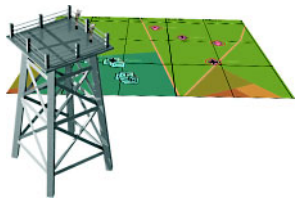

MÄK Plan View Display 2.6 Release Notes

This release note contains the following release-specific information for Plan View Display Release 2.6:

! MÄK Plan View Display 2.6 requires FLEXlm 9.2. You must update your license server. See the procedure in "FlexLm 9.2," on page 1-7.

Document ID: PVD-2.6-3-040212

Systems Supported

Plan View Display 2.6 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 Plan View Display 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 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.

For the IRIX version only, you must use VR-Link 3.9.1a.

Running the Plan View Display on SGI IRIX

If you want to run the Plan View Display on SGI IRIX, you must set the color depth to 24-bit.

Disk Space Requirements

A full installation of Plan View Display requires approximately 230 MB of disk space. Terrain databases use approximately 73 MB and documentation (primarily the class reference) uses 48 MB.

MÄK Product Compatibility

To build the Plan View Display, you must link with VR-Link 3.9.1 (3.9.1a for IRIX). If you are building for HLA and want to link with the MÄK RTI, use MÄK RTI 2.x-ngc.

MÄK Plan View Display 2.6 is a parallel release to MÄK VR-Forces 3.6. It is not compatible with previous releases of VR-Forces.

Qt Release Compatibility

If you want to do development using the Plan View Display API, you need to use Qt, a cross-platform GUI toolkit. The Plan View Display GUI was built with Qt release 3.1.2. A Plan View Display developer's license includes a Qt development license. MÄK provides you with a Qt license key. However, you must download the correct version of Qt from *www.trolltech.com*.

Third-Party Library Requirements

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

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

Plan View Display 2.6 was built against VR-Link 3.9.1 (3.9.1a for IRIX) and is compliant with:

- RPR-FOM 1.0 and a subset of 2.0 (draft 6)
- MÄK RTI 2.1
- DMSO RTI NG 4, and 6.

If you need to run the Linux version of the Plan View Display with the DMSO RTI, contact MÄK technical support.

Plan View Display 2.6 is compatible with applications that use earlier versions of VR-Link if they support versions of the RPR FOM listed above.

DIS only

Plan View Display 2.6 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

Plan View Display 2.6 code is not fully compatible with previous releases. See API changes described in "New Features and Updates," on page 1-6.

RPR FOM Versions Supported

Plan View Display 2.6 has built-in support for versions 1.0 and 2.0 (draft 6 and 17) of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, Plan View Display 2.6 uses RPR FOM 1.0.

