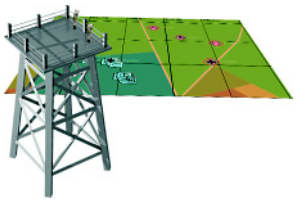

MÄK Plan View Display 2.8 Release Notes

This release note contains the following release-specific information for Plan View Display Release 2.8:

Document ID: PVD-2.8-3-050317

Systems Supported

Plan View Display 2.8 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
Red Hat Linux 9.0	GNU C++ (GCC) 3.2.2
Windows 2000 SP2, XP	Microsoft Visual C++ 6.0, Service Pack 5 Microsoft .NET 7.1

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

Patching the Microsoft Visual C++ 6.0 Compiler

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

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 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.

Compiler Compatibility on Windows

MÄK provides versions of product releases that have been compiled with Microsoft Visual C++ 6.0 and 7.1. When you run MÄK products together, for example, the VR-Forces and the Stealth, we strongly recommend that you run versions compiled with the same compiler. Mixing products compiled with different versions of the compiler can result in program instability.

MÄK Product Compatibility

To build the Plan View Display, you must link with VR-Link 3.9.3. The Plan View Display does not include a VR-Link developer's license. You must purchase it separately. If you are building for HLA and want to link with the MÄK RTI, use MÄK RTI 2.x.

MÄK Plan View Display 2.8 is a parallel release to VR-Forces 3.8. 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 from Trolltech. The Plan View Display GUI was built with Qt release 3.3.1. The Plan View Display does not include a Qt license. You must purchase a license from Trolltech.

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 2.8 was built against VR-Link 3.9.3 and is compliant with:

- RPR-FOM 1.0 and a subset of 2.0 (draft 6 and 17)
- MÄK RTI 2.3.3
- 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 2.8 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 2.8 supports DIS 2.0.4, IEEE 1278.1, 2.1.4, and IEEE 1278.1a, and can therefore interoperate with DIS applications of any of these versions.

Backwards Compatibility

In this release, the terrain file format has been incremented.

The FED file has a new entry to support remote control of the GUI.

If you rebuild the GUI, you must link to the following VR-Link libraries::

- *loggerControl{5, HLA13RT, HLA1516RT}.lib*
- *stealthControl{5, HLA13RT, HLA1516RT}.lib*

RPR FOM Versions Supported

Plan View Display 2.8 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 2.8 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 *DtRprFom-Mapper* 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 2.8 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 GUI features
  - Line-of-sight filtering
  - Map view mirroring
  - Freehand lines
  - Adjustable line width
  - Display of transmitter and signal PDUs
  - Range rings
  - Object transparency
  - Displaying entities as dots instead of using icons
  - Papermap transparency
  - Stealth Navigation toolbar
  - Environmental List and Environmental Information dialog boxes.
  - Fire and detonate lines can be configured for display independently
  - The display of fire and detonate interactions can enabled and disabled
  - Display of hazard clouds
  - Shadow text
- Configuration and startup changes
  - Configuration file changes and new MTL parameters
  - New startup options
  - Ability to change or hide the background font character
- Terrain database changes
  - Tdbtool updates
  - Support for Flat Earth
  - Ability to set RPF resolution
  - Updated Image Splitter utility
- GUI API changes
  - Remote control
  - Classes for transmitter display
  - Classes for transparency.
  - Symbol mapper updates
  - Appearance attributes
  - Removed classes
  - Changes to FED file for HLA to support view controls

- API changes
  - GeometryNative classes removed
  - New examples.

All new GUI and configuration features are fully documented in *MÄK Plan View Display User's Guide*.

Configuration and Startup Changes

This section describes new startup options and changes to configuration files.

New Startup Options

- `-F` *`ttl`* – specifies the mcastTtl value of the exercise connection socket. (DIS only)
- The `-R` startup option lets you load a script file at startup.

  `-R` *`script_file`*

  VR-Forces includes the following MTL commands that the GUI can respond to:

  - (`opendatabase` *`database_name`*) – Opens and displays the specified database as if the user selected File/Open Database and chose a database to open.
  - (`loadoverlay` *`overlay_file`*) – Loads an overlay file as if the user selected File/Load Overlays.
  - (`setlocalextent` (`list` *`x y z`*) (`list` *`x y z`*)) – Sets the front end extent (in local coordinates) to the specified North, East, and Southwest coordinates (as if you zoomed into a particular area.)

