MÄK VR-Forces 3.5-ngc Release Notes

This release note contains the following release-specific information for VR-Forces Release 3.5-ngc:

Document ID: VRF-3.5-3-031125

Systems Supported

VR-Forces 3.5-ngc 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 8.0	GNU C++ (GCC) 3.2
Windows NT 4.0 SP6, Windows 2000 SP2, XP	Microsoft Visual C++ 6.0, Service Pack 5 Microsoft .NET

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.

Disk Space Requirements

A full installation of VR-Forces requires approximately 420 MB of disk space. Terrain databases use approximately 73 MB and documentation (primarily the class reference) uses 145 MB.

MÄK Product Compatibility

To build VR-Forces applications, you must link with VR-Link 3.8.1-ngc. If you are building for HLA and want to link with the MÄK RTI, use MÄK RTI 2.x-ngc.

MÄK VR-Forces 3.5-ngc is a parallel release to MÄK Plan View Display 2.5-ngc. It is not compatible with previous releases of the Plan View Display.

Qt Release Compatibility

If you want to do development using the VR-Forces GUI API, you need to use Qt, a cross-platform GUI toolkit. The VR-Forces GUI was built with Qt release 3.1.2. 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.

Online Help

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

Network Compatibility

HLA only

VR-Forces 3.5-ngc was built against VR-Link 3.8.1-ngc and is compliant with:

- RPR-FOM 1.0 and a subset of 2.0 (draft 6)
- MÄK RTI 2.0.1-ngc
- DMSO RTI NG 4, and 6.

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

VR-Forces 3.5-ngc is compatible with applications that use earlier versions of VR-Link if they support versions of the RPR FOM listed above.

DIS only

VR-Forces 3.5-ngc supports DIS 2.0.4, IEEE 1278.1, 2.1.4, and IEEE 1278.1a, and can therefore interoperate with DIS applications of any of these versions.

Backward Compatibility

VR-Forces 3.5-ngc code is not fully compatible with previous releases. See API changes described in "New Features and Updates,".

RPR FOM Versions Supported

VR-Forces 3.5-ngc has built-in support for versions 1.0 and 2.0 (draft 6) of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, VR-Forces 3.5-ngc uses RPR FOM 1.0.

If you are building an application with the VR-Forces toolkit and you want to specify a version of the RPR FOM through code, pass the version number (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 we rely on the following custom MÄK extensions

- LgrControl
- ViewControl.

New Features and Updates

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

- Support for large databases (greater than 1 UTM zone).
- Updated Terrain API.
- More efficient GDB database format.
- Additional predicate classes.
- Ability to import shape file specification attributes from dBASE files.
- Updated GUI API supports overlays (tactical overlays) and the ability to save data to files. See "GUI API Migration Instructions," on page 1-19.
- New features in the VR-Forces front-end:
  - Ability to add and edit points on a route, area, or obstacle using the mouse
  - Ability to change the simulation speed during runtime
  - Ability to create pre-configured aggregate entities
  - Ability to save and restore the layout of the screen and display settings
  - Display of elapsed simulation time
  - Ability to change the colors used to display control objects and entity labels
  - Support for overlay graphics
  - Ability to specify the type of coordinate system used in dialog boxes, such as Set Location and Fire for Effect on Location
  - Ability to specify which types of intervisibility intersections to test for
  - Improved support for controlling the MÄK Stealth
  - Ability to display an entity's bounding box
  - Networking Options dialog box for setting thresholds
- New plan conditions: Entity-Under Fire and Has-Target.
- New Set Destroyed set data request.
- New Move to Altitude task.
- Support for take-off and landing of fixed-wing entities.

Support for Large Terrain Databases

VR-Forces now supports simulation on large terrain databases. Previous versions of VR-Forces were capable of simulating only in UTM (Universal Transverse Mercator) coordinates. Terrain databases were required to be built in UTM coordinates, and the play area could not span more than one UTM zone. (The UTM system divides the earth into 60 vertical slices or UTM zones. Each zone is 6-degrees of longitude in width (about 600km at the equator). Actuators and controllers assumed that constant Z meant constant altitude, and that moving in the positive X and Y directions meant moving east and north respectively.

In order to support large play areas, VR-Forces can now simulate in geocentric coordinates. (In the geocentric coordinate system, the origin is at the center of the earth. The positive X-axis passes through the prime meridian at the equator; the positive Y-axis passes through 90 degrees east longitude at the equator; and the positive Z-axis passes though the North Pole.) Of course, VR-Forces can still simulate in UTM coordinates.

Loading Geocentric Terrain Databases in VR-Forces

VR-Forces can now load terrain databases that are built in geocentric or geodetic (Latitude, Longitude, Altitude) coordinates. When a geodetic database is loaded by the simulation engine, it is automatically converted to geocentric, since it does not make sense to simulate in geodetic coordinates. Conversely, when a geocentric database is loaded by the VR-Forces GUI, it is automatically converted to geodetic coordinates, since it does not make sense for a 2D map to display geocentric coordinates.

In order to avoid the conversion during database loading, you can explicitly load a geodetic or UTM database into the GUI, while instructing the simulation engine (back-end) to load a corresponding geocentric database. See "Creating A Scenario," in *MÄK VR-Forces User's Guide* for the updated procedure.

