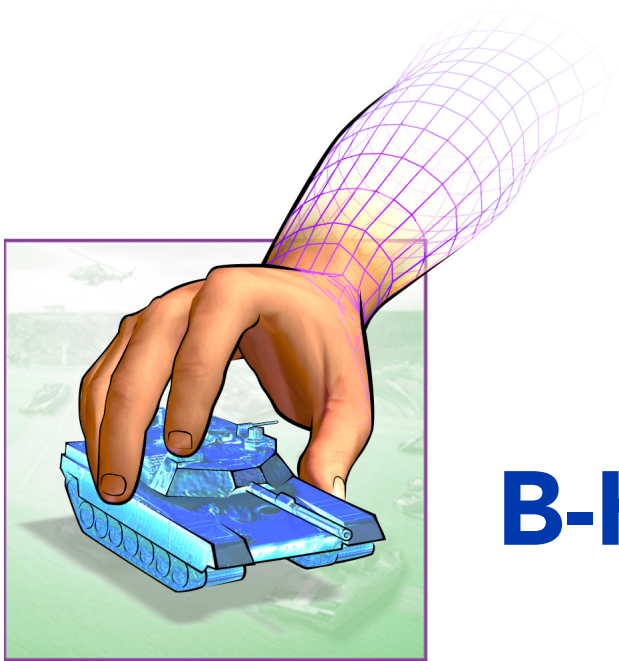
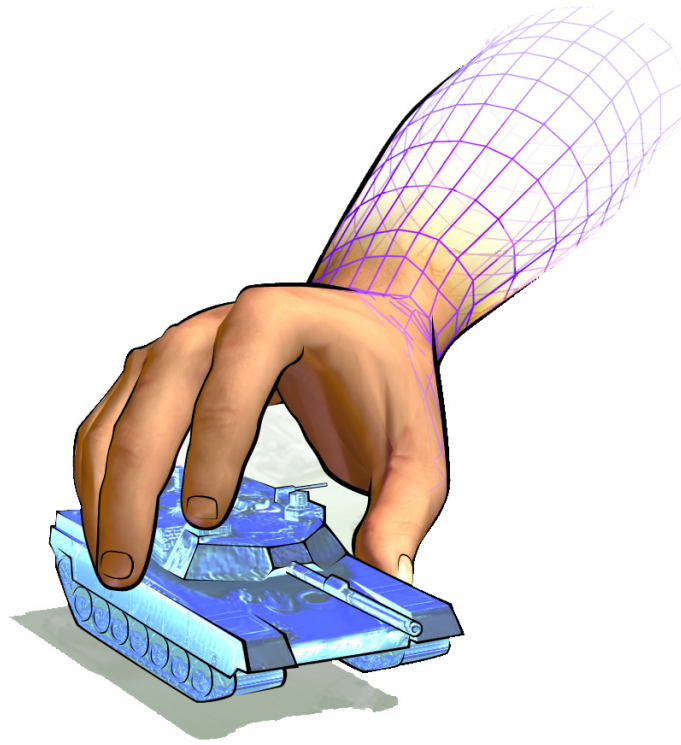




B-HAVE Module for VR-Forces

Users Guide



B-HAVE Module for VR-Forces

Users Guide

Copyright © 2009 VT MÄK
All rights Reserved. Printed in the United States.

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

VR-Exchange™ and VR-Vantage™ are trademarks of VT MÄK.
MÄK Technologies®, VR-Forces®, RTIspy®, B-HAVE®, and VR-Link® are registered trademarks of VT MÄK.

All other trademarks are owned by their respective companies.

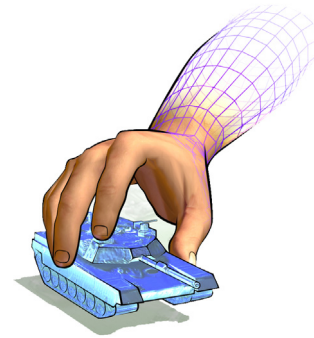
For third-party license information, please see [“Third Party Licenses,”](#) on page xii.

VT MÄK
68 Moulton St.
Cambridge, MA 02138 USA

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

info@mak.com
www.mak.com

Revision BHV-1.4-1-090710



Contents

Preface

How This Manual is Organized	vii
B-HAVE Module Documentation	viii
MÄK Products	viii
How to Contact Us	x
Document Conventions	xi
Hardware and Operating System Naming Conventions	xi
Mouse Button Naming Conventions.....	xii
Third Party Licenses	xii
Boost License.....	xii
libXML and libICONV	xiii
pThreads Library.....	xiii
EHS, PCRE, and PME	xiii
Kynapse License	xiii

Chapter 1 Getting Started

1.1. Introduction	1-2
1.1.1. The B-HAVE Module Architecture	1-2
1.1.2. Using the B-HAVE Module in a Simulation	1-3
1.2. Installing the B-HAVE Module	1-3
1.2.1. Configuring License Management for the B-HAVE Module ..	1-4
1.2.2. Localizing the B-HAVE Module	1-4

Chapter 2 B-HAVE Module Concepts

2.1. Topological Modeling	2-2
2.1.1. The Spatial Graph	2-2
2.1.2. The AI Mesh	2-3
2.1.3. How PathData is Generated	2-4
2.2. Path Finding	2-5