These commands are only supported locally and are not sent out over the network.

Configuration File Changes

The Plan View Display has some new configuration files and has made some changes to existing files, as follows:

- The character column in *symbolmap-uttf.mtl* file that appears after the font name has been removed.
- *capabilities.mtl* – stores settings for entity capabilities.
- *appearance.mtl* – stores settings for entity appearance attributes.
- *rangeRings.mtl* – configures threat rings for entities.

New configuration Parameters

The Plan View Display has the following new parameters:

- PVD_viewControlMessages – in *pvd.mtl,* enables or disables view control messages. See "Reflection of the Map View," on page 11.
- PVD_storeLineValuesMode – in *pvd.mtl*, specifies whether or not to save the settings for newly created lines to apply to the next new line. See "Storing Line Settings," on page 11.
- PVD_defaultFreehandSamplingRate – in *pvd.mtl*, specifies the default number of points to sample when drawing a freehand mode line. All points are recorded until the object is created, at which point only the sampled amount of points are saved.
- PVD_maximumSignalTimeDifference – in *pvd.mtl*, specifies the number of seconds between when a signal is sent on a certain transmitter frequency and another is sent on the same frequency to try and determine which entities might be responding to an originating signal.
- PVD_hideOffScreenSymbols – in *pvd.mtl*, specifies whether or not to hide symbols when they go off screen instead of destroying them. Setting this parameter to 1 may help with performance for large exercises.

Changing or Hiding the Background Character

You can now change or hide the character used to create the background color of symbols based on TrueType fonts. When you use TrueType fonts for entity symbols or any other symbols configured for the GUI, the background color, for example, blue or salmon, is determined by a character specified in the configuration file. By default, this is the @ character. (If you have ever run VR-Forces without installing the TrueType fonts, you may have seen @ characters displayed instead of the expected entity icons.)

The new configuration option lets you specify a different background character or eliminate it altogether. To specify an alternative background character, add it, in double quotes, to the end of the symbol's entry in the *symbolmap-ttf.mtl* file (or any other configuration file that specifies a "ttf" character. To specify no background, add empty double quotes, as follows:

```
(symbol-map (list  0 -1  -1  3 -1 -1 -1 -1) "ttf" "MapSym NK Sea" "0"
  "")   ;;   Clears out background character
```

This configuration option is optional. You do not have to update existing symbol map files if you do not need to use it.

Terrain Database Changes

The tdbtool has the following new options:

- Databases can now be converted to flat earth using the `-t` option.
- The origin of the database can now be manipulated by using `-setorigin` or `-moveorigin` and specifying the new origin with `-lat <D:M:S>`, `-lon <D:M:S>`, and `-zone <1 - 60>`.
  - `-setorigin` only moves the origin of the database. It does not reproject the database to the new origin (so, basically, moves the local origin.)
  - `-moveorigin` moves the database to the new origin location, changing the coordinates within the database (this is useful for moving a database from one zone to another.)
- New options have been added for specifying the expected number of vertices, surfaces, specifications expected in an imported terrain (any format). This can result in significant memory and performance savings for very large terrain files. The options are:
  - `-numvertices` *`expected_number_of_vertices`*
  - `-numsurfaces` *`expected_number_of_surfaces`*
  - `-numspecs` *`expected_number_of_specifications`*

  Example:

  ```
  tdbtool –numvertices 10000000 –o .\example.gdb
    .\example.flt
  ```
- The makland terrain database used in the maklanddemo scenario now uses a database that does not have polygonal features (buildings, trees, light poles). The result is that intervisibility will not recognize features unless you turn on feature intervisibility in the Intervisibility Options dialog box. Furthermore, you must increase the range of interest for intervsibility to recognize buildings. When computing line-of-sight for entity fire, the back-end may not adequately recognize features.

Flat Earth Support

Flat earth flight files can now be loaded up in the front-end and simulated in the back-end. Previous versions of the VR-Forces automatically converted flat earth databases to UTM. They are no longer converted, but are supported as flat earth.

RPF Resolution Keyword Addition

Chapter 2, *Working With Terrain Databases*, in *MÄK VR-Forces Configuration Guide* has added documentation for RPF paper map records. These record types have a new keyword that lets you specify the resolution to use. The default values are “1:1M” for CADRG and “5M” for CIB. The syntax is:

```
Resolution "resolution"
```

New GUI Features

This section describes new features in the GUI display.