Geocentric Terrain Databases

There are multiple ways to obtain terrain databases in geocentric coordinates. You can convert OpenFlight terrain databases in geocentric coordinates to MÄK GDB format using *tdbtool*, or import them directly into VR-Forces. You can convert any existing MÄK GDB file from its current coordinate system (UTM, geodetic) to geocentric using *tdbtool* (see "Using the Terrain Converter Tool," in *MÄK VR-Forces Configuration Guide* for details).

Creating Large Area Terrain Databases from DTED Data

Using *tdbtool*, you can create a set of terrain databases from one or more DTED files for simulating over large geographic areas. You can create a geodetic version for viewing in the front-end, and a geocentric version for simulating in the back-end.

To create a large area database from DTED data:

1. Create a *.map* file specifying the DTED file(s) to import. If the DTED data spans multiple UTM zones, set the Projected flag to false (keep the data in geodetic coordinates). Use *tdbTool* to generate a geodetic GDB database.
2. Load the geodetic database into *tdbTool* to generate a geocentric GDB version of the terrain.

Changes to the Terrain API to Support Large Terrain Databases

This section describes the changes to the Terrain API made to support large terrain databases. Additional changes to the Terrain API are described in "Updated Terrain API," on page 1-12.

Coordinate Systems

VR-Forces maintains the concept of local (database) coordinates. The local coordinate system is the coordinate system used by the terrain database. You can ask for and set an entity position, velocity, acceleration, or orientation in local coordinates. Through version 3.4 of VR-Forces, local coordinates have been synonymous with UTM coordinates, because that was the only terrain database coordinate system supported by VR-Forces. Now local coordinates may be UTM or geocentric coordinates, depending on the coordinate system of the terrain database.

To access the current coordinate system in the back-end:

```
Coordinate_System* coordinateSystem = simManager->terrainDatabase()-
  >coordinateSystem();
```

Converting between Local Coordinates and Other Coordinate Systems

The *Coordinate_System* class (*coordSystem.h*) provides member functions for converting between local coordinates and geocentric, topocentric, and geodetic coordinate systems.

For example, to obtain the altitude of a given position in local coordinates,

```
// Local position may be in geocentric or UTM coordinates, depending on
// the terrain database loaded.
DtVector localPosition;
. . .
DtGeodeticCoord geodeticPostion;
coordinateSystem->localToGeodetic(localPosition, geodeticPostion);

DtInfo("Altitude at local position %s = %f\n",
   localPosition.string().string(), geodeticPosition.alt());
```

Converting local coordinates to topocentric coordinates can be useful for performing 2-D calculations in the topocentric X-Y plane. You can ask the coordinate system for a rotation matrix (*DtDcm*, defined in *LibMatrix.h*) for transforming from the local coordinate system to a topocentric coordinate system at a given location.

For example, to find the downward and projected X-Y components of a local velocity vector:

```
DtVector localPosition;
DtVector localVelocity;
. . .
DtVector topoVelocity;

DtDcm localToTopoDcm;
coordinateSystem->localToTopo(localPosition, localToTopoDcm);

DtVector topoVelocity;
DtDcmVecMul(localToTopoDcm, localVelocity, topoVelocity);

DtInfo("The magnitude of the downward component of velocity is %f\n",
  topoVelocity[2]);
DtInfo("The projected magnitude of velocity (in the x-y plane) is %f\n",
  sqrt(topoVelocity[1]* topoVelocity[1] + topoVelocity[2] *
  topoVelocity[2]));
```

The Gravity Direction Vector

The *Coordinate_System* class has a member function that returns a direction vector pointing in the direction of gravity, at a given local position.

```
DtVector localPosition;
. . .
DtVector downDirectionVec;

coordinateSystem->down(localPosition, downDirectionVec);
```

Other API Changes

Other changes to the API are as follows:

Entity Models

The *DtAutomotiveActuatorComponent* (*autoAct.h*) has been refactored, to make it easier to override specific functionality in tick().

The *DtFixedWingActuator* (*fixWingAct.h*) class has been modified to support its ability to taxi on the ground, take-off, fly, and land.

Interface Messages

VR-Forces no longer specifies position information in database coordinates in interface messages. Positions are always specified in geocentric coordinates, to make it possible to use a different terrain database in the front-end than in the back-end. In particular, it makes it possible to load a geodetic version of a terrain database for display in the front-end, while loading a geocentric version of the terrain in the back-end for simulation.

Remote Control Interface

The remote control interface class, *DtVrfRemoteController* (*vrfController.h*), now requires that geocentric coordinates be passed in to any of the entity or control object creation functions (for example createEntity() or createRoute()). The underlying interface message class, *DtIfCreateVrfObject* (*ifCrtVrfObj.h*) requires vertices to be specified in geocentric coordinates. The newScenario() member function can now take two databases, one for the front-end and one for the back-end.

Removed Classes

DtObstacleManager class has been removed. There are no longer any references to *DtObstacleManager* in the *DtCgf* (*cgf.h*), *DtVrfCreator* (*vrfCreator.h*), and *DtSimManager* (*simMgr.h*) classes.

Deprecated API

In state repository classes (classes derived from *DtVrfBaseObjectStateRepository*), the following functions have been deprecated:

- setLocalOrientation()
- localOrientation()

These functions set and retrieve topocentric Euler angles for orientation. They will work only with UTM databases and have been retained for backward compatibility. They have been replaced by the following coordinate system-independent member functions:

- setLocalOrienationWrtDatabase()
- localOrienationWrtDatabase()

These functions set and retrieve Euler angles with respect to the current local (database) coordinate system. Similar functions for setting and retrieving local orientation in *DtVrfKinematicState* (*vrfKinState.h*) have been superseded by coordinate system-independent versions.