2.2.1. Planning a Path	2-5
2.2.2. Path Following	2-6
2.2.3. Dynamic Avoidance	2-7
2.3. Spatial Reasoning	2-8
2.3.1. Dynamic Topological Analysis	2-9
2.4. B-HAVE Module Tasks	2-10
2.4.1. The Go To Task	2-10
2.4.2. The Wander Task	2-10
2.4.3. The Flee and Hide Tasks	2-10
2.5. Road Traffic	2-11
2.5.1. Terrain Database Requirements to Support Road Traffic	2-11
2.5.2. Creating Background Road Traffic	2-12
2.5.3. Configuring Road Traffic	2-13
2.6. Advanced Planning Capabilities	2-13
2.7. Glossary	2-14

Chapter 3 Generating PathData

3.1. Introduction	3-2
3.1.1. Varying the Complexity of PathData Configurations	3-3
3.1.2. Creating PathData from Road Vector Networks	3-3
3.2. Loading a Terrain Database	3-4
3.3. Managing PathData Resources	3-5
3.3.1. Specifying Bounding Boxes	3-5
3.3.2. Specifying Seed Points	3-7
3.3.3. Specifying Seed Graphs	3-9
3.4. Configuring Generation of PathData	3-11
3.4.1. Creating a PathData Configuration	3-11
3.4.2. Displaying PathData Configurations	3-16
3.4.3. Editing PathData Configurations	3-17
3.5. Generating PathData	3-17
3.5.1. Monitoring the PathData Generation Process	3-19
3.6. Associating PathData with Entity Types	3-20
3.6.1. The BHAVEPathDataRecord	3-21
3.7. Testing PathData	3-22
3.8. Troubleshooting	3-23

Chapter 4 Using the B-HAVE Module in VR-Forces

4.1. Using the B-HAVE Module in a Simulation	4-2
4.1.1. Disabling the B-HAVE Module Plug-ins	4-2
4.2. Creating Multiple Entities	4-3
4.2.1. Adding an Area for Multiple Entity Creation	4-5
4.2.2. Specifying the Entities to Create	4-5
4.3. Sending Text Messages	4-6
4.4. B-HAVE Module Tasks	4-7
4.4.1. The Flee From Task	4-8

4.4.2. The Follow Entity Task	4-8
4.4.3. The Hide From Task	4-9
4.4.4. The Move to Waypoint Task	4-9
4.4.5. The Move to Location Task	4-10
4.4.6. The Move Along Route Task	4-11
4.4.7. The Wander Task	4-11
4.4.8. The Set Movement Mode Set Data Request	4-12
4.4.9. The Set Script Set Data Request	4-13
4.4.10. The View Target Set Data Request	4-13
4.5. Viewing B-HAVE Module Data in VR-Forces	4-14
4.6. Example Scenarios	4-15

Chapter 5 Writing B-HAVE Module Scripts

5.1. Introduction	5-2
5.2. Managing Scripts for VR-Forces	5-2
5.2.1. Specifying a Text Editor	5-3
5.2.2. Creating a New Script	5-3
5.2.3. Editing a Script	5-4
5.2.4. Deleting a Script	5-4
5.3. Writing Scripts	5-5
5.3.1. Using B-HAVE Module Tasks in Scripts	5-5
5.3.2. Considerations for Writing and Using Lua Scripts	5-6
5.3.3. Example Script	5-7
5.3.4. Sending Radio Messages	5-8
5.3.5. Custom Checkpointing	5-9
5.3.6. Extending Lua Scripts Using Modules and Custom Libraries	5-10
5.4. Displaying B-HAVE Module Data in the 3D Front End	5-13
5.4.1. Drawing PathData Around Entities	5-13
5.5. B-HAVE Module Functions	5-14
5.5.1. Entity Functions	5-14
5.5.2. Behavior Functions	5-17
5.5.3. Entity List Functions	5-28
5.5.4. Perception Functions	5-29
5.5.5. Utility Functions	5-31
5.5.6. VR-Forces Task and Set Data Request Functions	5-35

Chapter 6 B-HAVE Messages API

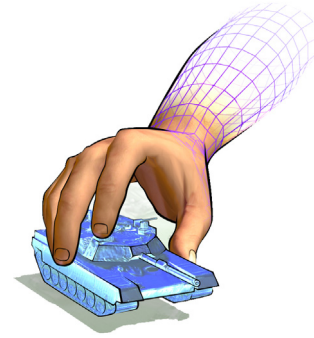
6.1. Introduction	6-2
6.2. Sending B-HAVE Tasks	6-2
6.3. Sending B-HAVE Set Data Requests	6-3
6.3.1. Set Script	6-3
6.3.2. Set Movement Mode	6-3
6.3.3. Set View Target	6-3
6.4. Requesting the Name of an Entity's Lua Script	6-4

