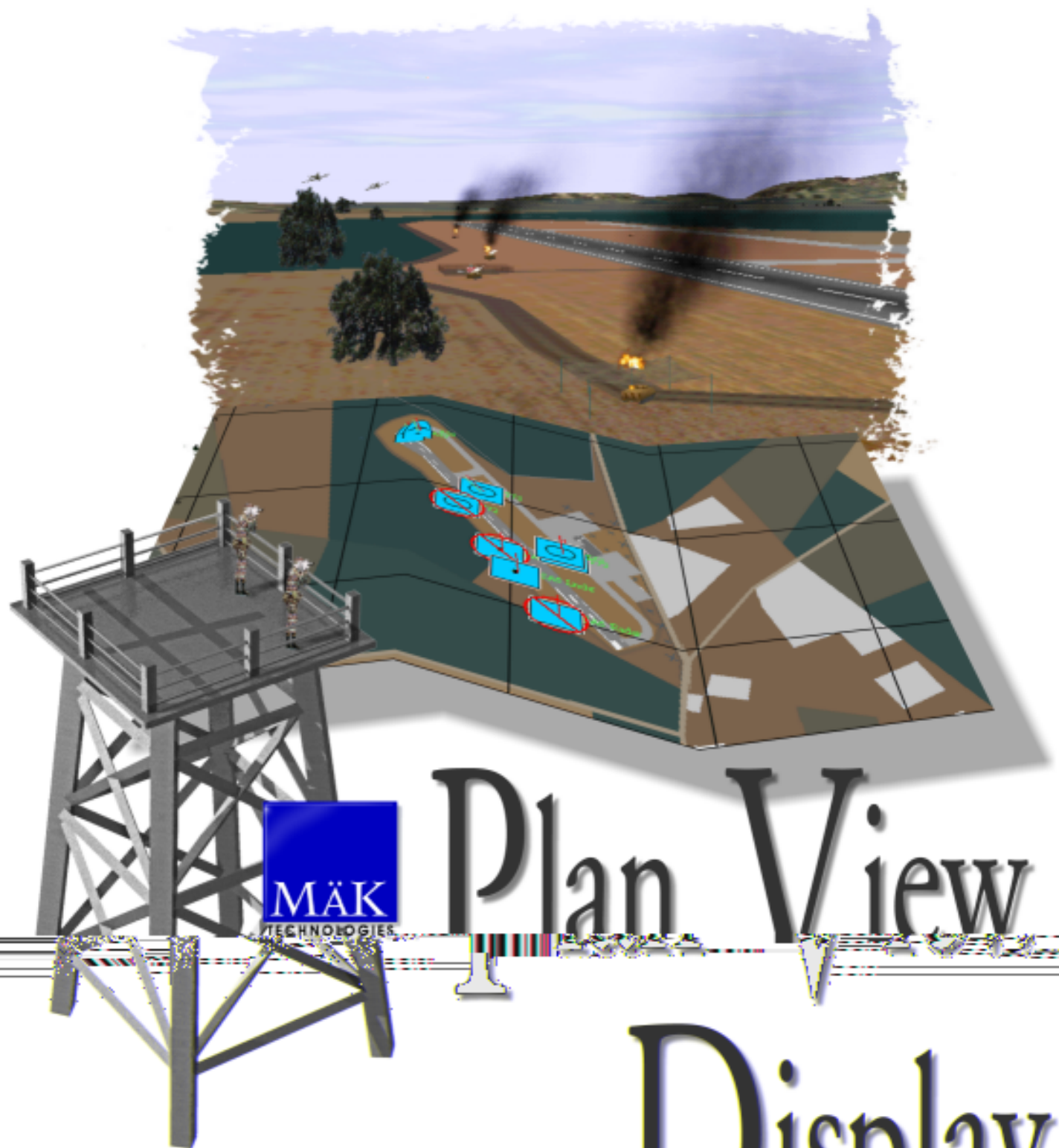
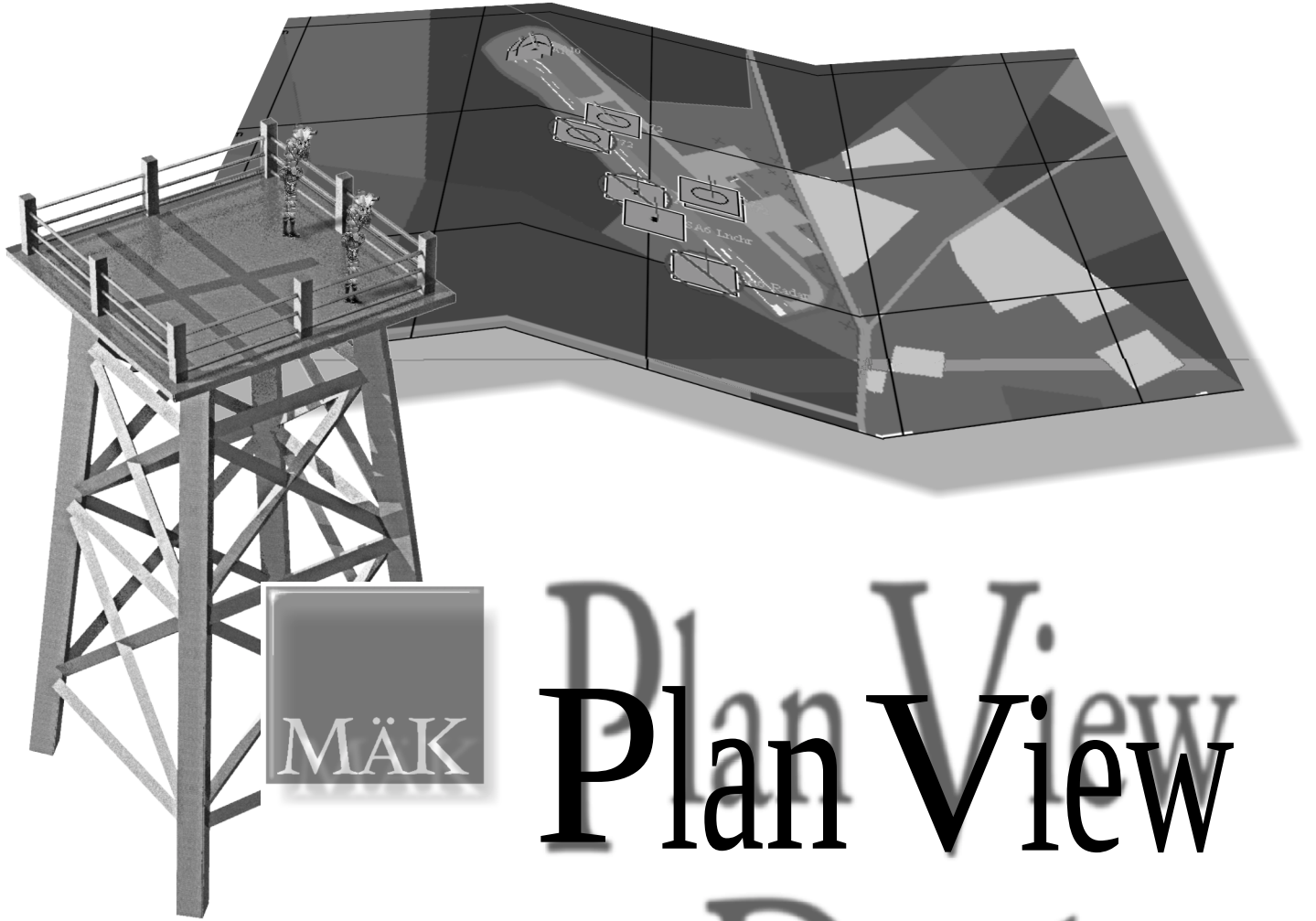




Plan View

Display

*The Real-Time 2D Virtual World
Viewer and Navigator*



Plan View Display

*The Real-Time 2D Virtual World
Viewer and Navigator*



MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02138 USA

Copyright 1998, 1999, MÄK Technologies
All rights Reserved. Printed in the United States.

Second Printing: May 1999.

Under copyright laws, no part of this document may be copied or reproduced in
any form without prior written consent of MÄK Technologies, Inc.

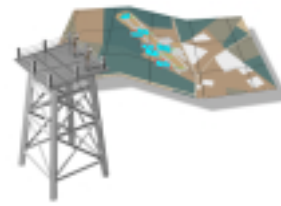
VR-Link® is a registered trademark of MÄK Technologies, Inc.
All other trademarks are owned by their respective companies.

MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02138 USA

Voice: 617-876-8085
Fax: 617-876-9208

info@mak.com
www.mak.com

Revision PVD-1.2-1-990513



Contents

Preface

About this Manual	ix
Other MÄK Products	xi
How to Contact Us	xii
Document Conventions	xiii

Chapter 1 Introducing the MÄK Plan View Display

1.1. Features of the Plan View Display	1-1
--	-----

Chapter 2 Installing the Plan View Display

2.1. System Requirements	2-2
2.1.1. Color Depth on PCs	2-2
2.2. Installing the Plan View Display On A PC	2-3
2.3. Installing the Plan View Display on UNIX	2-4
2.4. What's Installed	2-4
2.5. Setting Up and Using the FLEXlm License Manager	2-6
2.5.1. The FLEXlm Client-Server Architecture	2-6
2.5.2. Installing and Configuring the License Manager	2-7
2.5.3. Adding Additional License Files	2-10
2.5.4. Running the License Server	2-11
2.5.5. Troubleshooting the FLEXlm License Manager	2-11
2.6. Obtaining and Installing a Runtime Infrastructure (RTI)	2-14
2.6.1. Installing the MÄK RTI	2-14
2.6.2. Installing the DMSO RTI	2-15

Chapter 3 Getting Started

3.1. Starting the Plan View Display On A PC	3-2
3.2. Starting the Plan View Display in UNIX	3-3
3.2.1. Creating a Plan View	3-4
3.3. Exiting the Plan View Display	3-5
3.4. The Mak PVD Window	3-5
3.5. The Plan View Window	3-6
3.5.1. Plan View Menus	3-6
3.5.2. The Plan View Information Bar	3-8
3.5.3. Load Database Dialog Box	3-9
3.5.4. Save Database Dialog Box	3-10
3.6. Settings Dialog Box	3-11
3.6.1. Settings Dialog Box, Display Tab	3-12
3.6.2. Settings Dialog Box, Control Tab	3-14
3.6.3. Settings Dialog Box, Color Tab	3-15
3.7. Entity List Dialog Box	3-16
3.8. Entity Information Dialog Box	3-18
3.9. Stealth Control Dialog Box	3-20
3.10. Intervisibility Settings Dialog Box	3-22
3.11. Entity Labels Dialog Box	3-23

Chapter 4 Using the Plan View Display

4.1. Navigating a Plan View Map	4-2
4.1.1. Panning the Plan View Window	4-2
4.1.2. Zooming the Plan View Window	4-2
4.1.3. Resizing Plan View Windows	4-3
4.1.4. Moving to An Entity on A Plan View Window	4-3
4.2. Loading A New Terrain Database	4-5
4.3. Using Multiple Plan View Windows	4-6
4.3.1. Hiding and Displaying Plan View Windows	4-6
4.3.2. Closing Plan View Windows	4-7
4.3.3. Hiding Plan View Windows Versus Closing Them	4-7
4.4. Saving and Loading Layouts	4-8
4.4.1. Saving A Plan View Layout	4-8
4.4.2. Loading A Plan View Layout	4-8
4.5. Controlling the Stealth from the Plan View Display	4-9
4.5.1. Configuring the Stealth to Work with the Plan View Display ...	4-9
4.5.2. Controlling the Stealth	4-10
4.5.3. The No Action Radio Button	4-13
4.6. Working With Terrain Databases	4-14
4.6.1. Saving Terrain Databases in MAK Terrain Format	4-14
4.6.2. Using A Global Geodetic View	4-15
4.6.3. Terrain Metafiles	4-16
4.6.4. Creating Terrain Metafiles	4-17

Chapter 5 Working With Entities and Other Objects

5.1. Selecting Entities	5-2
5.1.1. Selecting Entities in A Plan View Window	5-2
5.1.2. Selecting Entities in the Entity List Dialog Box	5-3
5.1.3. Deselecting Entities	5-3
5.2. Working with Groups of Entities	5-5
5.2.1. Creating a User-Defined Group	5-5
5.2.2. Changing A User-Defined Group	5-5
5.2.3. Selecting Groups	5-6
5.2.4. Deleting User-Defined Groups	5-6
5.2.5. How Changes to Entities Affect Group Membership	5-6
5.3. Tracking Entities	5-7
5.3.1. Tracking Entities in the Plan View Display	5-7
5.3.2. Tracking An Entity Using the Stealth	5-7
5.4. Displaying Information About An Entity	5-9
5.5. Displaying Entity Labels	5-9
5.5.1. Understanding Visual Labels	5-10
5.5.2. Turning Labels On and Off	5-10
5.6. Displaying Intervisibility	5-11
5.7. Working With Markers	5-13
5.7.1. Placing Markers on A Canvas	5-13
5.7.2. Displaying Intervisibility for A Marker	5-13
5.7.3. Destroying A Marker	5-13
5.8. Interactions	5-13

Chapter 6 Customizing the Plan View Display

6.1. Startup Options	6-2
6.1.1. Protocol Independent Startup Options	6-3
6.1.2. Startup Options for HLA Federations	6-4
6.1.3. Startup Options for DIS Exercises	6-6
6.1.4. The Plan View Display Log File (PC Only)	6-6
6.2. Configuration File Options	6-7
6.2.1. Associating Bitmaps with Entity Types	6-7
6.2.2. Localizing the Plan View Display	6-9
6.2.3. Configuring Entity Labels	6-10
6.2.4. Configuring Plan View Window Settings	6-11
6.2.5. The X Resource File (UNIX Only)	6-13
6.3. Changing Settings	6-14
6.3.1. Specifying Which Information is Displayed	6-14
6.3.2. Setting Zoom Increments	6-15
6.3.3. Setting Panning Increments	6-15
6.3.4. Specifying the Frequency of Grid Lines	6-15
6.3.5. Specifying the Frequency of Contour Lines	6-15
6.3.6. Specifying the Duration of Interactions	6-16
6.3.7. Setting the Frame Rate	6-16

6.3.8. Displaying Terrain Shading	6-16
6.3.9. Setting Contribution Culling	6-17
6.3.10. Changing Colors	6-18
6.4. Icons	6-19

Chapter 7 Using Plug-ins With the Plan View Display

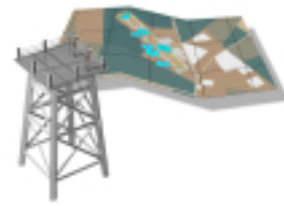
7.1. Introduction to Plan View Display Plug-ins	7-3
7.1.1. How the Plug-in API Works	7-3
7.2. Working with Plug-in Interfaces	7-4
7.3. Responding to Events	7-5
7.3.1. Registering Callbacks Directly with An Application	7-5
7.3.2. Responding to Plan View Display Ticks	7-6
7.4. Managing Windows From A Plug-in	7-7
7.4.1. Extending Menus	7-7
7.4.2. Getting Terrain Information	7-8
7.4.3. Working with Coordinate Information	7-8
7.5. Responding to User Events	7-10
7.5.1. Getting Selection Information	7-10
7.6. Working with Objects	7-10
7.6.1. Creating Objects	7-11
7.6.2. Looking Up An Object Interface	7-11
7.6.3. Getting A List of Objects	7-11
7.6.4. Tracking Objects	7-12
7.6.5. Timing Out Objects	7-12
7.6.6. Responding to Creation of Objects and Object Changes	7-12
7.6.7. Getting information about objects (state, coordinates, ID)	7-13
7.6.8. Setting Object Values	7-14
7.6.9. Removing objects	7-14
7.7. Building and Compiling A Plug-in	7-15
7.8. Plug-in Example	7-16
7.8.1. Planning a Plug-in	7-16
7.8.2. Implementing the Plug-in	7-20
7.9. Plug-in Classes and Header Files	7-24

Appendix A Plug-in Header Files

A.1. Class Hierarchy Diagram	A-2
plugAppInter.h	A-3
plugCanvInter.h	A-6
plugCircObjIf.h	A-10
plugEntObjIf.h	A-12
plugIntObjIf.h	A-13
plugLineObjIf.h	A-14
plugObjInter.h	A-16
plugObjLInter.h	A-19
plugPntObjIf.h	A-22

plugPolyObjIf.h	A-23
plugTextObjIf.h	A-25

Index



Preface

The MÄK Plan View Display (PVD) provides a two-dimensional, “big picture” of a virtual environment. It shows simulated entities, such as vehicles, soldiers, and tanks moving over a 2D-map display. The MÄK Plan View Display is compatible with both Distributed Interactive Simulation (DIS) and High Level Architecture (HLA) simulation standards.

About this Manual

This manual is written for persons who will install or use the MÄK Plan View Display. The manual assumes that you are familiar with the basic operation of your operating system and that you know how to work in your computer’s graphical window environment.

The Plan View Display is a dynamic product, constantly evolving to meet user needs. Consequently, you may find improvements or new features that have been added to the Plan View Display since this manual went to press. Please see the Release Notes for release-specific updates.

The manual is organized as follows:

Chapter 1, *Introducing the MÄK Plan View Display*, briefly describes the Plan View Display and its features.

Chapter 2, *Installing the Plan View Display*, explains how to install the Plan View Display, set up the FLEXlm license manager, and obtain an RTI for use with HLA.

Chapter 3, *Getting Started*, explains how to start and stop the Plan View Display and describes its windows and dialog boxes.

Chapter 4, *Using the Plan View Display*, explains high level tasks such as navigating a Plan View window, creating a new Plan View window, managing terrain databases, and controlling the MÄK Stealth from the Plan View Display.

Chapter 5, *Working With Entities and Other Objects*, explains how to select entities, create groups of entities, track entities, and assign entity labels.

Chapter 6, *Customizing the Plan View Display*, describes command line options and configuration files, and explains how to change settings that control the information displayed in Plan View windows.

Chapter 7, *Using Plug-ins With the Plan View Display* describes the plug-in interface to the Plan View Display and explains how to create plug-ins.

Appendix A, *Plug-in Header Files* lists the header files that support the plug-in feature.

i

The screen captures in the *MÄK Plan View Display User's Guide* were taken using the PC version of the Plan View Display. Therefore, there may be some slight variations between the manual and what you see on a UNIX screen. However, the functions and usage of the windows and dialog boxes are identical in both versions, except as noted in this manual.

Other MÄK Products

The Plan View Display is a member of the MÄK Technologies line of HLA/DIS software products designed to streamline the process of developing and using networked simulated environments.

In addition to the Plan View Display, the MÄK Technologies product line includes:

- ♦ The VR-Link[®] Network Toolkit. The Toolkit is an object oriented library of C++ functions and definitions that implement the HLA RPR FOM and the DIS protocol. This library minimizes the time and effort required to build and maintain new HLA/DIS-compliant applications, and to integrate such compliance into existing applications.

The Toolkit includes a set of sample debugging applications and their source code. The source code serves as an example of how to use the VR-Link Toolkit to write applications. The executables provide valuable debugging services such as generating a predictable stream of HLA or DIS messages, and displaying the contents of messages transmitted on the network.

VR-Link is the basis of the other MÄK products.

- ♦ The MÄK Stealth. The Stealth displays a realistic, three-dimensional view of your virtual world. You can view this world from the inside of a simulated moving vehicle, or place the eyepoint at another moving or stationary location. The Stealth lets you switch rapidly among several predefined viewpoints while the simulation is underway. The Stealth runs on PCs and the SGI computers and supports multi-channel views.
- ♦ The MÄK Logger. The Logger, also called the Data Logger, can record HLA and DIS exercises and play them back for after-action review. You can play a recorded file at speeds above or below normal, or in reverse, and locate areas of interest quickly. The Logger lets you choose between a graphical interface, or a shell-style, command-line interface. You can also control the Logger programmatically with logger control PDUs or state update messages.

The Logger comes with a set of utility programs that let you inspect, combine and edit Logger recordings.

- ♦ The MÄK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MÄK RTI is the first commercial run-time infrastructure for HLA simulations. It is optimized for real-time simulations.

How to Contact Us

For Plan View Display technical support, information about upgrades, and information about other MÄK products, you can reach us in the following ways:

For sales and upgrade information by e-mail:	info@mak.com
For technical support by e-mail:	support@mak.com
For license key files and information:	www.mak.com/support/keyintro.htm
The MÄK web site:	www.mak.com

When requesting support, please tell us the product you are using, the version, and the platform on which you are running.

Send postal correspondence to:	MÄK Technologies 185 Alewife Brook Parkway Cambridge, MA, USA 02138
--------------------------------	---

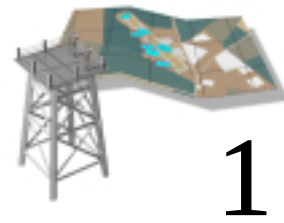
Call or fax us at:	Voice:	617-876-8085 (extension 3 for support)
	Fax:	617-876-9208

Document Conventions

This document uses the following typographic conventions:

<i>Monospaced Type</i>	Indicates commands or values you enter.
<i>Monospaced Italic</i>	Indicates variables that you replace with appropriate values.
[]	Indicates optional arguments.
	Separates options in a command where only one option may be chosen at a time.
/	Indicates a directory. Since MÄK products run on both UNIX and PC platforms, we use the / (slash) for generic discussions of pathnames. If you are running on a PC, substitute a \ (backslash) when you type pathnames.
Bold type	Indicates a new term.
<i>Italic</i>	Indicates a file name or path, or a class name.
sansSerif	Indicates a parameter or argument.
>	Indicates a one-step procedure.
Menu ③Option	Indicates a menu choice. For example, an instruction to select the Save option from the File menu appears as: Choose File Save.
	Indicates supplemental or clarifying information.
	Indicates additional information that you must observe to ensure the success of a procedure or other task.
	Indicates a warning about potential damage to the program or computer.

Directory names are preceded with dot and slash characters that show their position with respect to the Plan View Display home directory. For example, the directory *pvd/binRPR* appears in the text as *./binRPR*.



Introducing the MÄK Plan View Display

Imagine that you are a hawk flying high above a battlefield. Your wide field of view lets you take in the movements of large numbers of participants. At the same time, your acute vision lets you focus in on the smallest portion of the action. The MÄK Plan View Display (PVD) provides you this “birds-eye-view” of a virtual environment.

MÄK's Plan View Display provides you with new insights into a simulated battle by overlaying entities and information onto 2D views of tactical, strategic, and visual databases. Traditional 3D displays, with their perspective views, can make it difficult to see a strategic view of the battle. MÄK's Plan View Display answers typical questions about the placement of forces and how the terrain might affect the engagement more easily than do three-dimensional displays.

The MÄK Plan View Display is compatible with both Distributed Interactive Simulation (DIS) and High Level Architecture (HLA) simulation standards. It is compatible with several database formats, allowing easier and more intuitive use. The MÄK Plan View Display is interoperable with other MÄK products, such as the MÄK Stealth and the MÄK Data Logger.

1.1. Features of the Plan View Display

Using the Plan View Display, you can:

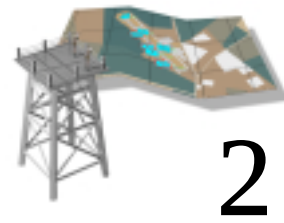
- ♦ Have multiple views open at any time
- ♦ Manipulate the map by zooming and panning
- ♦ Select and track individual entities and groups of entities
- ♦ Turn display features on and off
- ♦ Display entity labels that include several types of information
- ♦ Display intervisibility of entities and points.

The Plan View Display supports:

- ♦ The UTM projection and the Lambert conical conformal projection in CTDB C4B databases
- ♦ Open Flight UTM databases (PC only)
- ♦ MÄK Terrain Format (GDB)
- ♦ Paper maps, for example, CADRG and BMP files
- ♦ Digital terrain elevation data (DTED) databases
- ♦ Interoperability with the MÄK Stealth and Logger.



Although the UNIX version of the Plan View Display does not support the FLT database format, if you have access to the PC version, you can use it or the tdbtool to convert an FLT database to the MÄK Terrain Format (GDB), which you can open in the UNIX Plan View Display.



Installing the Plan View Display

This chapter explains how to install the Plan View Display. It covers the following topics:

System Requirements	2-2
Installing the Plan View Display On A PC	2-3
Installing the Plan View Display on UNIX	2-4
What's Installed	2-4
Setting Up and Using the FLEXlm License Manager	2-6
The FLEXlm Client-Server Architecture	2-6
Installing and Configuring the License Manager	2-7
Adding Additional License Files	2-10
Running the License Server	2-11
Troubleshooting the FLEXlm License Manager	2-11
Obtaining and Installing a Runtime Infrastructure (RTI)	2-14
Installing the MÄK RTI	2-14
Installing the DMSO RTI	2-15

2.1. System Requirements

The Plan View Display is available on the following platforms:

- ♦ SGI IRIX 6.2
- ♦ Sun Solaris 2.5
- ♦ Windows 95/98
- ♦ Windows NT 4.0.

See the *Plan View Display Release Notes* for minimum system requirements.

2.1.1. Color Depth on PCs

Set your PC to display at least 65K colors.

To set color depth:

1. From the Start menu, choose Settings → Control panel.
2. Open the Display applet.
3. Select the Settings tab.
4. Select 65536 Colors or a higher number from the Color Palette drop-down list.

2.2. Installing the Plan View Display On A PC

To install the Plan View Display on a PC:

1. Insert the Plan View Display CD into your CD-ROM drive. If the autorun feature is enabled, the MÄK Multiple Installation Program screen opens ([Figure 2-1](#)).

If the MÄK Multiple Installation Program does not open, open the Windows Explorer. In the root directory for your CD drive, double-click *setup.exe*. The MÄK Multiple Installation Program screen opens.

2. In the MÄK Multiple Installation Program screen, click Plan View Display.
3. Follow the instructions of the installation program.



Figure 2-1. MÄK Multiple Installation Program

2.3. Installing the Plan View Display on UNIX

To install the Plan View Display:

1. Mount the CD-ROM.
2. Run:
 `./install`
3. Follow the on-screen instructions.

You can install the Plan View Display into any directory for which you have write permissions.

2.4. What's Installed

The Plan View Display installation process on both PCs and UNIX creates a directory structure like the one shown in [Figure 2-2](#). On PCs, the default installation directory is `C:\Program Files\MAK\PVDx.x`.

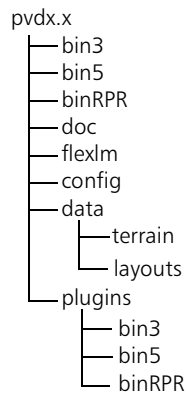


Figure 2-2. Directory structure of Plan View Display files

[Table 2-1](#) describes the contents of the directories installed by the Plan View Display.

Table 2-1: Contents of Plan View Display directories

Directory	Directory contents
<code>./bin3</code>	Executables for use with DIS version 2.0.3.
<code>./bin5</code>	Executables for use with DIS versions 2.0.4, 2.1.4, IEEE 1278, and IEEE 1278.1a
<code>./binRPR</code>	Executable for HLA.
<code>./config</code>	Configuration files. For a description of these files, see “Configuration File Options,” on page 6-7.
<code>./data</code>	The <code>./data</code> directory has several subdirectories that contain bitmaps and icons used for Plan View Display windows, the bitmaps used to represent entities in the Plan View window, and terrain databases. Saved layouts are stored in the <code>./data/layouts</code> directory.
<code>./doc</code>	Plan View Display documentation in Adobe portable document format (PDF).
<code>./flexlm</code>	Files for the FLEXlm license manager. For complete information see “Setting Up and Using the FLEXlm License Manager,” on page 2-6.
<code>./plugins</code>	Contains protocol-specific subdirectories into which you should place any plug-ins that you write for the Plan View Display. This directory also has a sub-directory with sample plug-ins.

2.5. Setting Up and Using the FLEXlm License Manager

The Plan View Display, like other MÄK Technologies products, uses the FLEXlm license manager. Before you can use the Plan View Display, you must obtain a valid license file and configure the license server and client machines. (For information about FLEXlm, view the FAQ page at www.globetrotter.com/flmfaq.htm.) Contact the MÄK Technologies sales department for information about special licensing agreements.

2.5.1. The FLEXlm Client-Server Architecture

This section explains the FLEXlm architecture to provide a context for the setup instructions that follow. FLEXlm uses a client-server architecture. The FLEXlm executable, *maklmgrd* runs on one computer, known as the license server. This machine services every machine (including possibly itself) on which you want to run a MÄK Technologies product. The machines on which the MÄK Technologies products are running are called clients.

The license server has a license file (supplied by MÄK Technologies). This file identifies the MÄK Technologies products for which you have licenses. On each client machine, the `MAKLMGRD_LICENSE_FILE` environment variable (which you must set) points to the server.

Licenses “float”, so you can install product software on more machines than you have licenses for, but at any given time, you can run only as many instances of a product as you have licensed.



The license key in the license file is tied to the host id of the server machine. Therefore, if you decide to change the server, you need to get a new license file. Then, you must update the `MAKLMGRD_LICENSE_FILE` environment variable on every client. This is not a trivial process, and is not recommended.

Figure 2-3 illustrates the client-server architecture for FLEXlm and MÄK products.

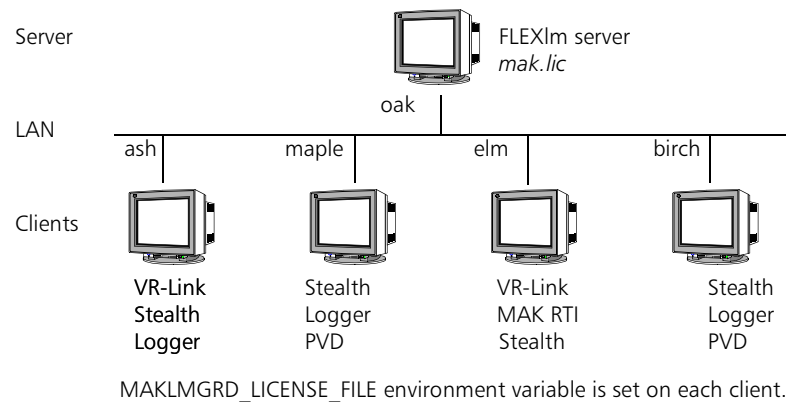


Figure 2-3. FLEXlm example architecture

2.5.2. Installing and Configuring the License Manager

The general procedure for installing and configuring license management is as follows (each step is explained in the rest of this section):

1. Install license manager software.
2. Request a license file.
3. Put the license file in the *flexlm* directory on the license server.
4. Set the MAKLMGRD_LICENSE_FILE environment variable on all client machines.

If you have MÄK Technologies products installed and you purchase additional licenses or products, you must get a license file for the new products or licenses. For this procedure, see [“Adding Additional License Files,”](#) on page 2-10.

Installing the License Manager Software

As part of the installation of all MÄK products, the FLEXlm software is installed in the *flexlm* directory under the root product directory.

1. Install your MÄK products on the machine(s) on which you want to run them.
2. If this is a networked system, and you did not install MÄK software on the license server, copy the *flexlm* directory from a client machine to the license server machine.