LOS Filter

The line-of-sight filter allows you to view entity line-of-sight within the boundaries of the filter. You can filter by force type and color the lines by force type.

Reflection of the Map View

You can control the map view of other instances of the MÄK Plan View Display and VR-Forces with the view control feature. You can set any individual instance of these applications to ignore view control messages. You can also enable or disable view control messages with the **PVD_viewControlMessages** MTL variable in *pvd.mtl*.

Freehand drawing

You can now draw freehand lines on overlays. Once you draw a freehand line, it behaves like all other lines.

When you create a freehand line, you can specify the number of points to create along the line.

Once a line has been down-sampled, the points removed during the down-sampling cannot be recovered.

Storing Line Settings

A new configuration parameter, **PVD_storeLineValuesMode**, lets you specify that new lines get created with the same color, width, sampling, and force settings as the previous line that was created. This variable has the following options:

- 0 - Do not store any values
- 1 - Store for freehand lines only
- 2 - Store for all lines.

This variable only applies to line-based overlay objects and is only valid for the current session.

Specifying Line Width

You can now set the width of individual lines. If no width is specified, the default width is used for the particular object type.

For network representation, the high four bits of the appearance of the object is used to represent this width.

Transmitter and Signal Support

The VR-Forces now supports display of reflected transmitters (using the VR-Link *DtReflectedRadioTransmitterList*) and captures signal PDUs/interactions to associate signals with the transmitters that might be sending them. The Display Options dialog box has options to display transmitters, display communication lines, and allow for transmitter selection.

You can view information about transmitters by looking at the Transmitters tab on the Entity Information dialog box. The Transmitter Information dialog box shows detailed information about a specific transmitter and any signals that might have been sent by that transmitter.

The transmitter settings are stored in the *settings.mtl* file, so, if you migrate to a new version of the software, you can copy the *settings.mtl* file to the new version to keep those frequency settings.

A new configuration parameter, **PVD_maximumSignalTimeDifference**, allows you to configure the number of seconds between when a signal is sent on a certain transmitter frequency and another is sent on the same frequency to try and determine which entities might be responding to an originating signal.

Threat Circles (Range Rings)

Threat circles show the potential threat radius of an entity. The threat radius is defined in *rangeRings.mtl* and lets you define multiple threat (range) rings for the weapons an entity might have associated with it. The default is to show a ring with cross hairs to more easily show which entity is associated with which range ring.

Range rings have the following display options:

- Drawing of transparent circles (this mode will significantly degrade the performance of the GUI.)
- Displaying of multiple rings leading to the outer ring (in the case of a very large range area, this can help to tie a ring to an entity)
- Displaying of range (distance) labels for the outer and inner range rings.

On small databases, an entity's range may be beyond the edge of the database display, in which case, you will not see it unless you zoom out.

Object Transparency

All overlay objects can now have transparency associated with them. You apply transparency using a slider on the More tab of the Create/Edit object dialog box. You can set transparency from 0 (opaque) to 100 (fully transparent). Transparency does not affect points and highlighting.

Replacing Entity Icons with Generic Symbols

You can replace entity icons force colored dots. This may increase performance. You can also show/hide entity symbols on a per-entity basis or for a selected group of entities.

Papermap Transparency

A new slider has been added to the Raster Map layers dialog box to make setting of transparency levels easier. Select the layer and move the slider to change the transparency level. If the Apply Changes Immediately check box is checked, any change on this dialog box take effect immediately.

Stealth Navigation Toolbar

The new Stealth Navigation toolbar lets you control the MÄK Stealth eyepoint and attachment modes, if there is a broadcasting Stealth available. The toolbar sets the mode of the selected Stealth or, if there is only one Stealth broadcasting, that Stealth.

If the Stealth Information dialog box has focus, you can use the keypad to move the Stealth in the same way as by using the keypad in the Stealth.

The Stealth Navigation toolbar requires the use of the MÄK Stealth 5.3 or higher.

Displaying Information About Environmental Objects

The new Environmentals List dialog box lists environmental objects (routes, waypoints, overlays, hazard clouds) that are published on the network. You can show all environmental objects or filter by a particular type.

You can also display information about individual environmental objects.

Fire/Detonate Option Separation

You can now configure the display of fire and detonate lines independently in the Display Options dialog box. The Options menu still has one control for both items. The menu option overrides the individual settings in the Display Options dialog box.