The *DtMissileStateRepository* class (*missileSR.h*) setStartLocalOrientation() member function has been deprecated. It sets the initial orientation of the missile in topocentric Euler angles. It will only work with UTM databases. It has been replaced by the member function setStartLocalOrientationWrtDatabase(), which specifies the initial orientation of the missile in Euler angles with respect to the local (database) coordinate system.

The *DtVrfObjectGeometry* class (*vrfObjGeom.h*) contains deprecated member functions for doing 2-D calculations, such as determining if a given point is left of a given list of vertices, or if a point lies inside a closed list of vertices. These functions assume all given local positions and local vertices are in UTM coordinates. Coordinate system-independent versions of these functions have been added to *kinTools.h*. They have the same names as those in the *DtVrfObjectGeometry* class, but will work with whatever the current local (database) coordinate system is.

Migrating Your Code to Support Large Databases

With the exception of the *DtAutomotiveActuator* (*autoAct.h*) and *DtFixedWingActuator* (*fixWingAct.h*) classes, if you have extended sensor, controller or actuator classes, they should continue to compile and function correctly on UTM terrain databases. If the only databases you intend to work with are UTM databases, no additional work should be required.

If you want your extended models to work with geocentric databases, they may require additional work to make them work with geocentric terrain databases (depending on the nature of the extensions). You will need to make changes to your models if they contain assumptions about UTM coordinates for an entity's state (position, orientation, velocity, acceleration.)

Getting the Downward Direction Vector

It is no longer appropriate to assume that the negative Z component of a given local position lies along the direction of gravity. This is only true when local coordinates are UTM coordinates. You can get the downward direction vector at a given local position in a coordinate system-independent manner through the *Coordinate_System* class. For example,

```
DtVector localPosition;
. . .
DtVector downDirectionVec;

coordinateSystem->down(localPosition, downDirectionVec);
```

Updating Local Orientation

Replace the following calls to set an entity's local orientation in its state repository (*DtVrfBaseObjectStateRepository*):

- localOrientation()
- setLocalOrientation()

with calls to the following functions (also in its state repository):

- localOrienationWrtDatabase()
- setLocalOrienationWrtDatabase()

The old localOrientation() and setLocalOrientation() member functions worked with topocentric Euler angles (heading, pitch, and roll). They will only work with UTM coordinates. The new setLocalOrientationWrtDatabase() and localOrientaitonWrtDatabase() set and retrieve local Euler angles (Euler angles representing orientation with respect the local (database) coordinate system.)

Terrain Intersections

The *DtTerrainDatabase* class (*terrainDb.h*) provides an interface for terrain intersections with arbitrary chords. In the VR-Forces back-end, we often need to perform terrain intersections pointing downward in the direction of gravity. The intersectTerrain() function declared in *kinTools.h* is a convenience function for performing this type of intersection.

Updated Terrain API

The Terrain API has been refactored and extended to improve ease of use as well as memory efficiency. New functionality has been added to facilitate the programmatic creation of custom terrains without limiting existing, more advanced controls over the internals of the terrain database. Several of the internal data structures have been reorganized or replaced in order to vastly improve memory usage and efficiency. The primary benefit to users of the API will be:

- Smaller number of steps necessary to create terrain
- Fewer objects to manage – the terrain database will now manage them
- Memory management integrated with the new creation functions presented by the terrain database making efficient memory usage easier
- Smaller data structures enabling the use of larger and more detailed terrain databases
- Cleaner, more consistent organization of the terrain database and its member objects.

Updated *DtGdbNode* Types

The types of *DtGdbNodes* have been updated as follows:

- The *DtPolygonImmediate*, *DtTriangleImmediate*, *DtSurfaceList*, *DtSpecificationList* and *DtVectorNetwork* types have been deprecated.

Updated *DtTerrainDatabase* Class

The *DtTerrainDatabase* class has been refactored to provide the following:

- Ability to transform the coordinate system of the entire terrain database. By passing in a new coordinate system, the terrain database will transform all geometry as appropriate.

For some conversions, such as converting to geocentric, the vector data may be lost as it cannot be translated at this time.

- To improve memory usage and reduce file sizes in memory, all immediate polygons and triangles have been deprecated. Indirect triangles and polygons are the geometric primitive supported.
- List classes (nodes) have been replaced with manager classes.
- Geometric feature data and vector feature data have been clearly split in the terrain database. Geometric feature data is stored in the features group node of the terrain database. Vector data is stored in the *DtVectorNetwork*, which is now a member of the terrain database and not of the features node.
- The *DtSpecificationManager*, which replaces the *DtSpecificationList* node, is now a member of the *DtVectorNetwork* and not of the features node.

- New functions for creating polygons and triangles.

See "Migrating to the Updated Terrain API," on page 1-25 for detailed migration instructions.

More Efficient GDB Format

The MÄK GDB database format has been updated to reduce the memory requirements of databases. When you use databases saved with the previous format, you will be prompted to save them to the updated format.

Opening GDB databases that were saved in the old format takes longer than opening updated files. Therefore, recommend that you open any databases that you have saved to GDB format and save them as GDB again.