Requesting A License File

License files are keyed to the host id of the license server. You need to supply MÄK with some information so that we can generate a license file for you.

To get a license file:

1. On the license server, in a command window, change to the *flexlm* directory.
2. Execute *lmhostid*. This program generates a unique number that MÄK uses to generate your license activation codes. Write down the number.
3. Go to <http://www.mak.com/support/keyintro.htm> and complete the license key request form. MÄK will e-mail you a file named *mak.lic*.

To get a license file, you need to know the name of the license server machine. To get the name of your machine follow the procedure for your operating system:

UNIX

At a command prompt, type *hostname*.

Windows 95/98/NT

1. On the Start menu, choose Settings → Control Panel.
2. Open the Network applet.
3. Select the Identification tab. The machine name is listed in the Computer Name field.

Putting the License File in the *flexlm* Directory

After you receive the license file, put it in the *flexlm* directory on the license server.



If you already have a *mak.lic* file in the *flexlm* directory, do not overwrite the file. See the instructions in [“Adding Additional License Files,”](#) on page 2-10.

Setting the MAKLMGRD_LICENSE_FILE Environment Variable

The MAKLMGRD_LICENSE_FILE environment variable identifies the server machine so that the FLEXlm software on a client can check out a license when you run a MÄK Technologies product. You must set MAKLMGRD_LICENSE_FILE on every client machine.



If you are running MÄK Technologies products on the license server machine, it is also a client, so you must set MAKLMGRD_LICENSE_FILE.

The syntax for the environment variable is: *@Server_name*. For example, if the server machine is oak, set the environment variable to *@oak*.

The following sections explain how to set environment variables on the different platforms that MÄK Technologies products run on.

Windows 95/98

On Windows 95/98, you set environment variables by adding or editing lines in the *Autoexec.bat* file. To set the MAKLMGRD_LICENSE_FILE variable, add a line similar to the following:

```
set MAKLMGRD_LICENSE_FILE=@oak
```



Do not put spaces around the equal (=) sign.

Windows NT

On Windows NT, you set environment variables in the Environment tab of the Properties dialog box for the machine:

1. Right-click the My Computer icon on the Desktop.
2. On the pop-up menu, select Properties. The System Properties dialog box opens ([Figure 2-4](#)).
3. Select the Environment tab.
4. In the Variable text field, type MAKLMGRD_LICENSE_FILE.
5. In the Value field, type the value, for example:
@oak

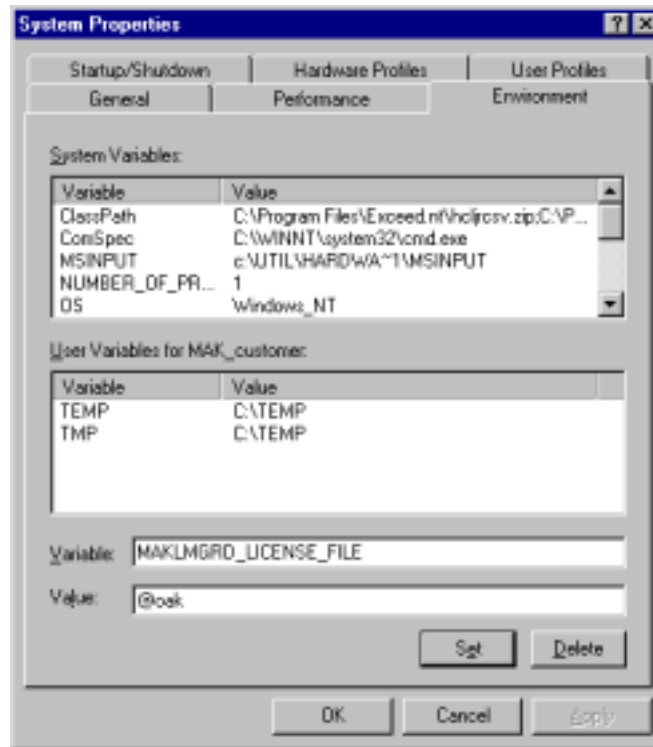


Figure 2-4. Setting an environment variable in Windows NT

UNIX

On UNIX, you set environment variables in your `.cshrc` (or equivalent startup file). Set the variable similarly to the following example:

```
setenv MAKLMGRD_LICENSE_FILE @oak
```

You are ready to run the license server and use your new licenses or MÄK products. See [“Running the License Server,”](#) on page 2-11.

2.5.3. Adding Additional License Files

If you buy additional MÄK products, or additional licenses for products you already have, you must get an additional license file.

- To request an additional license file, follow the same procedure as for your first license file. See [“Requesting A License File,”](#) on page 2-8.

To add a new license file to your license server:

1. When you get the new license file, rename it from *mak.lic* to something like *mak2.lic*, or some other name ending in *.lic*, so that it has a different name from the license file(s) currently in the *flexlm* directory on the license server.
2. Put the renamed file in the *flexlm* directory on the license server, with the other license files.

You can now use the new products or additional product licenses.

2.5.4. Running the License Server

The FLEXlm license server must be running on the server machine before you can run your MÄK applications.

- > To start the license server, on the server machine only, execute *runLm* from the *./flexlm* directory.
- > To obtain the current status of the server, run *lmstat*.



Do not execute more than one instance of *runLm* at a time. If you aren't sure whether or not the license server is running, execute *lmstat*.

Shutting Down the License Server

- > To force all applications to check in their licenses and shut down the license server, run *lmdown*.

2.5.5. Troubleshooting the FLEXlm License Manager

For general troubleshooting information, refer to the FLEXlm FAQ at www.globe-trotter.com/flmfaq.htm. It describes common problems, including the majority that have been reported to MÄK support engineers. Please consult the FAQ first when you have a problem.

Unable to Get A License

If you are unable to get a license for a MÄK application and the FLEXlm log file (*LOG*) indicates that the port is in use, it is possible that you have more than one *lmgrd* or *maklmgrd* processes running. It is also possible that an application that was using a license is thought to have terminated, but is still alive. To verify and resolve these possibilities:

1. See if more than one *lmgrd* or *maklmgrd* process is running.

On Windows 95/98/NT, check the Task Manager. To open the Task Manager, press Ctrl-Alt-Delete.

On UNIX, enter the following commands:

```
ps -aux | grep lmgrd
ps -aux | grep maklmgrd
```

If more than one instance of either process is running, write down the process ids (PID).



Some UNIX systems use *ps -aux*, others *ps -ef* to check processes. Use the command appropriate for your operating system.

2. If there are old processes present that may be using a license, kill them.

On UNIX, kill a process with the kill command, for example:

```
kill -9 1385 878
```

where 1385 and 878 are process IDs.

On Windows NT, select a process and click End Process.

It is important that you kill all the processes to ensure a clean restart for the license server.

3. If in step 2 you killed the license server, start it with the *runLm* command.

Preventing Multiple License Manager Processes

We recommend that you keep the license server running and that you not start and stop it as you run applications. If you prefer to stop the license server when you are not using it, always use *lmdown* to stop it.

Whenever you start the license manager, first run *lmstat* to be sure that there is not already a license server process running. Following this practice will ensure that you do not start multiple license servers.

License Manager Cannot Find MÄK License Management Executable

If your license manager gives you an error message indicating that it cannot find the license management executable, try putting the pathname in the license file as follows:

1. Open the license file (*mak.lic*) in a text editor.
2. Edit the license file to specify the location of the *maklmgrd* executable on the server machine. The file supplied by MÄK represents this location with a dot as shown below:

```
DAEMON maklmgrd .
```

Replace the dot with your absolute path to *maklmgrd*, for example:

```
DAEMON maklmgrd "C:\Program Files\MAK\flexlm\maklmgrd"
```



On PCs, you must enclose the path in quotes.
The drive letter must be uppercase.

2.6. Obtaining and Installing a Runtime Infrastructure (RTI)

An RTI (Run-Time Infrastructure) is a software library (and perhaps supporting executables) that implements the HLA Interface Specification. In HLA, applications exchange FOM data through RTI calls, which means that all HLA applications need to use an RTI.



Because of differences in the low-level network mechanisms used by different RTI implementations (which include, but are not limited to packet layout), applications that want to interoperate in the same federation execution must use the same RTI implementation.

Because RTIs are usually provided as dynamic libraries that implement a fixed API, a federation can often switch from one RTI implementation to another between runs (without even recompiling the applications), but during each run, all participants must agree on which RTI to use, much as they must also agree on which FOM to use.

The two RTI implementations against which all MÄK products have been tested are the MÄK RTI (which comes with all MÄK products), and the DMSO RTI (which is available from the DMSO web site). For the most recent information about the RTI versions supported by the Plan View Display, see the Product Versions page on the MÄK Technologies web site at: <http://www.mak.com/military/compatibility.html>.

2.6.1. Installing the MÄK RTI

The MÄK RTI is included with all MÄK products and you have the option of installing it as part of product installation.

UNIX

The installation process creates a link called *RTI* in each product's root directory, which points to the *makRti* directory, in which the MÄK RTI software is located.

PCs

The project files point to the default installation directory for the RTI.

Configuring Your System to Use the MÄK RTI

The RTI dynamic library needs to be located somewhere on your dynamic library search path. This path is indicated by an environment variable as follows:

- ♦ For IRIX6n32 use LD_LIBRARYN32_PATH
- ♦ On other UNIX platforms use LD_LIBRARY_PATH
- ♦ On PCs, use PATH

The MÄK RTI needs to know where to find the following configuration files:

- ♦ The FED file (*.fed*) for your federation execution (required)
- ♦ The RID file (*rid.mtl*) (optional)

Put the configuration files in the directory from which you are running, or set the environment variable `RTI_CONFIG` to the directory that contains them.

Running Applications with the MÄK RTI

You do not need an RTI server, such as the *rtiexec*, or central application to use the MÄK RTI, however you can run the *rtiexec* if you want to. Be sure the Flexlm license server is running, then start the applications, and they should be able to communicate. (For information about the license server, see “Setting Up and Using the FLEXlm License Manager”). See the *MÄK RTI Reference Manual* for information about the *rtiexec* and changing the networking parameters such as multicast addresses and UDP ports.

Documentation for the MÄK RTI

The documentation for the MÄK RTI consists of:

- ♦ *MÄK RTI Reference Manual*
- ♦ The documentation set for the DMSO RTI. Aside from the installation and configuration instructions, the documentation for the DMSO RTI also applies to the MÄK RTI. It is included on the MÄK product CD.

2.6.2. Installing the DMSO RTI

You can get the DMSO RTI from DMSO's web site (<http://hla.dmsolm.com>). Click on Software Distribution Center, where you can register for an account, then later download the RTI software. Documentation for the DMSO RTI is provided on the MÄK product CD because the documentation applies to the MÄK RTI as well.

Consult the DMSO RTI Release Notes to determine version information and whether or not any patches are required.

1. Install the RTI according to the instructions provided.
2. Copy the *VR-Link.fed* file from the MÄK products root directory to the DMSO RTI's config directory.

Configuring Your System to Use the DMSO RTI

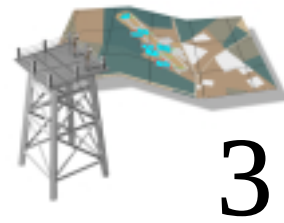
To configure the environment:

- > On UNIX systems, source the *RTI.env* file.
- > On Windows systems, set environment variables and add the RTI location to your path. The RTI's README file contains detailed instructions.

Running Applications Using the DMSO RTI

To use the DMSO RTI, you need to run the RTI executive (*rtiexec*). This executable resides in the RTI's *bin/platform_name* directory (UNIX) or in the RTI's *bin* directory (PCs). Run *rtiexec* only on one machine, regardless of how many HLA applications you are running.

Once the *rtiexec* is running, you can run HLA applications.



Getting Started

The Plan View Display has a control window, a display window, and a set of dialog boxes that let you configure the display. This chapter explains how to start and stop the Plan View Display and describes the following aspects of the graphical user interface:

Starting the Plan View Display On A PC	3-2
Starting the Plan View Display in UNIX	3-3
Creating a Plan View	3-4
Exiting the Plan View Display	3-5
The Mak PVD Window	3-5
The Plan View Window	3-6
Plan View Menus	3-6
The Plan View Information Bar	3-8
Load Database Dialog Box	3-9
Save Database Dialog Box	3-10
Settings Dialog Box	3-11
Settings Dialog Box, Display Tab	3-12
Settings Dialog Box, Control Tab	3-14
Entity List Dialog Box	3-16
Entity Information Dialog Box	3-18
Stealth Control Dialog Box	3-20
Intervisibility Settings Dialog Box	3-22
Entity Labels Dialog Box	3-23

3.1. Starting the Plan View Display On A PC

To start the Plan View Display on a PC:

1. On the Windows95/98/NT Start menu, choose Programs → MÄK Technologies → Plan View Display *protocol*. (The protocol is DIS3, DIS5, or HLA.) The Mak PVD window opens ([Figure 3-1](#)).

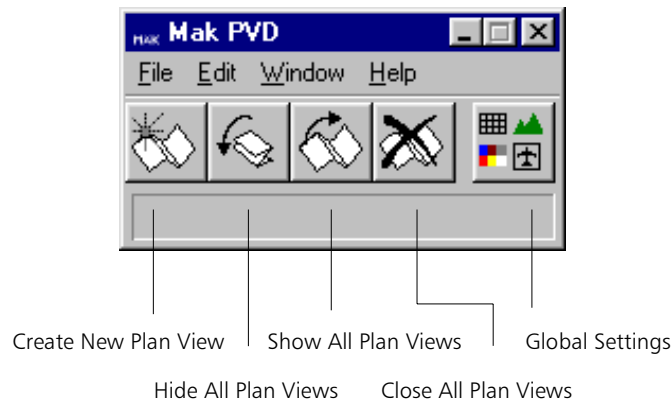


Figure 3-1. Mak PVD window

Notes

- ♦ If you open Plan View Display HLA, be sure that you have installed an RTI and configured your system to use it. See [“Obtaining and Installing a Runtime Infrastructure \(RTI\),”](#) on page 2-14.
 - ♦ When you start the Plan View Display, it loads the bitmap symbols used to represent objects. This can take a minute or so.
2. After the bitmaps finish loading, you can create a new Plan View window, or you can load a saved layout. These tasks are described in the following sections.

3.2. Starting the Plan View Display in UNIX

To start the Plan View Display in UNIX:

1. In a command window, change directory to the Plan View Display subdirectory for the protocol you want to use (DIS or HLA). For example, to run the HLA version of the Plan View Display, change to `./binRPR`.

2. Enter the `Pvd` command:

`Pvd`

For a list of command line options, see [“Startup Options,”](#) on page 6-2.

Notes

- ♦ If you run the HLA version of the Plan View Display, be sure that you have installed an RTI and configured your system to use it. See [“Obtaining and Installing a Runtime Infrastructure \(RTI\),”](#) on page 2-14.
 - ♦ When you start the Plan View Display, it loads the bitmap symbols used to represent objects. This can take a minute or so.
3. After the bitmaps finish loading, you can create a new Plan View window, or you can load a saved layout. These tasks are described in the following sections.

3.2.1. Creating a Plan View

A Plan View window displays a terrain database and any entities that are simulated in an exercise.

To create a new Plan View:

1. In the Mak PVD window, choose File → New Plan View. The Load Database dialog box opens.
2. In the Load Database dialog box, select a database to load. (The databases shipped with the Plan View Display are in `./data/terrain`.) You can filter the display to list databases of a particular type. For information about the types of databases supported, see [“Working With Terrain Databases,”](#) on page 4-14.
3. Click OK. A database is loaded and a Plan View window opens with the database displayed.

When you load a terrain database that is not in MÄK Terrain Format, the Plan View Display converts it to MÄK Terrain Format as part of the load process. Terrain databases that are in MÄK Terrain Format load more quickly than databases that must be converted. You can save non-MÄK Terrain Format databases in MÄK Terrain Format to speed up loading during later uses. For information about saving terrain databases in MÄK Terrain Format, see [“Saving Terrain Databases in MÄK Terrain Format,”](#) on page 4-14.

[Figure 3-2](#) illustrates a Plan View window with a terrain database loaded.

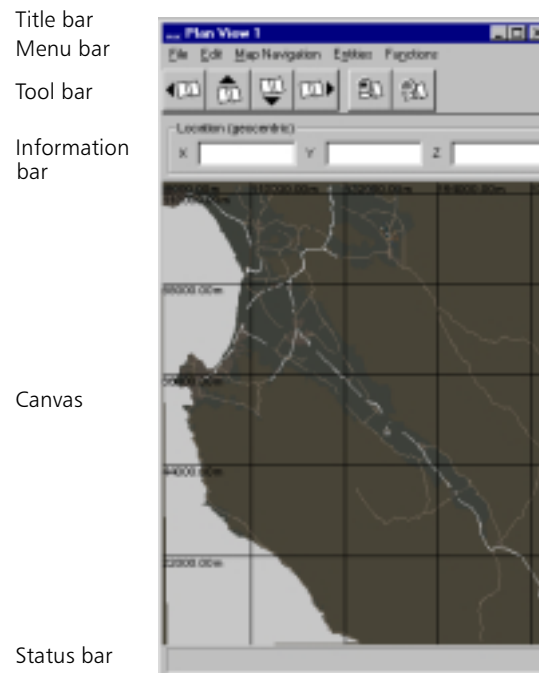


Figure 3-2. Plan View window

3.3. Exiting the Plan View Display

To exit the Plan View Display:

> In the Mak PVD window, choose File → Exit.

The Mak PVD window and all Plan View windows close, and the application terminates.

3.4. The Mak PVD Window

The Mak PVD window ([Figure 3-1](#)) is essentially a free-floating menu and button bar that lets you control high-level functions of the Plan View Display. The Mak PVD window is displayed automatically when you start the Plan View Display. [Table 3-1](#) lists the menus, menu options, and purpose of each option.

Table 3-1: Mak PVD window menus

Menu	Option	Purpose
File	New Plan View	Opens a new Plan View window.
	New Global Geodetic View	Displays a geodetic view with no land masses; essentially, a white screen. See “Using A Global Geodetic View,” on page 4-15.
	Close All Windows	Closes all Plan View windows.
	Load Layout	Loads a saved layout of Plan View windows. See “Saving and Loading Layouts,” on page 4-8.
	Save Layout	Saves the current layout of Plan View windows to a file.
	Exit	Closes the Mak PVD window and all Plan View windows and terminates the application.
Edit	Global Settings	Change settings for all open Plan View windows.
Window	Hide All Windows	Hides all Plan View windows.
	Show All Windows	Displays all hidden Plan View windows.
	List of Plan View windows (PC only)	Lists each Plan View window. Select a Plan View window to open it if it is hidden or give it focus if it is open.
Help	About	Provides information about this release of the Plan View Display.

3.5. The Plan View Window

The Plan View window ([Figure 3-2](#)) is the display window for the Plan View Display. You can have multiple Plan View windows open. Each Plan View has an ID number that is displayed in the title bar. A Plan View window has the following parts:

- ♦ Title bar - contains the ID for this Plan View
- ♦ Menu bar - menu options allow you to set all parameters for the Plan View
- ♦ Tool bar - buttons allow you to pan and zoom the display
- ♦ Information area - provides coordinate information for the cursor and last selected entity or ground point
- ♦ Canvas - displays the exercise
- ♦ Status bar - displays hints for selected options.

For the procedure for opening a new Plan View window, see [“Creating a Plan View,”](#) on page 3-4.

3.5.1. Plan View Menus

[Table 3-2](#) lists the menus on the Plan View window menu bar and their options.

Table 3-2: Plan View window menus

Menu	Option	Description
File	Load Terrain Database	Allows you to load the terrain database for the Plan View.
	Save Terrain Database	Allows you to save the loaded database in MÄK Terrain Format (<i>.gdb</i>).
	Close	Closes the Plan View and all dialog boxes associated with it.
Edit	Settings	Lets you change settings for this Plan View.
Map Navigation	Show Entire Database	Zooms the view out to show the entire database. This is the view shown when you create a new Plan View window.
	Pan Up	Moves the Plan View to view the area towards the top of the screen.
	Pan Down	Moves the Plan View to view the area towards the bottom of the screen.
	Pan Left	Moves the Plan View to view the area towards the left of the screen.
	Pan Right	Moves the Plan View to view the area towards the right of the screen.

Table 3-2: Plan View window menus

Menu	Option	Description
	Zoom In	Shows a smaller section of the Plan View in greater detail.
	Zoom Out	Shows a larger section of the Plan View in less detail.
	Move to Selection	Centers the Plan View on the selected entity or group of entities.
	Toggle Track Selection	Toggles tracking of selected entities on or off. Tracked entities are kept in the center of the Plan View. See “Tracking Entities in the Plan View Display,” on page 5-7.
	Go to Next Entity	Tracks the next entity in the entity list, starting at the top if no entities are selected, or at the first selected entity in the list if one or more entities are selected. See “Moving to the Next and Previous Entities,” on page 4-3.
	Go to Previous Entity	Tracks the previous entity in the entity list, starting at the first selected entity. If no entities are selected, starts with the last entity in the list. See “Moving to the Next and Previous Entities,” on page 4-3.
Entities	Entity List	Opens the Entity List dialog box, a list of all entities in the exercise.
	Invert Selection	Selects all entities that are currently unselected and deselects all entities that are currently selected.
	Deselect All	Deselects all entities.
Functions	Stealth Control	Provides information about a selected MÄK Stealth and allows you to control it.
	Redraw Map	Refreshes the Plan View.

3.5.2. The Plan View Information Bar

The Plan View window information bar (Figure 3-3), located below the toolbar, displays location information and selection information.

The location of the last selected entity or the last clicked ground point is displayed in both geocentric and database coordinates.

If a ground point is clicked, the information bar displays the soil type and terrain height.

i

Soil type is displayed only if it is specified in the database. Only CTDB databases specify soil type. However, if you save a CTDB database in MÄK Terrain Format, it retains the soil type information.

If an entity is selected, it displays the global object designator and markings instead of soil type and terrain height.



Figure 3-3. The Plan View information bar

3.5.3. Load Database Dialog Box

The Load Database dialog box is a standard Open dialog box for your windowing system. You can load terrain databases that have the following formats:

- ♦ MÄK terrain (.gdb).
- ♦ Open Flight (.flt) (PC only) If you want to load an FLT database on UNIX, convert it to the GDB format. For more information, see [“Saving Terrain Databases in MÄK Terrain Format,”](#) on page 4-14.
- ♦ CTDB (.c4b).
- ♦ MÄK Terrain Metafile (.map).

i

All databases are converted to MÄK Terrain Format when they are loaded.

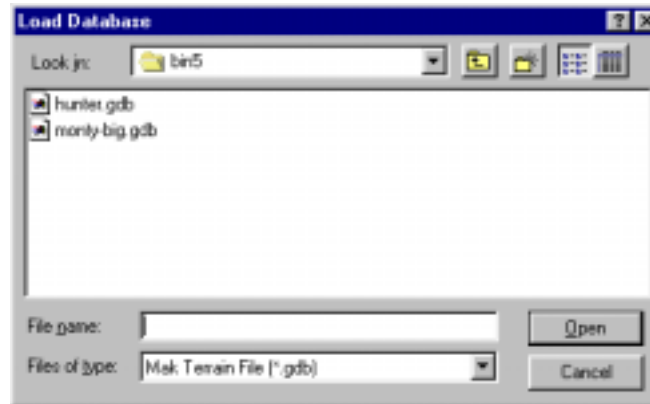


Figure 3-4. Load Database dialog box

For information about loading databases, see [“Loading A New Terrain Database,”](#) on page 4-5.

3.5.4. Save Database Dialog Box

The Save Database dialog box is a standard Save dialog box for your windowing system. It allows you to save your database in the MÄK Terrain Format (.gdb).

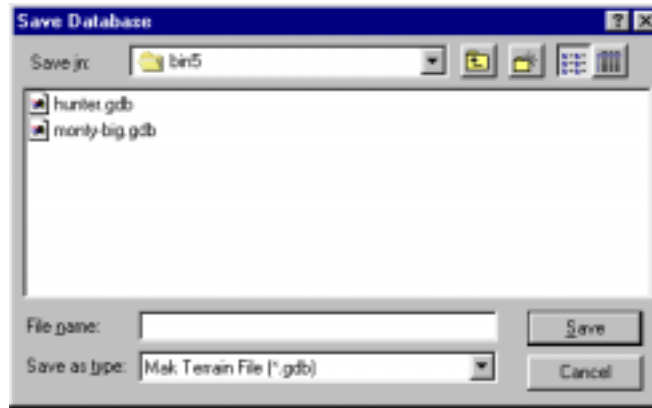


Figure 3-5. Save Database dialog box

For more information about saving databases, see [“Saving Terrain Databases in MÄK Terrain Format,”](#) on page 4-14.

3.6. Settings Dialog Box

The Settings dialog box (Figure 3-6) lets you define settings for a Plan View window. You can change settings globally, that is, for all open Plan View windows, or for a specific Plan View window.

- > To open the Settings Dialog Box for global settings, in the Mak PVD window, choose Edit → Settings.
- > To change settings for a particular Plan View window, in the Plan View window, choose Edit → Settings.

For more information about changing settings, see “Changing Settings,” on page 6-14.

Settings are divided into Display, Control, and Color tabs.

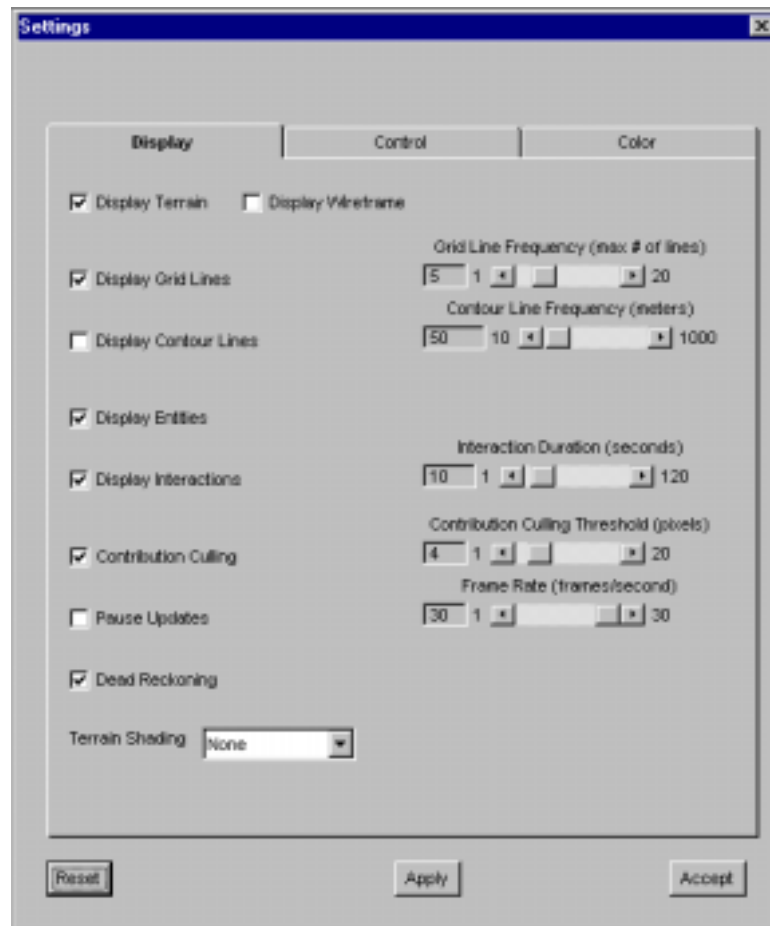


Figure 3-6. Settings dialog box, Display tab

[Table 3-3](#) describes the buttons common to the tabs on the Settings dialog box.

Table 3-3: Settings dialog box buttons

Button	Description
Reset	Resets display to the settings that were in effect when you opened the Settings dialog box.
Apply	Applies a change and leaves the dialog box open.
Accept	Closes the dialog box and applies all changes.

3.6.1. Settings Dialog Box, Display Tab

The Display tab on the Settings dialog box ([Figure 3-6](#)) toggles a number of aspects of the display on or off and lets you set values for other settings. [Table 3-4](#) describes the fields and their meanings.

Table 3-4: Settings dialog box, Display tab

Control	Description	Details
Display Terrain	Toggle display of the terrain on or off.	section 6.3.1
Display Grid Lines	Toggle grid lines on or off.	section 6.3.4
Display Contour Lines	Toggle terrain contour lines on or off.	section 6.3.5
Display Entities	Toggle display of entities on or off.	section 6.3.1
Display Interactions	Toggle display of interactions between entities, such as munitions fire, on or off.	section 6.3.6
Contribution Culling	Suppresses display of individual terrain features that are too small to be seen, thereby speeding up performance.	section 6.3.9
Pause Updates	Pauses updates to the Plan View.	section 6.3.1
Dead Reckoning	Toggles dead reckoning of entities on or off.	section 6.3.1
Terrain Shading	Selects shading based on the criteria selected from the drop-down list.	section 6.3.8
Display Wireframe	Toggle display of the terrain as wireframe on or off.	section 6.3.1
Grid Line Frequency	Specify the approximate number of grid lines in the longest dimension of the screen.	section 6.3.4
Contour Line Frequency	Specify the interval between contour lines in meters.	section 6.3.5

Table 3-4: Settings dialog box, Display tab

Control	Description	Details
Interaction Duration	Specifies in seconds how long an interaction such as an explosion stays on screen before it is removed.	section 6.3.6
Contribution Culling Threshold	Sets the threshold, in pixels, at which terrain features are not displayed.	section 6.3.9
Frame Rate	Sets the rate in frames per second at which the Plan View refreshes itself.	section 6.3.7

3.6.2. Settings Dialog Box, Control Tab

The Control tab on the Settings dialog box ([Figure 3-7](#)), lets you change settings that affect the Plan View window display and the MÄK Stealth. [Table 3-5](#) describes these controls.



Figure 3-7. Settings dialog box, Control tab

Table 3-5: Settings dialog box, Control tab

Control	Description	Details
Zoom Step Size	Specify the increments for zooming as a percentage of the screen.	section 6.3.2
Pan Step Size	Specify the increments of movement when panning as a percentage of the screen.	section 6.3.3
Stealth Linear Increment	Specify the linear increment used for Stealth control in meters.	
Stealth Angular Increment	Specify the angular increment used for Stealth control in radians.	

3.6.3. Settings Dialog Box, Color Tab

The Color tab of the Settings dialog box (Figure 3-8) lets you change the colors used for the intervisibility display. You can change the default colors in the `pvd.set` configuration file. For more information, see “Changing Colors,” on page 6-18.



Figure 3-8. Settings dialog box, Colors tab

Table 3-6 describes the options on the Color tab of the Settings dialog box.

Table 3-6: Settings dialog box, Control tab

Control	Description
Intervisibility Range Circle	Sets the color for the intervisibility range circle.
Intervisibility Fan Fill	Sets the color for the intervisibility fan.
Intervisibility Lines - Visible	Sets the color for intervisibility lines for entities that an entity can see or be seen by.
Intervisibility Lines - Obstructed	Sets the color for intervisibility lines for the portion of a sight line that is obstructed.

