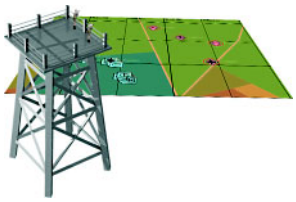

MÄK Plan View Display 2.5-ngc Release Notes

This release note provides the following release-specific information for MÄK Plan View Display 2.5-ngc:

Document ID: PVD-2.5-2-031125

Supported Systems and System Requirements

Plan View Display 2.5-ngc is being released for:

- PCs with Windows 2000/XP/NT 4.0
- Linux
- SGI IRIX.

If you are using the Plan View Display API, 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
PC with Windows NT 4.0 SP6, Windows 2000 SP2, XP	Microsoft Visual C++ 6.0, Service Pack 5 Microsoft .NET

Building on Linux or IRIX

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

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

MÄK Product Compatibility

To use the Plan View Display API, you must link with VR-Link 3.8.1-ngc. If you are building for HLA and want to link with the MÄK RTI, use MÄK RTI 2.x-ngc.

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

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.

System Requirements for PCs

To install the Plan View Display on a PC, you must have a system with the following minimum requirements:

- Processor: Intel Pentium 400 MHz or equivalent.
- RAM: The minimum required for your operating system, plus:
- 32 MB for DIS exercises
- 64 MB for HLA federates

Depending on the size of your terrain databases, you may need more RAM.

- Hard disk space - Approximately 185 MB. Most of this is for the terrain databases shipped with the Plan View Display and the class reference documentation. You may need considerably more space depending on the databases you plan to use.
- Two or more multiples of the amount of RAM for virtual memory. Running the Plan View Display in HLA may require increased amounts of virtual memory.

Network Compatibility

HLA only

Plan View Display 2.5-ngc was built against VR-Link 3.8.1-ngc and is compliant with:

- RPR-FOM 1.0 and a subset of 2.0 (draft 6 and 14)
- MÄK RTI 2.0.3-ngc
- DMSO RTI NG 6.

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

Plan View Display 2.5-ngc is compatible with applications that use earlier versions of VR-Link if they support versions of the RPR FOM listed above.

DIS only

Plan View Display 2.5-ngc supports DIS 2.0.4, IEEE 1278.1, 2.1.4, and IEEE 1278.1a, and can therefore interoperate with DIS applications of any of these versions.

FOM Support

Plan View Display 2.5-ngc has built-in support for versions 0.5, 0.7, 0.8, 1.0, and 2.0, drafts 6 and 14, of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, Plan View Display 2.5-ngc uses RPR FOM 1.0.

The keywords for specifying different versions of the RPR FOM with the `-f` startup option are:

- RPR FOM 0.5: `vlRPR05`
- RPR FOM 0.7: `vlRPR07`
- RPR FOM 0.8: `vlRPR08`
- RPR FOM 1.0: `vlRPR10`
- RPR FOM 2.0 darft 6: `vlRPR20006`
- RPR FOM 2.0 draft 14: `vlRPR20014`.

For information about FOM mapping and selecting the correct FOM mapper, see Startup Options for HLA Federations in Chapter 3, "Getting Started", in *MÄK Plan View Display User's Guide*.

New Features and Updates

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

- Support for large databases (greater than 1 UTM zone).
- Updated GUI API supports overlays (tactical overlays) and the ability to save data to files. See "GUI API Migration Instructions," on page 1-12.
- Updated Terrain API.
- More efficient GDB database format.
- Ability to import shape file specification attributes from dBASE files.
- New features in the Plan View Display front-end:
  - Ability to save and restore the layout of the screen and display settings
  - Support for tactical overlay graphics
  - Ability to specify which types of intervisibility intersections to test for
  - Ability to display an entity's bounding box
  - Improved support for controlling the MÄK Stealth.

Support for Large Databases

The Plan View Display shares the Terrain API with MÄK VR-Forces. The Terrain API has been updated to support large terrain databases (greater than one UTM zone). These changes have little effect on use of the Plan View Display, however this section records the changes to the Terrain API that have been introduced.

The Plan View Display can now load terrain databases that are built in geocentric or geodetic (Latitude, Longitude, Altitude) coordinates. When the Plan View Display loads a geocentric database, it is automatically converted to geodetic coordinates, since it does not make sense for a 2D map to display geocentric coordinates.

Coordinate Systems

The Plan View Display maintains the concept of local (database) coordinates. The local coordinate system is the coordinate system used by the terrain database. Through version 2.4 of Plan View Display, local coordinates have been synonymous with UTM coordinates, because that was the only terrain database coordinate system supported by the Plan View Display. Now local coordinates may be UTM or geocentric coordinates, depending on the coordinate system of the terrain database.