Enabling and Disabling the Display of Fire and Detonate Interactions

The Symbols tab on the Display Options dialog box now lets you enable and disable the display of fire and detonate interaction graphics. If these graphics are disabled, fire and detonation lines are also disabled.

Hazard Cloud Display

The GUI now displays hazard clouds that are generated using a protocol developed by ITT. The clouds can either be shown as ellipses with a radius (for each puff associated with a cloud) or as a bounding volume that encompasses the entire cloud. You can display an image instead of a colored ellipse; however, displaying images will dramatically decrease performance.

The Hazard Options dialog box lets you configure the display of hazard clouds. These settings are saved to the *settings.mtl* file.

You can display information about a cloud as a whole and each individual puff that makes up a cloud. Since puffs can be densely packed, only one puff can be selected at any given time.

Shadow Text

A new display option has been added to allow shadowing of text for greater label clarity on symbols.

API Changes

This section describes changes to the VR-Forces APIs.

PVD API

PVD Remote Control

This section describes the classes that support remote control (for example, loading a database) of Plan View Displays or VR-Forces front-ends by sending MTL commands.

PVD Remote Controller

The *DtPvdRemoteController* class constructs and sends MTL commands over the network to be received by other Plan View Displays or VR-Forces front-ends, for example, Open Database. This class uses the *DtMtlNetCommandDispatcher* class to send and receive MTL style commands that can be parsed and executed by the *DtMtlNetParser*. It wraps the *DtMtlNetParser* class, shielding the caller from having to know the syntax of the underlying MTL commands. It also provides a convenient place to collect all of your MTL commands. *DtPvdRemoteController* implements the Open Database and Reflect commands. The chat example demonstrates how to extend the PVD controller to add a new MTL command.

DtMtlVariable

The *DtMtlVariable* class knows how to create and parse itself for all of the intrinsic MTL types (list, string, fixed and float). It has convenience member functions for adding objects (such as a vector) to a command argument. *DtMtlVariables* are used by *DtMtlCommands* to construct an MTL string that can be parsed locally or sent over the network for parsing by remote Plan View Display and VR-Forces GUIs. Typically, you will create *DtMtlVariable* objects when you are creating a list. To add a variable of a specific type, you use the appropriate *DtMtlCommand* addVariable() member functions, described in this section.

The *DtMtlVariable* object is a list of *DtMtlVariable* objects.

To create and add a list MTL variable to a *DtMtlCommand*:

1. Create a new *DtMtlVariable* object.
2. Call addElement() on this object to add the list elements.
3. Add the *DtMtlVariable* object to the desired *DtMtlCommand* object by passing it into the *DtMtlCommand* through its addVariable() member.

While the *DtMtlVariable* must be allocated by the caller, once it is added to a *DtMtlCommand* object (through the addVariable() call), the *DtMtlCommand* object assumes responsibility for deleting the *DtMtlVariable*.

DtMtlCommand

DtMtlCommand is an abstract base class that defines how an MTL command is created. Subclassing and adding member functions for getting and setting variables allows for greater type safety.

You must implement the following member functions in derived classes to fully define a MTL command:

- `virtual DtString command() const = 0` – A command string used when creating the MTL representation of the command.
- `virtual void setupVariableList() = 0` – Sets up the variables that will be placed into the command string. It is called from the commandString() member function of the base class. You can add variables to the list of variables that will be added to the command by using the addVariable() member function.

For each MTL command you want to add to an MTL command dispatcher, add it to the parser as follows:

```
myMtlParser.registerCommand(DtMyMtlCommand().command(), callbackFunc,
  usrPtr);
```

This registers your new command with your MTL dispatcher so it can be referenced in MTL commands and scripts. Failure to register your command with the dispatcher will result in the command being ignored.

When a callback is invoked, a list of lisp tokens is supplied in the first parameter and the lisp environment in the second:

```
void DtlObjPtr callbackFunc(const DtlObjPtr& args, const DtlObjPtr& env)
```

You can use these arguments in their pure form, or you can restore a command object by passing these arguments to an MTL command constructor. You must create a constructor of this type, calling the base class constructor, if you want to automatically rebuild your command. This can be done as follows:

```
DtMyMtlCommand::DtMyMtlCommand(const DtlObjPtr& args, const DtlObjPtr&
  env) :
DtMtlCommand(args, env, command())
{
  myInt = (int)variable(0);
  myString = (const char*)variable(1);
}
```