3.7. Entity List Dialog Box

The Entity List dialog box (Figure 3-9) lists all entities in the exercise and allows you to select them. Selecting an entity in the entity list has the same effect as selecting it in the display. The Entity List dialog box lets you create groups of entities.

- > To open the Entity List dialog box, in a Plan View window, choose Entities → Entity List.

For information about manipulating entities, see Chapter 5, *Working With Entities and Other Objects*.

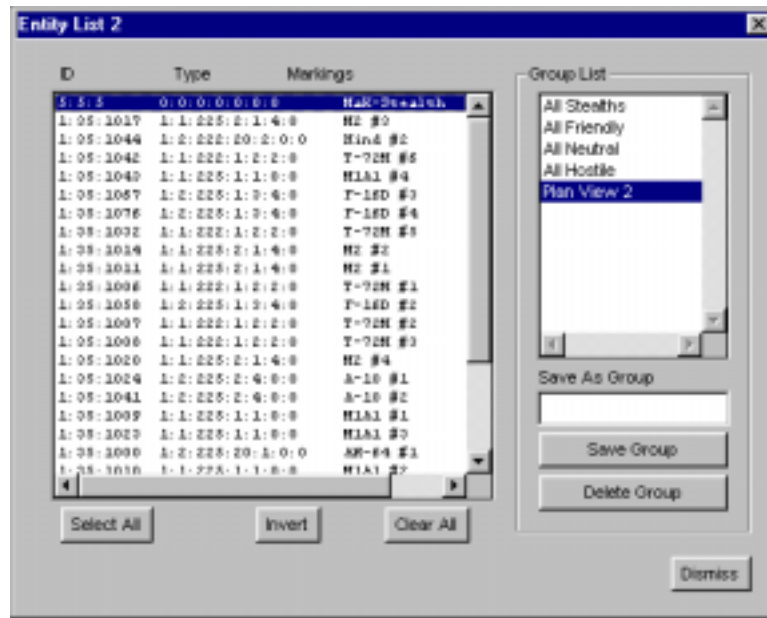


Figure 3-9. Entity List dialog box

Table 3-7 describes the controls in the Entity List dialog box.

Table 3-7: Entity List dialog box fields

Field	Description
Entity List	Lists all entities in the exercise with global object designator, type, and markings information
Group List	Lists all groups.
Save As Group	Specify the name of a group to assign to the selected entities.
Save Group	Save the selected entities as the group in the Group Name text box.
Delete Group	Deletes the group selected in the Group List window. Does not delete the entities that make up the group.

Table 3-7: Entity List dialog box fields

Field	Description
Select All button	Selects all entities in the list.
Invert button	Deselects all currently selected entities and selects all currently unselected entities.
Clear All button	Deselects all entities in the list.

3.8. Entity Information Dialog Box

The Entity Information dialog box (Figure 3-10) displays information about the selected entity. The information is not editable.

- > To open the Entity Information dialog box, right-click on an entity and select Entity Info from the menu.

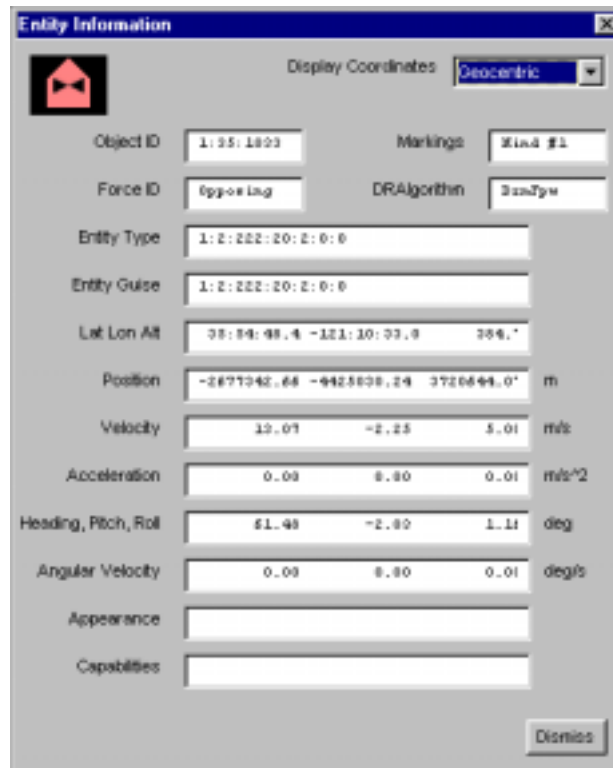


Figure 3-10. Entity Information dialog box

Table 3-8 lists the fields in the Entity Information dialog box.

Table 3-8: Entity Information dialog box fields

Field	Description
Entity Icon	Displays the icon for this entity.
Object ID	Global object designator.
Force ID	A description of the entity such as friendly, hostile, neutral, and so on.
Entity Type	A DIS enumeration that describes the vehicle type.
Entity Guise	A DIS enumeration that describes how the entity looks to the opposing force.

Table 3-8: Entity Information dialog box fields

Field	Description
Lat Lon Alt	Displays the location of the entity.
Position	The locations of the entity using the display coordinates. The coordinate system is identified in the Display Coordinates field.
Velocity	Speed of the entity in meters per second.
Acceleration	The rate at which the speed of the entity is increasing.
Heading, Pitch, Roll	Displays the heading, pitch, and roll of the entity.
Angular Velocity	The speed at which the entity is rotating.
Appearance	A string that determines various aspects of how an entity looks, such as, an open hatch on a tank.
Capabilities	A description of the capabilities of the entity. Refer to the DIS or HLA protocol documentation for a definition.
Display Coordinates	Drop-down list lets you specify the type of coordinate system to use.
Markings	A string used to carry extra information about the entity.
DR Algorithm	The dead reckoning algorithm.
Dismiss	Closes the dialog box.

For more information, see the *VR-Link Developer's Guide*.

3.9. Stealth Control Dialog Box

The Stealth Control dialog box (Figure 3-11) displays the position and view mode of the MÄK Stealth. You can attach the Stealth to an entity or a point in the terrain. You can also use controls in this dialog box to move the Stealth.



Before you can open the Stealth Control dialog box, you must select a Stealth.

The Stealth Control dialog box is not tied to a particular Plan View window. However, it is tied to a particular Stealth, usually running on a remote computer.

To open the Stealth Control dialog box:

1. In a Plan View window, or in the Entity List dialog box, select the icon of the Stealth that you want to control.
2. Choose Function → Stealth Control.

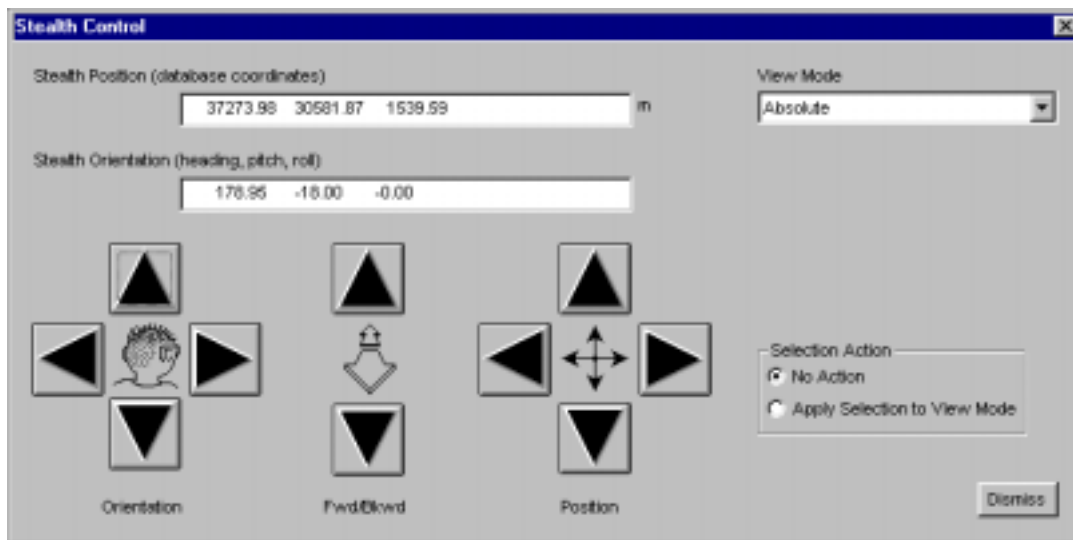


Figure 3-11. Stealth Control dialog box

Table 3-9 lists the fields and controls in the Stealth Control dialog box.

Table 3-9: Stealth Control dialog box controls

Field	Description
Stealth Position	The location of the eyepoint using UTM coordinates.
Stealth Orientation	Displays the heading, pitch, and roll of the attached body
View Mode	Select a mode from the drop-down list.

Table 3-9: Stealth Control dialog box controls

Field	Description
Orientation	Changes the view relative to the eyepoint in the context of the current view mode.
Fwd/Bkwd	Changes the depth perception of the view in the context of the current view mode.
Position	Moves the view relative to the terrain in the context of the current view mode.
No Action	Do not attach to selected objects.
Apply Selection to View Mode	Attach to selected entities or the ground point in the context of the view-mode.

You can have multiple Stealth Control dialog boxes open.

For information about how to use the Stealth from the Plan View Display, see [“Controlling the Stealth from the Plan View Display,”](#) on page 4-9.

3.10. Intervisibility Settings Dialog Box

The Intervisibility Settings dialog box (Figure 3-12) allows you to set multiple intervisibility options at one time rather than toggling them individually from the intervisibility menu.

To open the Intervisibility Settings dialog box for an entity or marker:

1. Place the cursor over the entity or marker.
2. From the right-click menu, choose Intervisibility → Intervisibility Settings.

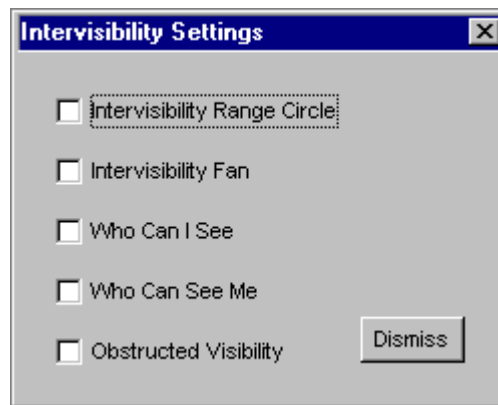


Figure 3-12. Intervisibility Settings dialog box

Table 3-10 describes the check boxes in the Intervisibility Settings dialog box.

Table 3-10: Intervisibility Settings dialog box fields

Field	Description
Intervisibility Range Circle	Displays a circle that encloses the area within which the Plan View Display calculates intervisibility.
Intervisibility Fan	Displays cross-hatching in areas of terrain that are not visible from the selected entity.
Who Can I See?	Displays intervisibility lines for the entities that the selected entity can see.
Who Can See Me?	Displays intervisibility lines for the entities that can see the selected entity.
Obstructed Visibility	Displays intervisibility lines for the entities that the selected entity cannot see and indicates where vision is obstructed.

For more information about intervisibility, see “[Displaying Intervisibility](#),” on page 5-11.

3.11. Entity Labels Dialog Box

The Entity Labels dialog box (Figure 3-13) allows you to specify information that is displayed on screen for an entity either graphically or using text. You can apply labels to individual entities or to groups of entities.

To open the Entity Labels dialog box:

1. Place the cursor over an entity or one of a group of selected entities.
2. From the right-click menu, choose Entity Labels.

For more information about entity labels, including the procedure for specifying labels for multiple entities, see “[Displaying Entity Labels](#),” on page 5-9.

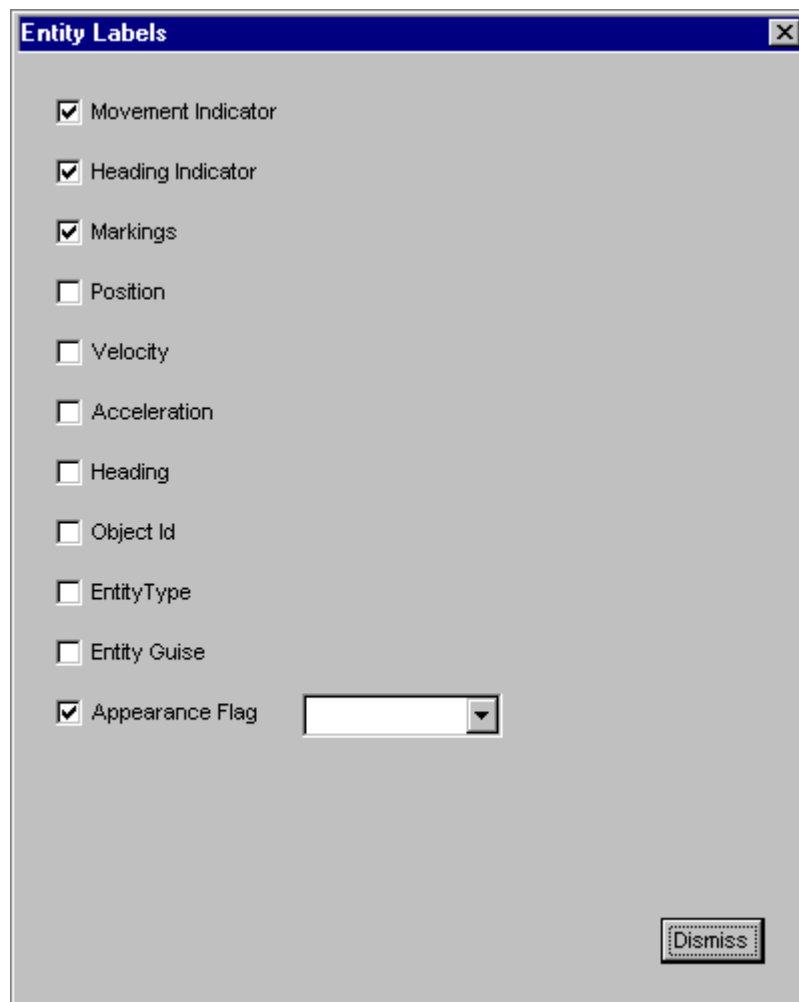
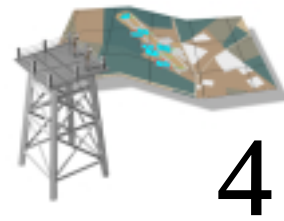


Figure 3-13. Entity Labels dialog box

Table 3-11 describes the options in the Entity Labels dialog box.

Table 3-11: Entity Labels dialog box fields

Field	Description
Movement Indicator	A line that points in the direction the entity is moving. Only visible when the entity is moving.
Heading Indicator	A line that points in the direction the entity is facing.
Markings	A string used to carry extra information about the entity.
Position	The locations of the entity in geocentric coordinates.
Velocity	Speed of the entity in meters per second.
Acceleration	The rate at which the speed of the entity is increasing, in meters/sec ² .
Heading	Displays the heading of the entity.
Object ID	Global object designator.
Entity Type	A DIS enumeration that describes the vehicle type.
Entity Guise	A DIS enumeration that describes how the entity looks to the opposing force.
Appearance Flag	A string that determines various aspects of how an entity looks, such as, an open hatch on a tank.
Dismiss	Closes the dialog box.



Using the Plan View Display

This chapter explains how to navigate and manipulate a Plan View window. It covers the following topics:

Navigating a Plan View Map.....	4-2
Panning the Plan View Window.....	4-2
Zooming the Plan View Window.....	4-2
Moving to An Entity on A Plan View Window.....	4-3
Loading A New Terrain Database.....	4-5
Using Multiple Plan View Windows.....	4-6
Hiding and Displaying Plan View Windows.....	4-6
Closing Plan View Windows.....	4-7
Hiding Plan View Windows Versus Closing Them.....	4-7
Saving and Loading Layouts.....	4-8
Saving A Plan View Layout.....	4-8
Loading A Plan View Layout.....	4-8
Controlling the Stealth from the Plan View Display.....	4-9
Configuring the Stealth to Work with the Plan View Display.....	4-9
Controlling the Stealth.....	4-10
The No Action Radio Button.....	4-13
Working With Terrain Databases.....	4-14
Saving Terrain Databases in MÄK Terrain Format.....	4-14
Using A Global Geodetic View.....	4-15
Terrain Metafiles.....	4-16
Creating Terrain Metafiles.....	4-17

4.1. Navigating a Plan View Map

The Plan View Display offers the following methods to change your view (the extents of the database visible in a Plan View window) of the exercise:

- ♦ Pan the map
- ♦ Zoom in or out
- ♦ Move the Plan View window to show a particular entity.

4.1.1. Panning the Plan View Window

You can pan the Plan View window in four directions. In the Settings dialog box, you can set the increment by which the map is panned. See [“Setting Panning Increments,”](#) on page 6-15.

To pan the Plan View window, use one of the following methods:

- > Click on the appropriate pan button in the Plan View toolbar.
- > In the Map Navigation menu, choose the Pan command for the direction in which you want to pan the Plan View window.

Centering the Window On A Point

- > To center the window on a point, Ctrl-Shift-click on the point.

4.1.2. Zooming the Plan View Window

You can zoom in and out on the Plan View window to vary the level of detail that you see. Use one of the following methods:

- > Click the Zoom In or Zoom Out buttons on the Plan View window toolbar.
- > In the Map Navigation menu, choose the Zoom command for the direction you want to zoom.

The Plan View zooms in or out around the center point of the canvas.

Zooming In On A Point

To zoom in or out on a point, use one of the following keyboard methods:

- Shift-left click to zoom in.
- Shift-right click to zoom out.

Zooming in On An Area of A Window

- To zoom in on a specific area, hold down the shift key and drag a box around the area which you want to zoom in on.



You cannot zoom in any closer than making the screen 0.2 x 0.2 meters.

In the Settings dialog box, you can set the increment by which the map is zoomed. See [“Setting Zoom Increments,”](#) on page 6-15.

4.1.3. Resizing Plan View Windows

In addition to zooming and panning the canvas, you can resize a Plan View window to change your view of the exercise. If a Plan View window is showing the entire database and you resize the window, it may not show the entire database, depending on how the proportions of the resized window compare to the proportions of the database.

4.1.4. Moving to An Entity on A Plan View Window

If you select one or more entities or a group, you can have the Plan View window move to them automatically.

- > To move to a selected entity, choose Map Navigation → Move to Selection.

The Plan View window moves to show the entity. If the entity is a group, the Plan View window may zoom to display the whole group. For information about selecting entities, see [“Selecting Entities,”](#) on page 5-2.

Moving to the Next and Previous Entities

You can move the focus from one entity to the next in the order in which they are listed in the Entity List dialog box.

- > To move to the next entity in the list, choose Map Navigation → Go To Next Entity.
- > To move to the previous entity in the list, choose Map Navigation → Go To Previous Entity.



Regardless of how many entities you have selected, choosing one of these options changes the selection to the next or previous entity.

[Table 4-1](#) explains how these options work.

Table 4-1: Moving to next and previous entities

Selection	Go To Next Entity	Go To Previous Entity
No entities selected	Starts at the top of the Entity List and moves down the list.	Starts at the bottom of the Entity List and moves up the list.
One entity selected	Moves to the next entity in the Entity List after the selected entity. If the last entity in the Entity List is selected, moves to the first entity in the list.	Moves to the entity in the Entity List that is above the selected entity. If the first entity in the Entity List is selected, moves to the last entity in the list.
Multiple entities selected	Moves to the next entity in the Entity List after the selected entity that is closest to the top of the list, regardless of whether or not the next entity is selected.	Moves to the entity in the Entity List above the selected entity that is closest to the top of the list, regardless of whether or not the previous entity is selected.

4.2. Loading A New Terrain Database

When you create a Plan View window, you specify a database for it to display. At a later point, you can load a different database into the window.

To load a terrain database:

1. In the Plan View window, choose File → Load Terrain Database. The Load Database dialog box opens.
2. Select the database you want to load.
3. Click Open.



When you load a database, it replaces the currently loaded database.

4.3. Using Multiple Plan View Windows

You can open multiple Plan View windows. This allows you to view various parts of an exercise at the same time or view the same portion of an exercise using different configuration settings.

Each Plan View window is identified by an ID in the title bar. The list of Plan View windows on the Window menu of the Mak PVD window lists the ID number for each window.

4.3.1. Hiding and Displaying Plan View Windows

You have the following options for hiding and displaying Plan View windows:

- ♦ Minimize a window - Minimized windows are visible as icons in your operating system's format
- ♦ Hide all windows - Hides all open Plan View windows. Hidden windows are not visible as icons.
- ♦ Restore a minimized window.
- ♦ Display a specific hidden window (PC only) or all hidden windows.



On a PC the Hide All Windows and Show All Windows commands do not affect Plan View windows that have been minimized.

To understand why you might choose one method over another, see [“Hiding Plan View Windows Versus Closing Them,”](#) on page 4-7.

Minimizing and Restoring Plan View Windows

You minimize and restore Plan View windows using the standard methods for your operating system.

- > To minimize a Plan View window, click the minimize button in the title bar.
- > To restore a minimized window, click its icon.

Hiding and Displaying Plan View Windows

You can only hide windows all at once. On a PC you can display them individually. On PCs and UNIX, you can display them all together.

- > To hide all Plan View windows, in the Mak PVD window, choose Window → Hide All Windows.
- > To display a hidden window, select it from the list in the Window menu on the Mak PVD window. (PC only)
- > To display all hidden windows, in the Mak PVD window, choose Window → Show All Windows.

4.3.2. Closing Plan View Windows

You can close Plan View windows individually or all at once, as follows:

- > To close an individual Plan View window, choose File → Close from the menu for that window.
- > To close all Plan View windows at once, in the Mak PVD window choose File → Close All Windows.

4.3.3. Hiding Plan View Windows Versus Closing Them

As described in [“Hiding and Displaying Plan View Windows,”](#) on page 4-6, you can hide or close Plan View windows. Either method has benefits and costs, as follows:

- ♦ Hidden or minimized windows redisplay very quickly, but because they are in memory, keeping them open but hidden could affect performance.
- ♦ When you close windows, you free up memory, thereby possibly improving performance, but when you open a new Plan View, you must wait for the database to load.

4.4. Saving and Loading Layouts

A terrain database can be quite large and the action in your exercise may take place in only one part of it, or in several distinct parts of it. This means that each time you start a new Plan View window, you will have to pan and zoom to get the display to focus on the part of the terrain that you are interested in. You can save yourself from repeating this effort by saving the layout of your Plan View windows. Once you've saved the layout, instead of starting a session by creating a new Plan View, you can open the layout and be ready to observe an exercise immediately.

When you save a window layout, the Plan View Display also saves the settings for the windows. If you open a saved layout and make changes to settings or the Plan View windows, these changes are not saved unless you explicitly save the layout again.

Layout files are protocol-independent. If you are running the HLA version of the Plan View Display and save a layout, you can open that layout using the DIS version. You can go from DIS to HLA, as well.

4.5. Controlling the Stealth from the Plan View Display

You can control a MÄK Stealth running on the same computer or another computer from the Plan View Display. Stealths are listed in the Entity List dialog box and displayed as an icon in the Plan View window. If you select a Stealth, you can control it from the Stealth Control dialog box in the Plan View window. The Stealth Control dialog box in the Plan View Display is virtually identical to the Stealth Control dialog box in the MÄK Stealth.

For complete information about controlling the Stealth, particularly for descriptions of view modes, refer to the *MÄK Stealth User's Guide*.

4.5.1. Configuring the Stealth to Work with the Plan View Display

To control the MÄK Stealth from the Plan View Display, you must configure each Stealth you want to control to transmit state information to other applications.

DIS only

For DIS, you must set the exercise ID for the Stealth to the exercise ID that the Plan View Display specifies.

HLA only

For HLA, you must specify the federation execution name that the Plan View Display is a part of.

Making the Stealth Visible to Other Applications

The Stealth can send state update messages to make its state information known to other applications. This makes it possible for a Plan View Display to represent the Stealth as an icon, and for the Stealth to be controlled from the Plan View Display.

- > To make the Stealth visible on the network, use the *-b* option at startup (on SGI).
- > To make this action the default (on SGI or PC), edit the **broadcast** parameter in *params.mtl* (located in the Stealth *./config* directory).

Set the argument to 1 for “send”.

DIS only

In DIS you can send Stealth information to just one other application, rather than to everyone in the exercise. To send the PDUs point-to-point (unicast), use the *-A* command-line option (SGI) or the **addressString** parameter (SGI or PC). (You still need to use the *-b* command-line option.)

Setting the Stealth's Exercise ID (DIS)

You must set the Stealth exercise ID to match that of the Plan View Display. See [“Startup Options for DIS Exercises,”](#) on page 6-6 for information about how to set the exercise ID for the Plan View Display.

- > To restrict the Stealth to one specific exercise, use the Exercise ID list in the Stealth Control dialog box ([Figure 4-1](#)).

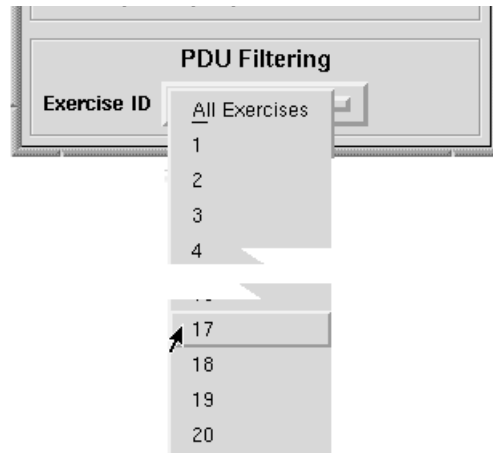


Figure 4-1. Selecting an exercise ID for the Stealth

By default, the Stealth listens to exercise 0 (zero), meaning all exercises.

- > To change the default, edit the argument to the `exerciseld` parameter in `params.mtl` (located in the Stealth `./config` directory).

When the Stealth sends data onto the network, the selected exercise ID is included in its PDUs. When All Exercises is selected, the PDUs contain 0 (zero) as the exercise ID.

Setting the HLA Federation Execution Name

For a Stealth to interoperate with the Plan View Display in HLA, they must both be in the same federation execution. This means they must both use the same FED file and RTI.

To specify the federation execution for the Plan View Display, use the `-x` startup operation. For more information, see [“Startup Options for HLA Federations,”](#) on page 6-4.

4.5.2. Controlling the Stealth

This section provides a brief discussion of controlling the Stealth from the Stealth Control dialog box.

To control a Stealth:

1. Select a Stealth (as you would any entity in the Plan View window. See [“Selecting Entities,”](#) on page 5-2).
2. Choose Functions → Stealth Control. The Stealth Control dialog box opens.
3. Control the Stealth as required.

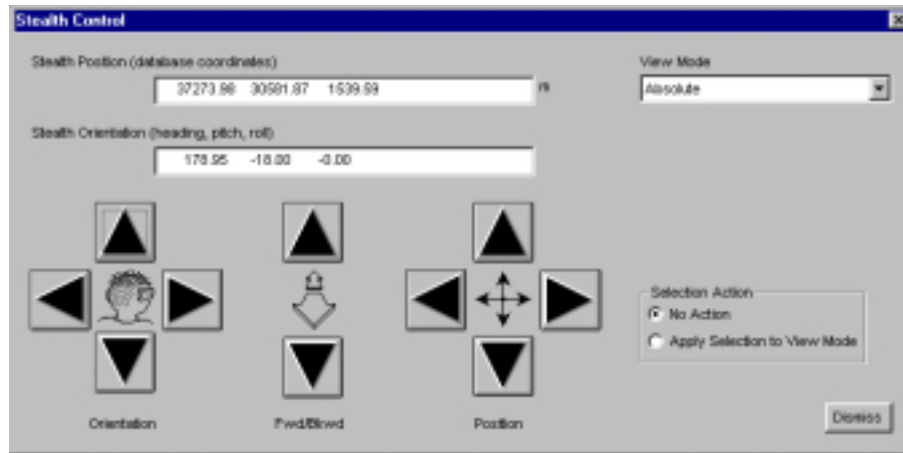


Figure 4-2. Stealth Control dialog box

Orientation, Perspective, and Position

The arrow buttons in the Stealth Control dialog box control the view of the attached entity. The effect of using these controls depends on the view mode you are using.

Changing position is similar to panning in the Plan View Display, it moves the view-point with respect to the attached entities and the terrain. The altitude, longitude, or latitude may change.

The Forward/Backward controls are similar to zooming.

The Orientation control has the effect of standing in place and rotating horizontally or vertically. The entity and terrain do not change position.

Attaching To and Tracking Entities

To attach the Stealth to an entity:

1. Select the Stealth to which you want to attach an entity.
2. Open the Stealth Control dialog box (Functions[]Stealth Control.)
3. Click the Apply Selection to View Mode button.
4. Select an entity, entities, or a group.

The Stealth attaches to the entity, entities, or group and focuses on it. If you change the selection, the Stealth tracks the new selection. If you want to keep the Stealth attached to a particular entity or group of entities, regardless of the selection in the Plan View window, see [“Keeping the Stealth Attached to A Particular Entity,”](#) on page 4-12.

Keeping the Stealth Attached to A Particular Entity

To keep the Stealth attached to a particular entity:

1. Attach the Stealth to an entity as described in [“Attaching To and Tracking Entities,”](#) on page 4-11.
2. In the Stealth Control dialog box, select the No Action option.

When you select the No Action option, the Stealth does not switch its tracking to new entities as it does when the Apply Selection to View Mode option is selected.

Changing the View Mode

View modes determine the relationship of the viewpoint to the attached entity or entities. Your choice of view mode affects how the orientation and position controls work. The default view mode is absolute mode.

- > To select the view mode, display the View Mode drop-down list and select the mode you want.

If you attach to one entity, you can be in the following view-modes:

- ♦ Track
- ♦ Compass
- ♦ Orbit
- ♦ Tether
- ♦ Mimic
- ♦ Turret
- ♦ Group
- ♦ Wide Group.

If you attach to two entities, you can be in tether track or mimic track view-mode. The first entity you select is the attached entity. The second entity you select is the tracked entity.

If you attach to a group, you can be in group or wide-group view-mode.

For descriptions of the view modes, refer to the *MÄK Stealth User's Guide*.

Detaching from An Entity

- > To detach the Stealth from an entity and stop tracking, change to absolute mode or click a point in the Plan View window (which changes you to absolute mode.)

Moving to a Point in the Plan View

You can move the Stealth to a point in the database if it is in absolute view-mode.

To move the Stealth to a point in the Plan View window:

1. Select a Stealth.
2. Open the Stealth Control dialog box (Functions▶Stealth Control.)
3. Click the Apply Selection to View Mode radio button.
4. Select a point in a Plan View window.

4.5.3. The No Action Radio Button

The Stealth Control dialog box has a radio button called No Action. This option has two uses:

- ♦ When you open a Stealth Control for a Stealth and you don't want to attach it to an entity, select No Action (the default.) You might want to do this if you want to keep the Stealth in absolute view-mode.
- ♦ To keep the Stealth attached to a particular entity regardless of the selection in a Plan View window. See [“Keeping the Stealth Attached to A Particular Entity,”](#) on page 4-12.

4.6. Working With Terrain Databases

This section describes your options for converting terrain databases and using terrain metafiles.