If you are building the Plan View Display 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 *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));
```

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 Plan View Display has the following updates and new features:

- Support for the HLA 1516 API
- New directory structure
- Support for FLEXlm 9.2
- Support for RPR FOM version 2.0 draft 17
- Display of trajectory histories for entities
- Display of fire and detonation lines
- Display of range and bearing for point-to-point intervisibility lines
- Ability to scale bitmap icons
- Support for terrain profiles
- Changes to the GUI API
- Changes to the drawing system for paper maps
- Terrain API and *tdbtool* changes
- Miscellaneous API changes.

Support for the HLA 1516 Specification

The Plan View Display 2.6 supports the SISO Dynamic Link Compatibility HLA API Product Development Group (HLA API PDG, *http://www.sisostds.org/index.cfm*) draft 1516 API. This API, unlike the IEEE 1516 API, supports dynamic link compatibility between different RTI implementations.

New Directory Structure

Plan View Display 2.6 has two changes to its directory structure. The *binRPR* directory has been replaced with *./binHLA13* and *./binHLA1516*, for the executables for the HLA 1.3 specification and HLA 1516 specification.

Protocol-independent utilities, such as *tdbtool* and *imagesplitter*, are no longer duplicated in each of the protocol-specific bin directories. They are in the *./bin* directory.

FlexLm 9.2

The Plan View Display 2.6 requires Flexlm 9.2. If you have not already updated your FlexLm directory as part of installing another MÄK product, you must update the license server.

If this is the first time you are installing a MÄK product, just follow the steps for installing license management that are in *MÄK Plan View Display User's Guide*.

To update your *flexlm* directory:

1. If the license server is running, shut it down with *lmdown*.
2. Back up the files in the *flexlm* directory on the license server (*C:\flexlm* on Windows, typically */usr/local/flexlm* on UNIX). (For example, to *flexlm.old*.)
3. Copy all of the files in *pvdx.x/flexlm92* to the *flexlm* directory on the license server.
4. Copy your license files (*.lic* or *.dat*) from the backup directory to the *flexlm* directory.
5. Start the license server.
6. Run the Plan View Display and verify that it is checking out a license.

FlexLM is backwards compatible, so you do not have to change your license file, and MÄK applications built with earlier versions of FlexLM will continue to work with FlexLM 9.2. If you have any problems or questions, contact MÄK support at support@mak.com.

Terrain Profile Tool

Terrain Profile windows are available for line objects that have height (in other words, routes and lines, but not phase lines), for intervisibility lines, and for an entity's track history. A terrain profile displays (numbers correspond to callouts in Figure 1:

- Location data for the start point and end point of each line segment (1)
- The total length of the line or track (2)
- The length of the selected line segment (3)
- Heading of each point (4)
- Elevation of each point (5)
- Location of any point specified in the plot window (6)
- A graph of the line or track against the terrain (7)
- Entities within a specified distance of the line or track (8).

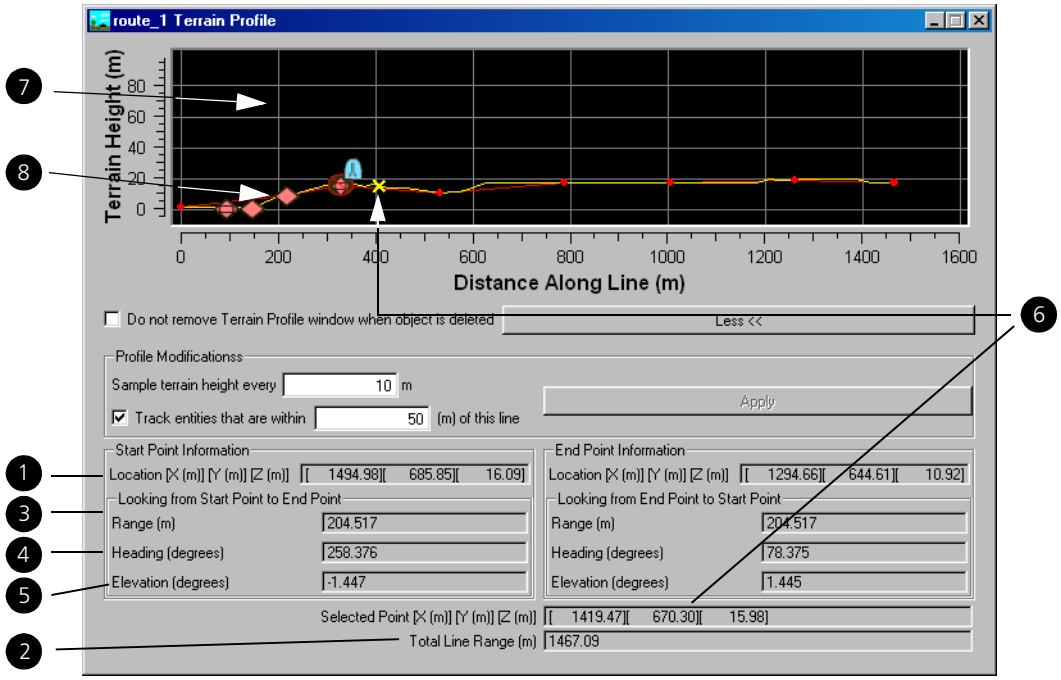

Figure 1. Terrain profile