6.5. Sending Interface Messages	6-5
6.6. Using the API	6-5
6.7. Building a Project	6-6

Chapter 7 A B-HAVE Module Tutorial

7.1. Introduction	7-3
7.2. Create a Basic Terrain Database and MTD File	7-3
7.2.1. Start the TDB Tool	7-3
7.2.2. Create and Save a New Project	7-4
7.3. Simple PathData Generation	7-5
7.3.1. Set Up PathData Resources	7-5
7.3.2. Configure Generation of PathData	7-10
7.3.3. Generate PathData	7-13
7.3.4. Associate PathData with Entity Types	7-14
7.3.5. Save the MTD File	7-15
7.3.6. View the Spatial Graph for the PathData	7-15
7.4. Simple PathData with Preferred Movement Enabled	7-17
7.4.1. Set Up PathData Resources	7-17
7.4.2. Configure Generation of PathData	7-20
7.4.3. Generate PathData	7-22
7.4.4. Associate PathData with Entity Types	7-22
7.4.5. Save the MTD File	7-23
7.5. Simple PathData with Local Zones	7-23
7.5.1. Set Up PathData Resources	7-23
7.5.2. Configure Generation of PathData	7-23
7.5.3. Generate PathData	7-26
7.5.4. Associate PathData with Entity Types	7-26
7.5.5. Save the MTD File	7-26
7.6. Generating PathData Using a Seed Graph	7-27
7.6.1. Set Up PathData Resources	7-27
7.6.2. Configure Generation of PathData	7-28
7.6.3. Generate PathData	7-29
7.6.4. Associate PathData with Entity Types	7-30
7.6.5. Save the MTD File	7-30
7.6.6. Examine the Spatial Graph	7-31
7.7. Generating PathData Using a Seed Graph and Local Zones	7-31
7.8. Conclusion	7-31

Index



Preface

This manual is written for persons who will install the B-HAVE® Module or use the B-HAVE Module. The manual assumes that you are familiar with your operating system and that you know how to work in your computer's graphical window environment.

Please see release documentation for information about changes and updates to the B-HAVE Module since this manual went to press.

How This Manual is Organized

The chapters are organized as follows:

Chapter 1, *Getting Started* introduces the features of the B-HAVE Module and explains how to install it and set up license management.

Chapter 2, *B-HAVE Module Concepts* explains how the B-HAVE Module is able to improve path planning and collision avoidance and analyze topology.

Chapter 3, *Generating PathData* explains how to prepare terrain databases for use with the B-HAVE Module.

Chapter 4, *Using the B-HAVE Module in VR-Forces* explains how to use the tasks and set data requests that the B-HAVE Module adds to VR-Forces.

Chapter 5, *Writing B-HAVE Module Scripts* describes how to write and manage Lua scripts for use with the B-HAVE Module.

Chapter 6, *B-HAVE Messages API* describes extensions to the VR-Forces remote control API that you can use to manage the B-HAVE Module.

Chapter 7, *A B-HAVE Module Tutorial* shows how to create PathData at various levels of complexity.

B-HAVE Module Documentation

The B-HAVE Module documentation set is as follows:

- ♦ *B-HAVE Module for VR-Forces Users Guide* describes how to install and use the B-HAVE Module program. It is distributed in printed format and as an online manual in Adobe Portable Document Format (PDF).
- ♦ *B-HAVE Module for VR-Forces Release Notes*. Release documentation consists of release notes provided as a PDF file and in printed format.

The PDF files are in the *./doc* directory.

Online Manuals

The online manuals (PDF) use hypertext links for the entries in the table of contents, for all index entries, and for cross references. PDF documentation also supports full text searching.

To view and print PDF documents, you need a copy of the free Acrobat Reader software (version 5.0 or later). To obtain a copy, visit www.adobe.com.

MÄK Products

The B-HAVE Module for VR-Forces is a member of the VT MÄK line of software products designed to streamline the process of developing and using networked simulated environments. The VT MÄK product line includes the following:

- ♦ VR-Link® Network Toolkit. VR-Link is an object-oriented library of C++ functions and definitions that implement the High Level Architecture (HLA), the Distributed Interactive Simulation (DIS) protocol, and the Test and Training Enabling Architecture (TENA). VR-Link has built-in support for the RPR FOM and allows you to map to other FOMs. This library minimizes the time and effort required to build and maintain new HLA or DIS-compliant applications, and to integrate such compliance into existing applications.

VR-Link 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.

- ♦ MÄK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MÄK RTI is optimized for high performance. It has an API, RTIspy®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface (GUI) for the MÄK RTI `rtiexec`.

- ♦ VR-Forces®. VR-Forces is a computer generated forces application and toolkit. It provides an application with a GUI, that gives you a two dimensional view of a terrain database.

You can create and view local entities, aggregate them into hierarchical units, assign tasks, set state parameters, and create plans that have tasks, set statements, and conditional statements. VR-Forces also functions as a plan view display for viewing remote entities taking part in an exercise. Using the toolkit, you can extend the VR-Forces application or create your own application for use with another user interface.

- ♦ VR-Vantage™. VR-Vantage is your 3D visualization solution. It includes two end-user applications, MÄK Stealth and Vantage IG, and the VR-Vantage Toolkit. The Stealth displays a realistic, 3D 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.

Vantage IG is a configurable desktop image generator for out the window (OTW) scenes and remote camera views. It has most of the features of the Stealth, but is optimized for its IG function.

The VR-Vantage Toolkit is a 3D visual application development toolkit. Use it to customize or extend Stealth and Vantage IG or to integrate VR-Vantage capabilities into your custom applications. VR-Vantage is built on top of OpenSceneGraph (OSG). The toolkit includes the OSG version used to build VR-Vantage.

