MÄK VR-Forces 3.4-ngc Release Notes

This release note contains the following release-specific information for VR-Forces Release 3.4-ngc:

Document ID: VRF-3.4-3-030702

Systems Supported

VR-Forces 3.4-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

Platform	Compiler
SGI IRIX6.5	MipsPro7.3 C++ compiler (available from SGI)
Red Hat Linux 8.0	GNU C++ (GCC) 3.2
PC with Windows NT 4.0 SP6, Windows 2000 SP2, XP	Microsoft Visual C++ 6.0, Service Pack 5

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 300 MB of disk space. Terrain databases use approximately 73 MB and documentation (primarily the class reference) uses 110 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 2.0.1-ngc.

MÄK VR-Forces 3.4-ngc is a parallel release to MÄK Plan View Display 2.4-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 3.4-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 and 14)
- 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 3.4-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 3.4-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 3.4-ngc code is not fully compatible with previous releases. Changes include the following:

- Scenario format - Scenarios built with previous 3.x-ngc releases can be read, but should be converted to the new format. See "New Formats for Scenario Files", on page 7.
- Object parameters database - If you have customized the object parameters database you must merge changes into the new format. See "Updating Object Parameters Database Entries", on page 40.

- Simulation API:
  - The API for connecting ports and port interfaces has changed. See "New Architecture for Connecting Components", on page 37.
  - The network representation of *DtSimTasks* has changed. See "Changes to Network Representation of DtSimTasks", on page 9.
- GUI API - The GUI has undergone substantial changes to the menu and event systems. See "VR-Forces GUI API Migration Instructions", on page 23.
- The Terrain API has been rewritten and is now documented. See the PDF version of *MÄK VR-Forces Developer's Guide*.
- Network compatibility - VR-Forces 3.4 is not network compatible with previous 3.x-ngc releases.

RPR FOM Versions Supported

VR-Forces 3.4-ngc has built-in support for versions 1.0 and 2.0 (draft 6 and 14) of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, VR-Forces 3.4-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, 2.0014) to the *DtRprFomMapper* constructor and pass the resulting object to the *DtExerciseConn* 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));
```

New Features and Updates

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

- The GUI API has been revised, with the following major changes:
  - New event handler architecture.
  - Elimination of map area classes.
  - Redesigned menu system. Changed how menus are constructed and modified.
  - New examples.

  See "VR-Forces GUI API Migration Instructions", on page 23.
- The Terrain API has been significantly changed, as follows:
  - The Terrain API now enables developers to create or modify the terrain surface and the vector network.
  - The API has new terrain reader and writer classes and corresponding factories for importing and exporting terrain database files.
  - VR-Forces include Terrain API examples (demonstrating how to create and modify terrain databases). See "Terrain API Examples", on page 8.
  - The *DtSimTerrain* class (a wrapper class around DtTerrainDatabase) has been removed.

  These changes are documented in *MÄK VR-Forces Developer's Guide*.
- The format for some scenario files has changed.
- Intercomponent communication (ports and port interfaces) has been redesigned. See "New Architecture for Connecting Components", on page 37.
- Joystick support for tanks.
- New startup options.
- Changes and additions to metafile records for vector data.
- New MTL parameters for *vrfGui.mtl*.
- Miscellaneous API changes.

- The network representation of DtSimTasks has changed. If you implement derived DtSimTasks you may have to update your implementation of netRepSize(), setToNet(), and setFromNet() in their derived task classes. See "Changes to Network Representation of DtSimTasks", on page 9.
- New features in the VR-Forces front-end:
  - Support for hiding all control objects.
  - Support for displaying the map in full screen mode, without toolbars.
  - Aggregate Information dialog box.
  - VR-Forces prompts users to save scenarios under certain conditions.
  - The Set Altitude, Set Location, and Set Heading dialog boxes show the current values for the selected entity.
  - Added Stealth Detach menu item to allow users to explicitly detach the Stealth from a given entity.
  - Echelon View now sorts by designator instead of by name, alphabetically.

Updated GUI API

The updated GUI API is documented in *MÄK VR-Forces Developer's Guide*. For directions for porting your existing GUI code to the new version, see "VR-Forces GUI API Migration Instructions", on page 23. The GUI API has the following new examples:

- filterMenu – This example will show you how to add new menus and menu items to an existing application and hook up those menu items to provide functionality for your new application.
- trackHistory – This example shows you how to use the DtMtlSymbolMapper and DtPvdSymbolUpdater to add new symbols to the application and then modify them with new data.
- autoRewind – This example shows you how to install and use an application level event controller and event handler.

New Formats for Scenario Files

As a result of changes made to plan files and other changes in this release, scenarios created with previous versions of VR-Forces may have problems running in VR-Forces 3.4-ngc. When you open a scenario created with a previous version of VR-Forces, you are prompted to upgrade the scenario to the new format. If you click Yes, a Save As dialog box opens. Save the scenario. (You may want to backup old scenarios before you upgrade to the new format.)

Terrain API Examples

VR-Forces has three new examples to support the Terrain API:

- createTerrainDb - shows how to create a simple terrain database using the terrain API.
- modifyTerrainDb - shows how to modify an existing terrain database (deform the terrain skin) using the terrain API.
- addTerrainFeatures - shows how to add terrain features (vector data) to a terrain database using the terrain API.

VR-Forces includes the following new terrain databases to support the examples:

- *simpleTerrain.gdb* - Simple flat (featureless) terrain generated by the createTerrain example.
- *simpleFeatures.gdb* - Generated by the addFeatures example. It is a modified version of *simpleTerrain.gdb* (added a few features to it (trees, building, river).
- *modifiedSimpleTerrain.gdb* - Generated by the modifyTerrain example. It is a modified version of *simpleTerrain.gdb* (terrain skin has been altered to have a 60 meter deep depression in one corner).

New Startup Options

VR-Forces has two new startup options:

- The `-c` option lets you specify a file other than *./config/vrf.ent* to populate the Create menu. The syntax is `-c file`.
- The `-z` option lets you run VR-Forces in combined mode without specifying a back-end. When you use `-z`, VR-Forces runs with the default back-end (vrfSim) using the default site and application ID values.

Changes and Additions to Vector File Metafile Records

The information contained in PvShapeMetafileRecords has been split into two record types: PvShapeMetafileRecord and ShapeVectorMetafileRecord. See "7.4.7 The SHP (Shapefile) Record (replacement for section 7.4.7)", on page 13 and "Update to 7.5.1, Setting Shapefile Parameters", on page 16.

DFAD data is now configured in DfadMetafileRecords. See "DFAD Records", on page 17.

New MTL Parameters

VR-Forces has two new MTL parameters that you can use in the *vrf.mtl* configuration file:

- **VRF_createMenuFilename** specifies an alternative file to populate the Create Entity menu. By default, the file *./config/vrf.ent* is loaded.

- **VRF_askSaveScenarioIfChanged** specifies whether or not to prompt a user to save a scenario when they run it for the first time or when they close it.
  0 = do not prompt.
  1 = prompt.

Changes to Network Representation of DtSimTasks

The *DtSimTask* class (base class for all tasks) has changed its implementation of netRepSize(), setToNet(), and setFromNet(). This is because the base *DtSimTask* class now reads and writes data to the network representation of the task. If you have created your own derived kind of *DtSimTask*, you must call down to the base class version of these functions in your derived class. See the addTaskSim or addTaskGui examples for an example of implementing these functions in a derived kind of *DtSimTask*.

In your implementation of netRepSize(), you must down to the base class netRepSize(), and add that to the size of the network representation for the task.

```
int DtRetreatTask::netRepSize() const
{
  int size;
  size = DtSimTask::netRepSize();
  size += sizeof(DtNetRetreatTask);
  size += myRetreatKind.length();
  size = (int) ceil(size / 8.) * 8;
  return size;
}
```

In your implementation of setFromNet(), you must call down to the base class version of setFromNet(), to ensure that base class data is retrieved from the network representation. Use the base class netRepSize() to find the correct offset into the network representation to start the derived class's data.