4.6.1. Saving Terrain Databases in MÄK Terrain Format

When the Plan View Display opens a database, it converts it into MÄK Terrain Format (*.gdb*). You can save databases in this format to speed up loading. You can save a converted database from the Plan View Display or convert it using the Terrain Converter Tool.

Saving a Terrain Database in MÄK Terrain Format in the Plan View Display

To save a database in the MÄK Terrain Format:

1. In the Plan View window, choose File → Save Terrain Database. The Save Database dialog box opens.
2. Enter a name for the database.
3. Click Save.



If you save a paper map file, the terrain database is saved, but not the map information.

Using the Terrain Converter Tool

The Plan View Display includes a command line tool for converting terrain databases to the MÄK Terrain Format. Although you can easily save databases from the Plan View Display interface, using the Terrain Converter Tool may save you some time, since you don't need to load a database into the Plan View Display to convert it. You can also use the Terrain Converter Tool in scripts and for batch conversion of databases.

The Terrain Converter Tool is available for each platform supported by the Plan View Display.

The syntax for the Terrain Converter Tool is as follows:

```
tdbtool [-f format] [-o outputfile] inputfile
```

where:

format is *flt* or *c4b*. If you don't specify the format, the Terrain Converter Tool looks at the extension of the input file.

outputfile is the path and name you want to give to the converted database. Include the extension *.gdb* in the name of the file. If you don't specify a name, the default is *out.gdb*.

inputfile is the pathname of the terrain database that you want to convert.

Database Formats Supported

The Terrain Converter Tool can convert only those terrain databases supported by the platform on which it is running. For example, on a PC, it can convert a FLT database, but it cannot convert a FLT database when running on a UNIX platform. The Terrain Converter Tool cannot convert DTED databases.

4.6.2. Using A Global Geodetic View

A global geodetic view provides a geodetic view of the world with no land masses. This is essentially a blank canvas. Use the geodetic view if you have a tape of an exercise and don't know where it is located. You can run the tape and check the latitude and longitude of the entities displayed. From this information you can calculate the location of the exercise and load the appropriate terrain database. [Figure 4-3](#) illustrates a global geodetic view with the *montydemo.lgr* tape running.

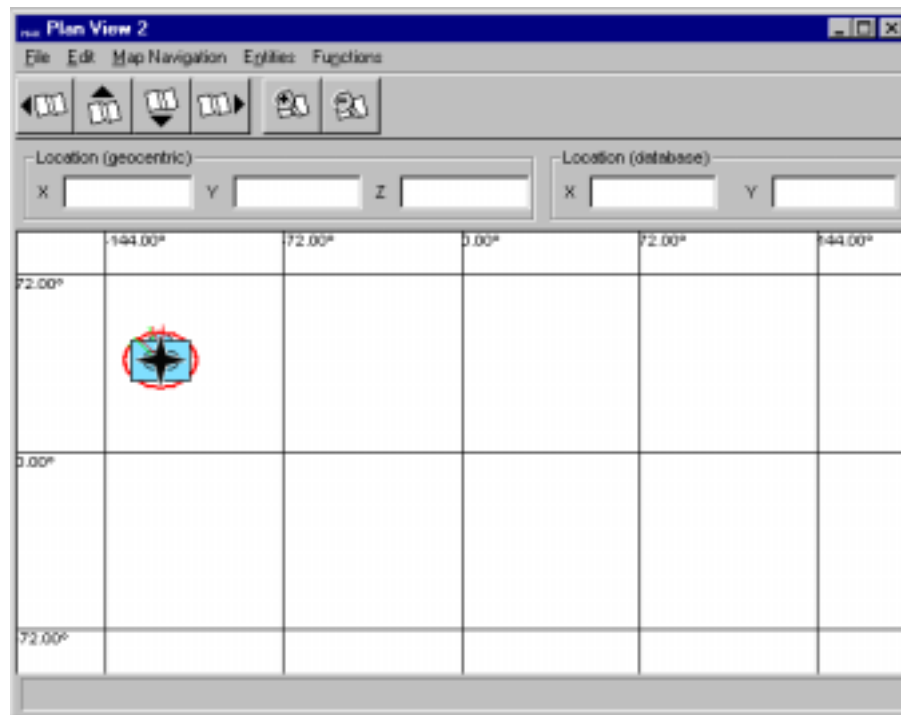


Figure 4-3. Global geodetic view

4.6.3. Terrain Metafiles

A Terrain Metafile (.map) packs together terrain databases and paper maps into a single file that can be loaded into the Plan View Display.

A terrain database is a list of polygons that carry three-dimensional information. With this information the Plan View Display can calculate height, intervisibility, and other three-dimensional information.

A paper map is either a bitmap file or a collection of Raster Product Format (RPF) files. A paper map file has no three-dimensional information. It is the equivalent of a paper map that you might display on a wall or use to plan an automobile trip. Paper maps are useful because they may display details not available in a terrain database, such as identifying text and information about terrain details and man-made structures. [Figure 4-4](#) shows a paper map of the same geographic area as the Monterrey database illustrated in [Figure 3-2](#).

i

You cannot display a paper map by itself. It must be associated with a terrain database.

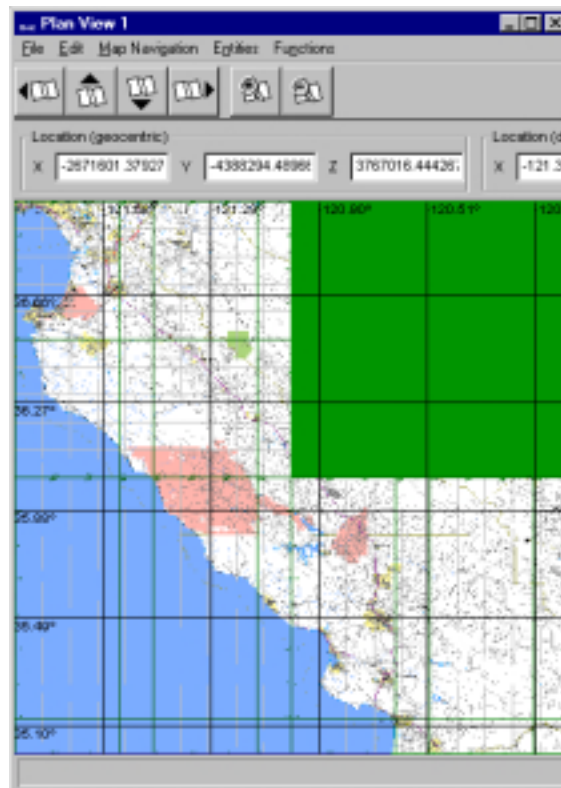


Figure 4-4. Paper map of Monterrey database

4.6.4. Creating Terrain Metafiles

A *.map* file consists of one Terrain Database Record, and zero or more Papermap Records. The following sections describe the syntax of the supported terrain database records and papermap records.

Terrain Database Records

The Plan View Display supports the following types of terrain database records:

- ♦ DTED
- ♦ FLT
- ♦ CTDB
- ♦ GDB.

DTED Terrain Database Records

DTED data is a grid of elevations. By default, the Plan View Display displays a DTED database as a blank green canvas. To see elevation detail, turn on terrain shading in the Settings dialog box.

In a DTED terrain database record, you specify the number of DTED samples per grid cell and one or more files with DTED data. The syntax is:

```
PvDtedMetafileRecord
{
  PostSpacing post_spacing
  File filename 1.dt0 or .dt1 or .dt2
  File filename 2
  .
  .
  .
  File filename n
}
```

where:

post_spacing is the number of DTED samples per grid cell, when constructing the polygon grid.

filename is a DTED file. DTED files typically have the extensions *.dt0* (for level 0 DTED), *.dt1* (for level 1 DTED), or *.dt2* (for level 2 DTED).

FLT Terrain Database Records

The FLT terrain database record specifies the name of a MultiGen OpenFlight database. The syntax is:

```
PvFltMetafileRecord
{
    File filename.flt
}
```



FLT files do not load textures.

CTDB Terrain Database Records

The CTDB terrain database record specifies the name of a Compact Terrain Database version 4 big endian database. The syntax is:

```
PvCtdbMetafileRecord
{
    File filename.c4b
}
```

GDB Terrain Database Records

The GDB record is the MÄK Terrain Format. The terrain database record specifies the name of the terrain database file. The syntax is:

```
PvGdbMetafileRecord
{
    File filename.gdb
}
```

Paper Map Records

The paper map record specifies the filename of the bitmap file, the coordinate system it uses, and the coordinates of the terrain database that it should overlay.

Paper Map Bitmap Record

The bitmap record specifies the bitmap used to overlay the terrain database and its location. The syntax is:

```
PvPaperMapBmpRecord
{
    File filename.bmp
    CoordinateSystem PvCoordinateSystemGeodetic
    SWCorner southwest_point
    NECorner northeast_point
}
```

where:

File is the name of the bitmap to load.

CoordinateSystem specifies the coordinate system that the **SWCorner** and **NECorner** extents are expressed in. Currently, the Plan View Display only supports the geodetic coordinate system for specifying paper map extents.

SWCorner and **NECorner** specify the latitude and longitude of the southwest corner and northeast corner of the papermap, for example:

```
SWCorner (-230, 34)
```

Paper Map RPF Record

An RPF paper map record specifies the standard used to arrange an RPF file, for example, ADRG. The syntax is:

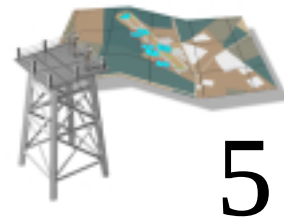
```
PvPaperMapRpfRecord
{
  File papermap_directory
  CoordinateSystem PvCoordinateSystemGeodetic
  Type CADRG
}
```

where:

File specifies the directory (directory only, not the file) of the papermap volume.

CoordinateSystem specifies the coordinate system that the RPF file uses. Currently, the Plan View Display only supports the geodetic coordinate system for specifying paper map extents.

Type is the kind of data in the papermap volume. Currently only CADRG. See the Release Bulletin for an updated list of supported paper map types.



Working With Entities and Other Objects

This chapter describes how to do work with entities, markers, and interactions:

Selecting Entities.....	5-2
Selecting Entities in A Plan View Window	5-2
Selecting Entities in the Entity List Dialog Box	5-3
Deselecting Entities	5-3
Working with Groups of Entities	5-5
Creating a User-Defined Group.....	5-5
Changing A User-Defined Group.....	5-5
Deleting User-Defined Groups.....	5-6
How Changes to Entities Affect Group Membership.....	5-6
Tracking Entities.....	5-7
Tracking Entities in the Plan View Display	5-7
Tracking An Entity Using the Stealth	5-7
Displaying Information About An Entity.....	5-9
Displaying Entity Labels	5-9
Understanding Visual Labels	5-10
Turning Labels On and Off	5-10
Displaying Intervisibility	5-11
Working With Markers.....	5-13
Placing Markers on A Canvas	5-13
Displaying Intervisibility for A Marker	5-13
Destroying A Marker.....	5-13
Interactions	5-13

5.1. Selecting Entities

You can select entities from the Entity List or directly in a Plan View window. The selection status of an entity is indicated by its border as follows:

- ♦ Black border - unselected
- ♦ White border - selected.

5.1.1. Selecting Entities in A Plan View Window

To select entities in a Plan View window, use the following mouse actions:

- ♦ Click - selects entity or a point in the display
- ♦ Ctrl-click - select multiple entities
- ♦ Marquee (hold down the left mouse button and drag a box) - select multiple entities
- ♦ Right click
 - If no entities are selected, right clicking an entity selects it and displays the right-click menu.
 - If one or more entities are selected, right-clicking a selected entity maintains the selection and displays the right-click menu. Right-clicking an unselected entity changes the selection to the entity you right-clicked.
 - Right-clicking terrain and dropping a marker does not affect the current selection of entities.

When you select an entity in the Plan View window, the Entity List dialog box highlights the entry for the entity to indicate that it is selected.

When entities are physically close to each other, they may appear to be one entity unless you zoom in very closely. The icons for each entity are stacked. The selection process is as follows:

- ♦ Click the top entity to select it.
- ♦ Click the selected entity again to send it to the bottom of the stack. It becomes unselected.
- ♦ Click the new top entity to select it.

5.1.2. Selecting Entities in the Entity List Dialog Box

You can select entities in the Entity List dialog box instead of directly in a Plan View window. The advantage to this method is that you can select entities that aren't visible in the window or which are bunched too closely together to easily discriminate on the screen or select using a marquee.

To select an entity in the Entity List dialog box:

1. Choose Entities → Entity List. The Entity List dialog box opens.
2. In the entity list, click the entity you want to select.



The Entity List supports multiple selection:

- Click to select
- Shift-click to select a contiguous range of entities
- Ctrl-click to select multiple non-contiguous entities.

5.1.3. Deselecting Entities

You can deselect entities in a Plan View window or in the Entity List dialog box. [Table 5-1](#) lists the options for deselecting an entity or entities in a Plan View window.

Table 5-1: Deselecting entities in a Plan View window

Action	Result
Click in the window away from the selected entity or entities.	Deselects all entities.
Click a selected entity.	Deselects the entity.
Control/click a selected entity.	Deselects the entity.

[Table 5-2](#) lists the options for deselecting an entity in the Entity List dialog box.

Table 5-2: Deselecting entities in the Entity List dialog box

Action	Result
Click the Clear All button.	Deselects all entities.
Click the Invert button.	Deselects all selected entities. Selects all previously unselected entities.
If one entity is selected.	Control/click the entity.
If a group of entities is selected: Control/click an entity.	Deselects the entity without affecting other selections.
Click an entity.	Selects the clicked entity and deselects all other entities.

Deselecting All Entities

To deselect all entities, choose Entities → Deselect All.

5.2. Working with Groups of Entities

You can create groups and assign entities to them. Groups are global, that is, they exist across all open Plan View windows. They are created and listed in the Entity List dialog box. When you create or change a group, the Entity List dialog boxes for all Plan View windows are updated with the new or changed group. The following groups exist by default:

- ♦ All selected entities in the Plan View window. This group is identified by the Plan View ID, for example Plan View 1.
- ♦ All stealths.
- ♦ All friendly entities.
- ♦ All hostile entities.
- ♦ All neutral entities.

5.2.1. Creating a User-Defined Group

To create a user-defined group of entities:

1. Choose Entities → Entity List. The Entity List dialog box opens.
2. In the Entity List, select the entities that you want to be in the group.
3. In the Group field, type the name for this group.
4. Click the Save Group button.

5.2.2. Changing A User-Defined Group

To change the members of a user-defined group:

1. Choose Entities → Entity List. The Entity List dialog box opens.
2. In the Group List, select the group that you want to change. The members of the group become highlighted in the Entity List. The name of the group is displayed in the Save As Group text box.
3. Add new members to the group or remove existing members by holding down the Ctrl key and clicking the entities you want to remove or add.

i

As soon as you add or remove an entity from a group, the group designation in the Group List changes to the Plan View group. This is because the entities selected no longer match the members of the user-defined group that you are changing. The group is not changed (redefined) until you click Save As Group.

4. Click the Save Group button.



You cannot change the membership of the default groups.

5.2.3. Selecting Groups

To select a group:

1. Choose Entities → Entity List. The Entity List dialog box opens.
2. In the Group List, select a group.

Using The Plan View Group

The Plan View group is a transient group created any time you select one or more entities in a Plan View window. The group is identified by the window ID, for example, Plan View 2. Unlike other groups, whose membership persists, if you deselect all entities, the Plan View group is empty until you select one or more entities again. Since groups exist across all Plan View windows, if you select an arbitrary group of entities in, for example, Plan View 2, you can quickly select the same entities in another Plan View window, for example, Plan View 3, by selecting the Plan View 2 group.

5.2.4. Deleting User-Defined Groups

To delete a user-defined group:

1. Choose Entities → Entity List. The Entity List dialog box opens.
2. In the Group List, select the group that you want to delete.
3. Click the Delete Group button.



You cannot delete the default groups. You can only delete user-defined groups.

5.2.5. How Changes to Entities Affect Group Membership

If an entity drops out of an exercise, its ID is not kept in the groups. Groups are updated to contain only the entities or stealths currently in the exercise.

5.3. Tracking Entities

You can track an entity or a group of entities in the Plan View Display or in the Stealth. When you track an entity in the Plan View Display, it keeps the entity roughly centered in the Plan View window.

5.3.1. Tracking Entities in the Plan View Display

Tracking of entities is toggled on and off with the Toggle Track Selection option on the Map Navigation menu. The tracking state is indicated in the status bar on the Plan View window.

- > To track selected entity(s) in a Plan View window, choose Map Navigation → Toggle Track Selection.

The Plan View window will track the entities that were selected when you toggled Track Selection on.



The status of the track selection toggle is displayed only if an entity is selected. It is not persistent, but will display again the next time you select this menu option.

Stopping Tracking in the Plan View Display

- > To stop tracking entities, choose Map Navigation → Toggle Track Selection. Tracking toggles off.

5.3.2. Tracking An Entity Using the Stealth

To track an entity with the Stealth:

1. Select the Stealth from which you want to track the entity.
2. Open the Stealth Control dialog box (Functions → Stealth Control.)
3. Click the Apply Selection to View Mode button.
4. Select an entity or a group. (See [“Selecting Entities,”](#) on page 5-2.)

The Stealth attaches to the selected entity or entities and tracks them (unless you are in absolute mode.)

Notes:

- ♦ If you change the selection in the Plan View window, the Stealth tracks the newly selected entity(s). To force the Stealth to track the original entity(s), you must select the No Action radio button. For more information, see [“The No Action Radio Button,”](#) on page 4-13.

- ♦ To attach to an entity and track it, the number of selected entities must be consistent with the type of view-mode you have chosen. For more information, see [“Attaching To and Tracking Entities,”](#) on page 4-11.

For more information about using the Stealth with the Plan View Display, see [“Controlling the Stealth from the Plan View Display,”](#) on page 4-9.

Stopping Tracking in the Stealth

You can stop tracking by the Stealth in two ways:

1. Open the Stealth Control dialog box (Functions → Stealth Control.)
2. Select the absolute view-mode.

or

- > Click any point in the terrain. The Stealth changes to absolute view-mode and stops tracking.

5.4. Displaying Information About An Entity

You can display information about an entity by opening the Entity Information dialog box. The information displayed is static. You cannot edit it.

- > To open the Entity Information dialog box, right-click on an entity and select Entity Info from the menu.

For information about the fields in the Entity Information dialog box, see [“Entity Information Dialog Box,”](#) on page 3-18.

5.5. Displaying Entity Labels

The Plan View Display allows you to display information about an entity in an on-screen label. By default, the label displays the heading indicator, movement indicator, and marking text. You can change the default colors and locations by editing the `pvd.lab` file. See [“Configuring Entity Labels,”](#) on page 6-10.

You can display the following additional information:

- ♦ Movement indicator
- ♦ Heading indicator
- ♦ Markings
- ♦ Position
- ♦ Velocity
- ♦ Acceleration
- ♦ Heading
- ♦ Object Id (global object designator)
- ♦ Entity type
- ♦ Entity Guise
- ♦ Appearance Flag.

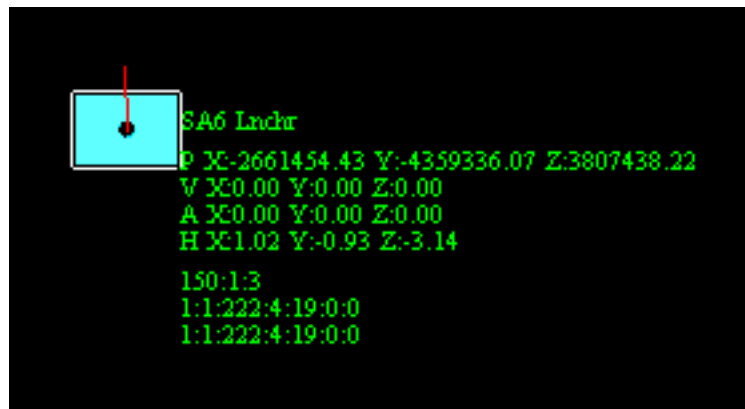


Figure 5-1. Entity labels

5.5.1. Understanding Visual Labels

Most labels are displayed in text to the right of the entity. The default color is green, but it is configurable. (See “[Configuring Entity Labels](#),” on page 6-10.) Some labels are displayed visually. The visual labels are illustrated in [Figure 5-2](#). For a description of each entity label, see [Table 3-11](#) in “[Entity Labels Dialog Box](#),” on page 3-23.

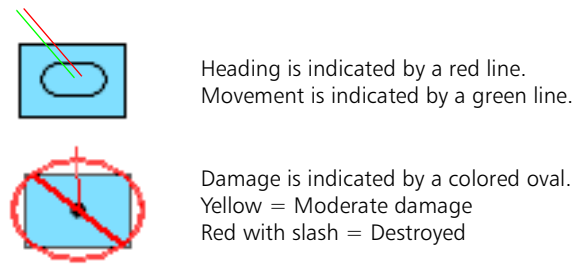


Figure 5-2. Graphical entity labels

5.5.2. Turning Labels On and Off

To specify which labels are displayed for an entity:

1. Place the cursor over the entity whose label you want to change.
2. From the right-click menu, choose Entity Labels. The Entity Labels dialog box opens ([Figure 3-13](#).)
3. Check the items that you want to display. Clear check boxes for labels that you do not want to display.
4. Click the Dismiss button.

To specify labeling for multiple entities:

1. Select the entities you want to label. See “[Selecting Entities](#),” on page 5-2 to learn how to select multiple entities.
2. Put the cursor over one of the selected entities.
3. From the right-click menu, choose Entity Labels. The Entity Labels dialog box opens.
4. Check the items that you want to display. Clear check boxes for labels that you do not want to display.
5. Click the Dismiss button.

5.6. Displaying Intervisibility

Intervisibility shows whether or not two entities or terrain points are within each other's line of sight. [Figure 5-3](#) illustrates the concepts of intervisibility. [Figure 5-4](#) illustrates how the Plan View Display shows intervisibility.

To display intervisibility:

1. Right-click an entity or marker.
2. On the right-click menu, choose Intervisibility → *intervisibility option*. The intervisibility options are described in [Table 5-3](#).

Table 5-3: Intervisibility options

Options	Description
Change Range	Allows you to adjust the sphere within which intervisibility is calculated.
Toggle Who Can I See?	Displays (or turns off) a line between the selected entity and each entity that it can see.
Toggle Who Can See Me?	Displays (or turns off) a line between the selected entity and each entity that can see it.
Toggle Intervisibility Fan	Toggles on or off display of terrain areas that are not visible.
Toggle Intervisibility Range Circle	Toggles on or off display of the range in which intervisibility is calculated. Represented as a blue circle, this is actually a sphere with the radius of the circle.
Toggle Obstructed Visibility	Toggles on or off display of obstructed sight lines. Obstructed portion of sight line is red.
Intervisibility Settings	Opens the Intervisibility Settings dialog box (Figure 3-12), which allows you to set all intervisibility settings at once.

i

In the Settings dialog box, you can change the colors used to display intervisibility. See “[Changing Colors](#),” on page 6-18. You can also change the default color settings. See “[Configuring Plan View Window Settings](#),” on page 6-11.

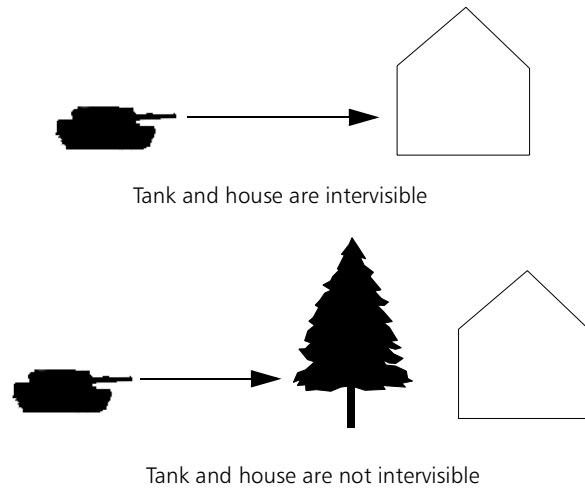


Figure 5-3. Intervisibility concepts

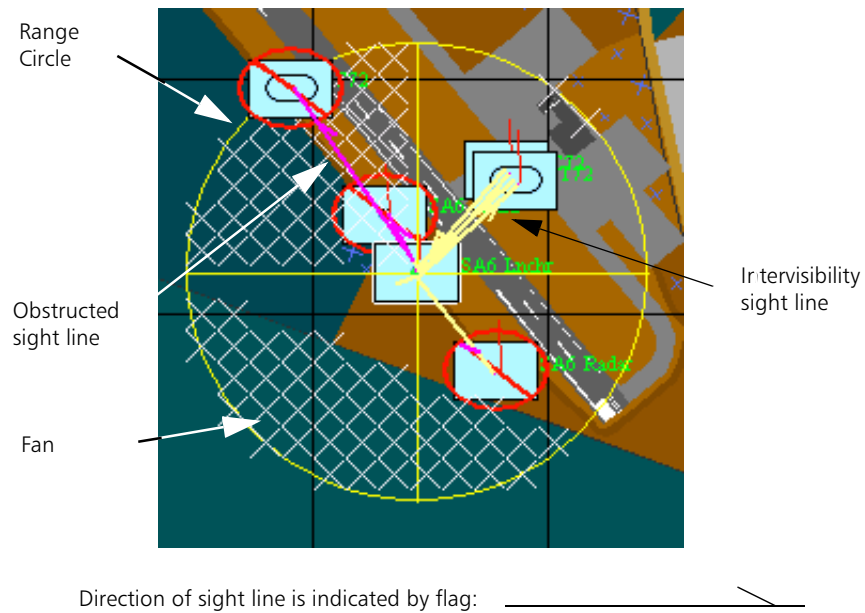


Figure 5-4. Intervisibility display in Plan View window

i

In [Figure 5-4](#), intervisibility colors have been changed from their default values.

5.7. Working With Markers

You can place markers on a Plan View window canvas. A marker is displayed as a green point. Markers can display intervisibility data. Therefore, using a marker, you can determine the concealment value of different locations in the terrain without placing entities in those locations.

5.7.1. Placing Markers on A Canvas

To place a marker on the canvas:

1. Move the mouse cursor to a point on the canvas that is not occupied by an entity.
2. From the right-click menu, choose Drop Marker. A point is placed on the canvas.

5.7.2. Displaying Intervisibility for A Marker

To display intervisibility for a marker:

1. Place the cursor over the marker.
2. From the right-click menu, choose the intervisibility options. For information about specifying intervisibility options, see [“Displaying Intervisibility,”](#) on page 5-11.

5.7.3. Destroying A Marker

To destroy a marker:

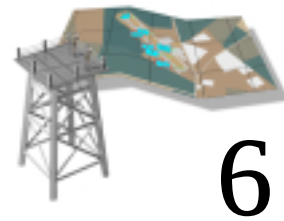
1. Place the cursor over the marker.
2. From the right-click menu, choose Destroy.

5.8. Interactions

The Plan View Display displays interactions that it receives in interaction messages or PDUs. You cannot select or otherwise manipulate them; you can only view them.

You can change the timeout settings for interactions and you can change the bitmaps associated with them.

For more information, see [“Specifying the Duration of Interactions,”](#) on page 6-16 and [“Configuration File Options,”](#) on page 6-7.



Customizing the Plan View Display

This chapter describes how to change various aspects of the Plan View and the information displayed in dialog boxes. The topics covered are:

Startup Options	6-2
Protocol Independent Startup Options	6-3
Startup Options for HLA Federations	6-4
Startup Options for DIS Exercises	6-6
The Plan View Display Log File (PC Only)	6-6
Configuration File Options	6-7
Associating Bitmaps with Entity Types	6-7
Localizing the Plan View Display	6-9
Configuring Entity Labels	6-10
The X Resource File (UNIX Only)	6-13
Changing Settings	6-14
Specifying Which Information is Displayed	6-14
Setting Zoom Increments	6-15
Setting Panning Increments	6-15
Specifying the Frequency of Grid Lines	6-15
Specifying the Frequency of Contour Lines	6-15
Specifying the Duration of Interactions	6-16
Setting the Frame Rate	6-16
Displaying Terrain Shading	6-16
Setting Contribution Culling	6-17
Icons	6-19

6.1. Startup Options

The Plan View Display has several command line options that you can set when you start the program.

On a PC, you can use startup options if you start the Plan View Display from the Run dialog box or you can add them to the command associated with the Plan View Display icon, a Plan View Display shortcut icon, or the entry in the Start menu.

In UNIX, since you start the Plan View Display from a command line, the startup options are always available.

The syntax for running the Plan View Display from the command line is:

```
Pvd [-A destination_address
      -a application_number
      -C
      -f RPRkeyword
      -f library:init_data
      -h
      -n notification_level
      -P port
      -S multicast_address
      -s site_ID
      -T
      -x {execution_name|exercise_ID}]
```

Arguments:

- c runs the Plan View Display with a console window open (meaningful only on PCs).
- h displays usage information in a console window (implies -c on PCs).
- n *notification_level* sets the level for messages written to the console or the Plan View Display log file, where *notification_level* is a number from 0 through 3. The least verbose message level is 0, the highest is 3. The default is 2.
- t runs the Plan View Display in single-threaded mode.

DIS only

- A *destination_address* specifies an address for point-to-point mode.
- a *application_number* specifies the DIS application number.
- P *port* specifies the port number for DIS (default 3000).
- S *multicast_address* specifies an address for multicast mode.
- s *site_ID* specifies the DIS site ID.
- x *exercise_ID* specifies the DIS exercise ID.

HLA only

- f specifies the FOM version to use or a shared library. See [“Selecting A FOM Mapper,”](#) on page 6-4.
- x *execution_name* specifies the HLA federation execution name.

Use of these options is described on the following pages.

6.1.1. Protocol Independent Startup Options

This section discusses startup options that you can use with both DIS and HLA.

Displaying A Console On A PC

You can run the Plan View Display with a console open to see the messages issued by the Plan View Display.

- > To display a console window with the Plan View Display, start it with the -C option, for example:

```
Pvd -C
```

Setting the Notification Level

You can control the number of messages sent to the console (and log file on a PC) by specifying a notification level.

- > To specify a notification level, use the -n option, for example:

```
Pvd -n 1
```

i

The UNIX version of the Plan View Display does not have a log file. Informational messages are sent to the window from which you start the Plan View Display. To capture informational messages, redirect output to a file.

Single-threaded Operation

The Plan View Display comes in single-threaded and multi-threaded versions. We recommend that you use the multi-threaded versions, which is the default, unless you have a particular reason for using the single-threaded version.

- > To run in single-threaded mode, start the Plan View Display with the `-t` command line option.

The single threaded version of the Plan View Display may drop packets. If you are having problems with packets being dropped, we recommend that you change `FED_BEST_EFFORT` to `FED_RELIABLE` in the *VR-Link.fed* file for the following attributes:

- ♦ `DRAAlgorithm`
- ♦ `EntityType`
- ♦ `EntityID`
- ♦ `ForceID`.



Changing to reliable transmission increases the bandwidth required and may affect performance.