- ♦ MÄK Data Logger. The Data Logger, also called the 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 and can quickly jump to areas of interest. The Logger has a GUI and a text interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application. The Logger editing features let you merge, trim, and offset Logger recordings.
- ♦ MÄK Plan View Display. The Plan View Display (PVD) provides a 2D, “big picture” of a virtual environment. It shows simulated entities, such as vehicles, soldiers, and tanks moving over a 2D-map display.
- ♦ VR-Exchange™. VR-Exchange allows simulations that use incompatible communications protocols to interoperate. For example, within the HLA world, using VR-Exchange, federations using the HLA RPR FOM 1.0 can interoperate with simulations using RPR FOM 2.0, or federations using different RTIs can interoperate. VR-Exchange supports HLA, TENA, and limited DIS translation.
- ♦ MÄK Gateway. The Gateway is a translator application that allows DIS exercises to operate as members of HLA federation executions that use the RPR FOM, thereby preserving the value of simulation code written to support DIS. The MÄK Gateway is a standalone application and does not require you to write additional code for its use.

How to Contact Us

For B-HAVE Module technical support, information about upgrades, and information about other MÄK products, you can contact us in the following ways:

Telephone

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

E-mail

Sales and upgrade information:	info@mak.com
Technical support:	support@mak.com
VR-Vantage support:	vrv-support@mak.com

Internet

MÄK web site home page:	www.mak.com
License key requests:	www.mak.com/support/licenses.php
Product version and platform information:	www.mak.com/support/productversions.php
For the free, unlicensed MÄK RTI:	www.mak.com/products/rti.php
Support FAQ:	www.mak.com/support/faq.php

Post

Send postal correspondence to:	VT MÄK 68 Moulton St. Cambridge, MA, USA 02138
--------------------------------	--

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

Document Conventions

This manual uses the following typographic conventions:

<code>Monospaced</code>	Indicates commands or values you enter.
Monospaced Bold	Indicates a key on the keyboard.
<i>Monospaced Italic</i>	Indicates command variables that you replace with appropriate values.
{ }	Indicates required arguments.
[]	Indicates optional arguments.
	Separates options in a command where only one option may be chosen at a time.
()	In command syntax, indicates equivalent alternatives for a command-line option, for example, (-h --help).
/	Indicates a directory. Since MÄK products run on both UNIX and Windows 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.
<i>Italic</i>	Indicates a file name, pathname, or a class name.
sans Serif	Indicates a parameter or argument.
➤	Indicates a one-step procedure.
Menu → Option	Indicates a menu choice. For example, an instruction to select the Save option from the File menu appears as: Choose File → Save .
i	Indicates supplemental or clarifying information.
!	Indicates additional information that you must observe to ensure the success of a procedure or other task.

Directory names are preceded with dot and slash characters that show their position with respect to the B-HAVE Module home directory. For example, the directory *vrforces/doc* appears in the text as *./doc*.

Hardware and Operating System Naming Conventions

Unless otherwise specified, references to UNIX include Linux, regardless of the type of computer it is running on. References to PCs refer to Intel processor-compatible computers running a version of Microsoft Windows.

Mouse Button Naming Conventions

An instruction to click the mouse button, refers to clicking the primary mouse button, usually the left button for right-handed mice and the right button for left-handed mice. The popup menu refers to the menu displayed when you click the secondary mouse button, usually the right button on right-handed mice and the left button on left-handed mice.

Third Party Licenses

MÄK software products may use code from third parties. This section contains the license documentation required by these third parties.

Boost License

VR-Link, and all MÄK software that uses VR-Link uses some code which is distributed under the Boost License. All header files that contain Boost code are properly attributed. The boost web site is: www.boost.org.

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the “Software”) to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libXML and libICONV

VR-Link and all MÄK software that uses VR-Link, links in libXML and libICONV. On some platforms the compiled libraries and header files are distributed with MÄK Products. MÄK has made no modifications to these libraries. For more information about these libraries please see the following web sites:

- ♦ The LGPL license is available at: <http://www.gnu.org/licenses/lgpl.html>
- ♦ Information about IconV is at: <http://www.gnu.org/software/libiconv/>
- ♦ Information about LibXML is at: <http://xmlsoft.org/>

pThreads Library

VR-Exchange links with the pThreads win32 library. The library is distributed under the GNU Lesser General Public License (LGPL). MÄK has made no modification to this library. For information about the pThreads win32 library please see: <http://sourceware.org/pthreads-win32/>

EHS, PCRE, and PME

The MÄK RTI links with the EHS, PCRE, and PME libraries. These libraries are distributed under the GNU Lesser General Public License (LGPL). MÄK has made modification to these libraries only to allow them to compile on the supported platforms. For more information about these libraries please see the following web sites:

- ♦ EHS: <http://xaxxon.slackworks.com/ehs/>
- ♦ PCRE: <http://www.pcre.org/>
- ♦ PME: <http://xaxxon.slackworks.com/pme/index.html>

Kynapse License

(c) 2009 Autodesk, Inc. All rights reserved. Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