Additional Predicate Classes

VR-Forces supports the following new predicate classes.

- Combining predicates (*DtCompositePredicate*, in *compositePred.h*). Objects pass evaluation by a composite predicate only if they also pass evaluation by all sub-predicates.
- Checking whether an object's echelon ID has changed since the last time it was evaluated (*DtEchelonIdChangedPredicate*, in *echelonIdChangedPred.h*). This predicate compares an object's echelon ID to the echelon ID in the predicate.
- Tests an object's *DtForceType* for inclusion in a set of *DtForceTypes* (*DtForceTypePredicate*, in *forceTypePred.h*).

New Front-End Features

The new front-end features are all documented in *MÄK VR-Forces User's Guide* and *MÄK VR-Forces Configuration Guide*.

- Ability to add and edit points on a route, area, or obstacle using the mouse. See "Editing An Object's Vertices," in *MÄK VR-Forces User's Guide*.
- Ability to change the simulation speed during runtime. See "Changing the Simulation Speed," in *MÄK VR-Forces User's Guide*.
- Ability to create pre-configured aggregate entities. Instead of creating individual entities and aggregating them, you can now create an aggregate and its entities in one operation. See "Creating A Pre-Configured Aggregate," in *MÄK VR-Forces User's Guide*.
- Ability to save and restore the layout of the screen and display settings. VR-Forces automatically saves your application settings. See "Saving VR-Forces Settings," in *MÄK VR-Forces User's Guide*.

- Display of elapsed simulation time. See "Simulation Time Toolbar," in *MÄK VR-Forces User's Guide*.
- Ability to change the colors used to display control objects and entity labels. You can change the colors of individual objects and of all objects of a given type. See "Changing the Color of An Object," in *MÄK VR-Forces User's Guide*.
- Support for overlay graphics. VR-Forces supports creation of multiple overlays on which you can draw overlay objects, such as points, lines, areas, symbols, and text. See Chapter 7, *Control Objects and Overlay Objects*, in *MÄK VR-Forces User's Guide*.
- Ability to specify the type of coordinate system used in dialog boxes, such as Set Location and Fire for Effect on Location. See "Specifying the Coordinate System for Dialog Boxes," in *MÄK VR-Forces User's Guide*.
- Ability to specify which types of intervisibility intersections to test for. You can test for polygons, features, and entities. In previous releases, VR-Forces just checked for terrain intersections. "Specifying the Types of Objects to Check for Intersections," in *MÄK VR-Forces User's Guide*.
- Improved support for controlling the MÄK Stealth. You can now specify one of three view modes when you attach the Stealth to an entity. You can also drag the Stealth icon to move the eyepoint. See "Interoperating with the MÄK Stealth," in *MÄK VR-Forces User's Guide*.
- Ability to display an entity's bounding box. You can display the entity bounding box used to calculate intervisibility. See "Displaying An Entity's Bounding Box," in *MÄK VR-Forces User's Guide*.
- Networking Options dialog box for setting thresholds. See "Setting Thresholds for Entity Updates," in *MÄK VR-Forces User's Guide*.

New Conditions

The Has-Target condition is true when an entity identifies a target. The Entity-Under-Fire condition is true when an entity detects that it is the target of hostile fire.

New Tasks and Set Data Requests

The Set Destroyed set data request lets you set an entity's state to destroyed. The Move to Altitude task directs an entity that can change altitude to fly or sink to the desired altitude in real time, rather than moving immediately to the altitude as they do under the Set Altitude set data request.

Fixed-Wing Takeoff and Landing

Fixed-wing entities can now take off and land while following a route, and they can take off towards a point. See "Fixed-Wing Entities," in *MÄK VR-Forces User's Guide* for complete details. You may need to update scenarios that use fixed-wing entities to accommodate the changes that accompany takeoff and landing.

If you are using a customized object parameters database, you must add the following parameters, with appropriate values, to your fixed-wing entities to support takeoff and landing:

```
(ordered-ground-speed 10)
(max-ground-speed 200)
(max-ground-stopping-deceleration 20)
(ground-turning-radius 5.000000)
(max-ground-slope 1)
```

The FixedWingMoveToController entries must add the following parameters, with appropriate values for your models:

```
(max-turnrate-cutoff-angle 0.5236) ;; 30. degrees
(taxiing-at-distance 2.0)
(taxiing-near-distance 35.0)
(taxiing-approach-speed 2.0)
```

The set-data controller entry must be changed to:

```
(set-data
  (component-descriptor-type "component-descriptor")
  (component-type  "fixed-wing-entity-set-controller")
)
```

Importing Shapefile Attributes

A ShapeVectorMetafileRecord can specify attributes. Attribute entries allow a dBASE field to be imported as a specification attribute. The type of attribute added is specified as 'integer', 'real' or 'string'. If the attribute type does not match the dBASE type, a conversion will be attempted, if possible. For instance, a dBASE field of type 'F' (floating point) can be converted to an integer attribute by specifying 'integer'. Not all dBASE types have a corresponding specification type in VR-Forces, but they can be converted. For instance you can read in a dBASE date field as a string specification. For details, see "ShapeVectorMetafileRecords" in Chapter 2, "Working with Terrain Databases," in *MÄK VR-Forces Configuration Guide*.

