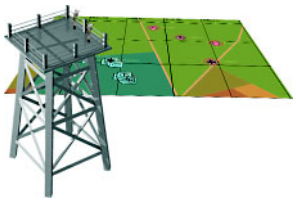

MÄK Plan View Display 2.7 Release Notes

This release note contains the following release-specific information for Plan View Display Release 2.7:

Document ID: PVD-2.7-3-040714

Systems Supported

Plan View Display 2.7 is available for the platforms listed in Table 1. For the availability of other platforms, contact your MÄK salesperson. For toolkit users, application code must be built with the indicated compilers in order to link to Plan View Display libraries.

Table 1: Platforms supported

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

Patching the Microsoft Visual C++ 6.0 Compiler

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

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

To install the patch:

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

Building on Linux or IRIX

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

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

Running 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.

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

Disk Space Requirements

A full installation of Plan View Display requires approximately 300 MB of disk space. Terrain databases use approximately 106 MB and documentation (primarily the class reference) uses 68 MB.

MÄK Product Compatibility

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

MÄK Plan View Display 2.7 is a parallel release to MÄK VR-Forces 3.7. 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 3.3.1. 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. Set the MAK_QWTDIR environment variable to point to the location of the QWT library.

Network Compatibility

HLA only

Plan View Display 2.7 was built against VR-Link 3.9.1 and is compliant with:

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

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

Plan View Display 2.7 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 2.7 supports DIS 2.0.4, IEEE 1278.1, 2.1.4, and IEEE 1278.1a, and can therefore interoperate with DIS applications of any of these versions.

Backward Compatibility

Plan View Display 2.7 code is not fully compatible with previous releases. See API changes described in "New Features and Updates," on page 7.

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

RPR FOM Versions Supported

Plan View Display 2.7 has built-in support for versions 1.0 and 2.0 (draft 6 and 17) of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, Plan View Display 2.7 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 Plan View Display 2.7 relies on the LgrControl and ViewControl custom MÄK extensions.

New Features and Updates

This release of MÄK Plan View Display has the following updates and new features:

- New and updated interface features
  - Ability to load symbol map files at runtime
  - Animated, representational, entity icons
  - Ability to select features and feature segments
  - Ability to create graphical objects based on feature data
  - Ability to show directional arrows on lines
  - Ability to print the map image or save it to a file
  - Reorganized entity information dialog box
  - Continuous navigation when pressing Navigation toolbar buttons
  - Reorganization of the main menu
- New special effects:
  - Display of electromagnetic emissions
  - Animation of munitions fire and detonation
  - Sound effects
- New Show Missile Impact example
- New startup option (load bounding volume configuration file at startup)
- API Changes
  - Changes to the Terrain API
  - PVD API changes
  - New build system to support customization of examples
- New online help browser (see "Documentation Updates," on page 19.)

Loading Symbol Map Files at Runtime

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

Animated Entity Icons

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

Helicopter Dismounted infantry M1A2 tank

Figure 1. Animated icon images

Selecting Features

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

Among the databases provided with the Plan View Display, the *makland* and *berkeley* databases contain feature data.

See *MÄK Plan View Display User's Guide* for details.

Creating Objects from Features

You can create routes from line features and points from point features. When you create a route from a line feature, the vertices of the route are placed at the intersection of each feature segment. You can create a point object at the location of a point feature.

See *MÄK Plan View Display User's Guide* for details.

Printing the Map Image or Saving it to A File

The Plan View Display supports the following new options for capturing images of the map area:

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

See *MÄK Plan View Display User's Guide* for details.

Reorganized Entity Information Dialog Box

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

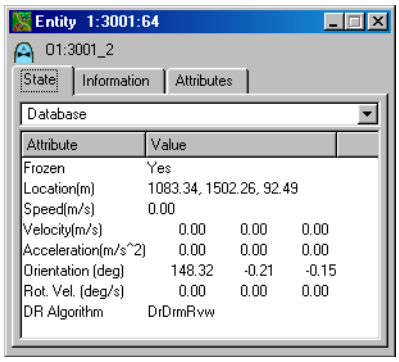

Figure 2. Entity Information dialog box

Continuous Navigation When Pressing Navigation Toolbar Buttons

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

Reorganization of the Main Menu

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

Display of Directional Arrows on Lines

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

To display a route's arrow:

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

New Special Effects

The Plan View Display supports the following special effects:

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

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

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

See "Using Special Effects," in *MÄK Plan View Display User's Guide* for details.

Show Missile Impact Example

The Show Missile Impact example (*./examples/showMissileImpact*) shows how to add new symbols and model data to the GUI.

Loading An Alternative Bounding Volume Configuration File

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

```
Pvd -b file
```

Changes to the Terrain API

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

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

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

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

Deprecated Terrain API Classes

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

Updates to the *DtVectorNetwork*

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

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

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

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

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

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

Clarification of Vector Network Intersection

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

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

Updates to Metrics Used to Calculate Intersections

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

Updates to the Display of Vector Point Features

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

PVD API Changes

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

Ability to Select Terrain Feature Objects

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

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

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

The Entity Information Dialog Boxes have been Redesigned

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

- *DtEntityInfoDialog*
  - *DtEntityStateInfoPage* – displays information about speed, location, velocity, and so on.
  - *DtBaseEntityInfoPage* – displays entity ID, force ID, entity type and guise.
  - *DtEntityAttributePage* – displays information about entity attributes.
- *DtAggregateInfoDialog*
  - *DtEntityStateInfoPage* – displays information about speed, location, velocity, and so on.
  - *DtAggregateInfoPage* – displays information about number of subordinates, dimensions, and so on.
  - *DtAggregateSubordinateInfoPage* – displays information about subordinates of this aggregate.