```
DtBoolean DtRetreatTask::setFromNet(const char* contentBuffer,
unsigned contentSize)
{
  if(!DtSimTask::setFromNet(contentBuffer,contentSize))
  {
    return DtFalse;
  }
  DtNetRetreatTask* taskBuffer;
  DtU16 stringLen;
  char buf[256];
  const char* stringPtr;
  const char* bufferOffset = contentBuffer + DtSimTask::netRepSize();
  taskBuffer = (DtNetRetreatTask*)bufferOffset;
  . . .
```

Similarly, in setToNet(), you must call down to the base class version of setToNet(), to ensure that the base class data is copied to the network representation. Use the base class netRepSize() to find the correct offset into the network to copy the derived class data to.

```
DtBoolean DtRetreatTask::setToNet()
{
  if(!DtSimTask::setToNet())
```

```
{
  return DtFalse;
}
if (!netRep())
{
  return DtFalse;
}
char* bufferOffset = myNetRep + DtSimTask::netRepSize();
DtNetRetreatTask* taskBuffer = (DtNetRetreatTask*)bufferOffset;
. . .
```

Miscellaneous API Changes

VR-Forces 3.4-ngc includes the following additional changes to the APIs:

- Added MIN_USER_RADIO_MESSAGE_TYPE #define to *radioMsgTypes.h*. This value represents the lowest (starting) value for user-defined DtSimTask types. Toolkit users should be aware of this value and choose integer types for their derived DtSimTask classes greater than or equal to this value.
- Added MIN_USER_INTERFACE_MESSAGE_TYPE and MAX_USER_INTERFACE_MESSAGE_TYPE to *msgTypes.h* to define a safe range of integer types toolkit users can use for their own custom interface messages (derived DtSimInterfaceContent types).
- Allow plans to be assigned by name, to overcome bug in which echelon ID is not always valid at the time plan is assigned.
- The simplePlan and simpleCGF examples are now console windows, reducing possibility of losing track of 'windowless' copies of the process.
- The *objectManager* example shows how to override a high-level manager.
- The target detection sensor tests for intersections with the vector network. Therefore, if your database has vector data, the sensor will be blocked by features such as buildings, trees, and so on.
- VR-Forces is now based on standard C++ IO streams.

New Front-End Features

VR-Forces 3.4-ngc has the following new features in the GUI.

Hiding All Control Objects

You can toggle the display of control objects on and of with the View → View Control Objects menu option.

Displaying the Map in Full Screen Mode, without Toolbars

You can display the map so that it covers the entire screen, with no toolbars displayed. Choose View → Full Screen.

Aggregate Information Dialog Box

If you select an aggregate entity and choose the Entity Information option, it displays information specific to the aggregate, including a list of subordinate entities.

VR-Forces Prompts Users to Save Scenarios Under Certain Conditions

The first time you run a simulation after you create a new scenario or load a scenario, VR-Forces asks you if you want to save the scenario. It also prompts you to save the scenario when you close it. You can disable this feature by setting `VRF_askSaveScenarioIfChanged` to 0.

Set Altitude, Set Location, and Set Heading Show the Current Values

Most Set Condition dialog boxes do not display the current values for the condition they control. They show default values or the last value entered. The Set Altitude, Set Location, and Set Heading dialog boxes now show the current value for the selected entity.

Stealth Detach Menu Option

The Stealth Detach menu item to allows you to explicitly detach the Stealth from an attached entity.

Echelon View Sort Order

The Echelon View window sorts by designator instead of by name, alphabetically.

Documentation Updates

The PDF version of *MÄK VR-Forces User's Guide* is updated to include all new features. Also, note the following clarifications to the printed version of the manual:

MÄK RTI Availability

The printed version of *MÄK VR-Forces User's Guide* states that the MÄK RTI comes with, or is included with, all MÄK products. This statement means that the MÄK RTI software is on product CDs. It does not mean that purchase of the VR-Forces includes a MÄK RTI license. The MÄK RTI is licensed separately from other MÄK products.

Extent Metafile Records

The description of ExtentMetafileRecords is unclear about the role and syntax for the coordinate system value.

CoordinateSystem specifies the coordinate system of the extent you are defining. The SWCorner and NECorner must be expressed in this coordinate system. Recall that VR-Forces is only capable of simulating on UTM databases. So if this extent is going to be used by the VR-Forces simulation engine (as opposed to a VR-Forces GUI that is used solely as an observer), CoordinateSystem must be Utm.

Regardless of the coordinate system used, you must specify a z value for SWCorner and NECorner. Since it is not actually used, you can use 0.0.

Specifying the Coordinate System for Paper Map Records

The coordinate system in which the Paper Map is defined must be aligned with the display coordinate system of the PVD or VR-Forces GUI, in order for the features on the map to line up correctly with their locations in the simulated world. (Display coordinate system of the PVD or VR-Forces GUI is dictated by the coordinate system indicated in the Terrain Database Record or Extent Record that precedes the Paper Map Record in your *.map* file).

For example, if your underlying terrain database or extent is defined in UTM coordinates, the X and Y axes of your paper map image must be aligned with the X and Y axes of the UTM coordinate system (lines of constant UTM easting and northing should be vertical and horizontal lines respectively, in your map.) On the other hand, if your underlying terrain database is defined in geodetic coordinate, then lines of latitude should run exactly horizontally, and lines of longitude should run exactly vertically in your papermap.

Recall that VR-Forces can simulate only in UTM coordinates, so if you are using a papermap with the VR-Forces GUI to control a VR-Forces simulation, you must use a UTM Terrain Database or Extent, which means that your papermap image must be aligned with the UTM axes.

Specifying the Coordinate System for Pixmap Paper Map Records

CoordinateSystem indicates the coordinate system along which your papermap image is aligned. SWCorner and NECorner must be expressed in this coordinate system. This coordinate system should match the coordinate system of the underlying terrain database or Extent Record. (See "Specifying the Coordinate System for Paper Map Records", on page 12).

SWCorner and NECorner specify the x, y, and z coordinates for the southwest corner and the northeast corner of the papermap. These coordinates must be expressed in the coordinate system designated by the CoordinateSystem attribute described above. In the Geodetic coordinate system, the x and y values define the corners in latitude and longitude (degrees). In the UTM and LCC coordinate systems, the x and y values define the corners in meters. The z value is not used, but you must include it. Setting it to zero satisfies the requirement.

CTDB Metafile Records

The syntax for CTDB metafile records has changed. The syntax is different for the two types of supported CTDB files, as follows:

CTDB Version 4

```
C4bMetafileRecord
{
   File filename
}
```

where **File** specifies the file name of a CTDB version 4 database. The filename must end with *.c4b*.

CTDB Version 7

```
C7bMetafileRecord
{
   File filename
}
```

where **File** specifies the file name of a CTDB version 7database. The filename must end with *.c7b*.

7.4.7 The SHP (Shapefile) Record (replacement for section 7.4.7)

To use shapefiles, you define a PvShapeMetafileRecord and, optionally, one or more -ShapeVectorMetafileRecords. For general information about shapefiles, see "Shapefiles," on page 7-16.

PvShapeMetafileRecord

A PvShapeMetafileRecord entry has terrain information and one or more File entries. The format for a PvShapeMetafileRecord entry is as follows:

```
PvShapeMetafileRecord
{
  TerrainType {postElevationData | poly | zpoly}
  [SeaRectangleAdded | GroundRectangleAdded]
  Projected {true | false}
  [Origin lat lon]
  [MinimumDistanceBetweenPoints distance]
  [ThresholdForSeaLevel threshold]
```

