the leader (the superior aggregate).
- Plans stop executing if they encounter a task that they cannot execute. For example, if the plan for a rotary-wing entity include Turn To Heading, which is not a valid task for rotary-wing entities, the plan stops executing.
- If you delete an entity that has a plan and create a new entity with the same name as the deleted entity, the old plan is displayed in the new entity's Edit Plan window. However, the new entity cannot execute that plan.
- Missiles get initialized at their maximum velocity. They do not accelerate to maximum velocity.
- If you assign a plan to an aggregate, remove the leader (via drag/drop in echelon view), retask the leader and run the simulation, the subordinates follow the old leader on its new task.
- Torpedoes ignore a polygon with soiltype water, but the intervisibility fan does not. So it does not show an accurate representation of who can see who.
- When a subsurface entity gets destroyed it jumps to altitude 0.
- It you task a rotary-wing entity to fly to a waypoint on the ground, it will land when it gets to the waypoint. If opposing force trucks get within range of the rotary wing it will fire. Since the rotary-wing entity is at a slight incline, it will fire directly into the ground and explode.

- Under some circumstances, the second entity in an aggregate does not receive the follow task for aggregate movement. The entity does not follow the leader. Other entities follow aggregate number 2.
- If you uninstall a VR-Forces 3.5.1 or earlier using the InstallShield uninstaller, the entity icon fonts get uninstalled. If you have another application that needs them, such asMÄK Plan View Display, you must manually install them. This is not a problem for VR-Forces 3.6 or later.
- Intervisibility works with linear and areal features, but not with point features.
- Occasionally, VR-Forces is unable to load a scenario. When this happens, delete the contents of the *./data/temp* directory and try loading again.
- Setting the heading of a rotary wing entity whose altitude is less than the **starting-height** parameter causes it to rise up to the **starting-height** value. The workaround is to set a very small (non-zero) **starting-height**.
- If a task is interrupted by a trigger that only has set statements, after the set statements are executed, the plan moves to the next task (if there is one) rather than continuing with the task that was interrupted, as it is supposed to. If there is no additional task in the plan, the entity resumes executing the interrupted task.
- Obstruction avoidance does not work with linear or areal features if their terrain location is greater than zero.