6.1.2. Startup Options for HLA Federations

The Plan View Display listens to whatever port is specified in the *RTI.rid* file. There is no mechanism to change the port from within the Plan View Display.

- > To specify the HLA federation execution name, use the `-x` option, for example:

```
Pvd -x vrlink
```

Selecting A FOM Mapper

The Plan View Display has built-in support for the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper.

You can specify which FOM Mapping code to use with the `-f` startup option, in one of two ways:

```
Pvd -f RPRkeyword
Pvd -f library:init_data
```

If the argument to the `-f` option is an RPR FOM keyword, the Stealth uses VR-Link's built-in RPR FOM mapper, configured for the RPR FOM version specified by the keyword.

i

Please see the Plan View Display Release Notes for a list of the RPR FOM versions supported by your version of the Plan View Display and the keywords to use.

If the argument to the `-f` option is not an RPR FOM keyword, it is interpreted as the name of a shared library (DSO or DLL) containing code for an alternate FOM Mapper that you would like the Stealth to use, optionally followed by any initialization data the library requires. The library name and initialization data must be separated by a colon.

Examples

```
Pvd -f vrlRPR05
```

This command instructs the Plan View Display to use the built-in RPR FOM Mapper, configured for RPR FOM version 0.5.

```
Pvd -f myFomMapLib:ABC
```

This command instructs the stealth to use the FOM Mapper contained in a shared library called *myFomMapLib.so* (or *myFomMap.dll*). The string “ABC” is passed to the FOM Mapper creation function in the shared library as an initialization parameter.

Specifying A FED File That is Appropriate for the FOM Mapper

!

The `-f` option only tells the application which FOM Mapping code to use. It does not ensure that you are using the right FED file for the FOM Mapper. Make sure that you are using a FED file that is consistent with the FOM Mapper you have chosen.

By default the application uses a FED file called *VR-Link.fed*, but you can specify an alternate FED file using the `-x` option. The `-x` option actually indicates the name of the federation execution name, but we append *.fed*, and use that as the FED file name as well.

The following example instructs the application to use the RPR FOM 0.5 FOM Mapping code (`-f vrlRPR05`), and also instructs it to use the *VR-Link05.fed* file (`-x VR-Link05`).

```
Pvd -f vrlRPR05 -x VR-Link05
```

As an alternative to using the `-x` option, you could rename the *VR-Link05.fed* file to *VR-Link.fed*.

i

Please see the Plan View Display Release Notes for a list of the FED files supplied with your version of the Plan View Display.

6.1.3. Startup Options for DIS Exercises

The following sections describe startup options that apply only to using the Plan View Display with DIS exercises.

Specifying the Port Number

By default, the Plan View Display listens to port 3000.

- > To change the port that the Plan View Display listens to, use the `-P` option, for example:

```
Pvd -P 3500
```

Specifying Point-to-Point or Multicast Operation

You can specify that the Plan View Display run in point-to-point mode or multicast.

- > To specify point-to-point, use the `-A` option, for example:

```
Pvd -A 123.123.12.1
```

- > To specify multicast, use the `-S` option, for example:

```
Pvd -S 233.233.22.3
```

You can specify more than one multicast address. The syntax is:

```
Pvd -S address -S address ...
```



The default is broadcast mode.

Specifying the Exercise ID

To control the MÄK Stealth from the Plan View Display, the Stealth must be configured to respond only to the exercise ID used by the Plan View Display.

- > To set the Plan View Display exercise ID, use the `-x` option, for example:

```
Pvd -x 6
```

6.1.4. The Plan View Display Log File (PC Only)

The Plan View Display creates a log file, *pvd.log* that records informational messages. These are the same messages that are sent to the console if you display a console. The log file is written in the directory from which you run *Pvd*.

6.2. Configuration File Options

The Plan View Display uses the following configuration files:

- ♦ *pvd.lab* - specifies label defaults, locations, colors, and bitmaps for entity labels.
- ♦ *pvd.set* - specifies the default global settings
- ♦ *pvdDIS3.smp*, *pvdDIS5.smp*, *pvdRPR.smp* - specify the bitmaps to associate with entity and interaction types. The files correspond to the different versions of DIS and to HLA. See [“Associating Bitmaps with Entity Types,”](#) on page 6-7.
- ♦ *pvd.stb* - contains string tables that you can change to internationalize the Plan View Display.
- ♦ *pvd.xrc* - configures colors and fonts for UNIX systems.

6.2.1. Associating Bitmaps with Entity Types

The *pvd.smp* configuration file specifies the bitmaps to associate with entity and interaction types. It has a section for entity symbols and one for event symbols.

Entity Symbol Map Entries

The first number on each line in the PvEntitySymbolMap section is the Force ID. The rest are the DIS enumeration for the entity. File names are relative to the *.smp* file. However, you can enter absolute pathnames if you want to.

The possible values for Force ID are:

- ♦ 1 - Friendly
- ♦ 2 - Hostile
- ♦ 3 - Neutral
- ♦ -1 - Wildcard.

The following code is a portion of a *pvd.smp* file:

```
PvEntitySymbolMap
{
    NumFields 8
    -1 -1 -1 -1 -1 -1 -1 -1 "../data/symbols/default.bmp"
    1 1 4 -1 -1 -1 -1 -1 "../data/symbols/subF.bmp"
    2 1 4 -1 -1 -1 -1 -1 "../data/symbols/subH.bmp"
    3 1 4 -1 -1 -1 -1 -1 "../data/symbols/subN.bmp"
    1 1 3 -1 -1 -1 -1 -1 "../data/symbols/waterF.bmp"
    2 1 3 -1 -1 -1 -1 -1 "../data/symbols/waterH.bmp"
    3 1 3 -1 -1 -1 -1 -1 "../data/symbols/waterN.bmp"
    1 1 1 -1 -1 -1 -1 -1 "../data/symbols/groundF.bmp"
    -1 0 0 0 0 0 0 0 "../data/symbols/stealth.bmp"
    ...
}
```

> To change the bitmaps associated with icons, change the pathnames on each line.

Notes

- ♦ Do not change the value for NumFields.
- ♦ Symbols are ordered from general to specific according to the fields of the DIS enumeration.

Event Symbol Map Entries

The PvEventSymbolMap entries contain an event identifier followed by the name of the bitmap to associate with the event. File names are relative to the *.smp* file. However, you can enter absolute pathnames if you want to.

Event Symbol Maps for HLA

The first element of the event symbol map is the name of the event (interaction). This is the name returned by the name() function in VR-Link.

```
PvEventSymbolMap
{
    DtFireInteraction "../data/symbols/fireInter.bmp"
    DtDetonationInteraction "../data/symbols/detInter.bmp"
}
```

Event Symbol Maps for DIS

The first element of the event symbol map is the interaction kind. This is the number returned by the kind() function in VR-Link.

```
PvEventSymbolMap
{
    2 "../data/symbols/fireInter.bmp"
    3 "../data/symbols/detInter.bmp"
}
```

6.2.2. Localizing the Plan View Display

To localize the Plan View Display interface, edit the quoted strings in the file *pvd*.stb*. The quoted strings are the text that appear on windows and dialog boxes in the Plan View Display.

Ampersands (&) indicate that the following letter should be underlined and used as a keyboard accelerator. For example, “&File” results in *F*ile being displayed and Alt-F opens the File menu.



If you change the strings, remember that you must not use the same letter as an accelerator twice in the same menu. You can repeat letters in different menus.

For example:

```
PvPlanViewMenubarStringTable
{
FileSubmenu "&File"
LoadTerrainDatabase "&Load Terrain Database"
LoadTerrainDatabaseHelp "Load a terrain database into this Plan View
                        window"
SaveTerrainDatabase "&Save Terrain Database"
SaveTerrainDatabaseHelp "Save the terrain database in MÄK Terrain Format"
Close "&Close"
CloseHelp "Close this Plan View window"
EditSubmenu "&Edit"
Settings "&Settings"
SettingsHelp "Change characteristics of the Plan View display"
...
}
```

6.2.3. Configuring Entity Labels

The configuration file *pvd.lab* lists the entity labels and their characteristics.

- > To change the bitmap associated with an appearance flag, edit its entry in the *pvd.lab* file.

Figure 6-1 illustrates the fields in label configuration setting. The fields have the following meaning:

- Name - the name of the label.
- Location - the location of the displayed label, in pixels from the center of the symbol.
- On/Off - the default setting, where one (1) is on and zero (0) is off.
- Color/pathname - For text and directional indicators, the color, in RGB values. For appearance flags, the pathname of the bitmap for the appearance flag. Pathnames are relative to the *.lab* file.
- Length/width - for directional indicators, the length and width of the indicator, in pixels.

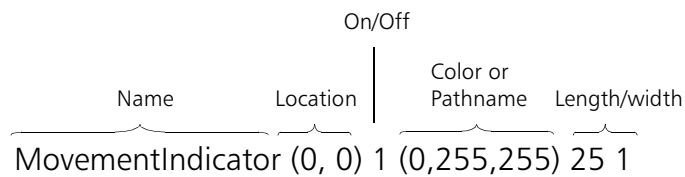


Figure 6-1. Label description fields

The default *pvd.lab* file is as follows:

```
PvEntityLabelsProperties
{
  MovementIndicator (-1, 0) 1 (0, 255, 0) 25 1
  HeadingIndicator (1, 0) 1 (255, 0, 0) 25 1
  MarkingsLabel (20, -10) 1 (0, 255, 0)
  PositionLabel (20, 5) 0 (0, 255, 0)
  VelocityLabel (20, 15) 0 (0, 255, 0)
  AccelerationLabel (20, 25) 0 (0, 255, 0)
  HeadingLabel (20, 35) 0 (0, 255, 0)
  ObjectIdLabel (20, 50) 0 (0, 255, 0)
  EntityTypeLabel (20, 60) 0 (0, 255, 0)
  EntityGuiseLabel (20, 70) 0 (0, 255, 0)
  ImmobilizedLabel (0, 0) 0 ""
  FirePowerDisabledLabel (0, 0) 0 ""
  EngineSmokeOnLabel (0, 0) 0 ""
  SmokePlumePresentLabel (0, 0) 0 ""
  FlamesPresentLabel (0, 0) 0 ""
  PowerPlantOnLabel (0, 0) 0 ""
  TentDeployedLabel (0, 0) 0 ""
  RampDeployedLabel (0, 0) 0 ""
  AfterburnerOnLabel (0, 0) 0 ""
  LauncherFlashPresentLabel (0, 0) 0 ""
}
```

```

DamageNoneLabel (0, 0) 0 ""
DamageSlightLabel (0, 0) 1 "..\data\symbols\damaged.bmp"
DamageModerateLabel (0, 0) 1 "..\data\symbols\damaged.bmp"
DamageDestroyedLabel (0, 0) 1 "..\data\symbols\destroyed.bmp"
TrailingEffectsNoneLabel (0, 0) 0 ""
TrailingEffectsSmallLabel (0, 0) 0 ""
TrailingEffectsMediumLabel (0, 0) 0 ""
TrailingEffectsLargeLabel (0, 0) 0 ""
CamouflageDesertLabel (0, 0) 0 ""
CamouflageWinterLabel (0, 0) 0 ""
CamouflageForestLabel (0, 0) 0 ""
PrimaryWeaponNoneLabel (0, 0) 0 ""
PrimaryWeaponStowedLabel (0, 0) 0 ""
PrimaryWeaponDeployedLabel (0, 0) 0 ""
PrimaryWeaponInFirePositionLabel (0, 0) 0 ""
SecondaryWeaponNoneLabel (0, 0) 0 ""
SecondaryWeaponStowedLabel (0, 0) 0 ""
SecondaryWeaponDeployedLabel (0, 0) 0 ""
SecondaryWeaponInFirePositionLabel (0, 0) 0 ""
}

```

6.2.4. Configuring Plan View Window Settings

The *pvd.set* configuration file contains the default settings for Plan View windows. To change the default settings, edit the file. [Table 6-1](#) describes each setting.

The default *pvd.set* configuration file is as follows:

```

PvInternalSettings
{
    DisplayWireframe 0
    DisplayGridLines 1
    DisplayContourLines 0
    DisplayInteractions 1
    DisplayTerrain 1
    DisplayEntities 1
    ContributionCulling 1
    PauseUpdates 0
    DeadReckoning 1
    TerrainShading 0
    GridLineFrequency 5
    ContourLineFrequency 50
    InteractionDuration 10
    ContributionCullingThreshold 4
    FrameRate 30
    StealthLinearIncrement 1
    StealthAngularIncrement 1
    ZoomStepSize 200
    PanStepSize 50
    IntervisibilityCircleColor (0, 0, 255)
    IntervisibilityFanColor (0, 0, 255)
    IntervisibilityShownLineColor (0, 255, 0)
    IntervisibilityObstructedLineColor (255, 0, 0)
}

```

Table 6-1: Settings parameters

Parameter	Value
DisplayWireframe	0 = Off; 1 = On
DisplayGridLines	0 = Off; 1 = On
DisplayContourLines	0 = Off; 1 = On
DisplayInteractions	0 = Off; 1 = On
DisplayTerrain	0 = Off; 1 = On
DisplayEntities	0 = Off; 1 = On
ContributionCulling	0 = Off; 1 = On
PauseUpdates	0 = Off; 1 = On
DeadReckoning	0 = Off; 1 = On
TerrainShading	0 = None; 1 = Height
GridLineFrequency	Approximate number of grid lines in the longest dimension of the screen.
ContourLineFrequency	Interval between contour lines in meters.
InteractionDuration	Number of seconds an interaction remains visible.
ContributionCullingThreshold	Number of pixels for contribution culling threshold.
FrameRate	Frames per second.
StealthLinearIncrement	Stealth Control linear increment in meters.
StealthAngularIncrement	Stealth Control angular increment in radians.
ZoomStepSize	Increments for zooming as a percentage of the screen. 200 means 2 times the screen size.
PanStepSize	Increment of panning as a percentage of the screen.
IntervisibilityCircleColor	Intervisibility range circle. Specified in RGB values.
IntervisibilityFanColor	Intervisibility terrain fan cross-hatching. Specified in RGB values.
IntervisibilityShownLineColor	Intervisibility line for entities that can see each other. Specified in RGB values.
IntervisibilityObstructedLine-Color	Portion of intervisibility line that is obstructed. Specified in RGB values.

6.2.5. The X Resource File (UNIX Only)

The Plan View Display for UNIX has an X resource file that lets you change the colors used in the background and foreground of windows and dialog boxes.

- > To customize the colors or fonts used in Plan View Display windows and dialog boxes, edit the file `./config/pvd.xrc`.

The default `pvd.xrc` file is similar to the following:

```
*foreground: black
*background: gray
*Menubar*background: gray
*popup*background: gray
*pulldown*background: gray
*panel*background: gray
*file_selector*background: gray
*panelBorder*background: gray
*textWindow*background: white
*client*background: gray
*fontList: -*-helvetica-medium-r-normal-*-*10-*-*-*-*-*
```

6.3. Changing Settings

You can change a number of settings that control the information displayed in a Plan View window. Most of them toggle an aspect of the display on or off. You can change them in all Plan View windows at once (globally) or for an individual window. When you save a Plan View Display layout, the settings are saved.

To change settings for all Plan View windows:

1. In the Mak PVD window, choose Edit → Settings. The Settings dialog box is displayed.
2. Change settings as desired.

To change settings for a Plan View window:

1. In the Plan View window, choose Edit → Settings. The Settings dialog box is displayed.
2. Change settings as desired.



Depending on the size of the database and the amount of system resources available, applying settings such as contour lines can take up to several minutes. If you only need to view a portion of the database, you can improve performance by zooming in on the map before you change the settings, thereby applying the changes to a smaller area.

6.3.1. Specifying Which Information is Displayed

You can turn the following items on and off:

- ♦ Grid lines
- ♦ Contour lines
- ♦ Interactions
- ♦ Wireframe
- ♦ Terrain
- ♦ Entities
- ♦ Stealths
- ♦ Dead reckoning
- ♦ Terrain shading
- ♦ Contribution culling.

To turn display of a feature on or off:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Display tab, check or clear the check box for the item you want to display or not display.

3. Click Apply to apply the change and keep the dialog box open, or click Accept to apply the change and close the dialog box.

6.3.2. Setting Zoom Increments

The zoom increment is specified as a percentage of the screen. The default increment for zooming the display is 200, which increases or decreases magnification by two times the image size.

To change the zoom increment:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Control tab, change the increment by moving the Zoom Step Size slider.

6.3.3. Setting Panning Increments

Panning is specified as a percentage of the page. The default increment for panning the display is 50% (1/2 of the page).

To change the pan increment:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Control tab, change the increment by moving the Pan Step Size slider.

6.3.4. Specifying the Frequency of Grid Lines

The Grid Line Frequency slider sets the number of grid lines displayed along the longest dimension of the Plan View window. The default grid line frequency is 5 lines. The smallest grid line increment is one meter.

To change the frequency of grid lines:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Display tab, change the frequency by moving the Grid Line Frequency slider.

6.3.5. Specifying the Frequency of Contour Lines

Frequency of contour lines is specified in meters. The default contour line frequency is 50 meters.

To change the frequency of contour lines:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Display tab, change the frequency by moving the Contour Line Frequency slider.

6.3.6. Specifying the Duration of Interactions

You can specify, in seconds, how long the results of an interaction, such as munitions fire, is displayed. The default duration is 10 seconds.

To change the duration of interactions:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Display tab, adjust the Interaction Duration slider.

6.3.7. Setting the Frame Rate

The frame rate is the fastest frequency at which the Plan View Display refreshes Plan View windows. The higher the frame rate, the greater the load on your computer. The default frame rate is one frame-per-second.

To change the frame rate:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Display tab, adjust the Frame Rate slider.

6.3.8. Displaying Terrain Shading

The default display of terrain for the Plan View Display is unshaded. If you turn on terrain shading, the display uses shading to suggest the terrain contour. Shading is based on a terrain feature such as height. [Figure 6-2](#) illustrates terrain shading.

To turn terrain shading on or off:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Display tab, in the Terrain Shading drop-down list, select a shading option to turn shading on or None to turn shading off.

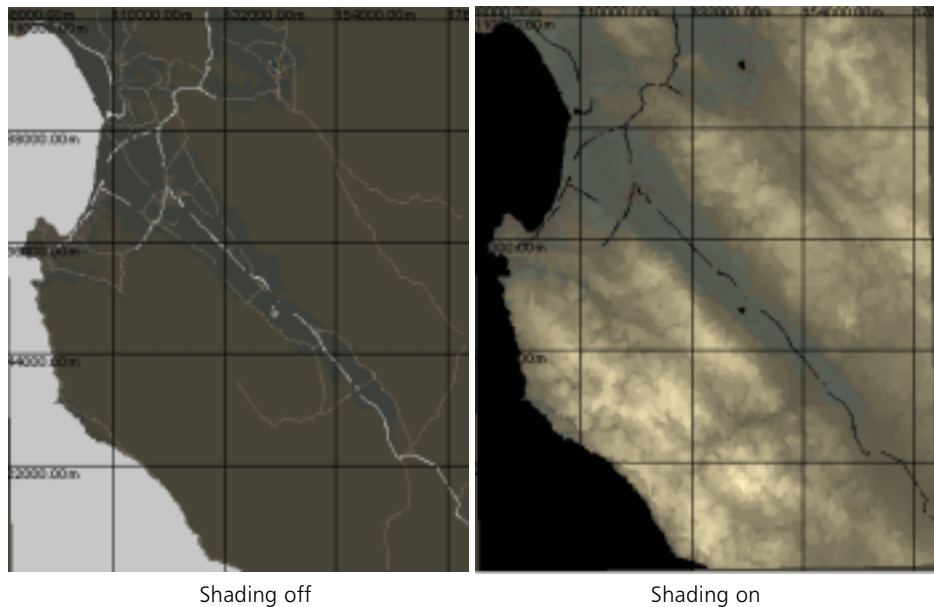


Figure 6-2. Terrain shading

6.3.9. Setting Contribution Culling

Contribution culling sets a threshold below which the Plan View Display does not try to display aspects of the terrain. By not calculating objects which you can't see anyway because they are too small, you can improve the performance of the Plan View Display.

- > To turn on contribution culling, check the Contribution Culling check box on the Display tab of the Settings dialog box.

The threshold for contribution culling is a pixel value, below which the Plan View Display does not try to display objects. The default value is four pixels.

i

Contribution culling is done on a per-object basis, so many small objects grouped together, such as the trees in a forest, could all be culled out, because they are not aggregated together. Zooming in on a Plan View would make such aggregates visible (or turning off contribution culling.)

To set the contribution culling threshold:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Display tab, adjust the Contribution Culling slider.

6.3.10. Changing Colors

You can change the colors assigned to intervisibility on the Color tab of the Settings dialog box. You can change other colors and the default colors for intervisibility in the *pvd.xxx* configuration file.

To change the color of an intervisibility setting:

1. Open the Settings dialog box as described in [“Changing Settings,”](#) on page 6-14.
2. On the Color tab ([Figure 3-8](#)), click the Change button for the intervisibility setting that you want to change. The Color dialog box opens. This is a standard color picker for your windowing system.
3. Select one of the existing colors or click Define Custom Colors to select any RGB value.
4. Click OK.
5. Click Apply to apply the change and leave the dialog box open. Click Accept to apply the change and close the Settings dialog box.

6.4. Icons

Table 6-2 illustrates the default icon set in the Plan View Display. The color and shape code is:

- ♦ Light blue - friendly
- ♦ Salmon - hostile
- ♦ Light green - neutral.

Symbols have a black border except as follows:

- ♦ If there is no border on the bottom, the entity domain is above ground, for example, helicopters.
- ♦ If there is no border on the top, the entity domain is below ground, for example, submarines.
- ♦ Selected entities have a white border.

The icons comply with MIL-STD-2525B Department of Defense Interface Standard for Common Warfighting Symbolology.

Table 6-2: Default entity symbol set

Icon	Meaning	Icon	Meaning
	Artillery: friendly		Helicopter: friendly
	Artillery: hostile		Helicopter: hostile
	Artillery: neutral		Helicopter: neutral
	Building: friendly		Missile: friendly
	Building: hostile		Missile: hostile

Table 6-2: Default entity symbol set

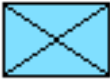
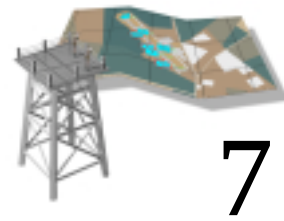
Icon	Meaning	Icon	Meaning
	Building: neutral		Missile: neutral
	Ground: friendly		Infantry: friendly
	Ground: hostile		Infantry: hostile
	Ground: neutral		Infantry: neutral
	Plane: friendly		Submarine: hostile
	Plane: hostile		Submarine: neutral
	Plane: neutral		Tracked entity: friendly
	Radar: friendly		Tracked entity: hostile
	Radar: hostile		Tracked entity: neutral
	Radar: neutral		Water: friendly

Table 6-2: Default entity symbol set

Icon	Meaning	Icon	Meaning
	Stealth		Water: hostile
	Submarine: friendly		Water: neutral



Using Plug-ins With the Plan View Display

This chapter describes how to create plug-ins for the Plan View Display. Plug-ins allow you to extend the capabilities of the Plan View Display.

i

This chapter assumes that you have VR-Link and are familiar with it. The plug-in API requires VR-Link.

This chapter covers the following subjects:

Introduction to Plan View Display Plug-ins	7-3
How the Plug-in API Works	7-3
Working with Plug-in Interfaces	7-4
Responding to Events	7-5
Registering Callbacks Directly with An Application	7-5
Responding to Plan View Display Ticks	7-6
Managing Windows From A Plug-in	7-7
Extending Menus	7-7
Getting Terrain Information	7-8
Working with Coordinate Information	7-8
Responding to User Events	7-10
Getting Selection Information	7-10
Working with Objects	7-10
Creating Objects	7-11
Looking Up An Object Interface	7-11
Getting A List of Objects	7-11
Tracking Objects	7-12
Timing Out Objects	7-12
Responding to Creation of Objects and Object Changes	7-12

Getting information about objects (state, coordinates, ID).....	7-13
Setting Object Values	7-14
Removing objects	7-14
Building and Compiling A Plug-in	7-15
Plug-in Example	7-16
Planning a Plug-in.....	7-16
Implementing the Plug-in	7-20
Plug-in Classes and Header Files	7-24

7.1. Introduction to Plan View Display Plug-ins

The Plan View Display has a plug-in interface that allows you to add your own capabilities to the Plan View Display. For example, you can:

- ♦ Create objects and place them on the canvas
- ♦ Add menus, menu items, and callbacks on menu items to the menu bar and add items to the right-click menu
- ♦ Respond to user events such as clicking an object or a location in the window
- ♦ Respond to the addition or removal of objects.

You can create and manipulate the following kinds of objects:

- ♦ Entity
- ♦ Interaction
- ♦ Point
- ♦ Line
- ♦ Polygon
- ♦ Circle
- ♦ Text.

7.1.1. How the Plug-in API Works

The `./plugins` directory has three subdirectories, `bin3`, `bin5`, and `binRPR`. When you start the Plan View Display, it loads all the plug-ins in the `./plugin/binxx` subdirectory that match the version of the Plan View Display that you are running, for example, the HLA version. The Plan View Display creates a `PvPluginAppInterface` and gives it to each plug-in. The plug-in communicates with the Plan View Display through a set of interfaces. These interfaces include:

- ♦ `PvPluginAppInterface` - a high level interface for interacting with the Plan View Display application. It maintains a list of all Plan View Display windows.
- ♦ `PvPluginCanvasInterface` - allows you to get information about and manipulate individual Plan View Display windows.
- ♦ `PvPluginObjectListInterface` - maintains a list of all objects and lets you create objects.
- ♦ `PvPluginObjectInterface` - a generic object interface with subclasses for each of the objects you can create.

In your plug-in, you register callbacks for the events you want to respond to. When such an event occurs, the callback calls the appropriate function within your plug-in.

The remainder of this chapter describes what you can do with the plug-in classes and functions. It ends with a sample plug-in.



The plug-in interface is not threadsafe. If your plug-in is multi-threaded, be sure to make all VR-Link and Plan View Display calls in the same thread.

7.2. Working with Plug-in Interfaces

This section discusses the functions for working with interfaces. An interface is an object that gives a plug-in access to part of the Plan View Display application, for example, a list of the objects in the Plan View Display, or a list of the Plan View windows.

Getting Lists of Interfaces

The plug-in classes have functions that let you get lists of the various interfaces. The functions are:

- ♦ `PvPluginAppInterface::canvasInterfaceList()` - returns a list of *PvPluginCanvasInterfaces*, which represent each Plan View window
- ♦ `PvPluginObjectListInterface::objectInterfaceList()` - returns a list of all *PvPluginObjectInterfaces*, which represent the objects in the simulation.

Getting Pointers to Interfaces

The plug-in classes have functions that let you get pointers to the various interfaces. The functions are:

- ♦ `PvPluginAppInterface::objectListInterface()` - returns a pointer to the *PvObjectListInterface*
- ♦ `PvPluginCanvasInterface::appInterface()` - returns a pointer to the *PvPluginAppInterface*
- ♦ `PvPluginObjectListInterface::appInterface()` - returns a pointer to the *PvPluginAppInterface*
- ♦ `PvPluginObjectInterface::objectListInterface()` - returns a pointer to the *PvPluginObjectListInterface*.

7.3. Responding to Events

Like other applications based on VR-Link, a plug-in uses callbacks to initiate responses to user events and application events such as state changes and interactions.

Callbacks should be regular global (C-style) functions or static member functions, but cannot be non-static member functions. The function prototype must match the prototype given in the particular add callback function.

The value passed to your callback function as *usr* is the value that was passed as the *usr* argument to the `addCallback()` function. Often, this *usr* argument passes a pointer to an object on which you would like to call a member function from within your callback. Cast *usr* back to the object's type within the callback. For example (from the sample plug-in):

```
void PvTest::windowCb(PvPluginCanvasInterface* source,
    PvPluginGenericMessageType type, void* usr,
    PvPluginAppInterface* appInterface)
{
    // If a new window has been created, then add the "Drop Bomb"
    // menu item.
    if (type == PvPluginGenericAddedMessage)
    {
        source->addItemToMenu("Functions", "Drop Bomb",
            "Detonates a bomb at the currently selected location",
            PvTest::bombCb, usr);
    }
}

void PvTest::bombCb(void* usr, void* canvasInterface)
{
    // Call back into an instance of the PvTest class.
    ((PvTest*) usr)->dropBomb((PvPluginCanvasInterface*) canvasInterface);
}
```

In cases where you would like to pass more than one object to your callback, you must create a simple structure consisting of pointers to your objects, then pass a pointer to the structure as *usr*.



Any pointer you pass as *usr* must be the address of a variable whose lifetime is at least as long as the period for which the callback function is registered.

7.3.1. Registering Callbacks Directly with An Application

You can register callbacks with the exercise connection of the Plan View Display that you are plugging into. Use `PvPluginAppInterface::exerciseConnection()`. For information about exercise connections and registering callbacks, see the VR-Link documentation.

7.3.2. Responding to Plan View Display Ticks

Your plug-in may need to take some action on a periodic basis, usually after each frame (or tick) of the application. You can add a callback that is run after each tick of the application with `PvPluginAppInterface::addTickCallback()`. Remove a callback with `PvPluginAppInterface::removeTickCallback()`. Remove all callbacks on the tick with `PvPluginAppInterface::removeAllTickCallbacks()`.

7.4. Managing Windows From A Plug-in

This section describes the functions that you can use to get information about and manipulate Plan View Display windows.

Responding To Window Creation And Deletion

Your application may want to respond when new Plan View Display windows are created and existing windows are deleted. *PvPluginAppInterface* has functions that let you add callbacks for window creation and deletion events. The callbacks are:

- ♦ `addCallbackForPlanViewWindow()`
- ♦ `removeCallbackForPlanViewWindow()`
- ♦ `removeAllCallbacksForPlanViewWindow()`.

Panning and Zooming Windows

PvPluginCanvasInterface has functions to change the view in a Plan View Display window. This allows you to, in essence, pan and zoom the window.

The function `getCanvasExtentsInDatabaseCoord()` returns upper left and lower right database coordinates that the window displays.

To change the view of the database, use `setCanvasExtentsInDatabaseCoord()`. This function takes the upper left and lower right coordinates of the view you want to display. For information about how you get coordinates and convert them between different formats, see [“Working with Coordinate Information,”](#) on page 7-8.

7.4.1. Extending Menus

PvPluginCanvasInterface has member functions that let you add sub-menus and menu items to the menus on the Plan View Display menu bar. You can also add items to the right-click menu.



Names of menus and sub-menus must be unique.

References to the default Plan View Display menus must match the names contained in the Plan View Display string tables. For information about the string tables, see [“Localizing the Plan View Display,”](#) on page 6-9.

To add an item to a menu or sub-menu, use `addItemToMenu()`. You must supply the following arguments with this function:

- ♦ The name of the menu
- ♦ The name of the item
- ♦ A brief description for status bar help
- ♦ The function to run when the menu item is selected.