The following are registered trademarks or trademarks of Autodesk, Inc., in the USA and other countries: 3DEC (design/logo), 3December, 3December.com, 3ds Max, ADI, Alias, Alias (swirl design/logo), AliasStudio, Alias/Wavefront (design/logo), ATC, AUGI, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk Envision, Autodesk Insight, Autodesk Intent, Autodesk Inventor, Autodesk Map, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSnap, AutoSketch, AutoTrack, Backdraft, Built with ObjectARX (logo), Burn, Buzzsaw, CAiCE, Can You Imagine, Character Studio, Cinestream, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Create>what's>Next> (design/logo), Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, DesignServer, DesignStudio, Design/Studio (design/logo), Design Web Format, Discreet, DWF, DWG, DWG (logo), DWG Extreme, DWG TrueCovert, DWG TrueView, DXF, Ecotect, Exposure, Extending the Design Team, Face Robot, FBX, Filmbox, Fire, Flame, Flint, FMDesktop, Freewheel, Frost, GDX Driver, Gmax, Green Building Studio, Heads-up Design, Heidi, HumanIK, IDEA Server, i-drop, ImageModeler, iMOUT, Incinerator, Inferno, Inventor, Inventor LT, Kaydara, Kaydara (design/logo), Kynapse, Kynogon, LandXplorer, LocationLogic, Lustre, Matchmover, Maya, Mechanical Desktop, Moonbox, MotionBuilder, Movimento, Mudbox, NavisWorks, ObjectARX, ObjectDBX, Open Reality, Opticore, Opticore Opus, PolarSnap, PortfolioWall, Powered with Autodesk Technology, Productstream, ProjectPoint, ProMaterials, RasterDWG, Reactor, FealDWG, Real-time Roto, REALVIZ, Recognize, Render Queue, Retimer, Reveal, Revit, Showcase, ShowMotion, SketchBook, Smoke, Softimage, Softimage/XSI (design/logo), SteeringWheels, Stitcher, Stone, StudioTools, Topobase, Toxik, TrustedDWG, ViewCube, Visual, Visual Construction, Visual Drainage, Visual Landscape, Visual Survey, Visual Toolbox, Visual LISP, Voice Reality, Volo, Vtour, Wire, Wiretap, WiretapCentral, XSI, and XSI (design/logo).

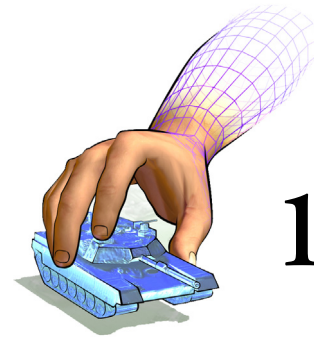
The following are registered trademarks or trademarks of Autodesk Canada Co. in the USA and/or Canada and other countries: Backburner, Multi-Master Editing, River, and Sparks.

The following are registered trademarks or trademarks of Moldflow Corp. in the USA and/or other countries: Moldflow MPA, MPA (design/logo), Moldflow Plastics Advisers, MPI, MPI (design/logo), Moldflow Plastics Insight, MPX, MPX (design/logo), Moldflow Plastics Xpert.

All other brand names, product names or trademarks belong to their respective holders.

Disclaimer

THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. "AS IS". AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.



Getting Started

This chapter provides a brief introduction to the B-HAVE Module and explains how to install it.

Introduction	1-2
The B-HAVE Module Architecture	1-2
Using the B-HAVE Module in a Simulation.....	1-3
Installing the B-HAVE Module.....	1-3
Configuring License Management for the B-HAVE Module.....	1-4
Localizing the B-HAVE Module	1-4

1.1. Introduction

The B-HAVE Module for VR-Forces¹ is a VR-Forces plug-in that extends the movement models for lifeforms and ground platforms. It improves:

- 3D path finding
- Simplicity of use of basic agents (GoTo)
- Complexity and realism of behaviors (Hide, Flee, Wander)
- Complexity and realism of scenarios.

The B-HAVE Module is powered by Kynapse®, the world's leading artificial intelligence (AI) technology solution, from Autodesk®. It is based on Kynapse's fundamental concepts of Topological Modeling for Path Finding, and Spatial Reasoning for Advanced Behaviors, which are integrated into VR-Forces as advanced planning capabilities.

The B-HAVE Module offers VR-Forces users the ability to:

- Make entities move autonomously without specifying routes
- Make entities navigate through complex environments, indoors and outdoors, while avoiding collisions with obstacles and other entities
- Quickly populate a 3D environment with dozens of entities wandering in buildings, on roads, and on pavement
- Develop complex scenarios in which entities hide from each other and flee from threats.

1.1.1. The B-HAVE Module Architecture

The B-HAVE Module is implemented as a set of plug-ins for VR-Forces and the MÄK Terrain Database Tool (TDB Tool), as follows:

- The TDB Tool plug-in allows you to generate PathData to associate with terrain databases. Entities use the PathData to understand the environment and plot a course.
- The VR-Forces front-end plug-in allows you to use the B-HAVE Module's tasks and run scripts through the GUI.
- The VR-Forces back-end plug-in performs all the AI-related runtime computations.

1. B-HAVE stands for Brains for Human Activity in Virtual Environments.

1.1.2. Using the B-HAVE Module in a Simulation

Before you can use the B-HAVE Module in a simulation, you must generate PathData for a terrain database and associate it with the terrain in its MTD file. Then, you create a scenario using the updated terrain.