Documentation Updates

MÄK VR-Forces User's Guide has been divided into two manuals, *MÄK VR-Forces User's Guide* and *MÄK VR-Forces Configuration Guide*. All manuals have been updated to include all new features. New printed versions of the manuals are being issued with this release. *MÄK VR-Forces User's Guide* now includes a Quick Reference Card for startup options, keyboard shortcuts, and navigation options. If there are discrepancies between the printed versions and the PDF versions, the printed version has the corrected information.

Bug Fixes

This release fixes the following problems:

- Subsurface entities now use correct icons.
- If you are using a DTED database and "Projected False" is specified in the PvDtedMetafileRecord, entities correctly execute movement tasks.
- The *world.gdb* file database now displays entity movement correctly.
- VR-Forces displays correct TrueType symbols for air aggregates.
- Published aggregate velocity and orientation is now computed as the average over all subordinates (to comply with the DIS/RPR standards). Previous versions of VR-Forces computed average velocity and orientation over healthy subordinates only.
- Set target works correctly now.

Known Problems

VR-Forces Release 3.5-ngc has the following known problems:

- On IRIX, icons may appear discolored or grainy unless the color depth is set to 24-bit.
- True Type fonts are not supported on SGI IRIX. Use bitmaps for entity icons.
- 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 we use 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.
- Uninstalling VR-Forces in Windows via InstallShield uninstalls the entity fonts. If you have another application that uses these fonts, such as the MÄK Plan View Display, the correct entity icons would not be displayed. You must manually install the fonts from their location in the other application that needs them.
- 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.
- The More section of the Entity Information dialog box is not editable, even when some states can otherwise be set through the GUI.
- 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.
- If an entity is given a Wait task while it is avoiding a collision, the Wait task does not immediately stop the entity. It continues with its collision avoidance behavior.
- 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.

GUI API Migration Instructions

The GUI API has been restructured to accommodate the new tactical overlay features. Parts of the toolkit have also been changed in order to make it easier to add new types of coordinate systems or unit conversions. Examples that rely on this API have been changed to reflect the new API. The sections that follow describe these changes as well as examples you can look at for using the new API.

Removed Classes

The following classes have been removed:

- *DtObstacleConfigWidget*
- *DtWaypointConfigWidget*
- *DtPhaseLineConfigWidget.*

New Classes

The following classes have been added:

- *DtArchiver* and archivable object classes.
- *DtPointConfigWidget* encapsulates *DtWaypointConfigWidget* to create any kind of object that would be a single point.
- *DtTerrainCoordinateSystemCollection* is a collection of coordinate systems. This class is used by all dialog boxes that want to set or display location information. This class is now the central point for adding new types of coordinate system displays. You no longer have to subclass every dialog box that needs to know about a coordinate system. The windowsCoordinateSystem example has been updated to show how to add and use a new coordinate system in this context.
- *DtTMUnitConverterCollection* is a collection of unit converter classes that are used by all dialog boxes and displays that care to translate meters to some other unit. This class is now the central point for adding new types of unit conversions. You no longer have to subclass every dialog box that needs to know about a unit conversion.
- *DtOverlaySymbolMapper* implements the ability to create overlay objects. *DtMtlSymbolMapper* now subclasses from this object rather than from *DtBaseSymbolMapper*.
- *DtVrfGuiSymbolMapper* is the new parent for any class that used to subclass from *DtMtlSymbolMapper*.
- *DtOverlayPublisherHandler* handles the creation and editing of overlay objects.
- *DtEnvironmentalModelData* handles the data used by overlay objects. *DtControlObjectModelData* classes now subclass from this class. Any symbol updater routine that used to reference *DtControlObjectModelData* now must reference *DtEnvironmentalModelData* instead.

- *DtTerrainLocationEntryWidget* can be used in dialog boxes that need to enter in positional information. When initialized, this widget can change itself to conform to changes in state of the coordinate system and convert between the current and new system automatically. If you have written code to take this into account, you may be able to replace it with this class. The imageSplitter example shipped with the application shows how to use this widget to set positional information to move items on the map area.
- *DtVrfGuiOverlayPublisherHandler* publishes overlay objects on the network.
- *DtVrfGuiSymbolUpdater* is the new parent for any class that used to subclass from *DtPvdSymbolUpdater*.

Changes to Existing Classes

The following classes have changed as indicated:

- *DtAreaConfigWidget* now subclasses from *DtLineConfigWidget*, rather than containing its own UI file.
- *DtLineConfigWidget* – The initialization function and the create and edit signals have changed.
- *DtMtlSymbolMapper* no longer takes a *DtMtlEnvironment* as a parameter. You must remove this parameter from any subclass of *DtMtlSymbolMapper* that has the parameter in the constructor and creation function. Also, the creation member function must now return a type of *DtBaseSymbolMapper** rather than a *DtMtlSymbolMapper**.