You can specify the distance between the points used when the VR-Forces samples the terrain to create the graph. You can also specify that a terrain profile for an entity remain on screen even if the entity is removed from the simulation. This is most useful for viewing the terrain profile of short-lived entities, such as missiles.

If you are viewing a geodetic database, you can display distance as a straight line (which may cut through the curvature of the earth, or as a great circle, which takes into account the earth's curvature.

Trajectory Histories

You can display the path that an entity follows as it moves over the terrain. The track uses the force color. The Plan View Display remembers the track of entities that are visible on the map and displays them as follows:

- When you display track histories, the Plan View Display displays the track history for each visible entity starting at the point that it became visible; it does not start the display of tracks at the current entity location.
- If you change the map view to include entities that were not visible when you displayed track histories, the tracks for the newly visible entities start at the point at which they become visible, not at the point when track histories were first displayed.
- Turning off the display of track histories does not discard the track history data. If you display track histories again the entire track history is displayed.
- You can discard the stored histories. Choose Options → Clear Track History.

Track histories degrade over time. The track is a line constructed from points laid down every 10 meters. After 60 seconds, the first point is removed from the history. Each successive point is removed 60 seconds after it was created. You can configure the distance between points and the degradation time with the **PVD_minHistoryMovement** and **PVD_historyTrackingTimeout** MTL parameters in *pvd.mtl*.

To improve performance, by default, track histories do not degrade for entities that are not moving or are destroyed. If you want track histories for these entities to degrade, edit the **PVD_decayTrackHistory** parameter in *pvd.mtl*.

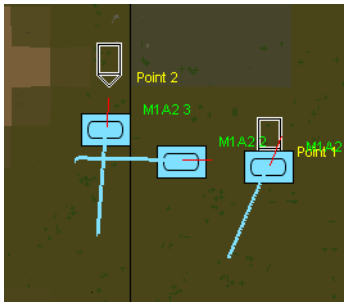

Figure 2. Entity track history

To display an entity's track history:

1. Choose Options → Display. The Display Options dialog box opens.
2. Select the Symbols tab.
3. To display the track history, in the Icon Labels group box, check Track History. To disable track histories, clear the check box.
4. Click Apply or OK.

Fire and Detonation Lines

When an entity fires a munition, the VR-Forces can display lines between the shooter and its target. Fire lines are dotted. Detonation lines are solid. The line uses the force color of the shooter.

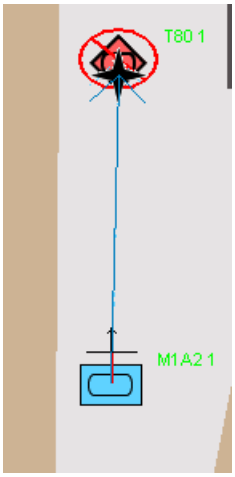

Figure 3. Detonation line

To display fire and detonation lines:

1. Choose Options → Display. The Display Options dialog box opens.
2. Select the Symbols tab.
3. To display the fire and detonation lines, in the Icon Labels group box, check Fire/Detonate Lines. To disable fire and detonation lines, clear the check box.
4. Click Apply or OK.

Scaling Bitmap Icons

In previous versions of the Plan View Display, only icons based on True Type fonts were affected by the Icon Scale slider. Now, bitmapped icons can be scaled.

Changes to the GUI API

The *DtCreateControlObjectHandler* appearance flag has been added as a parameter to the create and edit slots. Member functions that used to look like this:

```
virtual void create(QString name, QString label, QColor color,
  int forceType, bool publish, DtArchivableObject* userData);
virtual void completeEdit(QString name, QString label, QColor color,
  int forceType, bool publish, DtArchivableObject* userData);
```

Now look like this:

```
virtual void create(QString name, QString label, QColor color,
  int forceType, int appearance, bool publish,
  DtArchivableObject* userData);
virtual void completeEdit(QString name, QString label, QColor color,
  int forceType, int appearance, bool publish,
  DtArchivableObject* userData);
```

Bolded code indicates the changes made for this release.

The *DtLineConfigWidget* and *DtPointConfigWidget* classes have been changed to support passing of the appearance flag. Member functions that looked like this:

```
virtual void createActivated(QString name, QString label, QColor color,
  int forceType, bool publish, DtArchivableObject* userData);
virtual void editActivated(QString name, QString label, QColor color,
  int forceType, bool publish, DtArchivableObject* userData);
```

Now look like this:

```
virtual void createActivated(QString name, QString label, QColor color,
  int forceType, int appearance, bool publish,
  DtArchivableObject* userData);

virtual void editActivated(QString name, QString label, QColor color,
  int forceType, int appearance, bool publish,
  DtArchivableObject* userData);
```

The overlay publisher handler has also been changed to support the passing of the appearance flag. Member functions that looked like this:

```
virtual void create(const DtOverlaySymbolDescriptor* desc, QString name,
  QString label, QColor color, QPtrList<Dt3DPoint>& pnts,
  const DtEntityMapperKey& key, int force, bool publish,
  DtArchivableObject* userData);

virtual void edit(QString name, QString label, QColor color,
  QPtrList<Dt3DPoint>& pnts, DtModelData* data, int force, bool publish,
  DtArchivableObject* userData);
```

Now look like this:

```
virtual void create(const DtOverlaySymbolDescriptor* desc, QString name,
  QString label, QColor color, QPtrList<DtPoint>& pnts,
  const DtEntityMapperKey& key, int force, int appearance, bool publish,
  DtArchivableObject* userData);
```

```
virtual void edit(QString name, QString label, QColor color,
  QPtrList<DtPoint>& pnts, DtModelData* data, int force, int appearance,
  bool publish, DtArchivableObject* userData);
```

Changes to how Paper Maps are Drawn

The API for drawing paper maps has changed. To improve performance, all items that used to be treated as QImages are now treated internally as QImages only if the image being loaded from disk has an alpha channel associated with it. If the image has only a black and white mask or no mask at all, it is treated as a QPixmap.

Member functions for returning images to be drawn are no longer called getScaled-Pixmap() or getScaledImage(). They are called getScaledImage() and either a QPixmap or QImage is passed to each member function to return the image to draw. You can test to see if the particular papermap supports the drawing as an image by calling the drawAsImage() member function.

Terrain API and tdbtool Changes

The Terrain API and tdbtool have the following new features:

- You can clip vector data to the terrain extent and planarize the vector data
- The tdbtool has new options
- *DtTerrainDatabase* has new and changed functions
- The Plan View Display ships all libraries or DLLs required to read OpenFlight files
- The geometryNative library has been deprecated
- Logging and error checking in the c7b loader has been improved
- New class: *DtVectorNetworkIntersectionRecordList*.

Clipping Vector Data to the Terrain Extent

When you import any vector feature data, such as shape, DFD, and DFAD file data, you can clip the vector data to the terrain extent, with options in the map file or using the *tdbtool*.

Clipping the vector data clips all vector features to the terrain to reduce the size of the database as well as remove data that is unwanted.

Area features may be clipped incorrectly if more than one vertex in a row lies outside the clipping region. This will be fixed in an upcoming release. Clipping defaults to true in the map files, so if not specified, it occurs automatically.

Planarizing Vector Data

When you import vector data, you can planarize it with options in the map file or using *tdbtool*. Planarizing flattens the vector data and removes intersections of edges. It may take an extremely large amount of computation time on large vector datasets. Therefore, be careful about planarizing your datasets. Since most vector data other than DFD and DFAD files, such as shape files, is usually well formed, and because of processing and memory concerns, planarizing is not performed by default.

tdbtool Options

tdbtool now has the following options for planarizing vector data and clipping vector data to the terrain extent.

- `-p` *`tolerance`*`[, ` *`excessive_edge_count`*`]` planarizes vector data, where:
  - *`tolerance`* specifies the required tolerance parameter (0.0001 is usually a good default). The tolerance is used to determine if points are close enough to be considered the same.
  - *`excessive_edge_count`* stops processing when more than a certain number of edges are generated.
- `-v` clips vector data to the terrain.

New Map File Options

The following parameters are now supported in ShapeVectorMetafileRecords, DfdDfadMetafileRecords, and DfadMetafileRecords, or any other vector data metafile records:

- **Clip {true | false}** – Specifies whether or not to clip vector data to the extents of the polygonal terrain. Default: true.
- **Planarize {true | false}** – Specifies whether or not to planarize vector data. A planar network is constructed from the vector data and all edge intersections are detected and eliminated. This is an expensive operation and is usually useful only with DFAD and DFD data. Requires the **Tolerance** parameter and optionally takes the **ExcessEdgeCount** parameter. Default: false.
- **Tolerance tolerance** – Specifies a tolerance to use when deciding if edges intersect one another when planarizing data. If **Planarize** is true, **Tolerance** is required; otherwise, it is meaningless. It is a positive real number. Default: 0.0001.
- **ExcessEdgeCount count** – Specifies a threshold number of edges, as a positive integer, beyond which planarizing stops. Optional if **Planarize** is true; otherwise, meaningless. Default: 50000.

DtTerrainDatabase Functions

DtTerrainDatabase has the following new and changed member functions:

- terrainHeightAndSoilType() – This group of functions provides the height of the terrain at a specified location, either in local (database) coordinates or in latitude and longitude. If there is a successful terrain query at the specified location, the soil type of the polygon at the location is also returned.
- closestIntersection() – This group of functions returns the closest intersection to the specified location. Whereas the terrainHeight() functions return the topmost intersection, these functions return the intersection that is closest to the specified location, not necessarily the topmost. It is useful for positioning entities inside areas of the terrain that contain multiple levels, such as inside buildings, or in tunnels under the ground.
- intersectVectorNetwork(), intersectChordsWithVectorNetwork(), intersectLocationsOnVectorNetwork(), and allIntersectionsWithVectorNetworkAlongChordList() – These functions calculate intersections with the vector network according to the specified parameters. They return false if the vector network is empty or if any of the user inputs are NULL.
- intersectList() and allIntersectsAlongChord() – These functions now return a bool representing whether any intersection occurred or not. They take parameters that are set to the number of chords with intersections, or the number of intersections along the chord, respectively.

For details see the *DtTerrainDatabase* class description.

The geometryNative Library has been Deprecated

To simplify and improve the quality of the Terrain API, the *Dt3DPoint* and *Dt3DExtent* classes have been merged with the *DtPoint* and *DtExtent* classes, respectively. The *DtPoint* class now represents both the deprecated *Dt3DPoint* as well as the original *DtPoint* class. The same is true for the *DtExtent* class. The interface for each class is consistent with the original interfaces provided - the only change is the removal of the string() function from the *DtPoint* class. As a result, the *geometryNative* library has been deprecated and contents merged with the appropriate classes in the *geometry* library.

In order to maintain backwards compatibility, the header files *3DPoint.h* and *3DExtent.h* are still distributed with the product. However, they are now empty (do not declare any classes) and they produce a warning when included in a project. They include *point.h* and *extent.h*, respectively, but any references should still be updated appropriately. Instances of *3DPoint.h* or *3DExtent.h* in your files should be changed to be *point.h* or *extent.h* to prevent compilation problems in the future.

Additionally, *Dt3DPoint* and *Dt3DExtent* have been typedefined to be types of *DtPoint* and *DtExtent* inside of *point.h* and *extent.h*. As a result, all references to *Dt3DPoint* and *Dt3DExtent* should compile, but you should update references to these types as soon as possible to prevent problems with future releases.

You should update any references to the *geometryNative* library in any project files or makefiles to reference the *geometry* library instead.

Miscellaneous API Changes

- Corrected a spelling error in *shapeUtil.h* - caculatePolygonExtent() is now calculatePolygonExtent().

Documentation Updates

- If you create a DtSpecification for an area vector feature, that is, the specification has a FeatureType = 2 attribute, then the specification must also contain the following attributes:
  - DtXMin
  - DtXMax
  - DtYMin
  - DtYMax
  - DtZMin
  - DtZMax.

  Each of these attributes is a double. They represent the bounding box of the area feature. If these attributes are not present, the Plan View Display uses 0.0, which may result in unintended consequences.
- Chapter 9, Working with Terrain Databases, in *Plan View Display User's Guide*, has a new section that describes the terrain databases supplied with the Plan View Display.

Known Problems

Plan View Display Release 2.6 has the following known problems:

- True Type fonts are not supported on SGI IRIX. Use bitmaps for entity icons.
- If you are using the Plan View Display 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.
- 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.
- If you click the New Folder button in any of the file selection dialog boxes, the GUI may take a very long time to respond (it may appear to be hung). If this happens, shut down and restart the application.

- The Qt translation files (in *./translations*) for built-in Qt-level GUI items do not work properly. The translation files for the Plan View Display work correctly.
- 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.
- If you uninstall a Plan View Display 2.5.1 or earlier using the InstallShield uninstaller, the entity icon fonts get uninstalled. If you have another application that needs them, such as MÄK VR-Forces, you must manually install them. This is not a problem for Plan View Display 2.6 or later.
- The Entity In Area dialog box lists 1 Force and 2 Force as possible choices. These are valid only if you check Any Entity.