The B-HAVE Module includes terrain databases for which PathData has already been generated (Makland, VR-Village, and TerraSim_SampleUrban).

You create a scenario the same way that you would any VR-Forces scenario, except that lifeforms and ground platforms can now use B-HAVE Module tasks for movement in addition to or instead of the standard VR-Forces tasks. You can enhance the scenario by writing and using scripts (using the Lua language) that can implement additional advanced planning functions. Scripts can tell an entity what to perceive and what to do based on that perception. You can use them to fully control an entity.

1.2. Installing the B-HAVE Module

The B-HAVE Module is provided as an executable installation program. Please note the following important installation criteria:

- ♦ You must install the B-HAVE Module into the directory in which you have installed VR-Forces. The installer defaults to the default installation directory for the associated version of VR-Forces, for example, *C:\MAK\VR-Forces3.11*. If you did not install VR-Forces into the default directory, it is your responsibility to change the B-HAVE Module installation directory to your VR-Forces installation directory.

If the installer warns you that you are installing into an existing directory, ignore the warning.

- ♦ Each version of the B-HAVE Module is built for a specific version of VR-Forces and will work only with that version. For example, B-HAVE Module 1.2 for VR-Forces 3.11 will work only with VR-Forces 3.11. It would not work with VR-Forces 3.10.1.
- To install the B-HAVE Module, run the installation executable and follow the directions of the installation wizard.

Table 1-1 lists the files and directories installed in the VR-Forces directory by the B-HAVE Module.

Table 1-1: Files and directories installed into VR-Forces by the B-HAVE Module

File/Directory	Description
<i>./data/parameters/BHAVE</i>	Sample Lua scripts, PathData generation templates, and icons.
<i>./data/scenarios/B-HAVE_Demos</i>	Sample scenarios that use B-HAVE Module tasks.
<i>./data/terrain/</i>	MTD files and PathData for terrain databases.
<i>./flexlm.x/KYNOGON.exe</i>	Executable for the B-HAVE Module license management.
<i>./binx</i>	B-HAVE Module DLLs are installed in the various <i>bin</i> directories.
<i>./plugins/B-HAVE</i>	Plug-in DLLs for the B-HAVE Module.

1.2.1. Configuring License Management for the B-HAVE Module

The B-HAVE Module requires a FLEXlm license.

To set up license management for the B-HAVE Module:

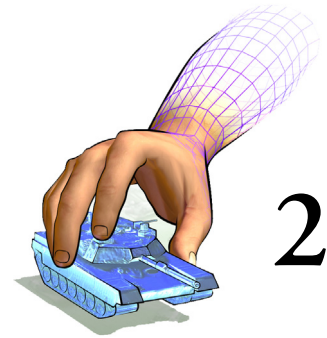
1. The B-HAVE Module installation process installs a file called *Kynogon.exe* in the *vrforcesx.x/flexlm.x* directory. Copy this file to the directory from which you are running the license server, for example, *C:\flexlm*.
2. On each computer on which you will run VR-Forces with the B-HAVE Module, set the KYNOGON_LICENSE_FILE environment variable to *@computer_name*, where *computer_name* is the name of the license server computer. Instructions for setting environment variables are in Chapter 2 of *VR-Forces Users Guide*.
3. Request a license using the license request procedure described in Chapter 2 of *VR-Forces Users Guide*. When you receive your license, give it a unique name, such as *bhave.lic* and place it in the directory from which you are running the license server.



If you run FLEXlm as a service, you must configure the path to the FLEXlm directory without a trailing slash, for example, *C:\flexlm*, not *C:\flexlm*. If you include the final slash, the Kynogon vendor daemon will not start.

1.2.2. Localizing the B-HAVE Module

You can localize the B-HAVE Module menus using Qt Linguist, as described in Section 2.5, “[Localizing the VR-Forces Interface](#)”, in *VR-Forces Users Guide*. An untranslated file for B-HAVE is included in the installation.



B-HAVE Module Concepts

This chapter provides conceptual background for the B-HAVE Module. Understanding these concepts will help you to use the MÄK Terrain Database Tool (TDB Tool) to prepare terrains for use with the B-HAVE Module. Concepts will also help you understand the behaviors to be expected from entities executing B-HAVE Module tasks.

Topological Modeling	2-2
The Spatial Graph	2-2
The AI Mesh	2-3
How PathData is Generated	2-4
Path Finding	2-5
Planning a Path	2-5
Path Following	2-6
Dynamic Avoidance	2-7
Spatial Reasoning	2-8
Dynamic Topological Analysis	2-9
B-HAVE Module Tasks	2-10
The Go To Task	2-10
The Wander Task	2-10
The Flee and Hide Tasks	2-10
Road Traffic	2-11
Terrain Database Requirements to Support Road Traffic	2-11
Creating Background Road Traffic	2-12
Configuring Road Traffic	2-13
Advanced Planning Capabilities	2-13
Glossary	2-14

2.1. Topological Modeling

In VR-Forces, when an entity is given a movement task, such as Move to Location or Move to Waypoint, it ordinarily moves directly towards the destination, taking relatively little account of topography, vector data, or other entities, except for basic collision avoidance. If path planning is enabled, the entity moves along roads if they are present. If you want an entity to move along an optimal route, you must draw the route and task the entity to follow it.