Migrating Code for Subclasses of *DtMtlSymbolMapper*

You must make the following changes to subclasses of *DtMtlSymbolMapper* in order to conform to the new class structure:

- The constructor has changed. You no longer need to have the *DtMtlEnvironment** as part of the class, so if your class constructor looked like:

  ```
  MyMtlSymbolMapper(DtMtlEnvironment* env = NULL);
  ```

  it now will look like:

  ```
  MyMtlSymbolMapper();
  ```

- The static creator must now return a *DtBaseSymbolMapper** type rather than a *DtMtlSymbolMapper** type:

  ```
  static DtMtlSymbolMapper* create(DtMtlEnvironment* env);
  ```

  changes to:

  ```
  static DtBaseSymbolMapper* create();
  ```

Migrating Code from *DtLineConfig* and other Config Widgets

The classes that used to create control objects (waypoints, phase lines, and so on) have been changed so that they can create tactical overlay objects. The classes have been organized into line, area and point classes. Any dialog boxes that subclassed from *DtWaypointConfigWidget* should now subclass from *DtPointConfigWidget*. Any class that subclassed from *DtLineConfigWidget* can continue to subclass from *DtLineConfig-Widget* (with class updates) and all other classes should subclass from *DtAreaConfig-Widget*.

If you used to create your widget by subclassing one of the handlers (line, area, point), you can now simply add the dialog box to be created via the addDialogCreator() member function defined in *DtCreateControlObjectHandler*. Whenever an object of the specified type is tried to be created, your dialog box will be instantiated.

You must also update the init() member functions and the createActivated() and editActivated() member function to supply the new parameters. For more information about these member functions and classes, please refer to the GUI API documentation in *MÄK VR-Forces Developer's Guide* that describes the new tactical overlay system.

The Minefield example shows how to subclass an existing dialog box and add your new information to it.

Migrating Code that Added Coordinate Systems or Unit Converters to Dialog Boxes

The coordinate system and unit conversion code has been centralized into two classes – *DtTerrainCoordinateSystemCollection* and *DtTMUnitConverterCollection*. These classes hold objects that know how to translate between coordinate systems or units (meters, feet, miles, and so on.)

If you have overridden classes that dealt with coordinate systems or unit converters, you must create a new coordinate system patterned after a *DtTerrainCoordinateSystem* or a *DtTMUnitConverter*. The old member functions and variables have been deprecated.

The windowsInfoExtension example shows how to implement a coordinate system so that it can be used in any dialog box that deals with a coordinate system. The widget classes that used to be subclassed to add this capability (the *DtInfoWidget* and *DtBaseEntityInfoWidget*) have been removed.

Display of UTM coordinate systems has changed. What used to be the UTM coordinate system for UTM databases (which would show an offset in meters from the origin of the database) now shows the true UTM coordinate of the database as Zone, Easting and Northing values. What used to be displayed under UTM is now displayed under the Database coordinate system, which shows the native coordinates of the database. This means that a Geodetic database shows radians and a UTM database shows meters as an offset from the origin of the database.

Updates to *vrfSim.opd* for Overlay Support

If you are using a customized version of *vrfSim.opd*, you must add the following parameter blocks to support overlay entries:

```
(circle-parameters
        (parameter-type "vrf-base-object-param")
        (object-type  1 (19 -1 -1 -1 -1 -1 -1))
        (category "Circle")
        (bounding-volume
           (local-bvol  0.000000 0.000000 0.000000)
           (offset  0.000000 0.000000 0.00000)
        )
;;         (min-tick-period 1.000) ;; 1 Hz
;;         (min-tick-period-variance 0.500)
        (local-objects
           (state-repository "vrf-overlay-object-state-repository")
           (net-interface "local-environmental-net-interface")
;;           (net-interface-min-tick-period 0.500)
;;           (net-interface-min-tick-period-variance 0.100)
        )
        (remote-objects
           (state-repository "vrf-overlay-object-state-repository")
           (net-interface "vrf-remote-environmental-net-interface")
;;           (net-interface-min-tick-period 0.500)
;;           (net-interface-min-tick-period-variance 0.100)
        )
     )
     (text-parameters
        (parameter-type "vrf-base-object-param")
        (object-type  1 (20 -1 -1 -1 -1 -1 -1))
        (category "Text")
        (bounding-volume
           (local-bvol  0.000000 0.000000 0.000000)
           (offset  0.000000 0.000000 0.00000)
        )
;;         (min-tick-period 1.000) ;; 1 Hz
;;         (min-tick-period-variance 0.500)
        (local-objects
           (state-repository "vrf-overlay-object-state-repository")
           (net-interface "local-environmental-net-interface")
;;           (net-interface-min-tick-period 0.500)
;;           (net-interface-min-tick-period-variance 0.100)
        )
        (remote-objects
```

```
            (state-repository "vrf-overlay-object-state-repository")
            (net-interface "vrf-remote-environmental-net-interface")
;;            (net-interface-min-tick-period 0.500)
;;            (net-interface-min-tick-period-variance 0.100)
         )
      )

      (symbol-point-parameters
         (parameter-type "vrf-base-object-param")
         (object-type  1 (101 -1 -1 -1 -1 -1 -1))
     (category "Symbol")
         (bounding-volume
            (local-bvol  0.000000 0.000000 0.000000)
            (offset  0.000000 0.000000 0.00000)
         )
;;         (min-tick-period 1.000) ;; 1 Hz
;;         (min-tick-period-variance 0.500)
         (local-objects
            (state-repository "vrf-overlay-object-state-repository")
            (net-interface "local-environmental-net-interface")
;;            (net-interface-min-tick-period 0.500)
;;            (net-interface-min-tick-period-variance 0.100)
          )
         (remote-objects
            (state-repository "vrf-overlay-object-state-repository")
            (net-interface "vrf-remote-environmental-net-interface")
;;            (net-interface-min-tick-period 0.500)
;;            (net-interface-min-tick-period-variance 0.100)
          )
      )
```