Part of this reconstruction will also be the assignment of the user pointer set when adding this member function. This can be referenced from the user() member function of the newly restored command. You can use this member function to call an internal member function from the static callback:

```
void DtlObjPtr callbackFunc(const DtlObjPtr& args, const DtlObjPtr& env)
{
  DtMyMtlCommand command(args, env);

  ((DtMyMtlCommand*)command->user())->process(command);
}
```

DtMtlCommandDispatcher

DtMtlCommandDispatcher is a convenience class that allows you to add registered commands to the MTL environment. It is a wrapper around the *DtMtlEnvironment* class that allows usage of the environment without having to know the lower level MTL calls.

DtMtlNetCommandDispatcher

DtMtlNetCommandDispatcher is a network enabled version of the command dispatcher that you can use to automatically register for network receipt of MTL commands and invoke registered callbacks when those messages are received. To register these commands, register the parser with the MTL command interaction as follows:

```
DtMtlNetCommandDispatcher parser;
parser.registerCommand(DtMyMtlCommand().command(), callbackFunc, usr);
DtMtlCommandInteraction::addParser(&exConn, &parser);
```

When an MTL command is received over the network, the callbackFunc registered will be invoked.

DtPvdMtlCommandDispatcher

DtPvdMtlCommandDispatcher checks to see if the user has decided to ignore messages received over the network.

Loading Script Files at Startup

The `-R` startup option lets you load a script file at startup. The script file can contain commands that have been registered with the default application parser. This parser can be retrieved from the commandDispatcher() member function in the *DtPvdAppEvent-Controller* object.

Object Transparency

A new class, *DtBaseSymbolDrawer,* has been added to support object transparency. If you have created your own drawer and want to take advantage of transparency drawing, instead of subclassing your drawer from *DtSymbolDrawer*, subclass from *DtBaseSymbolDrawer*. All other code can stay the same.

API Changes to Support Transmitters

The Connection Manager emits the following signals for transmitters:

```
transmitterAdded(const DtReflectedRadioTransmitter&)
transmitterRemoved(const DtReflectedRadioTransmitter&)
transmitterUpdated(const DtReflectedRadioTransmitter&)
```

It emits the following signal when a signal PDU is received:

```
signalInteraction(const DtSignalInteraction&);
```

When a new transmitter is added, an associated *DtTransmitterModelData* is created and a new model data object is added to the internal application model data list. This model data is associated with an owning entity model data. If no owning entity exists, the transmitter is not displayed. If the entity appears later in the simulation, the transmitter will be hooked up to that entity and displayed with that entity.

The symbol generator also emits the following signals when a signal PDU is received from the Connection Manager:

```
// Sent when a signal is first received and associated with a transmitter
signalReceived(const DtTransmitterModelData*,
  const DtSignalInteraction&);

// Sent when a transmitter has stopped transmitting after it had been
// transmitting
stoppedTransmitting(const DtTransmitterModelData*);
```

A new class *DtSignalMap* has been added. A *DtPvdApplication* object owns an instance of this class, and any time a new signal PDU is received it tries to map the signal to a possible sender of a signal on that same frequency. If no signal is matched, the signal is considered the first of that kind on that frequency and any subsequent signals received on that frequency (until a specified time has elapsed) are considered a match for that frequency and a communication link is made in the map. The display then draws a communication line based on this new information.

Symbol Mappers

The getColorForForce() and modifySymbol() member functions now take a `const DtEntityMapperKey&` as the last parameter.

This parameter will contain the entity type requesting the color.

Classed Removed

- The *DtPvdHelpMenu* has been removed and renamed to be a more generic *DtHelpMenu* included by *tmHelpMenu.h*. The basic functionality and menu items are the same.
- The *DtAboutBox* has been renamed to be *DtPvdAboutBox*. You must now include *pvdAboutBox.h* to override about box functionality.

Changes to the FED File

A new object has been added to the FED file. If you use a customized FED file, you must add the following to it:

```
(class PvdViewControl best_effort receive
  (parameter MtlCommandPdu)
)

(class MtlCommand best_effort receive
  (parameter MtlCommandPdu)
)
```

Terrain API Changes

This section summarizes changes to the Terrain API.

GeometryNative Files Removed

The geometryNative project and files that were deprecated in the 3.6 release have finally been removed. Any remaining references to them should be changed to use the appropriate files in the Geometry project.