When entities carry out B-HAVE Module tasks, they use previously computed topological models of the virtual 3D environment to compute, in real time, an optimized path to the destination (path finding). As they follow the path, they automatically avoid obstacles and other entities. The topological models, collectively called PathData, are implemented as a spatial graph and an AI mesh.

2.1.1. The Spatial Graph

A spatial graph is a 3D collection of vertices linked by edges. This graph covers the entire accessible world for a given entity.

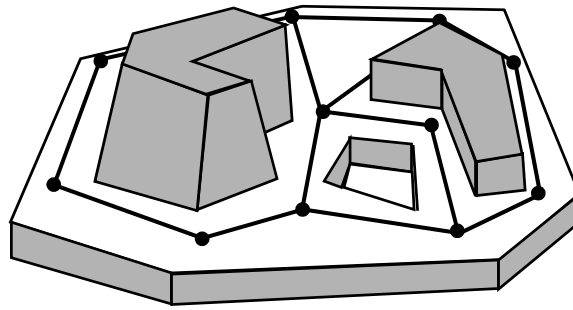


Figure 2-1. Spatial graph

Each type of entity uses its own spatial graph. A “type” of entity is defined roughly by its size (for example, lifeforms, similarly sized ground platforms, and so on) and its motion characteristics (its ability to go up or down a slope, climb objects such as steps, or jump).



Edges and vertices do not represent routes to be followed by entities. Entities can actually move anywhere.

2.1.2. The AI Mesh

An AI mesh is a 3D representation of the static physical collision world for the B-HAVE Module. It represents the physical boundaries of the accessible world in a light format.

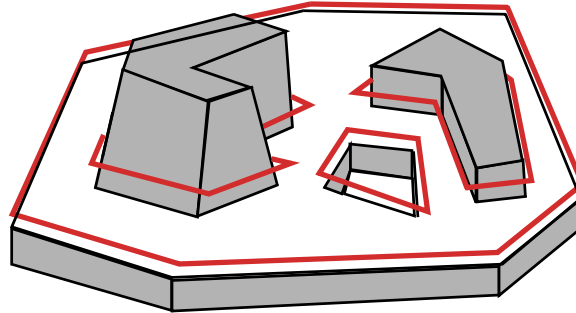


Figure 2-2. AI mesh

When an entity moves, it frequently tests the physical world. The B-HAVE Module performs its tests against the AI mesh, using optimal collision functions. Using an AI mesh significantly improves the performance of path following compared to standard AI path following algorithms.

2.1.3. How PathData is Generated

PathData is generated by the TDB Tool when you prepare a terrain database for use with the B-HAVE Module. PathData must be generated separately for each type of entity that you want to use with the B-HAVE Module. If the terrain is modified, the PathData must be generated again.

Path data is generated starting from a seed point. A seed point is the first accessible point, which you specify in the TDB Tool when you are generating PathData. From this point, a simple volume with the characteristics of the selected type of entity is moved all over the world. An ultra dense graph is generated during this exploration phase by sampling the world, at a specified interval, to see if movement is possible for this type of entity from the previous point, given its motion characteristics. This graph is a collection of accessible points.

The AI mesh is created based on a topological analysis of the ultra dense graph. Then the points are interconnected (propagation phase), generating a dense, connected spatial graph. To improve performance at runtime, this graph is simplified, becoming the spatial graph.

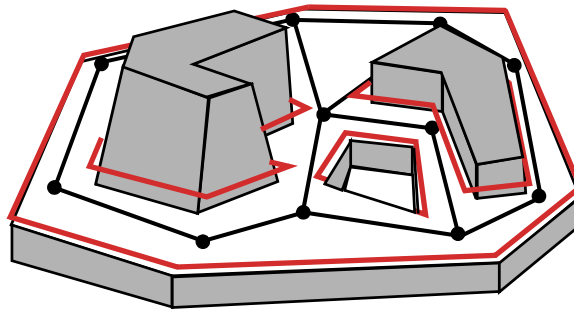


Figure 2-3. PathData (spatial graph and AI mesh)

The spatial graph and the AI mesh are saved to disk and referenced in the MTD file for the terrain database.



- ♦ One AI mesh is generated for each spatial graph. However, for performance purposes, different types of entities can share the same AI mesh at runtime.
 - ♦ The PathData generation can differentiate specific terrain information like roads, and store this information in the spatial graph.
-

For details about generating path data, please see Chapter 3, [Generating PathData](#).

PathData Templates

When the B-HAVE Module generates PathData, it uses a level of detail and density that you specify when you set up a PathData configuration. You specify detail and density by choosing among a set of templates provided with the B-HAVE Module.

The amount of detail in the PathData determines how precisely an entity can explore its environment. For example, at a higher level of detail, an entity could detect a doorway and walk through the door, or detect a lamp post and walk around it. At a lower level of detail, the PathData might not include information about the lamp post or doorway.

The density of the spatial graph determines the maximum length of edges. The denser the graph, the shorter the edges. Shorter edges result in smoother, more precise entity movement.

2.2. Path Finding

Path finding is the process by which an entity plans a path, follows the path, and avoids obstacles along the path.

2.2.1. Planning a Path

Every time an entity decides to move (either through an autonomous decision or a task assignment), it computes its path. Path computation, or path planning consists of running an A* algorithm, with a given heuristic, on the spatial graph.