To add a sub-menu to a menu, use `addMenuToMenu()`. You supply the following arguments:

- ♦ The name of the menu
- ♦ The name of the new submenu
- ♦ A brief description for status bar help.

After you add a submenu to a menu, you can add menu items to it.

To add an item to the right-click menu, use `addRightClickMenuItem()`. You supply the following arguments:

- ♦ The name of the menu item
- ♦ The function to run when it is selected.

7.4.2. Getting Terrain Information

PvPluginCanvasInterface has a function that lets you get terrain information.

- > To get the soil type, height, and slope of a given point, use `getTerrainInformation()`.
- > To get the name of the terrain database, use `PvPluginCanvasInterface::databaseName()`. It returns the full pathname of the database.

7.4.3. Working with Coordinate Information

PvPluginCanvasInterface has several functions that let you convert coordinate information between geocentric coordinates, terrain database coordinates, and the coordinates of a Plan View Display window. [Figure 7-1](#) displays the relationship of these different coordinates. The origin of the geocentric coordinate system is the center of the earth. The positive X-axis passes through the prime meridian at the equator; the positive Y-axis passes through 90 degrees east longitude at the equator; and the positive Z-axis passes through the north pole. A terrain database is some portion of the surface of the earth and coordinates are specified by the database. The canvas is a view of the terrain database, with the upper left corner having the coordinates 0,0.

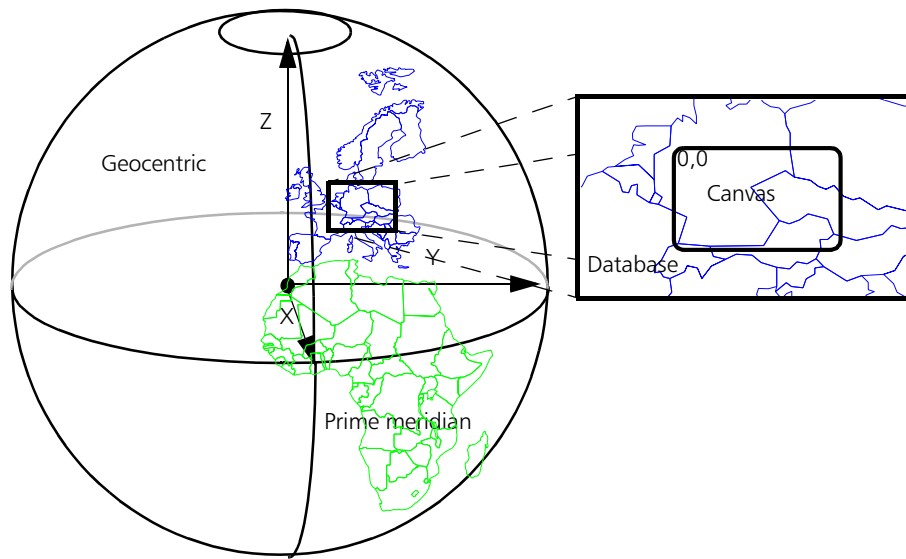


Figure 7-1. Relationship of geocentric, database, and canvas coordinates

Table 7-1 lists the conversion functions and the conversions they make.

Table 7-1: Coordinate conversion functions

Function	Description
<code>getCanvasCoordFromGeoc()</code>	Converts geocentric coordinates to coordinates in the canvas (screen coordinates.)
<code>getCanvasCoordFromDatabase()</code>	Converts database coordinates to coordinates on the canvas.
<code>getDatabaseCoordFromCanvas()</code>	Converts the canvas coordinates to database coordinates.
<code>getGeocCoordFromCanvas()</code>	Converts canvas coordinates to geocentric coordinates.
<code>getGeocCoordFromDatabase()</code>	Converts database coordinates to geocentric coordinates.
<code>getDatabaseCoordFromGeoc()</code>	Converts geocentric coordinates to database coordinates.

7.5. Responding to User Events

PvPluginCanvasInterface has callback functions for when a user changes their object selection or selects the ground. The callback functions are:

- ♦ `addCallbackForUserEvents()`
- ♦ `removeCallbackForUserEvents()`
- ♦ `removeAllCallbacksForUserEvents()`.

7.5.1. Getting Selection Information

Plan View Display users can select objects that are displayed in a Plan View window, and they can select points (locations) on the terrain. Plug-ins can get information about the current selection.

- > To get the list of interfaces of the objects selected by a user, use `PvPluginCanvasInterface::getSelectedObjectInterfaces()`.
- > To get the database coordinates of a point in a Plan View window that has been selected by a user, use `PvPluginCanvasInterface::getSelectedPoint()`.

i

Getting the location of a point that a user has selected is not the same thing as getting the coordinates of a point object. For information about point objects and how to get their location, see “[Working with Objects](#),” on page 7-10 and “[Getting Information About Geometric Objects](#),” on page 7-13.

7.6. Working with Objects

The plug-in API has several classes for working with objects.

PvPluginObjectListInterface has functions that allow you to:

- ♦ Add objects
- ♦ Remove objects
- ♦ Get a list of all objects
- ♦ Learn when objects are created and deleted.

PvPluginObjectInterface is a generic object interface with functions that let you perform activities common to all objects, such as:

- ♦ Adding and removing callbacks.
- ♦ Getting and setting the time-out value. See “[Timing Out Objects](#),” on page 7-12.
- ♦ Getting an object’s Plan View Display ID (`PvObjectId`).

Each object has an interface class with functions that allow you to get and set information specific to the object type. The object classes are:

- ♦ *PvPluginEntityObjectInterface*
- ♦ *PvPluginInteractionObjectInterface*
- ♦ *PvPluginPointObjectInterface*
- ♦ *PvPluginLineObjectInterface*
- ♦ *PvPluginPolygonObjectInterface*
- ♦ *PvPluginCircleObjectInterface*
- ♦ *PvPluginTextObjectInterface*.

7.6.1. Creating Objects

To create an object, use the appropriate `PvPluginObjectListInterface::addobjecttypeObject()` member function:

- ♦ Entity - `addEntityObject()`
- ♦ Interaction - `addInteractionObject()`
- ♦ Point - `addPointObject()`
- ♦ Line - `addLineObject()`
- ♦ Polygon - `addPolygonObject()`
- ♦ Circle - `addCircleObject()`
- ♦ Text - `addTextObject()`

7.6.2. Looking Up An Object Interface

When an object is created, it returns a `PvObjectId`. To look up an object interface using the `PvObjectId`, use `PvPluginObjectListInterface::lookup()`.

7.6.3. Getting A List of Objects

Since each object has an interface (not just those created by the plug-in, but all objects in the Plan View Display), getting a list of all object interfaces identifies all the objects. You do this with `PvPluginObjectListInterface::objectInterfaceList()`. From the interface list, you can get the `PvObjectId` for each object with `PvPluginObjectInterface::objectId()`.

7.6.4. Tracking Objects

One of the features of the Plan View Display is the ability to track an object. A plug-in might want to respond to a change in which objects are being tracked. *PvPluginCanvasInterface* has functions for adding and removing callbacks that respond when a user changes the object that is being tracked. The functions are:

- ♦ `addCallbackForTrackedObjects()`
- ♦ `removeCallbackForTrackedObjects()`
- ♦ `removeAllCallbacksForTrackedObjects()`.

7.6.5. Timing Out Objects

You can set timeout values (in seconds) on a Plan View Display-wide basis (all windows), for individual objects, or both. When an object times out, the Plan View Display removes its graphic representation from the canvas. If the object is a remote entity and the remote application sends an update, the object is added back to the canvas and the timeout period starts again.

- > To get the current timeout value, use `PvPluginObjectInterface::timeoutValue()`.
- > To set the timeout value for an object, use `PvPluginObjectInterface::setTimeoutValue()`.

A canvas argument of `NULL` causes the timeout to be applied to all Plan View windows. This is the default.

A timeout value of zero (0), causes the object to never be timed out. The default is the default of the Plan View Display application.

If you want to set a timeout for both individual objects and application-wide, set the application-wide timeout first. The individual time-outs will override the application-wide timeout.

7.6.6. Responding to Creation of Objects and Object Changes

You can set callbacks for the creation of objects, changes to an object, and deletion of objects.

The callbacks for creation and deletion of an object are:

- ♦ `PvPluginCanvasInterface::addCallbackOnObjectList()`
- ♦ `PvPluginCanvasInterface::removeCallbackOnObjectList()`
- ♦ `PvPluginCanvasInterface::removeAllCallbacksOnObjectList()`.

The callbacks for changes to an object's state and deletion of an object are:

- ♦ `PvPluginObjectInterface::addCallbackOnObject()`
- ♦ `PvPluginObjectInterface::removeCallbackOnObject()`
- ♦ `PvPluginObjectInterface::removeAllCallbacksOnObject()`.

7.6.7. Getting information about objects (state, coordinates, ID)

The functions of the individual object interfaces allow you to get object-specific information.

Getting Entity State Information

To get information about an entity, use the function `PvPluginEntityObjectInterface::esr()`.

You can get the global object designator for an entity. Use `PvPluginEntityObjectInterface::GlobalId()`.

i

The `PvObjectId` is an identifier specific to the Plan View Display. The global object designator is an identifier used to identify an object in an exercise.

Getting Interaction Data

To get data for an interaction, use `PvPluginInteractionObjectInterface::interaction()`.

Getting Information About Geometric Objects

You can get information about the location and characteristics of the geometric objects: points, lines, polygons, and circles. All location information is in geocentric coordinates. [Table 7-2](#) lists the object, type of information, and functions used to obtain it. In each case, the function is a member of the specific object class.

Table 7-2: Functions for getting geometric object data

Object	Data	Function
Point	Coordinates of the point.	<code>objectGeocLocation()</code>
Line	List of points that create the line.	<code>listOfVertices()</code>
Polygon	List of vertices that create the polygon.	<code>listOfVertices()</code>
Circle	Circumference and coordinates of the center	<code>centerGeocLocation()</code> <code>circumferenceGeocLocation()</code>
Text	Upper left point of the text string.	<code>objectGeocLocation()</code>

7.6.8. Setting Object Values

You can set values for some objects. All points are specified in geocentric coordinates. [Table 7-3](#) lists the objects, values you can set, and functions to set them. In each case, the function is a member of the specific object class.

Table 7-3: Functions for setting object values

Object	Value	Function
Entity	State attributes	setEsr()
Interaction		Use the functions for the interaction class.
Point	Coordinates	setObjectGeocLocation()
Line	Vertices	setVertices()
Polygon	Vertices	setVertices()
Circle	Center	setCenterGeocLocation()
	Circumference	setCircumferenceGeocLocation()
Text	Text string	setString()

7.6.9. Removing objects

You can remove an object by its interface or its `PvObjectId`. In either case, you use the `removeObject()` function, but supply different arguments.

- > To remove an object by its interface, use the function `PvPluginObjectListInterface::removeObject(PvPluginObjectInterface* objInterface)`.
- > To remove an object by its ID value, use the function `PvPluginObjectListInterface::removeObject(PvObjectId id)`.

7.7. Building and Compiling A Plug-in

The Plan View Display provides a sample makefile (UNIX) and project file (PC) for the sample plug-in. Use them as guides for setting up your own makefiles or project files.

To build and compile a plug-in, follow the instructions for building and compiling applications in the *VR-Link Developer's Guide*. In addition to the information provided in the VR-Link Developer's Guide, bear in mind the following points:

- ♦ When you build a plug-in, you are creating a shared library (.so or .dll) rather than an executable.
- ♦ When you build a plug-in, add the *./plugin/include* directory to your include path.

For PCs

- ♦ The project file for the drop bomb example is for use with Microsoft Visual C++ 5.0.
- ♦ Use the release build, not the debug build. It automatically copies the plug-in DLL to the proper location.
- ♦ For PCs, plug-ins must link with the RT version of the VR-Link libraries. To do this, the VR-Link DLLs need to be in the user's path at runtime.

After you compile a plug-in, put it in the subdirectory for the simulation standard with which you plan to use it.

7.8. Plug-in Example

This section presents an example of a plug-in. Although it doesn't use all the interfaces that the plug-in API provides, it demonstrates each of the essential components of a plug-in. The example is presented as a mini-tutorial to create some context and give you a sense of the thought process that might go into developing a plug-in.

7.8.1. Planning a Plug-in

Before you develop a plug-in, you need to decide what kind of functionality you'd like to add to the Plan View Display. In our example, we want to be able to drop bombs. These bombs will show up as detonations on the Plan View Display, and detonation interactions will be sent out on the network.

Planning the User Interface

After you decide what you want the plug-in to do, you need to decide how the user interface will work. The user of our plug-in needs to do two things: specify when to drop a bomb, and specify where to drop it. For specifying when to drop a bomb, we will add a menu item on the canvas menu that, when chosen, drops a bomb. For the location of the bomb, we will use the current selection: either a ground point or an entity.

The following code samples are part of the software distribution. They are in *./examples/dropbomb*.

A complete description of the sample plug-in follows the code listing.

Example Header File

The following code is the header file for the example plug-in:

```
1  #ifndef PvTest_H_
2  #define PvTest_H_
3
4  // class PvTest:
5  //
6  // Instances of PvTest are a dll to demo the plugins.
7
8  // It adds a menu item on the plan view window menu, under the
9  // functions category which causes a detonation interaction.
10
11 // Operation: Click on a point on the ground, or an entity, and
12 // then select the "drop bomb" menu item. A detonation symbol
13 // will appear on the canvas(es), and a detonation interaction
14 // will be output on the network.
15
16 #include "vlutil.h"
17 #include "plugAppInter.h"
18
19 class PvPluginCanvasInterface;
20
21 class PvTest
22 {
```

```

23     public:
24
25         // default constructor
26         PvTest();
27
28         // destructor
29         virtual ~PvTest();
30
31         // For initializing the plugin.
32         virtual void setup(PvPluginAppInterface* pluginInterface);
33
34         // Creates and sends the detonation at the selected location.
35         virtual void dropBomb(PvPluginCanvasInterface* canvasInterface);
36
37     public:
38
39         // Callback for the menu item.
40         static void bombCb(void* usr, void* canvasInterface);
41
42         // Callback for plan view windows.
43         static void windowCb(PvPluginCanvasInterface* source,
44                             PvPluginGenericMessageType type, void* usr,
45                             PvPluginAppInterface* myAppInterface);
46
47     protected:
48
49         PvPluginAppInterface* myAppInterface;
50     };
51
52     // These cause the entry points to be exported.
53     extern "C" EXPORT void* PvPluginCreator(PvPluginAppInterface*
54                                             pluginInterface);
55     extern "C" EXPORT void PvPluginDestroyer(void* plugin);
56
57     // This is the entry point for the PVD application to create and
58     // initialize the plugin.
59     extern "C"
60     {
61         inline void* PvPluginCreator(PvPluginAppInterface* pluginInterface)
62         {
63             // Check to make sure the versions match.
64             PvPluginAppInterface::PvPluginVersionInfo appVersion =
65                 pluginInterface->version();
66
67             #ifdef DtHLA
68                 if (appVersion != PvPluginAppInterface::PvPluginHLA)
69                 {
70                     return NULL;
71                 }
72             #else
73             #ifdef DtDISVERS != 3
74                 if (appVersion != PvPluginAppInterface::PvPluginDIS5)
75                 {
76                     return NULL;
77                 }
78             #else
79                 if (appVersion != PvPluginAppInterface::PvPluginDIS3)
80                 {
81                     return NULL;
82                 }
83             #endif
84             #endif
85         }
86     }
87     #endif

```

```
80     #endif
81
82     PvTest* myPvTest = new PvTest();
83     myPvTest->setup(pluginInterface);
84     return myPvTest;
85 }
86 }
87
88 // This is the entry point for the application to destroy the plugin.
89 extern "C"
90 {
91     inline void PvPluginDestroyer(void* plugin)
92     {
93         PvTest* aPlugin = (PvTest*) plugin;
94         DtDELETE(aPlugin);
95     }
96 }
97
98 #endif
99
100
```

Example Plug-in

The following code is the complete text of the example plug-in described in previous sections:

```
1  #include "test.h"
2  #include <stdio.h>
3  #include "vlprint.h"
4  #include "exerciseConn.h"
5  #include "detonateInter.h"
6  #include "plugAppInter.h"
7  #include "plugCanvInter.h"
8  #include "EntityTypes.h"
9  #include "plugEntObjIf.h"
10 #include "entitySR.h"
11 #include "plugObjLInter.h"
12
13
14 PvTest::PvTest() :
15     myAppInterface(NULL)
16 {
17 }
18
19
20 PvTest::~~PvTest()
21 {
22     // Delete the callbacks we added.
23     myAppInterface->removeCallbackForPlanViewWindow(PvTest::windowCb);
24 }
25
26
27 void PvTest::setup(PvPluginAppInterface* pluginInterface)
28 {
29     DtWarn("Setting up demo plugin (drop bomb)...\n");
30
31     // Keep track of the app interface, because we'll need it later.
32     myAppInterface = pluginInterface;
```

```
33
34     // Add a callback so that when a plan view window is created,
35     // I can add menu items to it.
36     myAppInterface->addCallbackForPlanViewWindow(PvTest::windowCb,
37         this);
38 }
39
40 void PvTest::windowCb(PvPluginCanvasInterface* source,
41     PvPluginGenericMessageType type, void* usr,
42     PvPluginAppInterface* appInterface)
43 {
44     // If a new window has been created, then add the "Drop Bomb"
45     // menu item.
46     if (type == PvPluginGenericAddedMessage)
47     {
48         source->addItemToMenu("Functions", "Drop Bomb",
49             "Detonates a bomb at the currently selected location",
50             PvTest::bombCb, usr);
51     }
52 }
53
54 void PvTest::bombCb(void* usr, void* canvasInterface)
55 {
56     // Call back into an instance of the PvTest class.
57     ((PvTest*) usr)->dropBomb((PvPluginCanvasInterface*)
58         canvasInterface);
59 }
60
61 void PvTest::dropBomb(PvPluginCanvasInterface* canvasInterface)
62 {
63     DtWarn("Dropping Bomb\n");
64
65     // Assemble the detonation interaction.
66     DtDetonationInteraction* interaction = new DtDetonationInteraction();
67     interaction->setVelocity(DtVector(0, 0, 0));
68     interaction->setRate(0);
69     interaction->setQuantity(1);
70     interaction->setFuseType(DtFuzeOther);
71     interaction->setMunitionType(DtEntityType(DtMunition, DtStrategic,
72         DtUnitedStates, 0, 0, 0, 0));
73     interaction->setWarheadType(DtWarheadOther);
74     interaction->setMunitionId(DtGlobalObjectDesignator());
75     interaction->setEventId(myAppInterface->exerciseConn()-
76         >nextEventId());
77     interaction->setEntityLocation(DtVector(0, 0, 0));
78
79     DtVector dbLoc;
80     if (canvasInterface->getSelectedPoint(dbLoc))
81     {
82         // If we clicked on a point, set up a ground detonation.
83         interaction->setResult(DtDetResDetonation);
84         DtVector worldLoc;
85         canvasInterface->getGeocCoordFromDatabase(dbLoc, worldLoc);
86         interaction->setWorldLocation(worldLoc);
87     }
88     else
89     {
```

```
90         // If we clicked on an entity, set up an entity detonation.
91         interaction->setResult(DtDetResEntityImpact);
92         DtList* objList = new DtList();
93         canvasInterface->getSelectedObjectInterfaces(objList);
94         // Only if we've selected a single object, which is an entity.
95         if (objList->count() == 1)
96         {
97             PvPluginObjectInterface* obj = (PvPluginObjectInterface*)
98                 objList->first()->data();
99             PvPluginEntityObjectInterface* entity =
100                 dynamic_cast<PvPluginEntityObjectInterface*>(obj);
101             if (entity)
102             {
103                 interaction->setWorldLocation(entity->esr()->location());
104                 interaction->setTargetId(entity->globalId());
105             }
106             else
107             {
108                 DtDELETE(interaction);
109                 return;
110             }
111         }
112         else
113         {
114             // If nothing is selected, abort.
115             DtDELETE(interaction);
116             return;
117         }
118     }
119
120     // Send out the interaction.
121     myAppInterface->exerciseConn()->sendStamped(*interaction);
122
123     #ifdef DtHLA
124         // Add the interaction event. In HLA, we don't read our own
125         // packets.
126         // The PVD will take care of deleting the interaction we created.
127         myAppInterface->objectListInterface()-
128             >addInteractionObject(interaction);
129     #else
130         // In DIS, we read our own packets.
131         DtDELETE(interaction);
132     #endif
133 }
```

7.8.2. Implementing the Plug-in

1. We need to create the entry points for the Plan View Display to create and destroy the plug-in. The following code is from the header file, *test.h*:

```
// These cause the entry points to be exported.
extern "C" EXPORT void* PvPluginCreator(PvPluginAppInterface*
    pluginInterface);
extern "C" EXPORT void PvPluginDestroyer(void* plugin);

// This is the entry point for the PVD application to create and
// initialize the plugin.
extern "C"
{
```

```
inline void* PvPluginCreator(PvPluginAppInterface* pluginInterface)
{
    // Check to make sure the versions match.
    PvPluginAppInterface::PvPluginVersionInfo appVersion =
        pluginInterface->version();
#ifdef DtHLA
    if (appVersion != PvPluginAppInterface::PvPluginHLA)
    {
        return NULL;
    }
#else
#ifdef DtDISVERS
    if (appVersion != PvPluginAppInterface::PvPluginDIS5)
    {
        return NULL;
    }
#else
    if (appVersion != PvPluginAppInterface::PvPluginDIS3)
    {
        return NULL;
    }
#endif
#endif
    PvTest* myPvTest = new PvTest();
    myPvTest->setup(pluginInterface);
    return myPvTest;
}

// This is the entry point for the application to destroy the plugin.
extern "C"
{
    inline void PvPluginDestroyer(void* plugin)
    {
        PvTest* aPlugin = (PvTest*) plugin;
        DtDELETE(aPlugin);
    }
}
```

2. We want to add the menu item to all of the canvas windows, so we register a callback on Plan View window creation as soon as the plug-in is attached to the Plan View Display. This ensures that every Plan View window that gets created will have our menu item. The following code starts at line 27 in the example. We add the callback in line 36. It calls the function `windowCb()`. Since window creation is at the highest level of the Plan View Display, we access it through the *PvPluginAppInterface*.

```
void PvTest::setup(PvPluginAppInterface* pluginInterface)
{
    DtWarn("Setting up demo plugin (drop bomb)...\n");

    // Keep track of the app interface, because we'll need it later.
    myAppInterface = pluginInterface;

    // Add a callback so that when a plan view window is created,
    // I can add menu items to it.
    myAppInterface->addCallbackForPlanViewWindow(PvTest::windowCb, this);
}
```

```
}
```

In the next-to-last line of code, “this” is a pointer to an instance of a class. Callbacks use static C-style functions, and in order to access a class instance, the static member function needs a pointer to it.

3. The callback is in lines 40 to 52. Menus are added to Plan View windows, which are accessed through the *PvPluginCanvasInterface*. The menu item added is “Drop Bomb”.

```
void PvTest::windowCb(PvPluginCanvasInterface* source,
    PvPluginGenericMessageType type, void* usr,
    PvPluginAppInterface* appInterface)
{
    // If a new window has been created, then add the "Drop Bomb"
    // menu item.
    if (type == PvPluginGenericAddedMessage)
    {
        source->addItemToMenu("Functions", "Drop Bomb",
            "Detonates a bomb at the currently selected location",
            PvTest::bombCb, usr);
    }
}
```

The types referred to in the “if” statement are listed in the header files.

4. When someone selects the Drop Bomb menu item, the function bombCb() gets called (lines 55 - 59). Selecting Drop Bomb is a user event, which we respond to with the callback dropBomb().

```
void PvTest::bombCb(void* usr, void* canvasInterface)
{
    // Call back into an instance of the PvTest class.
    ((PvTest*) usr)->dropBomb((PvPluginCanvasInterface*) canvasInterface);
}
```

5. The callback function dropBomb() is in lines 62 - 118. In lines 66 - 77, we assemble the interaction. In lines 79 - 87, we get the selection and check to see if it is a point. If it is, we do a ground detonation. The following code segment gets the coordinates of the selected point and converts them from database coordinates to geocentric coordinates.

```
DtVector dbLoc;
if (canvasInterface->getSelectedPoint(dbLoc))
{
    // If we clicked on a point, set up a ground detonation.
    interaction->setResult(DtDetResDetonation);
    DtVector worldLoc;
    canvasInterface->getGeocCoordFromDatabase(dbLoc, worldLoc);
    interaction->setWorldLocation(worldLoc);
}
```

If the selection is not a point, it is an entity. If it is one entity, we want to do an entity detonation (lines 88 - 105). This code segment demonstrates getting the interfaces of the objects selected by a user and checking the length of the list of interfaces.

```
else
```

```
{
    // If we clicked on an entity, set up an entity detonation.
    interaction->setResult(DtDetResEntityImpact);
    DtList* objList = new DtList();
    canvasInterface->getSelectedObjectInterfaces(objList);
    // Only if we've selected a single object, which is an entity.
    if (objList->count() == 1)
    {
        PvPluginObjectInterface* obj = (PvPluginObjectInterface*)
            objList->first()->data();
        PvPluginEntityObjectInterface* entity =
            dynamic_cast<PvPluginEntityObjectInterface*>(obj);
        if (entity)
        {
            interaction->setWorldLocation(entity->esr()->location());
            interaction->setTargetId(entity->globalId());
        }
    }
}
```

In this code segment we use both a hard cast and a dynamic cast. A dynamic cast does the cast only if it is possible. Otherwise it sets the pointer to NULL. A hard cast forces the cast regardless of the result, which could be a pointer to the wrong class. In the hard cast in this code segment, we know that the cast is possible.

To use dynamic casting, you need to compile with runtime type information. The project file and makefile supplied with the example plug-in include the runtime type information. For more information about casting and run-time type information, see your C++ references.

If the selection is neither a point or a single entity, we don't do anything (lines 106 - 118).

6. In line 121, we send the interaction.

```
myAppInterface->exerciseConn()->sendStamped(*pdu);
```

7. This example is designed to be protocol-independent: you can use it with HLA or DIS. However the protocols handle interactions differently, so we have conditional code for adding the interaction (lines 123 - 130):

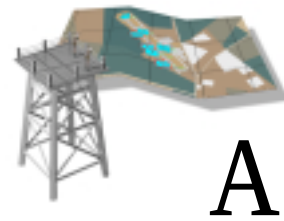
```
#ifdef DtHLA
    // Add the interaction event. In HLA, we don't read our own packets.
    // The PVD will take care of deleting the interaction we created.
    myAppInterface->objectListInterface()-
        >addInteractionObject(interaction);
#else
    // In DIS, we read our own packets.
    DtDELETE(interaction);
#endif
```

7.9. Plug-in Classes and Header Files

Table 7-4 lists the plug-in classes and the header files that contain them.

Table 7-4: Plug-in classes and header files

Class	Header File
PvPluginApplInterface	plugApplInter.h
PvPluginCanvasInterface	plugCanvInter.h
PvPluginCircleObjectInterface	plugCircObjIf.h
PvPluginEntityObjectInterface	plugEntObjIf.h
PvPluginInteractionObjectInterface	plugIntObjIf.h
PvPluginLineObjectInterface	plugLineObjIf.h
PvPluginObjectInterface	plugObjInter.h
PvPluginObjectListInterface	plugObjLInter.h
PvPluginPointObjectInterface	plugPntObjIf.h
PvPluginPolygonObjectInterface	plugPolyObjIf.h
PvPluginTextObjectInterface	plugTextObjIf.h



Plug-in Header Files

This appendix contains the code for the plug-in header files.

Class Hierarchy Diagram	A-2
plugAppInter.h	A-3
plugCanvInter.h	A-6
plugCircObjIf.h	A-10
plugEntObjIf.h	A-12
plugIntObjIf.h	A-13
plugLineObjIf.h	A-14
plugObjInter.h	A-16
plugObjLInter.h	A-19
plugPntObjIf.h	A-22
plugPolyObjIf.h	A-23
plugTextObjIf.h	A-25

A.1. Class Hierarchy Diagram

Figure A-1 is a diagram of the Plan View Display plug-in classes.