DtSpecificationAttribute Refactored

The *DtSpecificationAttribute* class has been updated to contain a symbol string instead of an actual string. This significantly reduces the amount of memory needed to store large numbers of specifications attributes in the specifications. Most specifications contain the same attributes, and so this reduction is non-trivial for terrain files with large amounts of vector data. You should not have to do anything to update your code. Additionally, the label member has been made private, but with public accessors so that derived classes can easily override the functionality to store the label another way.

New GDB File Version

The GDB file format has been incremented to account for the spatial subdivision changes. The VR-Forces can read GDB files using earlier formats, but previous versions of the VR-Forces cannot read GDB files created with the release 2.83.8 tdbtool.

The new GDB version no longer explicitly stores the entire spatial subdivision in the file. It stores only the configuration, so that it can be recreated every time the file is imported. This only marginally affects the performance of importing GDB files, and drastically reduces versioning issues when reading GDB files.

Vertex Functions in DtTriangleIndirect have Changed

The vertex0(), vertex1(), and vertex2() functions in the *DtTriangleIndirect* class are now non-virtual, and all higher level functions have been refactored to comply with this. As part of the refactoring, which was done to allow those functions to be inlined, the getEdge(), getNormal(), and getPlane() functions have either been replaced with edge(), normal() and plane() functions to be consistent with the current naming conventions and interfaces. This affects *DtAllTypesPolygon* (*allPoly.h*), *DtBasePolygon* (*basePoly.h*), and *DtBaseTriangle* (*baseTri.h*). Classes derived from *DtBaseTri* or *DtAllTypesPolygon* that were using the vertex0/1/2 functions should now use getVertex() with the appropriate index.

Change to *DtFltReader*

The xlateGroupChildrenIfAny() function now takes a reference to a pointer to a *DtGroup*, instead of an actual *DtGroup* pointer, so that it can allocate the group when and if a child node is translated. This avoids creation of *DtGroup* nodes that have no children.

New Examples

The Plan View Display has the following new examples:

- envProcessListen demonstrates how to listen for and display state updates for environmental process objects (control and overlay objects) generated by VR-Forces.
- addAttr demonstrates how to extend the RPR FOM. It is derived from the VR-Link addAttr example.
- chat demonstrates how to derive a GUI remote control class and install it.

Documentation Updates

The PDF version of *MÄK Plan View Display User's Guide* has been updated for this release.

Bug Fixes

This release fixes the following problems:

- The Plan View Display now correctly displays velocity and acceleration in UTM databases in the Entity Information dialog box.
- The Plan View Display no longer crashes if you run a logger recording with an incorrect database loaded or no database opened.
- When selecting a feature type that might be both an areal and point feature, make sure to select both kinds.
- Shape files containing NULL records are now being imported properly.
- There was a problem with calculation of the intersection record normal when the terrain contained coordinate system nodes. The correct normal will now be returned if requested.
- The Ocean polygons in the Berkeley terrain have been fixed to no longer have the grass soil type.
- DFAD files are now imported correctly.
- Several bugs in creating surface characteristics from texture file names, using the *surfChar.map* file, when importing OpenFlight files have been fixed. The surface characteristic for Sand is now properly created when a texture name contains the string "sand". In general, creating surface characteristics from the texture names and the surface characteristics map has been improved.
- Specifying a larger minimum than maximum longitude or latitude in DtedMetafileRecords no longer causes an application crash. If the minimum is larger than the maximum, the specified values are ignored and a warning is output.
- The CTDB file importer now treats linear features with only a single vertex as point features representing trees. Previously these features were ignored, which was incorrect.
- OpenFlight files that contained relative pathnames to referenced files, textures, and so on, now import correctly.
- DtGdbNodes marked for removal are now not written out to disk in the GDB file. This results in a small improvement in memory usage and performance.
- The UTM Zone specified in the OpenFlight header is now respected. Previously, the UTM zone was calculated from the specified origin latitude/longitude. (Users seeing apparent projection errors when importing OpenFlight files created by MPI's CTS product should see an improvement.)
- The map datum for Open Flight files was being read and set incorrectly. This has been fixed. If you have Open Flight files with a non-WGS-84 map datum, you should reimport them into the GDB format.

Known Problems

Plan View Display Release 2.8 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.
- Vector clipping does not work properly with areal features.
- 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.
- Intervisibility works with linear and areal features, but not with point features.
- You cannot use translation files to translate entity labels.