You must also change the state repository type of control objects to be vrf-overlay-object-state-repository. If you do not make this change a warning will be printed to the simulation window and the control object will not be created. The following parameter blocks are examples of an old control object entry and an updated entry:

Old Control Object Entry:

```
      (generic-area-parameters
         (parameter-type "vrf-base-object-param")
         (object-type  1 (18 -1 -1 -1 -1 -1 -1))
         (category "Area")
         (bounding-geometry-type Area)
         (bounding-volume
            (local-bvol  0.000000 0.000000 0.000000)
            (offset  0.000000 0.000000 0.00000)
         )
;;         (min-tick-period 1.000) ;; 1 Hz
;;         (min-tick-period-variance 0.500)
       (local-objects
            (state-repository "vrf-object-base-state-repository")
            (net-interface "local-environmental-net-interface")
;;            (net-interface-min-tick-period 0.500)
;;            (net-interface-min-tick-period-variance 0.100)
          )
         (remote-objects
            (state-repository "vrf-object-base-state-repository")
            (net-interface "vrf-remote-environmental-net-interface")
;;            (net-interface-min-tick-period 0.500)
```

```
;;            (net-interface-min-tick-period-variance 0.100)
           )
        )
```

New Control Object Entry:

```
        (generic-area-parameters
           (parameter-type "vrf-base-object-param")
           (object-type  1 (18 -1 -1 -1 -1 -1 -1))
           (category "Area")
           (bounding-geometry-type Area)
           (bounding-volume
              (local-bvol  0.000000 0.000000 0.000000)
              (offset  0.000000 0.000000 0.00000)
           )
;;           (min-tick-period 1.000) ;; 1 Hz
;;           (min-tick-period-variance 0.500)
          (local-objects
              (state-repository "vrf-overlay-object-state-repository")
              (net-interface "local-environmental-net-interface")
;;              (net-interface-min-tick-period 0.500)
;;              (net-interface-min-tick-period-variance 0.100)
           )
           (remote-objects
              (state-repository "vrf-overlay-object-state-repository")
              (net-interface "vrf-remote-environmental-net-interface")
;;              (net-interface-min-tick-period 0.500)
;;              (net-interface-min-tick-period-variance 0.100)
           )
        )
```

You must make these changes for the following object parameters database entries:

- generic-point-parameters
- control-point-parameters
- target-point-parameters
- generic-line-parameters
- phase-line-parameters
- route-parameters
- generic-area-parameters
- control-area-parameters
- obstacle-parameters.

Migrating to the Updated Terrain API

This section describes the changes you need to make to your code to migrate it to the updated Terrain API. The major changes are:

- You do not need to create vertex or surface lists
- The specification manager no longer resides outside the vector network
- Functions for creating common types of nodes are provided and strongly recommended.

All code for doing any of the above should be modified to use the new interfaces.

New Manager Classes

Three new classes, *DtSpecificationManager*, *DtSurfaceManager*, and *DtVertexManager*, replace the *DtSpecificationList*, *DtSurfaceList*, and *DtGdbVertexList* classes. *DtSpecificationList* and *DtSurfaceList* have been deprecated entirely and are no longer supported at all. *DtGdbVertexList* remains supported for other uses, but is no longer used to store the vertices shared by triangles in the terrain database.

The new manager classes are automatically created by their owners, the *DtTerrainDatabase* and *DtVectorNetwork* respectively, during construction. As such, you no longer need to create instances of the list nodes and add them to the terrain database. Additionally, the manager classes automatically manage their own storage needs. You do not have to manage, and possibly reallocate, the memory for these data structures.

Unlike the list classes, the manager classes are not types of *DtGdbNode*. The *DtSurfaceManager* and *DtVertexManager* are members of the *DtTerrainDatabase* and not located in the terrain group node. The *DtSpecificationManager* is a member of the *DtVectorNetwork*, itself now a member of the *DtTerrainDatabase*, and is not a member of the features node.

Additionally, only one instance of the *DtVertexManager*, *DtSurfaceManager*, and *DtSpecificationManager* should be used, as opposed to the multiple list nodes. This is to increase memory efficiency and the ease of use of the API. Triangle and polygons created by the *DtTerrainDatabase* are automatically associated with the member manager classes.

Migrating from List Nodes to the New Manager Classes

The largest difference between the *DtVertex*, *DtSurface*, and *DtSpecificationList* classes and the new manager classes is that the manager classes extend themselves when they require addition storage space. As a result, you no longer need code for allocating a new array, copying over the elements, and resetting the array owned by the list class.

The manager classes are created upon construction of a terrain database. Therefore, there is no need to create an instance and add it to the terrain database. They are already constructed and ready to be used. You can still set them in the terrain database (inside a *DtVectorNetwork* in the case of the specification manager), but this is not recommended as it may cause problems with any IDs that have already been assigned.

Additionally, since only one instance should be used, if you have extended the file readers you must ensure that they are referencing the main terrain database's manager classes and not creating new managers where new list nodes were previously created.

Creating and Adding *DtPolygonIndirects* and *DtTriangleIndirects*

You no longer need to (nor should you) create a new POLYGON_INDIRECT or TRIANGLE_INDIRECT node, add the surface lists pointer and vertex list pointer, and then add the vertices and surfaces.