Migrating Code to the New Entity Information API

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

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

Table 2: Entity Information dialog box objects and member functions

Dialog Box Class	Object	Member Function
DtEntityInfoDialog	DtEntityStateInfoPage	createEntityStatePage()
	DtBaseEntityInfoPage	createEntityInfoPage()
	DtEntityAttributePage	createEntityAttributePage()
DtAggregateInfoDialog	DtEntityStateInfoPage	createEntityStatePage()
	DtAggregateInfoPage	createAggregateInfoPage()
	DtAggregateSubordi-nateInfoPage	createAggregateSubordi-nateInfoPage()

3. In your subclass of the page, override initPageContents() to add your items to the display. To add an item, call the addInfoItem() member function, providing a unique ID and the title of the item. The item is added to the list box that is displayed on the page. The first column displays the title; the second displays the value of the item.
4. To update the value of the item you added, override the updatePageContents() member function. The model key and the state repository of the item being updated are passed into this member function. To update a particular item, call infoItem() to retrieve the QListViewItem created when you added your item. Then pass the unique ID you used when creating the item. Once you have the item, set its value by calling the Qt member function setText(1, *value*) on it.

Adding A New Page to An Information Dialog Box

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

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

The init() member function should look like this:

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

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

  return myInfoPage;
}

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

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

  return initPageContents();
}
```

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

Creating Graphical Objects

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

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

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

Other Changes to the PVD API

The following changes have been made to other objects:

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

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

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

New Build System

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

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

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

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

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

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

To generate a new project:

1. Open up a DOS command window and change your directory to the *./scripts* directory (for example, *C:\MAK\PVD2.7\SCRIPTS*)
2. Add your new source file names to *files-cc.txt*, your new header file names to *files-h.txt* and your new Qt UI files to *files-ui.txt.*

3. Run the *buildProject* program. The syntax for buildProject is:

   ```
   buildProject source_directory project_name
   ```

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

   ```
   buildProject examples scenarioStatistics
   ```

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

   ```
   buildProject examples\condition conditionGui
   ```

   The program creates six files – separate debug and release projects for DIS, HLA13 and HLA1516 projects. The projects named **D.dsp* (or **D.vcproj* if you are using .NET) are debug projects and only the debug configuration of that file is valid. The **R.dsp* (or **R.vcproj* if you are using .NET) are release projects and only the release configuration of that file is valid.

4. Create a new workspace.

5. Add the project files to the workspace that you want to build.

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

Documentation Updates

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

- Online help is now displayed in the Qt Assistant help browser, not the default web browser for your computer. The new browser opens more quickly and supports full text search, bookmarks, and the ability to increase the size of the text displayed in the window.
- The PDF version of *MÄK Plan View Display User's Guide* provided with this release has been updated to describe all new user features, the changes to *tdbtool* and map file configuration, changes to the PVD API, and changes to the Terrain API.
- If you create a *DtSpecification* for an areal vector feature, that is, the specification has a FeatureType = 2 attribute, then the specification must also contain the following attributes:
  - DtXMin
  - DtXMax
  - DtYMin
  - DtYMax
  - DtZMin
  - DtZMax.

  Each of these attributes is a double. They represent the extent of the areal feature. If these attributes are not present, 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.
- Importation of OpenFlight terrain databases has the following restrictions:

  The following nodes are not imported or displayed:
  - Mesh Nodes
  - Switch Nodes
  - Light Points
  - Light Point System
  - DOF Nodes
  - Text Nodes
  - Sound.

The following featrues are not imported:

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

The following face attributes are not supported:

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

Bug Fixes

This release fixes the following problems:

- A bug in vector data clipping has been fixed. It's highly recommended that terrains that were clipped, planarized, or both, be reimported from their original format.
- Shapefiles containing elevation values for the data are now correctly imported.
- A fix has been added for incorrectly generated Soil Types. It will not catch all types of soil type issues, and may incorrectly map them, so regeneration from original data is highly recommended.
- OpenFlight files containing instances will now load correctly. Some OpenFlight files contained instances that were not properly interpreted. This has been fixed.
- A bug in the `-k` option for *tbdTool* has been fixed. Failure to load properly will also be reported.
- Paged GDB files are now imported and loaded correctly. Any paged Open Flight files imported in Plan View Display 2.5.1 or 2.6 should be regenerated from source data as they were incorrectly created.
- The AddSimpleFeatures example now uses the new Terrain API properly and has the proper specification attribute values.
- The terrainHeight() function has been fixed to no longer return an empty intersection record.
- When intersecting the vector network, areal features were not intersected properly. This has been fixed and will improve correctness of selection process as well as performance.
- *DtNetworkEdges* were not being clipped properly, but only for linear features. Previously, this would create a degenerate edge. This has been fixed, although it is highly recommended that the GDB files be recreated from source data.
- Clicking the New Folder button in any of the file selection dialog boxes no longer causes the GUI to appear to be hung.
- Selected entities now show up correctly when moving from off screen on to the display. Previously, a map zoom or pan event was required to get them to appear when moving back onto the display.
- Some older GDB files that contained spatial indexing would not render correctly. These files now load correctly.
- Some older GDB files with spatial indexes would not load. These files will now load, however it is recommended that customers re-import older databases. The command to use with tdbtool to remove spatial indexing is:

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

  To regenerate a database, use:

  ```
  tdbtool -s x,y,z
  ```

Known Problems

Plan View Display Release 2.7 has the following known problems:

- 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.
- 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.
- Intervisibility works with linear and areal features, but not with point features.