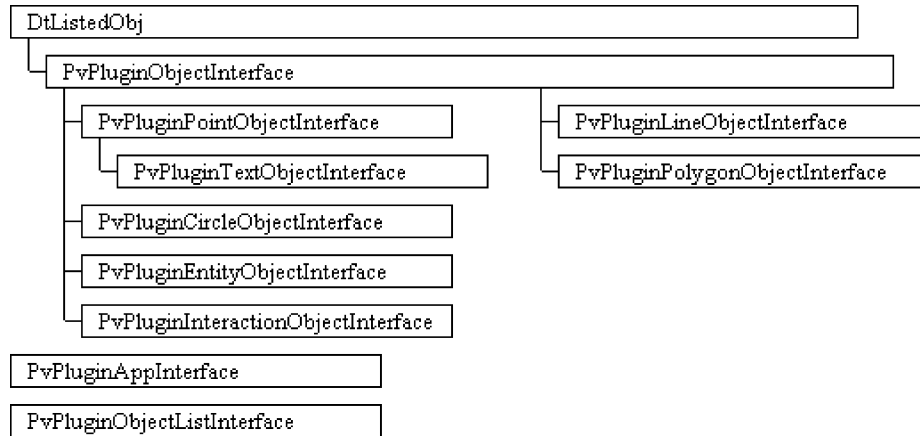


Figure A-1. Plan View Display plug-in class hierarchy

plugApplInter.h

```
#include "rMsgConduit.h"
#include "rMsgConNotf.h"
#include "vlTime.h"
#include "reflEntList.h"

#ifdef WIN32
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

enum PvPluginGenericMessageType
{
    PvPluginGenericAddedMessage,
    PvPluginGenericChangedMessage,
    PvPluginGenericDeletedMessage,
    PvPluginGenericOtherMessage
};

typedef void (*PvPluginPvCbFunc)(PvPluginCanvasInterface* source,
    PvPluginGenericMessageType type, void* usr,
    PvPluginAppInterface* myAppInterface);
typedef void (*PvPluginTickCbFunc)(void* usr,
    PvPluginAppInterface* myAppInterface);

// class PvPluginAppInterface:
//
// Instances of PvPluginAppInterface are the interface between the
// plugin and the application.

class PvPluginAppInterface
{
public:
    enum PvPluginVersionInfo
    {
        PvPluginDIS3,
        PvPluginDIS5,
        PvPluginHLA
    };

public:
    // default constructor
    PvPluginAppInterface(PvPvdApp *app);

    // destructor
    virtual ~PvPluginAppInterface();
};
```

```
// Returns the exercise connection.
virtual DtExerciseConn* exerciseConn();

// Add/Remove a callback on plan view window creation and destruction.
virtual void addCallbackForPlanViewWindow(PvPluginPvCbFunc cb,
    void* usr);
virtual DtBoolean removeCallbackForPlanViewWindow(
    PvPluginPvCbFunc cb);
virtual void removeAllCallbacksForPlanViewWindow();

// Add/Remove a function from the tick callback list
virtual void addTickCallback(PvPluginTickCbFunc cb, void* usr);
virtual DtBoolean removeTickCallback(PvPluginTickCbFunc cb);
virtual void removeAllTickCallbacks();

// Get the object list interface.
virtual PvPluginObjectListInterface* objectListInterface();

// Get the list of canvas interfaces.
virtual DtList* canvasInterfaceList();

// Get at the internals - requires additional header files and libraries
virtual PvPvdApp* accessInternals();

// Tick function, executed every cycle by the pvd app.
// Not intended as part of the interface for the plugin creator.
virtual void tick();

// Returns the version info, so that the plugin can match with the app.
virtual PvPluginVersionInfo version();

// Processes a message from the main frame.
// Not part of the interface for the plugin creator.
virtual DtBoolean processMessage(
    const PvCanvasWindowConduitMessage* message);

protected:

    // copy constructor
    PvPluginAppInterface(const PvPluginAppInterface& orig);

    // assignment operator
    PvPluginAppInterface& operator=(const PvPluginAppInterface& orig);

    // The remove functions above merely mark the callbacks for deletion.
    // These functions actually delete them.
    // This is done to prevent someone from corrupting the list while the
    // list is executing its callbacks.
    virtual void deleteMarkedTickCallbacks();
```

```
    virtual void deleteMarkedPlanViewCallbacks();

protected:

    PvPvdApp* myApp;

    PvPluginObjectListInterface* myObjectListInterface;

    DtList* myCanvasInterfaceList;

    DtList* myTickCbFunctions;
    DtList* myPvCbFunctions;

    DtRIGHT_END_NOTIFIER(PvPluginAppInterface)*
        myRightEndMsgConduit;
};
DtDECLARE_DEFINE_RIGHT_END_CLASS(PvPluginAppInterface,
    const PvCanvasWindowConduitMessage*);
```

plugCanvInter.h

```
#include "mathTypes.h"
#include "str.h"
#include "rMsgConduit.h"
#include "rMsgConNotf.h"

enum PvPluginUserEventType
{
    PvPluginObjectSelectionChanged,
    PvPluginGroundSelection,
};
enum PvPluginSurfaceType
{
    PvSurfaceWet,
    PvSurfaceWetOcean,
    PvSurfaceWetLake,
    PvSurfaceWetRiver,
    PvSurfaceWetSwamp,
    PvSurfaceWetMuck,
    PvSurfacePaved,
    PvSurfacePavedRoad,
    PvSurfaceUnpaved,
    PvSurfaceUnpavedSlowGo,
    PvSurfaceUnpavedNoGo,
    PvSurfaceOther,
    PvSurfaceUnspecified
};

typedef void (*PvPluginMenuCbFunc)(void* usr, void* canvasInterface);
typedef void (*PvPluginObjectMenuCbFunc)(void* usr, void* canvasInterface);

typedef void (*PvPluginTrackedObjCbFunc)(
    DtList* trackedObjectInterfaces,
    void* usr, PvPluginCanvasInterface* myCanvasInterface);

typedef void (*PvPluginUserEventCbFunc)(
    PvPluginUserEventType type,
    DtConstVector geocCoord, DtList* objectInterfaces,
    void* usr, PvPluginCanvasInterface* myCanvasInterface);

// class PvPluginCanvasInterface:
//
// Instances of PvPluginCanvasInterface are the interface between the plugin
// and a specific plan view window.

class PvPluginCanvasInterface
{
```

```
public:

    // constructor
    PvPluginCanvasInterface(PvPlanView* planView, PvPluginAppInterface*
appInterface);

    // destructor
    virtual ~PvPluginCanvasInterface();

    // Convert geocentric world/database coordinates from canvas coordinates.
    virtual void getCanvasCoordFromGeoc(DtConstVector geocCoord, DtVector&
canvasCoord);
    virtual void getCanvasCoordFromDatabase(DtConstVector databaseCoord,
DtVector& canvasCoord);

    // Convert geocentric world/canvas coordinates to database coordinates.
    virtual void getDatabaseCoordFromGeoc(DtConstVector geocCoord, DtVector&
databaseCoord);
    virtual void getDatabaseCoordFromCanvas(DtConstVector canvasCoord,
DtVector& databaseCoord);

    // Convert database/canvas coordinates to geocentric world coordinates.
    virtual void getGeocCoordFromDatabase(DtConstVector databaseCoord,
DtVector& geocCoord);
    virtual void getGeocCoordFromCanvas(DtConstVector canvasCoord, DtVector&
geocCoord);

    // Set/Get the rectangle that the canvas is viewing.
    virtual void setCanvasExtentsInDatabaseCoord(DtConstVector upperLeft,
DtConstVector lowerRight);
    virtual void getCanvasExtentsInDatabaseCoord(DtVector& upperLeft,
DtVector& lowerRight);

    // Get the height of terrain, soil type (if any), and ground normal at a
particular.
    virtual void getTerrainInformation(DtConstVector databaseCoord,
DtVector& intersectionCoord,
        PvPluginSurfaceType& surfaceType, DtVector& intersectNormal);

    // Get the filename of the database.
    virtual DtString databaseName();

    // Add an item to a top-level menu on the plan view menubar.
    virtual void addItemToMenu(DtString menu, DtString itemName, DtString
itemHelp,
        PvPluginMenuCbFunc cb, void* usr);

    // Add a menu to a top-level menu on the plan view menubar.
    virtual void addMenuToMenu(DtString menu, DtString newMenuName, DtString
menuHelp);
```

```
// Add/Remove a callback that is triggered when the tracked object(s)
change.
virtual void addCallbackForTrackedObjects(PvPluginTrackedObjCbFunc cb,
void* usr);
virtual DtBoolean removeCallbackForTrackedObjects(PvPluginTrackedObjCb-
Func cb);
virtual void removeAllCallbacksForTrackedObjects();

// Add/Remove a callback that is triggered for a user event.
virtual void addCallbackForUserEvents(PvPluginUserEventCbFunc cb, void*
usr);
virtual DtBoolean removeCallbackForUserEvents(PvPluginUserEventCbFunc
cb);
virtual void removeAllCallbacksForUserEvents();

// Add a right click menu item.
virtual void addRightClickMenuItem(DtString name, PvPluginObjectMenuCb-
Func cb, void* usr);

// Get a list of interfaces to the currently selected objects.
// SelectedList should be an empty list which will be filled
// with object interfaces.
virtual void getSelectedObjectInterfaces(DtList* selectedList);

// Gets the currently selected point in database coordinates.
// Returns DtFalse if no point is selected.
virtual DtBoolean getSelectedPoint(DtVector& databaseCoord);

// Get the app interface.
virtual PvPluginAppInterface* appInterface();

// Get at the internals - requires additional header files and libraries
virtual PvPlanView* accessInternals();

// Processes a message from the main frame.
// Not part of the interface for the plugin creator.
virtual DtBoolean processMessage(
    const PvEntityGroupListConduitMessage* message);

// Processes a message from the canvas.
// Not part of the interface for the plugin creator.
virtual DtBoolean processMessage(
    const PvCanvasSelectionConduitMessage* message);

protected:

// copy constructor
PvPluginCanvasInterface(const PvPluginCanvasInterface& orig);

// assignment operator
PvPluginCanvasInterface& operator=(const PvPluginCanvasInterface& orig);
```

```

// The remove functions above merely mark the callbacks for deletion.
// These functions actually delete them.
// This is done to prevent someone from corrupting the list while the
// list is executing its callbacks.
virtual void deleteMarkedUserEventCallbacks();
virtual void deleteMarkedTrackedObjectCallbacks();

protected:

// Handle the right click menu callbacks
static void rightClickCb(void* info, void* planViewCanvas);

// Handle the menubar menu callbacks
static void menuCb(void* info, void* planView);

protected:

int myIdForMenuCreation;

PvPlanView* myPlanView;
PvPluginAppInterface* myAppInterface;

DtList* myTrackingCbFunctions;
DtList* myUserEventCbFunctions;

DtRIGHT_END_NOTIFIER(PvPluginCanvasInterface)*
    myRightEndMsgConduit;
DtRIGHT_END_NOTIFIER_WITH_MOD(PvPluginCanvasInterface, Selection)*
    mySelectionRightEndMsgConduit;
};
DtDECLARE_DEFINE_RIGHT_END_CLASS(PvPluginCanvasInterface,
    const PvEntityGroupListConduitMessage*);

DtDECLARE_DEFINE_RIGHT_END_CLASS_WITH_MOD(PvPluginCanvasInterface,
    Selection,
    const PvCanvasSelectionConduitMessage*);

```

plugCircObjIf.h

```
#include "plugObjInter.h"

// class PvPluginCircleObjectInterface:
//
// Instances of PvPluginCircleObjectInterface are the interface between
// the plugin and a circle sim object.

class PvPluginCircleObjectInterface : public PvPluginObjectInterface
{
public:

    enum PvPluginFillStyle
    {
        PvPluginFillStyleSolid,
        PvPluginFillStyleHorizontalHatch,
        PvPluginFillStyleVerticalHatch,
        PvPluginFillStyleCrossHatch,
        PvPluginFillStyleDiagonalCrossHatch,
        PvPluginFillStyleDiagonalForwardHatch,
        PvPluginFillStyleDiagonalBackwardHatch,
        PvPluginFillStyleTransparent,
    };

public:
    // constructor
    PvPluginCircleObjectInterface(PvCircleSimObject* simObject,
        PvPluginObjectListInterface* interfaceList);

    // destructor
    virtual ~PvPluginCircleObjectInterface();

    // Set/Get the circle's center location in geocentric world coordinates.
    virtual void setCenterGeocLocation(DtConstVector geocCoord);
    virtual DtConstVector centerGeocLocation();

    // Set/Get the circle's circumference location in geocentric world coordinates.
    virtual void setCircumferenceGeocLocation(DtConstVector geocCoord);
    virtual DtConstVector circumferenceGeocLocation();

    // Set/Get the circle's fill color on a per canvas basis.
    // A NULL value for the canvas sets it application wide.
    virtual void setFillColor(unsigned char red, unsigned char green,
        unsigned char blue, PvPluginCanvasInterface* canvas = NULL);
    virtual void getFillColor(unsigned char* red, unsigned char* green,
        unsigned char* blue, PvPluginCanvasInterface* canvas);
```

```
// Set/Get the circle's fill style on a per canvas basis.
// A NULL value for the canvas sets it application wide.
virtual void setFillStyle(PvPluginFillStyle style,
    PvPluginCanvasInterface* canvas = NULL);
virtual PvPluginFillStyle fillStyle(PvPluginCanvasInterface* canvas);

protected:

    // copy constructor
    PvPluginCircleObjectInterface(const PvPluginCircleObjectInterface&
orig);

    // assignment operator
    PvPluginCircleObjectInterface& operator=(const PvPluginCircleObjectInt-
erface& orig);

protected:

    PvCircleSimObject* myCircleSimObject;
};
```

plugEntObjIf.h

```
#include "plugObjInter.h"

// class PvPluginEntityObjectInterface:
//
// Instances of PvPluginEntityObjectInterface are the interface between
// the plugin and a single entity sim object.

class PvPluginEntityObjectInterface : public PvPluginObjectInterface
{
public:

    // constructor
    PvPluginEntityObjectInterface(PvSingleEntitySimObject* simObject,
        PvPluginObjectListInterface* interfaceList);

    // destructor
    virtual ~PvPluginEntityObjectInterface();

    // Set/Get the entity state repository.
    virtual void setEsr(DtEntityStateRepository* esr);
    virtual DtEntityStateRepository* esr();

    // Get the entity's object's global object designator
    virtual const DtGlobalObjectDesignator& globalId() const;

protected:

    // copy constructor
    PvPluginEntityObjectInterface(const PvPluginEntityObjectInterface&
orig);

    // assignment operator
    PvPluginEntityObjectInterface& operator=(const
PvPluginEntityObjectInterface& orig);

protected:

    PvSingleEntitySimObject* myEntitySimObject;
};
```

plugIntObjIf.h

```
#include "plugObjInter.h"
#include "interaction.h"
// class PvPluginInteractionObjectInterface:
//
// Instances of PvPluginInteractionObjectInterface are the interface
// between the plugin and a base interaction.

class PvPluginInteractionObjectInterface : public PvPluginObjectInterface
{
public:
    // constructor
    PvPluginInteractionObjectInterface(PvBaseInteractionSimObject* simOb-
ject,
    PvPluginObjectListInterface* interfaceList);

    // destructor
    virtual ~PvPluginInteractionObjectInterface();

    // Get the interaction
    virtual DtInteraction* interaction();

protected:
    // copy constructor
    PvPluginInteractionObjectInterface(const PvPluginInteractionObjectInter-
face& orig);

    // assignment operator
    PvPluginInteractionObjectInterface& operator=(const PvPluginInteraction-
ObjectInterface& orig);

protected:
    PvBaseInteractionSimObject* myInteractionSimObject;
};
```

plugLineObjlf.h

```
#include "plugObjInter.h"

// class PvPluginLineObjectInterface:
//
// Instances of PvPluginLineObjectInterface are the interface between
// the plugin and a line sim object. Vertices are represented by
// DtVectors in geocentric world coordinates.

class PvPluginLineObjectInterface : public PvPluginObjectInterface
{
public:

    // constructor
    PvPluginLineObjectInterface(PvLineSimObject* simObject,
        PvPluginObjectListInterface* interfaceList);

    // destructor
    virtual ~PvPluginLineObjectInterface();

    // Set/Get a particular vertex.
    // Vertices are numbered starting at zero.
    // Returns a zero vector if the index is not found.
    virtual DtConstVector vertex(unsigned int vertexNumber);
    virtual void setVertex(unsigned int vertexNumber,
        DtConstVector geocCoord);

    // Adds a vertex after another vertex.
    // Returns DtTrue if successful.
    virtual DtBoolean addVertexAfterVertex(DtConstVector geocCoord,
        unsigned int vertexNumber);

    // Adds a vertex to the start/end
    virtual void addVertexToStart(DtConstVector geocCoord);
    virtual void addVertexToEnd(DtConstVector geocCoord);

    // Removes a vertex.
    // Returns DtTrue if successful.
    virtual DtBoolean deleteVertex(unsigned int vertexNumber);

    // Gets the number of vertices
    virtual unsigned int numberOfVertices();

    // Get the list of vertices. The list items contain pointers to
    // DtVectors in geocentric world coordinates.
    // It is recommended that you use the other functions to access
    // or modify the list.
    // If you do use this function to modify the list, be sure to
```

```
// call update() afterwards.
virtual DtList* listOfVertices();

protected:

    // copy constructor
    PvPluginLineObjectInterface(const PvPluginLineObjectInterface& orig);

    // assignment operator
    PvPluginLineObjectInterface& operator=(const PvPluginLineObjectInter-
face& orig);

protected:

    PvLineSimObject* myLineSimObject;
};
```

plugObjInter.h

```
#include "rMsgConduit.h"
#include "rMsgConNotf.h"
#include "str.h"
#include "mathTypes.h"
#include "vlTime.h"
#include "pvdTypes.h"
#include "plugAppInter.h"

typedef void (*PvPluginObjectCbFunc)(PvPluginGenericMessageType type,
    void* usr, PvPluginObjectInterface* myObjInterface);

// class PvPluginObjectInterface:
//
// Instances of PvPluginObjectInterface are the interface between
// the plugin and an object (sim object) in the PVD.

class PvPluginObjectInterface : public DtListedObj
{
public:

    enum PvPluginDrawingStyle
    {
        PvPluginDrawingStyleSolid,
        PvPluginDrawingStyleDot,
        PvPluginDrawingStyleDash,
        PvPluginDrawingStyleTransparent,
    };

public:

    // constructor
    PvPluginObjectInterface(PvSimObject* object,
        PvPluginObjectListInterface* listInterface);

    // destructor
    virtual ~PvPluginObjectInterface();

    // Add/Remove callback on the object.
    // It is especially easy to create a loop which can corrupt the callback
list
    // and crash the program.
    // For example, do not, in your callback, delete a callback and do
anything
    // which causes the entity to update.
    virtual void addCallbackOnObject(PvPluginObjectCbFunc cb, void* usr);
    virtual DtBoolean removeCallbackOnObject(PvPluginObjectCbFunc cb);
    virtual void removeAllCallbacksOnObject();
```

```

// Get the object id.
virtual PvObjectId objectId();

// Get the object list interface
virtual PvPluginObjectListInterface* objectListInterface();

// Force an update.
virtual void update();

// Set/Get the object's annotation.
virtual void setAnnotation(const DtString& str);
virtual const DtString& annotation() const;

// Set/Get the object's selection status on a per canvas basis.
// A NULL value for the canvas sets it application wide.
virtual void setSelection(DtBoolean value, PvPluginCanvasInterface*
canvas = NULL);
virtual DtBoolean selection(PvPluginCanvasInterface* canvas);

// Set/Get the object's hidden status on a per canvas basis.
// A NULL value for the canvas sets it application wide.
virtual void setHidden(DtBoolean value, PvPluginCanvasInterface* canvas =
NULL);
virtual DtBoolean hidden(PvPluginCanvasInterface* canvas);

// Set/Get the object's timeout value on a per canvas basis.
// A zero value for the time will disable timeout processing.
// A NULL value for the canvas sets or gets it application wide.
virtual void setTimeoutValue(DtTime time, PvPluginCanvasInterface*
canvas = NULL);
virtual DtTime timeoutValue(PvPluginCanvasInterface* canvas = NULL);

// Set/Get the object's color on a per canvas basis.
// If the object does not have settable colors, these do nothing.
// A NULL value for the canvas sets it application wide.
virtual void setColor(unsigned char red, unsigned char green, unsigned
char blue,
    PvPluginCanvasInterface* canvas = NULL);
virtual void getColor(unsigned char* red, unsigned char* green, unsigned
char* blue,
    PvPluginCanvasInterface* canvas);

// Set/Get the object's drawing style on a per canvas basis.
// If the object does not have settable drawing style, these do nothing.
// A NULL value for the canvas sets it application wide.
virtual void setDrawingStyle(PvPluginDrawingStyle style,
    PvPluginCanvasInterface* canvas = NULL);
virtual PvPluginDrawingStyle drawingStyle(PvPluginCanvasInterface*
canvas);

```

```
// Set/Get the object's pen width on a per canvas basis.
// If the object does not have settable pen width, these do nothing.
// A NULL value for the canvas sets it application wide.
virtual void setDrawingWidth(int width, PvPluginCanvasInterface* canvas =
NULL);
virtual int drawingWidth(PvPluginCanvasInterface* canvas);

// Get at the internals - requires additional header files and libraries
virtual PvSimObject* accessInternals();

// Processes a message from the main frame.
// Not part of the interface for the plugin creator.
virtual DtBoolean processMessage(
    const PvBaseObjectConduitMessage* message);

protected:

    // copy constructor
    PvPluginObjectInterface(const PvPluginObjectInterface& orig);

    // assignment operator
    PvPluginObjectInterface& operator=(const PvPluginObjectInterface& orig);

    // The remove function above merely mark the callbacks for deletion.
    // This function actually deletes them.
    // This is done to prevent someone from corrupting the list while the
    // list is executing its callbacks.
    virtual void deleteMarkedObjectCallbacks();

protected:

    PvSimObject* myObject;
    PvPluginObjectListInterface* myObjectListInterface;

    DtList* myObjectCbFunctions;

    DtRIGHT_END_NOTIFIER(PvPluginObjectInterface)*
        myRightEndMsgConduit;
};
DtDECLARE_DEFINE_RIGHT_END_CLASS(PvPluginObjectInterface,
    const PvBaseObjectConduitMessage*);
```

plugObjLInter.h

```
#include "mathTypes.h"
#include "pvdTypes.h"
#include "rMsgConduit.h"
#include "rMsgConNotf.h"
#include "plugAppInter.h"

typedef void (*PvPluginObjectListCbFunc)(PvPluginGenericMessageType type,
    PvObjectId id, void* usr, PvPluginObjectListInterface* myObjListInter-
    face);

// class PvPluginObjectListInterface:
//
// Instances of PvPluginObjectListInterface are the interface between
// the plugin and the sim object list.

class PvPluginObjectListInterface
{
public:

    // constructor
    PvPluginObjectListInterface(PvSimObjectList* objList,
        PvPluginAppInterface* appInterface);

    // destructor
    virtual ~PvPluginObjectListInterface();

    // Add objects of various types.
    virtual PvObjectId addEntityObject (DtEntityStateRepository* esr,
        DtGlobalObjectDesignator globalId);
    virtual PvObjectId addInteractionObject(DtInteraction* interaction);
    virtual PvObjectId addPointObject(DtConstVector geocCoord);
    virtual PvObjectId addLineObject(DtList* listOfGeocCoords);
    virtual PvObjectId addPolygonObject(DtList* listOfGeocCoords);
    virtual PvObjectId addCircleObject(DtConstVector centerGeocCoord,
        DtConstVector circumferenceGeocCoord);
    virtual PvObjectId addTextObject(DtConstVector geocCoord,
        const DtString& text);

    // Remove an object.
    virtual DtBoolean removeObject(PvPluginObjectInterface* objInterface);
    virtual DtBoolean removeObject(PvObjectId id);

    // Get the list of object interfaces.
    virtual const DtList* objectInterfaceList();

    // Lookup an object interface.
    virtual PvPluginObjectInterface* lookup(PvObjectId id);
```

```
// Add/Remove a callback on the sim object list.
virtual void addCallbackOnObjectList(PvPluginObjectListCbFunc cb, void*
usr);
virtual DtBoolean removeCallbackOnObjectList(PvPluginObjectListCbFunc
cb);
virtual void removeAllCallbacksOnObjectList();

// Get the plugin app interface.
virtual PvPluginAppInterface* appInterface();

// Get the interface factory.
virtual PvPluginObjectInterfaceFactory* interfaceFactory();

// Get at the internals - requires additional header files and libraries
virtual PvSimObjectList* accessInternals();

// Processes a message from the main frame.
// Not part of the interface for the plugin creator.
virtual DtBoolean processMessage(
    const PvBaseObjectConduitMessage* message);

protected:

// copy constructor
PvPluginObjectListInterface(const PvPluginObjectListInterface& orig);

// assignment operator
PvPluginObjectListInterface& operator=(
    const PvPluginObjectListInterface& orig);

// Adds/Removes an object interface from the list only.
virtual PvPluginObjectInterface* addListObject(PvObjectId id);
virtual void removeListObject(PvObjectId id);

// The remove function above merely mark the callbacks for deletion.
// This function actually deletes them.
// This is done to prevent someone from corrupting the list while the
// list is executing its callbacks.
virtual void deleteMarkedObjectListCallbacks();

protected:

PvSimObjectList* mySimObjectList;

PvPluginAppInterface* myAppInterface;

DtHashlist* myObjectInterfaceList;

DtList* myObjectListCallbacks;
```

```
DtRIGHT_END_NOTIFIER(PvPluginObjectListInterface)*
    myRightEndMsgConduit;

    PvPluginObjectInterfaceFactory* myInterfaceFactory;
};
DtDECLARE_DEFINE_RIGHT_END_CLASS(PvPluginObjectListInterface,
    const PvBaseObjectConduitMessage*);
```

plugPntObjIf.h

```
#include "plugObjInter.h"
#include "mathTypes.h"

// class PvPluginPointObjectInterface:
//
// Instances of PvPluginPointObjectInterface are the interface between
// the plugin and a point sim object.

class PvPluginPointObjectInterface : public PvPluginObjectInterface
{
public:

    // constructor
    PvPluginPointObjectInterface(PvPointSimObject* simObject,
        PvPluginObjectListInterface* interfaceList);

    // destructor
    virtual ~PvPluginPointObjectInterface();

    // Set/Get the point location
    virtual void setObjectGeocLocation(DtConstVector geocCoord);
    virtual DtConstVector objectGeocLocation();

protected:

    // copy constructor
    PvPluginPointObjectInterface(const PvPluginPointObjectInterface& orig);

    // assignment operator
    PvPluginPointObjectInterface& operator=(const
PvPluginPointObjectInterface& orig);

protected:

    PvPointSimObject* myPointSimObject;
};
```

plugPolyObjIf.h

```
#include "plugObjInter.h"

// class PvPluginPolygonObjectInterface:
//
// Instances of PvPluginPolygonObjectInterface are the interface
// between the plugin and a polygon sim object. Vertices are
// represented by DtVectors in geocentric world coordinates.
// The last vertex is connected to the first.

class PvPluginPolygonObjectInterface : public PvPluginObjectInterface
{
public:

    enum PvPluginFillStyle
    {
        PvPluginFillStyleSolid,
        PvPluginFillStyleHorizontalHatch,
        PvPluginFillStyleVerticalHatch,
        PvPluginFillStyleCrossHatch,
        PvPluginFillStyleDiagonalCrossHatch,
        PvPluginFillStyleDiagonalForwardHatch,
        PvPluginFillStyleDiagonalBackwardHatch,
        PvPluginFillStyleTransparent,
    };

public:

    // constructor
    PvPluginPolygonObjectInterface(PvPolygonSimObject* simObject,
        PvPluginObjectListInterface* interfaceList);

    // destructor
    virtual ~PvPluginPolygonObjectInterface();

    // Set/Get a particular vertex.
    // Vertices are numbered starting at zero.
    // Returns a zero vector if the index is not found.
    virtual DtConstVector vertex(unsigned int vertexNumber);
    virtual void setVertex(unsigned int vertexNumber,
        DtConstVector geocCoord);

    // Adds a vertex after another vertex.
    // Returns DtTrue if successful.
    virtual DtBoolean addVertexAfterVertex(DtConstVector geocCoord,
        unsigned int vertexNumber);

    // Adds a vertex to the start/end
```

```
virtual void addVertexToStart(DtConstVector geocCoord);
virtual void addVertexToEnd(DtConstVector geocCoord);

// Removes a vertex.
// Returns DtTrue if successful.
virtual DtBoolean deleteVertex(unsigned int vertexNumber);

// Gets the number of vertices
virtual unsigned int numberOfVertices();

// Get the list of vertices. The list items contain pointers to
// DtVectors in geocentric world coordinates.
// It is recommended that you use the other functions to access
// or modify the list.
// If you do use this function to modify the list, be sure to
// call update() afterwards.
virtual DtList* listOfVertices();

// Set/Get the polygon's fill color on a per canvas basis.
// A NULL value for the canvas sets it application wide.
virtual void setFillColor(unsigned char red, unsigned char green,
    unsigned char blue, PvPluginCanvasInterface* canvas = NULL);
virtual void getFillColor(unsigned char* red, unsigned char* green,
    unsigned char* blue, PvPluginCanvasInterface* canvas);

// Set/Get the polygon's fill style on a per canvas basis.
// A NULL value for the canvas sets it application wide.
virtual void setFillStyle(PvPluginFillStyle style,
    PvPluginCanvasInterface* canvas = NULL);
virtual PvPluginFillStyle fillStyle(PvPluginCanvasInterface* canvas);

protected:

    // copy constructor
    PvPluginPolygonObjectInterface(const PvPluginPolygonObjectInterface&
orig);

    // assignment operator
    PvPluginPolygonObjectInterface& operator=(const PvPluginPolygonOb-
jectInterface& orig);

protected:

    PvPolygonSimObject* myPolygonSimObject;
};
```

plugTextObjIf.h

```
#include "plugPntObjIf.h"

// class PvPluginTextObjectInterface:
//
// Instances of PvPluginTextObjectInterface are the interface between
// a plugin and a text sim object.

class PvPluginTextObjectInterface : public PvPluginPointObjectInterface
{
public:
    enum PvPluginFontFamilyType
    {
        PvPluginFontFamilyRoman,
        PvPluginFontFamilyModern,
        PvPluginFontFamilyDefault
    };

    enum PvPluginFontStyleType
    {
        PvPluginFontStyleNormal,
        PvPluginFontStyleItalic
    };

    enum PvPluginFontWeightType
    {
        PvPluginFontWeightNormal,
        PvPluginFontWeightBold
    };

public:
    // constructor
    PvPluginTextObjectInterface(PvTextSimObject* simObject,
        PvPluginObjectListInterface* interfaceList);

    // destructor
    virtual ~PvPluginTextObjectInterface();

    // Set/Get the object's text string.
    virtual void setString(const DtString& str);
    virtual const DtString& string() const;

    // Set/Get the object's font information.
    // The get routines requires a canvas.
    // The set routines will apply the change to all canvases if none is
    provided.
    // For now, these routines cannot be called from within an object added
    callback

```

```
// because the canvas may not have added the object yet.
virtual void setFontFamily(PvPluginFontFamilyType family,
    PvPluginCanvasInterface* canvas = NULL);
virtual PvPluginFontFamilyType fontFamily(PvPluginCanvasInterface*
canvas);
virtual void setFontStyle(PvPluginFontStyleType style,
    PvPluginCanvasInterface* canvas = NULL);
virtual PvPluginFontStyleType fontStyle(PvPluginCanvasInterface*
canvas);
virtual void setFontWeight(PvPluginFontWeightType weight,
    PvPluginCanvasInterface* canvas = NULL);
virtual PvPluginFontWeightType fontWeight(PvPluginCanvasInterface*
canvas);
virtual void setFontSize(int pointSize,
    PvPluginCanvasInterface* canvas = NULL);
virtual int fontSize(PvPluginCanvasInterface* canvas);

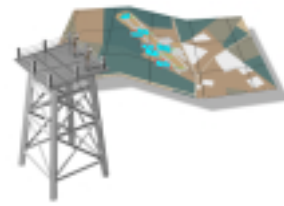
protected:

    // copy constructor
    PvPluginTextObjectInterface(const PvPluginTextObjectInterface& orig);

    // assignment operator
    PvPluginTextObjectInterface& operator=(const PvPluginTextObjectInter-
face& orig);

protected:

    PvTextSimObject* myTextSimObject;
};
```



Index

A

- A option 6-6
- About 3-5
- absolute view mode 4-13
- Acceleration 3-19, 3-24
- Accept button 3-12
- addCallbackForPlanViewWindow() function 7-7
- addCallbackForTrackedObjects() function 7-12
- addCallbackForUserEvents() function 7-10
- addCallbackOnObject() function 7-12
- addCallbackOnObjectList() function 7-12
- addCircleObject() function 7-11
- addEntityObject() function 7-11
- addInteractionObject() function 7-11
- addItemToMenu() function 7-7
- addLineObject() function 7-11
- addMenuToMenu() function 7-8
- addPointObject() function 7-11
- addPolygonObject() function 7-11
- addPostDrainCallback() function 7-5
- address
 - MÄK Technologies xii
- addRightClickMenuItem() function 7-8
- addTextObject() function 7-11
- addTickCallback() function 7-6
- angular increment 3-14
- Angular Velocity 3-19
- Appearance 3-19
- Appearance Flag 3-24
- appInterface() function 7-4
- Apply 3-12
- Apply Selection to View Mode 3-21
- attached entities
 - detaching 4-12

- attaching
 - entities to, in Stealth 4-11
 - Stealth to entities 4-11

B

- bin* directories 2-5
- bitmaps
 - associating with entity types 6-7
 - location of 2-5
- border
 - entity, described 5-2
 - icon 6-19