The interface provided by *DtTerrainDatabase* expects the following input to create polygons and triangles:

- *DtTriangleIndirect*:
  - 3 vertices
  - 1 surface

The vertices and surface will be managed by the terrain database's manager classes and the actual created triangle will only have IDs associated with the managed items.

- *DtTriangleIndirect* (with IDs):
  - 3 vertex IDs that reference vertices managed by the terrain database's vertex manager.
  - 1 surface ID that references surfaces managed by the terrain database's surface manager.
- *DtPolygonIndirect*:
  - A *DtGdbVertexList* of the vertices of the new polygon
  - A surface for the new polygon.

The vertices and surface will be managed by the terrain database's manager classes and the actual created triangle will only have IDs associated with the managed items.

The constructor allocates its own list of vertices and copies the specified vertices. Thus the new node does not take ownership of the specified *DtGdbVertexList*, and, as a result, the caller is responsible for deleting any memory allocated by him.

- *DtPolygonIndirect* (with IDs):
  - A pointer to a list of vertex IDs.
  - The number of vertices that the polygon will have.
  - A surface ID

The constructor allocates its own list of vertex IDs and copies the specified vertex IDs. Thus the new polygon node does not take ownership of the specified memory, and, as a result, the caller is responsible for cleaning up any memory allocated by him.

All of the functions mentioned also take two optional parameters:

- A parent node pointer if the terrain database's root terrain node is not to be the parent of the new triangle node.
- A Boolean flag which defaults to false specifying whether the node should be added to the beginning or end of the list of children of the parent node.

New Code Examples

For examples of how to restructure code to create triangles and polygons using the API, see the example projects:

- createTerrainDb.dsw
- addTerrainFeatures.dsw
- modifyTerrainDb.dsw

Memory Usage

Several changes were made to the API to improve memory usage and efficiency.

- *DtTriangleIndirect* no longer store extent information. In order to drastically decrease the size of the *DtTriangleIndirect* data structure, the stored extent information was removed.
- The new functions for creating polygons and triangles automatically use the terrain database's memory manager.
- The manager classes implement internal memory management.
- New functionality for reducing vertex/surface reduction.
- The function polygonsToTriangles() has been renamed polygonNodesToTriangleNodes() to more accurately reflect what is has been performing. All three-sided polygons will be converted to triangle nodes. Polygons with greater-than three sides remain polygons. (This may change in an upcoming version, but probably through separate tessellation code.)

Duplicate Vertex/Surface Reduction

The reduction of duplicate vertices and surfaces is not new functionality, but its importance is. When creating triangles and polygons using the direct interface – where actual vertices and surfaces are specified instead of vertex and surface IDs, then the eliminateDuplicate functions should be called to ensure that memory is efficiently used. Because of the processing time necessary to remove duplicates, it is recommended that these functions are only called once all the geometry nodes have been created. You should also note that the tolerance taken as a parameter to the eliminateDuplicateVertices function should be carefully chosen. All vertices within tolerance distance of each other will be welded into a single vertex.

Optimizing Performance

In order to improve performance, several options are recommended. While not actually new additions to the terrain API, because of several bug fixes and the reorganization of the API, they are worth discussing in briefly.

Spatial Index: The spatial index divides the terrain into spatially organized "buckets" for improved intersection calculation time. The trade-off for the improved efficiency is increased memory usage, but for the default values of 100, 100, 1, the additional memory usage is negligible. Users are strongly encouraged to create a Spatial Index when importing data into GDB format. The default settings are conservative so as to not generate too much memory. However, as a result, for finely tuning performance, users are encouraged to experiment with different settings to find more appropriate values for their specific terrains.

Modification of the Spatial Index

When a spatial index is present, any modifications to the terrain must also be reflected in the spatial index. You have two options:

- Delete and recreate the spatial index. This is really only viable after very large numbers of modifications, but for small terrain databases, may be acceptable.
- Modification of "buckets" in the spatial index: Currently, the spatial index only supports manual searching and updating. Users must find the cells that contain the nodes they have changed in particular and update their contents appropriately, as well as any new cells that have been affected. For example, if a user deletes a polygon and adds several new triangles, all references to the polygon in the Spatial Index should be remove and the new triangles should be added, possibly as a single group.

Tree Balancing

Tree balancing attempts to "balance" the tree by reorganizing it into a tree with branching and leafing specified by the user. Ideally, this would result is more efficient organization for intersection querying and modification. As with spatial indexing, users will have to experiment to find a level of balancing appropriate for their terrains. This does not give nearly the improvement in performance as a Spatial Index, but can still result in further improvements.

Caveat: A known issue with the Tree Balancing prevents it from being "on" by default. For some terrains, the reorganization of the geometry causes draw errors in the front end, related to polygon ordering (similar to z buffer issues). This does not affect intersection information, and does not affect all terrains. It is primarily an issue with terrains that contain coplanar or nearly coplanar polygons (such as a sea level polygon "underneath" a ground polygon). If any drawing issues occur, discontinue the use of the balancing and focus on improved performance using the spatial index.

Deprecated Classes

The following classes are deprecated and should not be used. If not already removed from the Terrain API, they will be soon.

- *DtTriangleImmediate*
- *DtPolygonImmediate*
- *DtGrid*
- *DtGridPostList*
- *DtMinefield*
- *DtRoughTerrain*
- *DtBodyOfWater*
- *DtBridge.*