Converting between Local Coordinates and Other Coordinate Systems

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

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

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

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

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

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

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

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

DtVector topoVelocity;
```

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

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

The Gravity Direction Vector

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

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

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

Terrain Intersections

The *DtTerrainDatabase* class (*terrainDb.h*) provides an interface for terrain intersections with arbitrary chords. The intersectTerrain() function declared in *kinTools.h* is a convenience function for performing this type of intersection.

Updated Terrain API

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

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

Updated *DtGdbNode* Types

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

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

Updated *DtTerrainDatabase* Class

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

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

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

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

See "Migrating to the Updated Terrain API," on page 1-14 for detailed migration instructions.

More Efficient GDB Format

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

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

New Front-End Features

The new features are all documented in the updated PDF version of *MÄK Plan View Display User's Guide*.

- Ability to save and restore the layout of the screen and display settings. The Plan View Display automatically saves your application settings. See "Saving Plan View Display Settings," in *MÄK Plan View Display User's Guide*.
- Support for overlay graphics. The Plan View Display supports creation of multiple overlays on which you can draw overlay objects, such as points, lines, areas, symbols, and text. See Chapter 7, *Control Objects and Overlay Objects*, in *MÄK Plan View Display User's Guide*.
- Ability to specify which types of intervisibility intersections to test for. You can test for polygons, features, and entities. In previous releases, the Plan View Display just checked for terrain intersections. "Specifying the Types of Objects to Check for Intersections," in *MÄK Plan View Display User's Guide*.
- Ability to display an entity's bounding box. You can display the entity bounding box used to calculate intervisibility. See "Displaying An Entity's Bounding Box," in *MÄK Plan View Display User's Guide*.
- Improved support for controlling the MÄK Stealth. You can now specify one of three view modes when you attach the Stealth to an entity. You can also drag the Stealth icon to move the eyepoint. See "Interoperating with the MÄK Stealth," in *MÄK Plan View Display User's Guide*.

Importing Shapefile Attributes

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

Documentation Updates

The PDF version of *MÄK Plan View Display User's Guide* is updated to include all new features. It now includes a Quick Reference Card that summarizes startup options, keyboard shortcuts, and procedures for navigating the map. Also, note the following clarifications to the printed version of the manual:

- In Chapter 10, *The Plan View Display API*, "Capturing State Updates and Displaying Information," the following section incorrectly capitalized connection-Manager().

To register a callback that is to be executed when a message is received, you must get a handle to the exercise connection. You can do this in *main.cxx* (or any *.cxx* file that can get access to the main application object) as follows:

```
DtPvdApplication* pa =
  dynamic_cast<DtPvdApplication*>(DtTMApplication::app());

pa->connectionManager()->exerciseConn()->
  addPduCallback(DtPduKind(MyPduKind), myPduCallback, this);
```

MÄK RTI Availability

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

Bug Fixes

This release fixes the following problems:

- If you are using TrueType fonts, the symbols for air aggregates are now displayed correctly.

Known Problems

Plan View Display Release 2.5-ngc has the following known problems:

- On IRIX, icons may appear discolored or grainy unless the color depth is set to 24-bit.
- True Type fonts are not supported in SGI IRIX. Use bitmaps for entity icons.
- If you are using the Plan View Display in a large DIS exercise with heavy network traffic, you may experience problems with dropped packets and entities may time out. You alleviate this problem by using asynchronous I/O, by starting with the `-I` parameter.
- Uninstalling the Plan View Display in Windows via InstallShield uninstalls the entity fonts. If you have another application that uses these fonts, such as MÄK VR-Forces, the correct entity icons would not be displayed. You must manually install the fonts from their location in the other application that needs them.
- The Qt translation files (in *./translations*) for built-in Qt-level GUI items do not work properly. The translation files for the Plan View Display work correctly.
- Changing the measurement unit used in dialog boxes does not change the measurement unit for contour lines.

GUI API Migration Instructions

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

New Classes

The following classes have been added:

- *DtArchiver* and archivable object classes.
- *DtPointConfigWidget* encapsulates *DtWaypointConfigWidget* to create any kind of object that would be a single point.
- *DtTerrainCoordinateSystemCollection* is a collection of coordinate systems. This class is used by all dialog boxes that want to set or display location information. This class is now the central point for adding new types of coordinate system displays. You no longer have to subclass every dialog box that needs to know about a coordinate system. The windowsCoordinateSystem example has been updated to show how to add and use a new coordinate system in this context.
- *DtTMUnitConverterCollection* is a collection of unit converter classes that are used by all dialog boxes and displays that translate meters to some other unit. This class is now the central point for adding new types of unit conversions. You no longer have to subclass every dialog box that needs to know about a unit conversion.
- *DtOverlaySymbolMapper* implements the ability to create overlay objects. *DtMtlSymbolMapper* now subclasses from this object rather than from *DtBaseSymbolMapper*.
- *DtOverlayPublisherHandler* handles the creation and editing of overlay objects.
- *DtEnvironmentalModelData* handles the data used by overlay objects. *DtControlObjectModelData* classes now subclass from this class. Any symbol updater routine that used to reference *DtControlObjectModelData* now must reference *DtEnvironmentalModelData* instead.
- *DtTerrainLocationEntryWidget* can be used in dialog boxes that need to enter in positional information. When initialized, this widget can change itself to conform to changes in state of the coordinate system and convert between the current and new system automatically. If you have written code to take this into account, you may be able to replace it with this class. The imageSplitter example shipped with the application shows how to use this widget to set positional information to move items on the map area.

Changes to Existing Classes

The following classes have changed as indicated:

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

Migrating Code for Subclasses of *DtMtlSymbolMapper*

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

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

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

  it now will look like:

  ```
  MyMtlSymbolMapper();
  ```

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

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

  changes to:

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

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

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

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

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

Migrating to the Updated Terrain API

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

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

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

New Manager Classes

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

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

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

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

Migrating from List Nodes to the New Manager Classes

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

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

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

Creating and Adding *DtPolygonIndirects* and *DtTriangleIndirects*

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

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

- *DtTriangleIndirect*:
  - 3 vertices
  - 1 surface

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

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

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

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

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

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

All of the functions mentioned also take two optional parameters:

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

New Code Examples

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

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

Memory Usage

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

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

Duplicate Vertex/Surface Reduction

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

Optimizing Performance

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

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

Modification of the Spatial Index

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

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

Tree Balancing

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

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

Deprecated Classes

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

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