C

- C option 6-3
- callback 7-22
 - object state 7-12
 - on object 7-12
 - plug-in 7-5
 - registering 7-5
 - user event 7-10
- canvas
 - interface 7-4
- canvasInterfaceList() function 7-4
- Capabilities 3-19
- centerGeocLocation() function 7-13
- centering
 - window on point 4-2
- Change Range 5-11
- changing
 - bitmaps for entities 6-7
 - entity groups 5-5
 - intervisibility color 6-18

- settings 6-14
- strings in interface 6-9
- checking
 - intervisibility 5-11
- circle 7-13, 7-14
- class
 - PvPluginAppInterface* 7-3
 - PvPluginCanvasInterface* 7-3, 7-7
 - PvPluginCircleObjectInterface* 7-11
 - PvPluginEntityObjectInterface* 7-11
 - PvPluginInteractionObjectInterface* 7-11
 - PvPluginLineObjectInterface* 7-3
 - PvPluginObjectInterface* 7-3
 - PvPluginObjectListInterface* 7-3
 - PvPluginPointObjectInterface* 7-11
 - PvPluginPolygonObjectInterface* 7-11
 - PvPluginTextObjectInterface* 7-11
- Clear All button 3-17
- Close 3-6
- Close All Windows 3-5, 4-7
- closing
 - compared to hiding 4-7
 - Plan View 4-7
 - Plan View Display 3-5
 - windows 4-7
- color
 - intervisibility settings 6-18
- Color tab 6-18
- compiling
 - plug-in conditionally 7-23
- conditional compilation 7-23
- configuration file
 - location of 2-5
 - pvd*.smp 6-7
 - pvd.lab 6-7
 - pvd.set 6-7
 - pvd.stb 6-7
 - pvd.xrc 6-7
- configuring
 - entity labels 6-10
 - settings 6-11
 - Stealth 4-9
 - system for DMSO RTI 2-15
 - system for MÄK RTI 2-14
- console 6-2
 - messages sent to 6-3
 - opening 6-3
- contour 6-16
- Contour Line Frequency 3-12
- contour lines
 - frequency of 6-15
- ContourLineFrequency configuration setting 6-12
- Contribution Culling 3-12
- contribution culling
 - threshold 3-13
 - turning on and setting pixel size 6-17
- Contribution Culling Threshold 3-13
- ContributionCulling configuration setting 6-12
- ContributionCullingThreshold configuration setting 6-12
- Control tab
 - Settings dialog box 3-14
- controlling
 - messages 6-3
 - Plan View Display 3-5
 - Stealth 4-10
- conventions xiii
 - manual xiii
- converting
 - coordinates in plug-in 7-8
 - database format 4-14
- coordinate
 - converting in plug-in 7-8
 - setting and getting in plug-in 7-7
- coordinate types
 - supported 1-1
- coordinates
 - entity or ground point 3-8
- creating
 - entity groups 5-5
 - objects 7-11
 - callbacks on 7-12
 - Plan View window 3-4
 - terrain metafile 4-17
 - windows in plug-in 7-7
- CTDB database 3-9
- culling
 - contribution 6-17

D

- data
 - directory 2-5
- database, terrain
 - converting format 4-14
 - Load Database dialog box 3-9
 - loading 4-5
 - Save Database dialog box 3-10
 - saving in MÄK Terrain Format 4-14
 - supported 1-2

-
- databaseName() function 7-8
 - Dead Reckoning 3-12
 - DeadReckoning configuration setting 6-12
 - Delete Group 3-16
 - deleting
 - groups 5-6
 - markers 5-13
 - objects
 - callbacks on 7-12
 - windows in plug-in 7-7
 - Deselect All 3-7
 - deselecting
 - entities 5-4
 - destroying
 - marker 5-13
 - detaching
 - Stealth from entity 4-12
 - details
 - entity 3-18
 - dialog box
 - Entity Information 3-18
 - Entity Labels 3-23
 - Entity List 3-16
 - Intervisibility Settings 3-22
 - Load Database 3-9
 - Save Database 3-10
 - Settings 3-11
 - Stealth Control 3-20
 - directory
 - bin* 2-5
 - data 2-5
 - installed 2-4
 - plugins 7-15
 - DIS
 - application number 6-2
 - executables 2-5
 - exercise ID, specifying 6-2, 6-6
 - port number 6-6
 - site ID 6-2
 - specifying broadcast modes 6-6
 - startup options 6-2, 6-6
 - Dismiss 3-19
 - display
 - settings 3-11
 - Display Contour Lines 3-12
 - Display Coordinates 3-19
 - Display Entities 3-12
 - Display Grid Lines 3-12
 - Display Interactions 3-12
 - Display tab 3-12
 - Display Terrain 3-12
 - Display Wireframe 3-12
 - DisplayContourLines configuration setting 6-12
 - DisplayEntities configuration setting 6-12
 - DisplayGridLines configuration setting 6-12
 - displaying
 - labels 5-10
 - Plan View window 4-6
 - DisplayInteractions configuration setting 6-12
 - DisplayTerrain configuration setting 6-12
 - DisplayWireframe configuration setting 6-12
 - DMSO RTI 2-15
 - configuring 2-15
 - documentation, location of 2-5
 - DR Algorithm 3-19
 - dropping
 - marker 5-13
 - DTED 4-17
 - duration of interactions
 - setting 6-16
- ## E
- entities 3-12
 - entity
 - associating bitmaps with 6-7
 - attaching Stealth to 4-12
 - attaching to and tracking 4-11
 - attaching to, in Stealth 4-11
 - border and status 5-2
 - changes to in groups 5-6
 - changing group membership 5-5
 - coordinates 3-8
 - deleting groups 5-6
 - deselecting 5-4
 - detaching in Stealth 4-12
 - groups of 5-5
 - icon 3-18
 - icons 6-19
 - information 3-18
 - labels 5-9, 6-10
 - list of 3-16
 - moving from one to another 4-3
 - moving to in Plan View 4-3
 - selecting 5-2
 - selecting in Entity List dialog box 5-3
 - setting value 7-14
 - state information in plug-in 7-13
 - stopping tracking in Stealth 4-12
 - symbol map 6-7

- tracking in Plan View 5-7
- tracking in Stealth 4-11, 5-7
- type 3-18, 3-24
- Entity Guise 3-18, 3-24
- Entity Icon 3-18
- Entity Information dialog box 3-18
- Entity Labels dialog box 3-23
- Entity List 3-7, 3-16
- Entity List dialog box 3-16
 - selecting entities in 5-3
- Entity Type 3-18, 3-24
- entry point
 - plug-in 7-20
- environment variable
 - license manager 2-9
 - MAKLMGRD_LICENSE_FILE 2-9
 - RTI_CONFIG 2-15
- esr() function 7-13
- event
 - plug-in 7-5
 - symbol map 6-8
- example plug-in 7-16
- executables
 - DIS 2-5
 - installation locations 2-4
- execution name, HLA
 - specifying 6-4
- exercise connection 7-5
- exercise ID
 - setting 4-9
 - specifying 6-2, 6-6
 - Stealth 4-9
- Exercise ID menu 4-9
- exerciseConnection() function 7-5
- Exit 3-5
- exiting
 - Plan View Display 3-5

F

- fax
 - MÄK Technologies xii
- features 1-1
- FED file 2-15
- federation execution name
 - setting 4-10
 - specifying 6-4
- file
 - FED 2-15
 - installed 2-4

- Plan View log 6-6
- pvd*.smp 6-7
- pvd.lab 6-7, 6-10
- pvd.set 6-7, 6-11
- pvd.stb 6-7
- pvd.xrc 6-7, 6-13
- Raster Product Format 4-16
- resource 6-7
- RID 2-15
- RTI.RID 2-15
- RTI.rid 6-4
- X resource 6-13
- FLEXlm license manager 2-6
- FLT 4-18
- Force ID 3-18
 - values for 6-7
- format
 - converting database 4-14
- Frame Rate 3-13
- frame rate 3-13
 - setting 6-16
- FrameRate 6-12
- FrameRate configuration setting 6-12
- frequency
 - contour line 6-15
 - grid lines, setting 6-15
- function
 - addCallbackForPlanViewWindow() 7-7
 - addCallbackForTrackedObjects() 7-12
 - addCallbackForUserEvents() 7-10
 - addCallbackOnObject() 7-12
 - addCallbackOnObjectList() 7-12
 - addCircleObject() 7-11
 - addEntityObject() 7-11
 - addInteractionObject() 7-11
 - addItemToMenu() 7-7
 - addLineObject() 7-11
 - addMenuToMenu() 7-8
 - addPointObject() 7-11
 - addPolygonObject() 7-11
 - addPostDrainCallback() 7-5
 - addRightClickMenuItem() 7-8
 - addTextObject() 7-11
 - addTickCallback() 7-6
 - appInterface() 7-4
 - canvasInterfaceList() 7-4
 - centerGeocLocation() 7-13
 - databaseName() 7-8
 - esr() 7-13
 - exerciseConnection() 7-5

-
- getCanvasCoordFromDatabase() 7-9
 - getCanvasCoordFromGeoc() 7-9
 - getCanvasExtentsInDatabaseCoord() 7-7
 - getDatabaseCoordFromCanvas() 7-9
 - getDatabaseCoordFromGeoc() 7-9
 - getGeocCoordFromCanvas() 7-9
 - getGeocCoordFromDatabase() 7-9
 - getSelectedObjectInterfaces() 7-10
 - getSelectedPoint() 7-10
 - getTerrainInformation() 7-8
 - globalId() 7-13
 - interaction() 7-13
 - listOfVertices() 7-13
 - lookup() 7-11
 - objectGeocLocation() 7-13
 - objectId() 7-11
 - objectInterfaceList() 7-4, 7-11
 - objectListInterface() 7-4
 - PvObjectId() 7-13
 - PvPluginAppInterface::addTickCallback() 7-6
 - PvPluginAppInterface::canvasInterfaceList() 7-4
 - PvPluginAppInterface::exerciseConnection() 7-5
 - PvPluginAppInterface::findGlobalId() 7-13
 - PvPluginAppInterface::objectListInterface() 7-4
 - PvPluginAppInterface::removeAllTickCallbacks() 7-6
 - PvPluginAppInterface::removeTickCallback() 7-6
 - PvPluginCanvasInterface::addCallbackOnObjectList() 7-12
 - PvPluginCanvasInterface::appInterface() 7-4
 - PvPluginCanvasInterface::databaseName() 7-8
 - PvPluginCanvasInterface::getSelectedObjectInterfaces() 7-10
 - PvPluginCanvasInterface::getSelectedPoint() 7-10
 - PvPluginCanvasInterface::removeAllCallbacksOnObjectList() 7-12
 - PvPluginCanvasInterface::removeCallbackOnObjectList() 7-12
 - PvPluginEntityObjectInterface::esr() 7-13
 - PvPluginInteractionObjectInterface::interaction() 7-13
 - PvPluginObjectInterface::addCallbackOnObject() 7-12
 - PvPluginObjectInterface::objectId() 7-11
 - PvPluginObjectInterface::objectListInterface() 7-4
 - PvPluginObjectInterface::removeAllCallbacksOnObject() 7-12
 - PvPluginObjectInterface::removeCallbackOnObject() 7-12
 - PvPluginObjectInterface::setTimeoutValue() 7-12
 - PvPluginObjectInterface::timeoutValue() 7-12
 - PvPluginObjectListInterface::appInterface() 7-4
 - PvPluginObjectListInterface::lookup() 7-11
 - PvPluginObjectListInterface::objectInterfaceList() 7-11

- getDatabaseCoordFromGeoc() function 7-9
- getGeocCoordFromCanvas() function 7-9
- getGeocCoordFromDatabase() function 7-9
- getSelectedObjectInterfaces() function 7-10
- getSelectedPoint() function 7-10
- getTerrainInformation() function 7-8
- getting
 - coordinate in plug-in 7-7
 - coordinates of a point 7-10
 - entity state information 7-13
 - geometric information 7-13
 - interaction data 7-13
 - plug-in interfaces 7-4
 - pointers to interfaces 7-4
 - selected objects 7-10
- global geodetic view 4-15
- Global Settings 3-5
- global settings
 - Settings dialog box 3-11
- globalId() function 7-13
- Go To Next Entity 4-3
- Go to Next Entity 3-7
- Go To Previous Entity 4-3
- Go to Previous Entity 3-7
- Grid Line Frequency 3-12
- grid lines
 - frequency of 6-15
- GridLineFrequency configuration setting 6-12
- ground point
 - coordinates 3-8
- group
 - changing members 5-5
 - creating entity 5-5
 - deleting 5-6
 - list of 3-16
 - selecting 5-6
 - status of entities which change 5-6
- Group List 3-16
- guise 3-18, 3-24

H

- header file
 - plugAppInter.h* A-3
 - plugCanvInter.h* A-6
 - plugCircObjIf.h* A-10
 - plugEntObjIf.h* A-12
 - plugIntObjIf.h* A-13
 - plugLineObjIf.h* A-14
 - plugObjInter.h* A-16

- plugObjLInter.h* A-19
- plugPntObjIf.h* A-22
- plugPolyObjIf.h* A-23
- plugTextObjIf.h* A-25
- Heading 3-24
- Heading Indicator 3-24
- heading, pitch, and roll 4-11
- Heading, Pitch, Roll 3-19
- Hide All Windows 3-5, 4-6
- hiding
 - compared to closing 4-7
 - Plan View window 4-6
- HLA
 - federation execution name 4-10
 - federation execution name, specifying 6-4
 - startup options 6-2, 6-4

I

- icon
 - borders 6-19
 - entity 3-18
 - location of 2-5
 - meaning of 6-19
- ID
 - exercise 6-6
 - exercise for Stealth 4-9
 - Force 3-18, 6-7
 - object 3-18, 3-24
 - plan view 3-6
 - site 6-2
 - window 4-6
- information
 - entity 3-18
- information bar
 - Plan View window 3-8
- installing
 - DMSO RTI 2-15
 - MÄK RTI 2-14
 - PVD on a PC 2-3
 - PVD on UNIX 2-4
- interaction 3-12, 5-13
 - data in plug-in 7-13
 - duration 3-13
 - duration of 6-16
 - setting value 7-14
 - symbol map 6-8
- Interaction Duration 3-13
- interaction() function 7-13
- InteractionDuration configuration setting 6-12

- interface 7-3
 - list of 7-4
 - object, looking up 7-11
 - pointers to 7-4
- internationalizing PVD 6-9
- intervisibility 5-11
 - for markers 5-13
 - line colors 6-18
- Intervisibility Fan 3-22
- Intervisibility Fan Fill 3-15
- Intervisibility Lines - Obstructed 3-15
- Intervisibility Lines - Visible 3-15
- Intervisibility Range Circle 3-15, 3-22
- Intervisibility Settings 5-11
- Intervisibility Settings dialog box 3-22
- IntervisibilityCircleColor configuration setting 6-12
- IntervisibilityFanColor configuration setting 6-12
- IntervisibilityObstructedLineColor configuration setting 6-12
- IntervisibilityShownLineColor configuration setting 6-12
- Invert button 3-17
- Invert Selection 3-7

L

- label
 - configuration file 6-10
 - entity 5-9
 - setting 5-10
 - visual versus textual 5-10
- Lat Lon Alt 3-19
- level
 - notification for messages 6-3
- license manager 2-6
 - location of 2-5
- license server
 - running 2-11
 - status 2-11
 - stopping 2-11
- license, environment variable 2-9
- line 7-13
 - setting vertices 7-14
- linear increment 3-14
- list
 - entities and groups 3-16
 - object 7-11
- List of Plan View windows 3-5
- listOfVertices() function 7-13

- lmdown program 2-11
- lmhostid program 2-8
- lmstat program 2-11
- Load Database dialog box 3-9
- Load Layout 3-5
- Load Terrain Database 3-6, 4-5
- loading
 - terrain database 4-5
- location
 - moving Stealth to 4-13
- log 6-2
- log file 6-6
 - messages sent to 6-3
- Logger xi
- looking up
 - object interface 7-11
- lookup() function 7-11

M

- maintenance agreement xii
- MÄK Logger xi
- Mak PVD window 3-2, 3-5
- MÄK RTI xi
 - configuring 2-14
 - installing 2-14
- MÄK Stealth xi
- MÄK Technical Support xii
- MÄK Terrain Format 3-10, 4-14
- MÄK VR-Link Network Toolkit xi
- MAKLMGRD_LICENSE_FILE environment variable 2-9
- manual
 - conventions xiii
- manuals
 - location of 2-5
- map 4-16
- marker 5-13
 - destroying 5-13
- Markings 3-19, 3-24
- menu
 - changing in plug-in 7-7
 - Mak PVD window 3-5
 - plan view window 3-6
- message
 - setting levels 6-3
- messages 6-2
 - viewing in console window 6-3
- metafile
 - terrain 4-16

- minimizing
 - Plan View windows 4-6
- mode
 - single and multi-threaded 6-4
- modes
 - specifying 6-6
- Move to Selection 3-7, 4-3
- Movement Indicator 3-24
- moving
 - entities to in Plan View 4-3
 - Stealth to location 4-13
- moving in window 4-2
- multicast mode
 - specifying 6-6
- multiple Plan Views
 - using 4-6
- multi-threaded mode 6-4

N

- n option 6-3
- name
 - federation execution 4-10
- navigating
 - Plan View window 4-2
- New Global Geodetic View 3-5
- New Plan View 3-5
- New Plan View button 3-4
- No Action 3-21, 4-13
- notification level
 - setting 6-3

O

- object
 - callbacks 7-12
 - controlled by plug-in 7-3
 - creating 7-11
 - geometric
 - information 7-13
 - interface 7-4
 - list 7-11
 - looking up interface 7-11
 - removing in plug-in 7-14
 - state callbacks 7-12
 - supported by plug-in 7-10
 - timeout in plug-in 7-12
 - tracking 7-12
- Object ID 3-18
- object ID 3-18, 3-24

- objectGeocLocation() function 7-13
- objectId() function 7-11
- objectInterfaceList() function 7-4, 7-11
- objectListInterface() function 7-4
- Obstructed Visibility 3-22
- Open Flight database 3-9
- options
 - startup 6-2
- Orientation 3-21
- orientation of Stealth 4-11

P

- P option 6-6
- Pan Down 3-6
- Pan Left 3-6
- Pan Right 3-6
- Pan Step Size 3-14
- Pan Up 3-6
- panning
 - in plug-in 7-7
 - increments 6-15
 - Plan View 4-2
- PanStepSize configuration setting 6-12
- paper map
 - description of 4-16
- paper map bitmap record 4-18
- paper map RPF record 4-19
- Papermap Record 4-18
- Pause Updates 3-12
- PauseUpdates configuration setting 6-12
- PC
 - installation on 2-3
 - starting PVD on 3-2
 - startup options 6-2
- performance
 - hiding compared to closing Plan View windows 4-7
- perspective of Stealth 4-11
- phone number
 - MÄK Technologies xii
- pitch 4-11
- plan view
 - ID 3-6
- Plan View window 3-4, 3-6
 - closing 4-7
 - controlling Stealth from 4-10
 - creating 3-4
 - displaying and hiding 4-6
 - information bar 3-8

- minimizing 4-6
- moving to entity in 4-3
- panning 4-2
- selecting entities in 5-2
- tracking entities in 5-7
- zooming 4-2
- planning
 - user interface for plug-in 7-16
- platforms supported 2-2
- plugAppInter.h* header file A-3
- plugCanvInter.h* header file A-6
- plugCircObjIf.h* header file A-10
- plugEntObjIf.h* header file A-12
- plug-in
 - callbacks 7-5
 - converting coordinates 7-8
 - creating and deleting windows 7-7
 - creating objects 7-11
 - description of API 7-3
 - entity state information 7-13
 - entry point 7-20
 - events 7-5
 - example 7-16
 - extending menus 7-7
 - getting lists of interfaces 7-4
 - getting selected objects and points 7-10
 - interaction data 7-13
 - object callbacks 7-12
 - object list 7-11
 - object timeout 7-12
 - objects manipulable 7-3
 - panning and zooming 7-7
 - pointers to interfaces 7-4
 - removing object 7-14
 - setting object values 7-14
 - thread-safety 7-4
 - ticks 7-6
 - tracking objects 7-12
 - uses for 7-3
- plugins directory 7-15
- plugIntObjIf.h* header file A-13
- plugLineObjIf.h* header file A-14
- plugObjInter.h* header file A-16
- plugObjLInter.h* header file A-19
- plugPntObjIf.h* header file A-22
- plugPolyObjIf.h* header file A-23
- plugTextObjIf.h* header file A-25
- point 7-13
 - centering window on 4-2
 - setting coordinates 7-14
 - zooming to 4-2
- point-to-point mode
 - specifying 6-2, 6-6
- polygon 7-13
 - setting vertices 7-14
- port number
 - specifying at startup 6-6
- Position 3-19, 3-21, 3-24
- position of Stealth 4-11
- products
 - Logger xi
 - RTI xi
 - Stealth xi
 - technical support xii
 - upgrades xii
 - VR-Link xi
- projections
 - supported 1-1
- protocol-independence 7-23
- pvd*.smp file 6-7
- pvd.exe 6-2
- pvd.lab file 6-7, 6-10
- pvd.log 6-2, 6-6
- pvd.set file 6-7, 6-11
- pvd.stb file 6-7
- pvd.xrc file 6-7, 6-13
- PvEntitySymbolMap 6-7
- PvEventSymbolMap 6-8
- PvObjectId 7-11, 7-14
- PvObjectId() function 7-13
- PvPluginAppInterface* class 7-3
- PvPluginAppInterface::addTickCallback()*
 - function 7-6
- PvPluginAppInterface::canvasInterfaceList()*
 - function 7-4
- PvPluginAppInterface::exerciseConnection()*
 - function 7-5
- PvPluginAppInterface::findGlobalId()* function 7-13
- PvPluginAppInterface::objectListInterface()*
 - function 7-4
- PvPluginAppInterface::removeAllTickCallbacks()*
 - function 7-6
- PvPluginAppInterface::removeTickCallback()*
 - function 7-6
- PvPluginCanvasInterface* class 7-3, 7-7
- PvPluginCanvasInterface::addCallbackOnObjectList()* function 7-12
- PvPluginCanvasInterface::appInterface()* function 7-4

PvPluginCanvasInterface::databaseName()
function 7-8
PvPluginCanvasInterface::getSelectedObjectInter-
faces() function 7-10
PvPluginCanvasInterface::getSelectedPoint()
function 7-10
PvPluginCanvasInterface::removeAllCallback-
sOnObjectList() function 7-12
PvPluginCanvasInterface::removeCallbackOnOb-
jectList() function 7-12
PvPluginCircleObjectInterface class 7-11
PvPluginEntityObjectInterface class 7-11
PvPluginEntityObjectInterface::esr() function 7-
13
PvPluginInteractionObjectInterface class 7-11
PvPluginInteractionObjectInterface::interaction()
function 7-13
PvPluginLineObjectInterface class 7-11
PvPluginObjectInterface class 7-3
PvPluginObjectInterface::addCallback-
OnObject() function 7-12
PvPluginObjectInterface::objectId() function 7-11
PvPluginObjectInterface::objectListInterface()
function 7-4
PvPluginObjectInterface::removeAllCallback-
sOnObject() function 7-12
PvPluginObjectInterface::removeCallback-
OnObject() function 7-12
PvPluginObjectInterface::setTimeoutValue()
function 7-12
PvPluginObjectInterface::timeoutValue()
function 7-12
PvPluginObjectListInterface class 7-3
PvPluginObjectListInterface::appInterface()
function 7-4
PvPluginObjectListInterface::lookup() function 7-
11
PvPluginObjectListInterface::objectInter-
faceList() function 7-4, 7-11
PvPluginObjectListInterface::removeObject()
function 7-14
PvPluginPointObjectInterface class 7-11
PvPluginPolygonObjectInterface class 7-11
PvPluginTextObjectInterface class 7-11

R

Raster Product Format file 4-16
record
papermap 4-18

Redraw Map 3-7
registering
callbacks 7-5
removeAllCallbacksForPlanViewWindow()
function 7-7
removeAllCallbacksForTrackedObjects() function
7-12
removeAllCallbacksForUserEvents() function 7-
10
removeAllCallbacksOnObject() function 7-12
removeAllCallbacksOnObjectList() function 7-12
removeAllTickCallbacks() function 7-6
removeCallbackForPlanViewWindow() function
7-7
removeCallbackForTrackedObjects() function 7-
12
removeCallbackForUserEvents() function 7-10
removeCallbackOnObject() function 7-12
removeCallbackOnObjectList() function 7-12
removeObject() function 7-14
removeTickCallback() function 7-6
removing
marker 5-13
Reset 3-12
RID file 2-15
roll 4-11
RPF file. See Raster Product Format file
RTI xi
definition of 2-14
DMSO, installing 2-15
MÄK, installing 2-14
RTI.RID file 2-15
RTI.rid file 6-4
RTI_CONFIG environment variable 2-15
rtiexec 2-16
runLm program 2-11
running
license server 2-11

S

-S option 6-6
Save As Group 3-16
Save Database dialog box 3-10
Save Group 3-16
Save Layout 3-5
Save Terrain Database 3-6, 4-14
saving
databases 3-10
terrain database in MÄK Terrain Format 4-14

-
- Select All button 3-17
 - selecting
 - entities 5-2
 - groups 5-6
 - setCanvasExtentsInDatabaseCoord() function 7-7
 - setCenterGeocLocation() function 7-14
 - setCircumferenceGeocLocation() function 7-14
 - setEsr() function 7-14
 - setObjectGeocLocation() function 7-14
 - setText() function 7-14
 - setTimeoutValue() function 7-12
 - setting
 - changing 6-14
 - configuring 6-11
 - contour line frequency 6-15
 - contribution culling 6-17
 - coordinate in plug-in 7-7
 - duration of interactions 6-16
 - frame rate 6-16
 - grid line frequency 6-15
 - HLA federation execution name 4-10
 - intervisibility colors 6-18
 - labels 5-10
 - notification level 6-3
 - object values in plug-in 7-14
 - panning increments 6-15
 - zoom increments 6-15
 - setting center and radius 7-14
 - Settings 3-6
 - Settings dialog box 3-11
 - Color tab 6-18
 - Control tab 3-14
 - Display tab 3-12
 - setVertices() function 7-14
 - shading
 - terrain 6-16
 - Show All Windows 3-5, 4-6
 - Show Entire Database 3-6
 - showing. See displaying
 - shutting down
 - license server 2-11
 - sight lines, checking 5-11
 - single-threaded mode 6-4
 - site ID
 - specifying 6-2
 - specifying
 - console window 6-3
 - DIS application number 6-2
 - DIS exercise ID 6-2, 6-6
 - DIS site ID 6-2
 - HLA federation execution 6-4
 - point-to-point and multicast modes 6-2, 6-6
 - port number at startup 6-6
 - starting
 - PVD in UNIX 3-3
 - PVD on PC 3-2
 - startup
 - options 6-2
 - state updates
 - sending from the Stealth 4-9
 - status
 - license server 2-11
 - Stealth xi
 - attaching to and tracking entities 4-11
 - attaching to entities 4-11
 - attaching to entity 4-12
 - configuring to work with Plan View Display 4-9
 - controlling from Plan View 4-10
 - controlling from Plan View Display 4-9
 - detaching from entity 4-12
 - linear increment and angular increment 3-14
 - moving to location 4-13
 - stopping tracking 4-12, 5-8
 - tracking entities 4-11, 5-7
 - tracking entities with 5-7
 - view mode 4-12
 - Stealth Angular Increment 3-14
 - Stealth Control 3-7
 - No Action option 4-13
 - Stealth Control dialog box 3-20
 - Stealth Control window 4-9
 - Stealth Linear Increment 3-14
 - Stealth Orientation 3-20
 - Stealth Position 3-20
 - StealthAngularIncrement configuration setting 6-12
 - StealthLinearIncrement configuration setting 6-12
 - stopping
 - license server 2-11
 - tracking in Plan View 5-7
 - tracking in Stealth 5-8
 - string table 6-7
 - strings
 - changing 6-9
 - support, technical xii
 - supported
 - terrain databases 1-2
 - supported features 1-1
 - supported platforms 2-2
 - symbol map

- entity 6-7
- event 6-8
- symbols
 - location of 2-5

T

- T option 6-4
- tdbtool 4-14
- technical support xii
- terminating
 - Plan View Display 3-5
- terrain
 - displaying 3-12
 - shading 6-16
- Terrain Converter Tool 4-14
- terrain database
 - coordinates 7-8
 - description of 4-16
 - Load Database dialog box 3-9
 - loading 4-5
 - Save Database dialog box 3-10
- terrain database record
 - DTED 4-17
 - FLT 4-18
 - GDB 4-18
- terrain metafile 4-16
 - creating 4-17
- Terrain Shading 3-12
- TerrainShading configuration setting 6-12
- text
 - setting string 7-14
- thread safety 7-4
- threshold
 - contribution culling 3-13, 6-17
- tick
 - PVD 7-6
- timeoutValue() function 7-12
- timing out objects 7-12
- Toggle Intervisibility Fan 5-11
- Toggle Intervisibility Range Circle 5-11
- Toggle Obstructed Visibility 5-11
- Toggle Track Selection 3-7
- Toggle Who Can I See? 5-11
- Toggle Who Can See Me? 5-11
- tracking
 - entities 4-11, 5-7
 - entities with Stealth 4-11
 - objects 7-12
 - stopping in Plan View 5-7

- stopping in Stealth 4-12, 5-8
- translating text in interface 6-9
- type
 - vehicle 3-18, 3-24

U

- UNIX
 - installation on 2-4
 - resource file 6-7
 - starting PVD 3-3
 - startup options 6-2
 - X resource file 6-13
- unselecting
 - entities 5-4
- upgrades xii
- user event callback 7-10
- using
 - multiple Plan Views 4-6

V

- values
 - Force ID 6-7
- variable
 - environment 2-9
 - license environment 2-9
- vehicle type 3-18, 3-24
- Velocity 3-19, 3-24
- view
 - geodetic 4-15
 - Stealth 3-20
- view mode
 - absolute 4-13
 - Stealth 4-12
- View Mode drop-down list 3-20
- VR-Link xi, 7-1

W

- Who Can I See? 3-22
- Who Can See Me? 3-22
- window
 - centering on point 4-2
 - closing Plan View 4-7
 - displaying and hiding Plan View 4-6
 - hiding compared to closing 4-7
 - ID 4-6
 - Mak PVD 3-2, 3-5

- minimizing 4-6
- moving in 4-2
- multiple Plan View 4-6
- navigating 4-2
- panning and zooming in plug-ins 7-7
- Plan View 3-6
- plan view 3-4
- plan view ID 3-6
- Plan View information bar 3-8
- Plan View, creating 3-4
- selecting entities in Plan View 5-2
- zooming Plan View 4-2
- wireframe 3-12

X

- x option 6-4
 - DIS 6-6
- X resource file 6-7, 6-13

Z

- zoom
 - setting increments 6-15
- Zoom In 3-7
- Zoom Out 3-7
- Zoom Step Size 3-14
- zooming
 - in plug-in 7-7
 - Plan View 4-2
- ZoomStepSize configuration setting 6-12



MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02138
617.876.8085