```
    File filename
    elevationFactorToMeters elevation
    {elevationInData | elevationField field_name | elevationValue
    elevation}
    ...
}
```

where:

- **TerrainType** defines how to interpret the data stored in the shapefiles. The possible values are:
  - **postElevationData** specifies that data in the shapefiles is used as points in an irregular grid. The type of shapefile is ignored.
  - **poly** specifies that data is stored as polygons. Valid only with polygon or multipatch shapefiles.
  - **zpoly** specifies that data is stored as polygons with elevation. Valid only with zpolygon or multipatch shapefiles.
- **SeaRectangleAdded** adds a surface at sea level across the entire database. Use this parameter if the shapefile does not include information for the entire area covered by the extent of the database. This parameter is optional and is mutually exclusive with GroundRectangleAdded. See Figure 7-1.
- **GroundRectangleAdded** adds a surface at the lowest ground elevation across the entire database. Use this parameter if the shapefile does not include information for the entire area covered by the extent of the database. This parameter is optional and is mutually exclusive with SeaRectangleAdded. See Figure 7-1.
- **Projected** specifies that the data is UTM (true) or lat, lon (false).
- **Origin** *lat lon* specifies the origin of the database in latitude, longitude, in degrees.
- **MinimumDistanceBetweenPoints** specifies the minimum distance between points, in meters. Used with postElevationData. If two points are closer than this distance, the second one is omitted. Default:150 m.
- **ThresholdForSeaLevel** specifies the elevation below which terrain is considered to be at sea level. Useful when using postElevationData to set new points used to build the GDB and for setting soil types. Default: 0.

The File Entry in A PvShapeMetafileRecord

A PvShapeMetafileRecord must have at least one File entry. The first File entry is used to calculate the origin of the database, if it is not specified with the Origin parameter. If two or more File entries overlap, the VR-Forces uses the highest elevation. The format of the File entry is as follows:

```
File filename
elevationFactorToMeters elevation
{elevationInData | elevationField field_name | elevationValue
elevation}
...
```

where:

- **File** specifies the name of the shapefile. You must have a shapefile (.shp) and dBase file (.dbf) named filename.
- **elevationFactorToMeters** specifies a factor to use to transform the elevation units into meters. Default: 1.
- **elevationInData** | **elevationField** *field_name* | **elevationValue** *elevation* specifies how to calculate the elevation data, as follows:
  - **elevationInData** specifies that the shapefile contains elevation information, for example, zpoly files.
  - **elevationField** specifies the name of the field in the .dbf file that contains the elevation data.
  - **elevationValue** specifies an arbitrary elevation.

ShapeVectorMetafileRecords

A ShapeVectorMetafileRecord entry specifies feature data information. It has one or more **File** entries. You can specify one or more ShapeVectorMetafileRecords to provide feature data for any of the supported database types, for example, GdbMetafileRecord, DfadMetafileRecord, or PvShapeMetafileRecord.

! The origin in a ShapeVectorMetafileRecord must match the origin in the corresponding metafile record.

The format for a ShapeVectorMetafileRecord entry is as follows:

```
ShapeVectorMetafileRecord
{
  Projected { true | false }
  [Origin lat lon]

  File filename
  type type
  ...

}
```

where:

- **Projected** specifies that the data is UTM (true) or lat, lon (false).
- **Origin** *lat lon* specifies the origin of the database in latitude, longitude, in degrees.

The File Entry in A ShapeVectorMetafileRecord

The format of a File entry is as follows:

```
  [File filename
  type type
]
  ...
```

where:

- **File** specifies a file to be imported as feature data. The file must be a .shp file.
- **type** specifies the type of information in the feature data file, according to the description in *./data/importDfad/dfad_sedris_map.txt*.

The color is set based on the file *sedris_colormap.txt*. (See "Importing DFAD or DFD Files into GDB," on page 7-5.) If you want to import feature data with a different specification, add a new entry to dfad_sedris_map.txt.

Update to 7.5.1, Setting Shapefile Parameters

The following is an update to the examples in section 7.5.

Examples

This section has several examples of shapefile records.

```
PvShapeMetafileRecord
{
  TerrainType zpoly
  SeaRectangleAdded
  Projected true
  Origin 47.05 -35.90

  File grb_505_2.shp
  elevationFactorToMeters 1
  elevationInData
}


PvShapeMetafileRecord
{
  TerrainType postElevationData
  SeaRectangleAdded
  Projected true
  MinimumDistanceBetweenPoints 150
  Origin 17.05 -61.90
  ThresholdForSeaLevel 25

  File ancoast.shp
  elevationFactorToMeters 1
  elevationField ELEVATION

  File antconto.shp
  elevationFactorToMeters 1
  elevationField HTM

}


ShapeVectorMetafileRecord
{
  Projected true
  Origin 17.05 -61.90
```

```
  File antspot.shp
  type 774

  File antroad.shp
  type 240

  File acoast.shp
  type 948

  File antcorals.shp
  type 915

  File antponds.shp
  type 940

  File antcliff.shp
  type 924

  File antdrain.shp
  type 945

  File antmangr.shp
  type 953
}


PvShapeMetafileRecord
{
  TerrainType poly
  GroundRectangleAdded
  Projected false

  File gl_eda_elevation.shp
  elevationFactorToMeters 1
  elevationField ELEV_METER
}
```

DFAD Records

You can import DFAD data directly into the VR-Forces using DfadMetafileRecords. The syntax for the record is:

```
DfadMetafileRecord
{
  [ImportDescriptorFile import_desc_filename]
  [FidSmcFile fid_smc_filename]
  [DfadSedrisMapFile dfad_sedris_map_filename]
  [ColorMapFile color_map_filename]
  [Cropping {true|false}]

  File filename
  [File filename]
  ...
}
```

where:

- **ImportDescriptorFile** is the name of a file that defines the list of DFAD/DFD features to import. Each possible feature specifies IMPORT or NOIMPORT. By default it loads the file *..\data\importDfad\importFile.txt.*
- **FidSmcFile** is the name of a file that defines the list of FID and SMC codes for DFAD and DFD basic features. By default it loads the file *..\data\import-Dfad\FID_SMC.txt*.
- **DfadSedrisMapFile** - Maps each type of DFAD/DFD feature to a default SEDRIS feature. You can edit this file or specify a different map file with the `tdbTool -e` option. By default it loads the file *..\data\importDfad\dfad_sedris_map.txt*.
- **ColorMapFile** is the name of a file that defines the color representation for each subcategory in the SEDRIS code, in the format RGBA. By default it loads the file *..\data\importDfad\sedris_colormap.txt*.
- **Cropping** is a boolean flag that indicates whether or not to clip the incoming vector data to the extent of the terrain.
- **File** is the name of a DFAD file to load. You can specify any number of files.

Example:

```
DfadMetafileRecord
{
  File prague.dfad
}
```

prague.dfad is the name of the DFAD file to load.

DFD Records (Windows Only)

On Windows platforms, Multigen DFD data can be imported using DfdDfadMetafile-Record. This record is almost identical to a DfadMetafileRecord, except that you can specify Multigen DFD files to load in addition to DFAD files. The syntax for the record is:

```
DfdDfadMetafileRecord
{
  [ImportDescriptorFile filename]
  [FidSmcFile filename]
  [DfadSedrisMapFile filename]
  [ColorMapFile filename]
  [Cropping {true|false]

  File filename
  [File filename]
  ...
}
```

where:

- **File** is the name of either a Multigen DFD file or a DFAD file to import. You can specify any number of files.

Example:

```
DfdDfadMetafileRecord
{
  File prague1.dfd
  File prague.dfad
}
```

MÄK VR-Forces Developer's Guide

The body of *MÄK VR-Forces Developer's Guide* has been updated to document the GUI API, Terrain API, and new component connection mechanisms. The model descriptions in Appendix A, "Component Model Descriptions," have not been updated to the new port and port group mechanism. However, since the old and new methods are conceptually fairly similar, you should be able to transpose the old descriptions to the new classes.

This release of VR-Forces implements Plan File version 2.0. The description of plan file syntax has not been fully updated to match this new version.

Bug Fixes

This release fixes the following problems:

- A DtedMetafileRecord that does not have a valid File entry no longer hangs vrfGui. 1740
- Support for repeatability in VR-Forces has been restored.
- The Object Manager now creates any sub-objects specified in an object parameters entry in the object parameters database.
- VR-Forces can now load c4b and c7b files from terrain metafiles (*.map* files).
- Fixed a problem drawing contour lines on terrain databases containing reference nodes, which can happen when terrain is imported from OpenFlight files.
- Fixed various subtasking bugs. The subtasking flag is now included in the network representation of task messages.
- Aggregate symbols now freeze in the GUI when a simulation is paused (they no longer drift due to dead reckoning).
- Newly added icons are now always created with the current icon scale, rather than the default icon size.
- The icon scaling slider now works correctly when you click on it to resize, as opposed to click and drag.
- Icon scaling behavior was inconsistent when zooming in or out of the display. Symbols that were not in the view before a zoom operation were appearing with the incorrect scale when brought into view. This has been fixed.
- A crash bug when configuring a fire-for-effect task on an entity through the GUI has been fixed.

- Creation of concave areas and obstacles in the GUI generates a warning message. Previously, it allowed you to create the objects, but failed (silently) to create them in the back-end.
- VR-Forces no longer fails completely when adding an object with a NULL object name (this can happen with a non-VR-Forces entity that does not have a unique marking text string). When adding an object with an invalid or duplicate name, we print a warning letting you know that the object will not be able to be looked up by name. The object will still be accessible through other means (for example, by its global ID), but will not be able to be referenced by name in tasks and plans.
- VR-Forces had problems handling objects with NULL identifiers. Now, if VR-Forces detects an object that has a NULL identifier, it ignores it.
- Fixes to the indirect fire controller and damage actuator have improved how entities take damage from indirect fire.
- If you specified too small of a clipping area in DTED metafiles, VR-Forces crashed.
- The aspect ratio of a rubber band was not being maintained correctly when zooming in on an area.
- Fire-for-effect dialog boxes are no longer hidden by the Edit Plan window.
- Aggregate entities now always initially appear in a collapsed state in the GUI. Previously, the initial display state of aggregate (collapsed or expanded) was unpredictable.
- The log file *pvd.txt* has been renamed to *pvd.log*, to be consistent with other products.
- Zooming out from an aggregate and back in always showed it expanded.
- We removed the 'Entity List' and 'Echelon View' entries from control objects' right-click popup menus. These options were not appropriate in that context.
- The *DtTargetDetectionPSR* class now maintains state for checkpointing. It saves the time to the next detection attempt and the list of previously detected targets. This allows the *DtTargetDetectionSensor* to be restored to its state.
- Missiles can now target lifeforms, munitions, cultural features, and platforms. Previously, they could only target platforms.
- Joysticks now work correctly with DirectInput. 1226
- We improved the configurability of the target detection sensor. (It will now detect object types based on types specified in object parameters database). Previously, it was tied to platform entity types (DIS definition) for targets (it ignored lifeforms and missiles).

Known Problems

VR-Forces Release 3.4-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.1189
- If you are using a DTED database and "Projected False" is specified in the PvDtedMetafileRecord, entities do not execute movement tasks. 99
- The *world.gdb* file is based on a DTED database. If you use this database, some entities may not execute tasks. Some entities will not display properly.
- 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.
- Surface entities are not configured to fire weapons or be fired upon. You can customize the object parameters database entries to enable this.
- The pseudo-aggregate formation controller ignores entities in a formation that are independently tasked when determining if the promotion order needs to be updated. Currently it has no way to determine if remote entities are independently tasked. 995
- 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. 1565
- 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.
- Subsurface entities use incorrect icons. 1649
- If you run VR-Forces with the DMSO RTI on Windows in combined mode, when you exit VR-Forces (the GUI), the back-end does not resign from the federation execution. The GUI and back-end console go away, but the vrfGui process does not shut down. You have to go into Task Manager to kill it. If you do not restart the rtiexec after shutting down VR-Forces, restarting VR-Forces after exiting takes much longer than with a fresh rtiexec, and the behavior of VR-Forces may be erratic. 1838
- The Set Formation request does not work properly on aggregate entities that are subordinate members of larger aggregate units. 1842
- 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. 1851
-

- 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. 995
- 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.
- If you are using TrueType fonts, the symbols for air aggregates are incorrect. 1641

VR-Forces GUI API Migration Instructions

The VR-Forces GUI API has been rewritten to organize its functionality into smaller class units and make the subclassing and adding of functionality easier. *MÄK VR-Forces Developer's Guide* fully describes the updated API. The following sections summarize the changes made and provide migration instructions for applications written with previous versions of the API. The examples have been modified to work with the updated API, and where appropriate, the old code has been commented out and new code added to show how to migrate from the old API to the new.

Removed Classes

The *DtVrfGuiPvdMapArea* class has been removed. Any class that you have derived from *DtVrfGuiPvdMapArea* must now be changed to subclass *DtPvdMapArea*.

New Classes

Most of the functionality that used to exist in the *DtVrfGuiPvdMapArea* and *DtVrfGuiPvdWindow* objects has been divided among the following classes:

- *DtVrfGuiPvdEventController*
- *DtVrfGuiPvdAppEventController*
- Various menu object (*DtVrfGuiPvdFileMenu*, for example) that encapsulate the creation of menus and menu items
- *DtVrfGuiPvdRightClickMenu*
- *DtPvdMiddleClickMenu*.

These classes are explained in detail in *MÄK VR-Forces Developer's Guide*.

The Names of Factory Classes have Changed

The names of the default factories have changed to include the word "Default" in them. If you used to reference *DtVrfGuiPvdFactory*, you must now reference *DtVrfGuiPvdDefaultFactory*. Similarly, if you referenced *DtPvdFactory* you must now reference *DtPvdDefaultFactory*.

Changes to Existing Classes

The *DtMtlSymbolMapper* and *DtPvdSymbolUpdater* classes have been changed to allow for easier subclassing and modification of symbols.

The *DtMtlSymbolMapper* Class

DtMtlSymbolMapper has been reorganized from large member functions that create entire symbols into individual parts that allow you to override a very specific piece of functionality that you might want to change. Refer to *mtlSymbolMapper.h* for more information about the structure of the class.

How to Change Code that Subclasses from DtMtlSymbolMapper

If you override the createSymbol() member function, then you do not need to make any changes. If you override member functions such as getEntitySymbolFromData(), getAggregateSymbolFromData(), and so on, those member functions have changed to the following:

```
virtual DtSymbol* processCreateSymbol(const DtEntityModelData& data,
  void *usr);
virtual DtSymbol* processCreateSymbol(const DtAggregateModelData& data,
  void *usr);
virtual DtSymbol* processCreateSymbol(const DtControlObjectModelData&
  data, void *usr);
virtual DtSymbol* processCreateSymbol(const DtFireInterModelData& data,
  void *usr);
virtual DtSymbol* processCreateSymbol(const DtDetonateInterModelData&
  data, void *usr);
```

By overriding the new version of the member function, you should be able to use your code exactly as it exists today in the new member function.

The *DtPvdSymbolUpdater* Class

Like *DtMtlSymbolMapper*, *DtPvdSymbolUpdater* has been reorganized to provide more member functions that target specific functionality. Please refer to *pvdSymbolUpdater.h* more information about the structure of the class.

How to Modify Code that Subclasses from DtPvdSymbolUpdater

If you currently subclass *DtPvdSymbolUpdater*, look at the header file. If you have all your code in the updateEntitySymbol() member function, there may be new member functions that are more appropriate for you to override (for example, in one of the specific update member functions). The general update member functions now have the following prototypes:

```
// Update entity attributes
virtual void updateSymbol(DtPvdEntitySymbol* es,
  const DtEntityModelData& d, void* usr) const;
// Update aggregate attributes
virtual void updateSymbol(DtPvdEntitySymbol* es,
  const DtAggregateModelData& d, void* usr) const;
// Update control object attributes
virtual void updateSymbol(DtNameLabelSymbol* es,
  const DtControlObjectModelData& d, void* usr) const;
// Note that this updateOptions is used by both entity and aggregate
  symbols
virtual void updateOptions(DtPvdEntitySymbol* es,
  const DtPvdDisplayOptions* opt, void* usr) const;
// Update options for name label symbols (waypoints, areas, etc.)
```

```
virtual void updateOptions(DtNameLabelSymbol* nls,
  const DtPvdDisplayOptions* opt, void* usr) const;
// Update options for interations symbols (fire, detonate)
virtual void updateOptions(DtInterSymbol* nls,
  const DtPvdDisplayOptions* opt, void* usr) const;
```

You can override the member function that meets your need (*DtNameLabelSymbols* now represent control objects), calling the parent implementation first.

DtModelData and its subclasses now return the type of data that they represent. If you have a subclass of *DtModelData* that is a new type, you can assign the particular type of that model data by using the DtNewModelDataType macro to create a new model data type. You can then use this type to register creation and update functions in the *DtMtlSymbol-Creator* and *DtPvdSymbolUpdater* classes.

Changes to the Menu System

The menu system has been removed from various init() functions in the window classes (*DtVrfGuiPvdWindow.*) and placed into discrete objects that are subclassed from *DtMenuWidget* and *DtPopupMenu* (in *menuWidget.h* and *popupMenu.h*). These classes provide a simple interface to adding new menu items and action functions to menus. The header file *menuItems.h* describes the menus and menu items that are contained in the application and the base variables that you can use.

The *DtTaskActionGroup* and *DtSetActionGroup* Classes

DtTaskActionGroup and *DtSetAction* no longer have their creation functions set in the *DtVrfGuiPvdFactory*. Their creation functions are now registered in the *DtMenuWidget* class.

Migrating Code that Subclasses from *DtTaskActionGroup* and *DtSetActionGroup*

To register your task and set menus to be created, you must do the following in *main.cxx*:

Old code:

```
// Tell the factory to create our instance of the task group
f.setTaskActionGroupCreatorFcn(MyTaskActionGroup::create);
```

New code:

```
#include "menuItems.h"
DtMenuWidget::globalReplacePopupMenu(DtTaskMenu,
  MyTaskActionGroup::create);
```

or:

```
DtMenuWidget::globalReplacePopupMenu(DtSetMenu,
  MySetActionGroup::create);
```

Change the create() prototype and object constructors to follow the new prototype:

Old code:

```
MyTaskActionGroup(QObject* parent, const char* name = 0,
  bool exclusive = TRUE);
static DtTaskActionGroup* create(QWidget* p, const char *name,
  bool enabled);
```

New code:

```
MyTaskActionGroup(QObject* parent, const char* name = 0);
static DtPopupMenu* create(QObject* parent, const char *name);
```

Replacing the myMapArea Member Variable

- The **myMapArea** member variable has been removed and can be replaced with a reference to the member function mapArea().

Migrating Code that Adds Menu Items.

The init() function no longer exists and has been changed to a function called initMenuItems().

Any menu items that you have created have to be created in a different form. Menu items are no longer created as QActions. They are now scheduled to be created in the initMenuItems() member function and then created and added to the menu in the createMenuItems() member function. Using the addTaskGui as an example, if your old code looks like this:

```
bool MyTaskActionGroup::init(DtPvdMapArea* mapArea)
{
  if(!DtTaskActionGroup::init(mapArea))
  {
    return false;
  }

  myTaskRetreatAction = new QAction( this, "myTaskRetreatAction" );
  myTaskRetreatAction->setText( trUtf8( "Retreat..." ) );
  myTaskRetreatAction->setMenuText( trUtf8( "Retreat..." ) );
  myTaskRetreatAction->setToolTip( trUtf8(
    "Task selected entity/aggregate to retreat." ) );

  // Connect the actions.
  connect
  (
    myTaskRetreatAction,
    SIGNAL(activated()),
    this,
    SLOT(taskSelectionRetreatTask())
  );

  return true;
}
```

You must change it to look like this:

```
bool MyTaskActionGroup::initMenuItems()
{
```

```
  if(!DtTaskActionGroup::initMenuItems())
  {
    return false;
  }

  insertMenuItem(DtNoMenuItem, DtUserMenuItemStart,
    trUtf8( "Retreat..." ), trUtf8( "Retreat..." ),
    trUtf8( "Task selected entity/aggregate to retreat." ));

  addActionAndValidatorForType(DtUserMenuItemStart, this,
    SLOT(taskSelectionRetreatTask()));

  return true;
}
```

Notes

- The changes that need to be made to subclasses of *DtTaskActionGroup* must also be made to subclasses of *DtSetActionGroup*. Use the description of changing *DtTaskActionGroup* subclasses as a template for changing *DtSetActionGroup* subclasses. If you look at the prototype of the addActionAndValidatorForType() member function, you will see that you can add a validator after the action in the final two parameters.
- The base DtUserMenuItemStart constant was used. If you define more than one type of menu item in your application, you must create different constant values starting at the *DtUserMenuItemStart* constant, using those as references in your insertMenuItem() and addActionAndValidatorForType() member functions. This will ensure that your new menu types do not conflict with any types. defined by VR-Forces.

The application objects define the menu changes they want to make for the application (*DtVrfGuiApplicationDtPvdApplication*), in the initMenuSytems() member function called from the init() member function. If you want to make modifications to the menu system, be sure to make them after the init() member function of the application object. You may also create your own subclass of the application object to create your menu system:

```
bool MyApplication::initMenuSystem()
{
  if (!DtPvdApplicationDtVrfGuiApplication::initMenuSystem())
  {
    return false;
  }

  DtMenuWidget::globalReplacePopupMenu(DtViewMenu, MyViewMenu::create);
  DtMenuWidget::globalReplacePopupMenu(DtCreateMenu,
    MyCreateMenu::create);
  DtMenuWidget::globalReplacePopupMenu(DtEntitiesMenu,
    MyEntityMenu::create);
  DtMenuWidget::globalReplacePopupMenu(DtTaskMenu,
    MyTaskActionGroup::create);
  DtMenuWidget::globalReplacePopupMenu(DtOptionsMenu,
    MyOptionsMenu::create);

  return true;
}
```

See the addTaskGui and mergedSet examples for details of migrating code. See the simpleScenarioPlayer example for how to globally configure the main menu bar through the *main.cxx* file.

Adding Menu Items to Existing Menus

To add menu items to existing menus (for example the View menu), use one of the following methods.

- Use the window member function:

  ```
  bool MyMainWindow::init(const DtTMFactory* factory, DtTMMiniMap* miniMap)
  ```

  If you override this member function, you can add your new menu item to the View menu after calling the parent's init() member function.

This method will only allow you to append menu items to an exsisting menu.

The following code is from the fogOfWar example:

```
bool MyMainWindow::init(const DtTMFactory* factory, DtTMMiniMap* miniMap)
{
  if(!DtVrfGuiWindowDtPvdWindow::init(factory, miniMap))
  {
    return false;
  }

  // Now, add our new menu item -- uses new 3.4 toolkit changes
  DtPopupMenu* viewMenu = myMenuWidget->menuForType(DtViewMenu);

  viewMenu->addMenuItemSeparator(0);
  viewMenu->insertMenuItem(DtNoMenuItem, DtUserMenuItemStart,
    trUtf8( "Switch to Friendly" ), trUtf8( "Switch to Friendly" ),
    trUtf8(" Switch between showing all, friendly or opposing forces in
    fog of war filter"));
  viewMenu->addActionAndValidatorForType(DtUserMenuItemStart, this,
    SLOT(choiceSwitch()));

  return true;
}
```

- Override the menu.

  If you want to create your own menu class, you must override the *DtVrfGuiPvd-ViewMenu* class, install your subclass as the menu to be created, and then create the new menu item as follows:

  ```
  // The main.cxx file contains global changes to the default
  // DtVrfGuiPvdApplication and factory objects.
  // It will also allow for modification of the default menu system

  #include "vrfGuipvdDefaultFactory.h"
  #include "vrfGuipvdDefaultFactory.h"
  #include "vrfGuipvdApplication.h"
  #include "menuItems.h"
  #include "menuWidget.h"
  #include "myViewMenu.h"
  ```

```
int main(int argc, char* argv[])
{
  // Set our aggregate filter handler to create a new fog of war filter.
  // This must be done first as when the app is inited it will use the
  // default handler

  DtEventController::addHandler(DtEchelonViewHandlerType,
    MyFogOfWarHandler::create);

  DtVrfGuiPvdDefaultFactory factory;
  DtVrfGuiPvdApplication app(argc, argv);

  if (!app.init(&factory))
  {
    return -1;
  }

  // Replace the default view menu with our new view menu. This member
  // function must be after the applications init() member function such
  // that our definition is not overwritten.
  DtMenuWidget::globalReplacePopupMenu(DtViewMenu, MyViewMenu::create);

  app.fileNewWindow();

  return app.exec();
}

#ifndef MYVIEWMENU_H
#define MYVIEWMENU_H

#include "vrfGuipvdViewMenu.h"

class MyViewMenu : public DtVrfGuiPvdViewMenu
{
  Q_OBJECT

public:
  MyViewMenu ( QObject * parent, const char * name = 0 );
  virtual ~MyViewMenu ();

  // Pure virtual overrides
protected:
  // Initializes and maps the following menu items. Note these items are
  // inserted as toggle menu items
  //
  // DtUserMenuItemStart to myOwner->choiceSwitch()
  // DtVrfGuiPvdViewMenu items (see vrfGuipvdViewMenu.h)
  virtual bool initMenuItems();

public:
  // Creator function that will return an instance of this object
  static DtPopupMenu* create(QObject* parent, const char* name = NULL);
};

#endif

#include "myViewMenu.h"
#include "menuItems.h"
```

```
MyViewMenu::MyViewMenu(QObject* parent, const char* name) :
DtVrfGuiPvdViewMenu(parent, name)
{
}

// Initialize our new version of the view menu by creating the stock view
  // menu first and then adding our new view item which will switch the
  // force type of the fog of war filter
bool MyViewMenu::initMenuItems()
{
  if (!DtVrfGuiPvdViewMenu::initMenuItems())
  {
    return false;
  }

  addMenuItemSeparator(0);

  insertMenuItem(DtNoMenuItem, DtUserMenuItemStart, trUtf8( "Switch to
  Friendly" ),
    trUtf8( "Switch to Friendly" ), trUtf8(" Switch between showing all,
    friendly or opposing forces in fog of war filter"));
  addActionAndValidatorForType(DtUserMenuItemStart, myOwner,
    SLOT(choiceSwitch()));}
}

DtPopupMenu* MyViewMenu::create(QObject* parent, const char* name)
{
  return new MyViewMenu(parent, name);
}
```

Refer to the fogOfWar example for an example of how to modify an existing menu from the init() member function. Refer to the filterMenu example for an example of how to create and install a new popup menu.

Validators for Menu Items

The validator map has been replaced with a function that uses Qt signals and slots to call validation functions for menu items. The addActionAndValidatorForType() member function allows you to assign actions and validator member functions for the menu items being created. Actions are called when a user selects a menu item. Validators are called each time the user displays a popup menu.

(From *popupMenu.h*)

```
// Adds a new action/validator pair. Supply NULL to have no action. The
// items passed are SLOTS that will be called via a connection to
// either the activated signal from the QAction (for the action) or
// the validateItems signal if something needs to be validated. The
// QObject* supplied is the item that will be signalled in response to
// both action and validator
virtual void addActionAndValidatorForType(int iType, QObject* aCall,
  const char* action, QObject* vCall = NULL,
  const char* validator = NULL);

#include "menuItems.h"

// Define our new menu item constant so we do not conflict with product
// menu items
```

```
const int DtMyMenuItem = DtUserMenuItemStart;

bool DtMyMenu::initMenuItems()
{
  // Call parent class such that our new menu item is inserted after all
  // other
  if (!DtVrfGuiPvdViewMenu::initMenuItems())
  {
    return false;
  }

  // Now add our new item and validation
  insertMenuItem(DtNoMenuItem, DtMyMenuItem, trUtf8("My Action"),
    trUtf8("My Action"));
  addActionAndValidatorForType(DtMyMenuItem, this, SLOT(myAction()),
    this, SLOT(myValidator()))

  return true;
}

// Enable menu item only if a terrain database is loaded. Called when the
// popup menu is displayed or activated
void DtMyMenu::myValidator()
{
  // Get a handle to our menu action
  QAction* action = menuItemForType(DtMyMenuItem);

  if (DtTMApplication::app()->terrainDatabase())
  {
    action->setEnabled(true);
  }
  else
  {
    action->setEnabled(false);
  }
}
```

See the unitStatus example for an example of creating a view menu subclass.

The View Menu

If you have created custom toolbars that you added to the View menu, and you toggle those menu items on and off, the way that you add the toolbars to the View menu has changed. To convert to the new system, you must do the following:

1. Subclass the *DtVrfGuiPvdViewMenu* as your own class:

```
#ifndef DtMyViewMenu_H
#define DtMyViewMenu_H

// VRF includes
#include "vrfGuipvdViewMenu.h"

class DtMyViewMenu : public DtVrfGuiPvdViewMenu
{
  Q_OBJECT

public:
  DtMyViewMenu ( QObject * parent, const char * name = 0 );
```

```
  virtual ~DtMyViewMenu ();

  // Initializes and maps the following menu items
  virtual bool initMenuItems();

public:
  static DtPopupMenu* create(QObject* parent, const char* name = NULL);
};

#endif
```

2. Create an object that adds your the options to the View menu. If you have more than one option, start at *DtUserMenuItemStart* and increase from there:

```
#include <myViewMenu.h>
#include <menuItems.h>

DtNewMenuItemType(DtMySwitchItem, 1);

DtMyViewMenu::DtMyViewMenu( QObject * parent, const char * name):
  DtVrfGuiPvdViewMenu( parent, name )
{
}

DtMyViewMenu::~DtMyViewMenu()
{
}

bool DtMyViewMenu::initMenuItems()
{
  // By calling the view menu parent method last, this ensures our menu
  // item will be shown first
  insertToggleMenuItem(DtNoMenuItem, DtMySwitchItem, trUtf8( "Switch
  Toolbar" ),
    trUtf8( "Switch Toolbar" ), trUtf8( "Activates my toolbar." ));

  // Unless you want to do some custom handling, all toolbars are now
  // turned on/off via the toggleToolbar() member function.
  // The toggleToolbar() function is defined in baseWindow.h and will use
  // the myViewMenuToolbarMap (see below) to match our menu item to a
  //specific toolbar
  addActionAndValidatorForType(DtMySwitchItem , myOwner,
    SLOT(toggleToolbar(bool)));

  if (!DtVrfGuiPvdViewMenu::initMenuItems())
  {
    return false;
  }

  return true;
}

DtPopupMenu* DtMyViewMenu::create(QObject* parent, const char* name)
{
  return new DtMyViewMenu(parent, name);
}
```

3. Subclass *DtVrfGuiPvdWindow* and override the initViewMenuToolbarMap() member function to map your new toolbar menu item to the toolbar it affects:

```
// Must do this mapping to be sure our toolbar can be shown/hidden
void DtMyUserWindow::initViewMenuToolbarMap()
{
  myViewMenuToolbarMap.insert(DtUserMenuItemStart , myNewToolBar);
  DtVrfGuiPvdWindow::initViewMenuToolbarMap();
}
```

4. Install your new View menu in *main.cxx* after you create the application object:

```
#include "vrfGuipvdDefaultFactory.h"
#include "vrfGuipvdApplication.h"

#include "menuItems.h"
#include "menuWidget.h"
#include "myViewMenu.h"

int main(int argc, char* argv[])
{
  DtVrfGuiPvdDefaultFactory f;
  DtVrfGuiPvdApplication a(argc, argv);

  if (!a.init(&f))
  {
    return -1;
  }

  // Must be done after init so application init does
  // not override changes
  DtMenuWidget::globalReplacePopupMenu(DtViewMenu,
    DtMyViewMenu::create);

  a.fileNewWindow();

  return a.exec();
}
```

5. Remove any code in your window subclass that used to turn the toolbar on and off.

Right and Middle Click Menus

Right-click and middle-click menus are now their own objects and are set in the factory using setCreateRightClickContextMenuCreatorFcn() and setCreateMiddleClickContextMenuCreatorFcn(). The functions that used to be in the windows classes (for example, updateRightClickSingleEntity()) are now in the *DtVrfGuiPvdRightClickMenu* class (see *vrfGuiRightClickMenu.h*). If you overrode these member functions to add your own right click functionality, you must now subclass *DtVrfGuiPvdRightClickMenu* and reimplement the member functions that you implemented in the windows class. Look at addMenuItem() in *contextMenu.h*. This member function takes a menu item type you want to add (as defined in *menuItems.h* or your own menu item types), and optionally, a separator and where to place the item in the menu:

```
void DtVrfGuiPvdRightClickMenu::updateRightClickSingleEntity()
{
  addMenu(tr("Task"), DtTaskMenu);
  addMenu(tr("Set"), DtSetMenu, true);
  addMenuForGroup(tr("Intervisibility"), DtIntervisibilityMenu,
    DtIntervisibilityAction, true);
```

```
  addMenuItem(DtEditAction, true);
  addMenuItem(DtEntitySkipTaskAction);
  addMenuItem(DtEntityEditPlanAction);
  addMenuItem(DtEntityViewPlanAction);
  addMenuItem(DtEntityAbandonPlanAction, true);
  addMenuItem(DtCollapseAggregateAction, true);
  addMenuItem(DtEntityInfoAction);
  addMenuItem(DtEntityListAction);
  addMenuItem(DtEchelonPanelAction, true);
  addMenuItem(DtCenterSelectionAction);
  addMenuItem(DtTrackSelectionAction);
  addMenuItem(DtClearTrackingAction, true);
  addMenuItem(DtEntityEnableJoystickAction);
  addMenuItem(DtEntityDisableJoystickAction, true);
  addMenuItem(DtStealthAttachAction);
  addMenuItem(DtStealthDetachAction);
  addMenuItem(DtStealthTeleportAction, true);
  addMenuItem(DtEntityDeleteAction);
}
```

Event Controllers and Event Handlers

Most of the functionality that was contained in the map areas, applications, and windows has been moved into event controllers (classes such as *DtVrfGuiPvdEventController and DtVrfGuiPvdAppEventController*) and event handlers (classes such as *DtCreateAreaHandler* and *DtBatchScenarioHandler*). An event controller is a manager of pieces of functionality (event handlers). It installs and runs events based upon actions taken in the interface. You can also install default handlers that are always present and perform some piece of functionality every tick, or when some keyboard or mouse event takes place (for example, the navigation handler or the drag and drop handler). *MÄK VR-Forces Developer's Guide* describes the different types of handlers and the event controllers that are in the system.

Accessing the Remote Controller

If you used to access the remote controller as:

```
DtVrfGuiPvdApplication* app =
  dynamic_cast<DtVrfGuiPvdApplication*>(DtTMApplication::app());
DtVrfRemoteController* controller = app->remoteController();
```

This has changed to:

```
DtVrfRemoteController* controller =
  DtVrfGuiPvdAppEventController::remoteController();
```

Also, if you subclass the application object and reference the remote controller member variable, you can change that to reference the remote controller as above.

Creating Your Own Remote Controller

The remote controller has been moved to the *DtVrfGuiPvdAppEventController*, and an accessor placed within that handler to set your version of the remote controller. If you used to create a version of the remote controller by subclassing the setupRemoteController() member function, that member function no longer works. After you initialize the application, you can reference the application controller as follows to set your new remote controller:

```
#include "vrfGuipvdDefaultFactory.h"
#include "vrfGuipvdApplication.h"
#include "vrfGuipvdAppEventController.h"

#include "myRemoteController.h"

int main(int argc, char* argv[])
{
  DtVrfGuiPvdDefaultFactory f;

  DtVrfGuiPvdApplication a(argc, argv);

  if (!a.init(&f))
  {
    return -1;
  }

  MyRemoteController* remoteControl = new MyRemoteController;

  remoteControl->init(DtVrfGuiPvdAppEventController::appController()->
    messageInterface());
  DtVrfGuiPvdAppEventController::appController()->
    setRemoteController(remoteControl);

  a.fileNewWindow();

  return a.exec();
}
```

You no longer need to subclass in order to set your remote controller.

The *DtEntityInfoDialog* Class

The Entity Information dialog box has been reorganized to allow for easier construction and code reuse by the Aggregate Information dialog box.

How to Modify Code That Subclasses from *DtEntityInfoDialog*

If you override the constructor to add new coordinate systems for the coordinate system list, you must now override and install a separate class in the factory. A new class called *DtBaseEntityInfoWidget* has been created to hold the coordinate system and common display information (algorithm, location, velocity, and so on) that is also used by the *DtAggregateInfoDialog* class.

You must subclass the *DtBaseEntityInfoWidget* class and install it as the widget to be created when an entity information or aggregate information window is created via the setBaseEntityInfoWidgetCreatorFcn() member function in the factory. You can then override the init() member function of the *DtBaseEntityInfoWidget* class, call the parent init(), and then remove your code from your subclass of the *DtEntityInfoDialog* class and place it in the DtBaseEntityInfoWidget init(). The various update member functions for updating velocity, location, and so on have also been moved to this new class. If your *DtEntityInfoDialog* implements these functions, then move your implementation to the *DtBaseEntityInfoWidget* class.

Editing Plan View Display Examples for Use with VR-Forces

VR-Forces includes examples that were written specifically for the MÄK Plan View Display, for example, the xMarksTheSpot example. When you compile and run the examples, they will look like the Plan View Display, not VR-Forces. To add full VR-Forces functionality to the examples, you must make the following changes to them:

1. Change instances of *DtPvdDefaultFactory* and *DtPvdApplication*, to *DtVrfGuiPvdDefaultFactory* and *DtVrfGuiPvdApplication*.
2. If there is a window class in the example, subclass that window from *DtVrfGuiPvdWindow* rather than *DtPvdWindow*.
3. Add the following libraries to the link line:
   - vrfqt*protocol*.lib
   - vrfguieventhandler*protocol*.lib
   - vrfmsgs*protocol*MD.lib
   - vrftasksMD.lib
   - vrfutilMD.lib
   - vrfcontrol*protocol*MD.lib
   - vrfplan*protocol*MD.lib
   - vrfcore*protocol*MD.lib.

where *protocol* is 5 for DIS or RPR for HLA.

When you rebuild the example, it will look like a VR-Forces example not a Plan View Display example.

New Architecture for Connecting Components

The mechanism for inter-component communication has been redesigned to make it more flexible, and easier to configure. Under the old approach, components that used *DtPortInterfaces* to connect to other components had to know about the class type of the other component that it was connecting to. This created an undesirable class dependency between the components being connected. This problem no longer exists under the new approach.

The classes used to represent data ports and connections between components have changed. The way in which connections between components are specified in the object parameters database has also changed. Details about the new architecture are in *MÄK VR-Forces Developer's Guide*. If you have written code to create components or extend components, or you have modified the object parameters database, you may be affected by this change. This section explains how to migrate to the new architecture for creating component.

The model descriptions in Appendix A, "Component Model Descriptions," have not been updated to the new port and port group mechanism.

Migrating to the New Port/Port Group Architecture

The *DtPort* and *DtPortInterface* classes are now obsolete. However, they are roughly analogous to some of the new classes.

Table 2:

Old Class	New Class
DtPort	*DtInputPort* (*inputPort.h*)
	DtOutputPort (*outputPort.h*)
DtPortInterface	*DtInputPortGroup* (*inputPortGroup.h*)
	DtOutputPort (*outputPort.h*)

In the old approach, components (*DtSimComponent*) that provided data had one or more *DtPorts*, and components that accepted data as input had one or more *DtPortInterfaces*. Under the new architecture, components that provide data may have one or more *DtOutputPorts*, *DtOutputPortGroups* (in a has-a relationship) or both, and components that accept data as input may have one or more *DtInputPorts*, *DtInputPortGroups* (also in a has-a relationship) or both. *DtInputPortGroup* and *DtOutputPortGroup* are collections of ports and provide a convenient interface to manipulate ports as a group.

There are several derived *DtInputPort* and *DtOutputPort* classes, representing the different kinds of data that can be exchanged through a port. For example the *DtIntegerOutputPort* and *DtIntegerInputPort* classes (both declared in *integerIOPort.h*) are used to send and receive integer values, respectively. These specific types of input and output port classes correspond to the old *DtIntegerPort* class. There are analogous *DtInputPort* and *DtOutputPort* derived classes for all of the old derived *DtPort* class types.

The new *DtInputPortGroup* and *DtOutputPortGroup* classes loosely correspond to the old *DtPortInterface* class. They represent collections of either *DtInputPort* or *DtOutputPort* objects. Use of *DtInputPortGroup* and *DtOutputPortGroups* are optional. They simply provide an interface that makes it convenient to test or set the status of a group of ports.

Creating Ports

If you have created or extended components that generate data and send it through *DtPort* data members, replace them with *DtOutputPorts*. For example, if you have created a type of component that has a *DtIntegerPort* data member, replace it with a *DtIntegerOutputPort* (*integerIOPort.h*).

Similarly, if you have created a component that accepts data through its own *DtPortInterface* data member, replace it with the corresponding *DtInputPort* data members, and (optionally) a *DtInputPortGroup*. For example, the *DtAutomotiveActuator* used to have a *DtAutoActuatorPortInterface* data member. It now has the following data members instead:

```
DtAnalogInputPort* myThrottlePort;
DtAnalogInputPort* mySteeringPort;
DtAnalogInputPort* myBrakePort;
DtIntegerInputPort* myGearPort;
DtBooleanInputPort* myParkingBrakePort;
DtInputPortGroup* myAutomotiveControlPortGroup;
```

To create your new port data members, override createPorts() in your component. In your implementation of createPorts(), create your new port, choosing a name for the port that you know will be unique within the component. These names will be used in the parameter database to specify how the component will be connected to another component.

After you create your new port data member, add it to the appropriate list of ports in the component, or to a port group owned by the component. This will enable the port to be looked up and connected to another port by the component manager. Components maintain lists of *DtInputPorts* and *DtOutputPorts*, and *DtInputPortGroups* and *DtOutputPortGroups*. If your new port is not meant to be part of a port group, then just add it to the appropriate list of ports on the component, for example:

```
myTargetListPort = new DtEntityListOutputPort("target-list");
myTargetListPort->init();
addOutputPort(myTargetListPort);
```

If your new port is supposed to be part of a port group, then you need to add it to the port group, instead of the component's port list, e.g.

```
mySteeringPort = new DtAnalogInputPort("steering");
myAutomotiveControlPortGroup->addPort(mySteeringPort);
```

Add your port to the component's port list or to a port group, but not to both.

! Port groups are always created before ports, so you can always safely add ports to port groups in createPorts().

To create new port group members, override createPortGroups(). In your implementation, create the port group with a name you know will be unique within the component. The name will be used to configure the connection with another component in the object parameters database. After you instantiate the component, add it to the appropriate port group list in the component, for example:

```
myAutomotiveControlPortGroup =
  new DtInputPortGroup("automotive-control");
addInputPortGroup(myAutomotiveControlPortGroup);
```

Changes to Components that Send Data through Ports

You must make the following additional changes to components that send data to migrate your code:

- Replace calls to DtSimComponent::takeControlOfPortInterface() with DtOutputPort::attemptToActivate().

  Under the old architecture, components took control of a port interface before placing values on a port with a call to the component's takeControlOfPortInterface() member function. Replace these calls with a call to the *DtOutputPort*'s attemptToActivate() member function. This function indicates whether the port connection is active (that is, the recipient will receive data placed on the port – there is no component with a higher priority currently connected). These calls are typically made at the beginning of a component's tick, so it can determine whether or not to compute and set values on its output ports.

- Replace calls to DtSimComponent::giveUpControlOfPortInterface() with DtOutputPort::deactivate().

  In the old design, once a component was done providing data in its port, it relinquished the connection to the port interface with a call to giveUpControlOfPortInterface(). This call should be replaced with a call to the *DtOutputPorts*' deactivate() member function.

- Replace calls to DtPortInterface::setOwnerPriority(-1) with DtOutputPort::allowDeactivation().

  Relinquishing ownership of a port connection without disconnecting ('passive disconnect') was done under the old design by calling the setOwnerPriority(-1) function in the *DtPortInterface*. This had the effect of moving the currently connected port to the lowest possible priority, while still maintaining the connec-

tion. Replace these calls with a call to the *DtOutputPort*'s allowDeactivation() member function.

Calls to set values in the new *DtOutputPort* classes are identical with those in the old *DtPort* classes.

Changes to Components that Receive Data through Ports and Port Interfaces

You must make the following changes to components that receive data to migrate your code:

- Replace calls to test for non-NULL DtPort* with DtInputPort::activeConnection().

  Under the old architecture, a port was connected to a port interface by setting a pointer to it in the port interface. In the component's tick function, the pointer was checked to see if it was valid (that is, non-NULL). If it was a valid pointer, then the component would assume the connection to the port was valid and retrieve values from it. Replace these tests for non-NULL *DtPort* pointers with a call to the corresponding *DtInputPort*'s active() member function.

- Replace calls to DtPortInterface::dataReceived() with DtInputPort::dataReceived().

  Under the old architecture, the DtPortInterface::dataReceived() call was used to notify the *DtPort* that the data had been received. Replace this call with a call to the corresponding DtInputPort::dataReceived() member function.

  Calls to retrieve values in the new *DtInputPort* classes are identical with those in the old *DtPort* classes.

Updating Object Parameters Database Entries

If you have created or modified any entries in the object parameters database, you may need to modify those entries to be compliant with the new Port/Port Group scheme.

! We recommend that you back-up the object parameters database file before making changes.

For a given entity entry in the object parameters database, you do the following:

1. In the object parameters database file, remove all attached-to-list entries in the entity's component descriptors. These entries were used to indicate which port interface a given component should be connect to:

```
(move-to
  (component-descriptor-type "ground-move-to-descriptor")
  (component-type  "ground-auto-move-to-controller")
  (attached-to-list
    (powertrain
      (port-interface "automotive-input-port")
    )
```

```
  )
  (at-distance 1.0)
  (near-distance 25.0)
)
```

2. To specify how all of an entity's components are connected to each other, add a new connections entry to the entity's parameter entry. It will look something like the following:

```
(connections
  (connect joystick:automotive-control powertrain:automotive-control)
  (connect collision-avoidance:automotive-control
    powertrain:automotive-control)
  (connect move-to:automotive-control powertrain:automotive-control)
  . . .
)
```

   Each item in the connections entry specifies a connection between a given port or port group. See section 12.3.2, "Common Elements of Component Descriptors" in the *MÄK VR-Forces User's Guide* for details on the format of the connections entry.