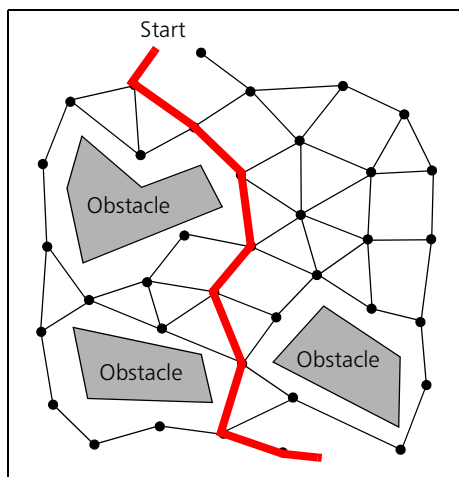


Figure 2-4. Planning a path through the spatial graph

A path finding heuristic is a cost estimate for the shortest path to reach a target. This cost can be measured according to various criteria, such as distance, time, and danger. The path finding heuristic is critical for the performance of the A* algorithm used by the B-HAVE Module.

The standard path finding heuristic is the Euclidian Heuristic. In this heuristic, the cost of moving from one vertex to another vertex is the Euclidian length of the connecting edge. The B-HAVE Module also uses a Road Constrained Heuristic, in which a ground platform computing its path is forced to move only on roads.

2.2.2. Path Following

Once a path is determined for an entity, it can start moving to its final goal. As illustrated in [Figure 2-4](#), a path is a set of vertices and edges. However, entities do not necessarily follow the exact path. The B-HAVE Module smooths the entity's movement.

The first intermediary destination (or subgoal) of the entity is the nearest vertex. Before moving towards this point, the entity checks to see if it can move directly to the next vertex in its path. This test is performed against the AI mesh. If the next vertex is reachable, it becomes the new subgoal. The entity continues testing successive vertices until it finds a vertex that it cannot reach directly. The final trajectory is a smooth line that connects the various subgoals.

Figures 2-5 and 2-6 illustrate this process. Using the prospective path defined in [Figure 2-4](#), the entity computes its actual path. In [Figure 2-5](#), view 1, it tests the first three vertices (dashed lines) and finds that it can reach all of them directly. In [Figure 2-5](#), view 2, it tests the fourth vertex and finds that it can reach it. It then tests the fifth vertex and finds it cannot reach it directly. Therefore, the fourth vertex becomes the subgoal for this portion of the path. [Figure 2-6](#) shows the final calculated path (dashed line). Notice how the path to the fourth vertex is much smoother than a path that strictly follows the vertices and edges would be, because the entity determined that it could move directly to that vertex.

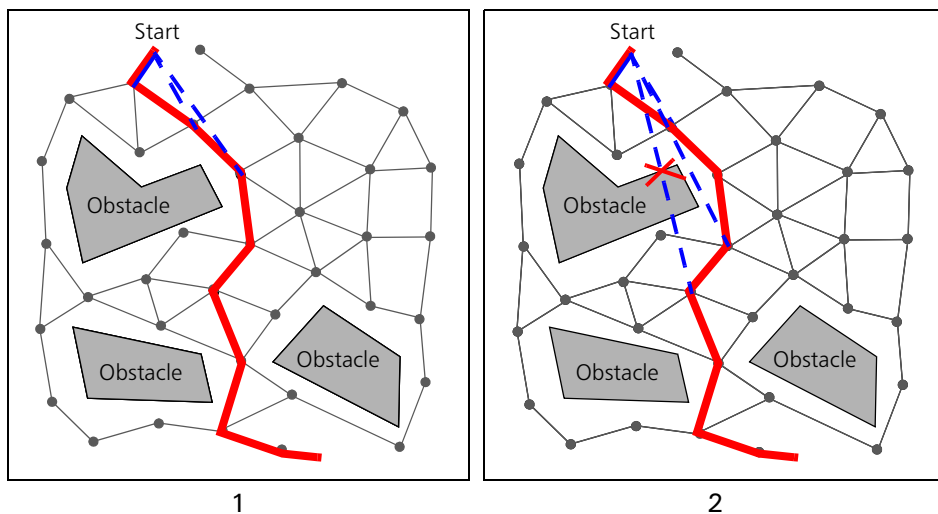


Figure 2-5. Checking next vertex

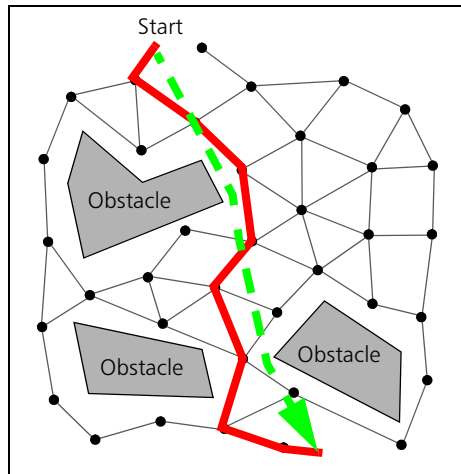


Figure 2-6. Final calculated path

2.2.3. Dynamic Avoidance

While following its path, an entity may encounter other entities. VR-Forces models include collision avoidance. However in complex scenarios, the scenario developer may have to spend a considerable amount of time creating routes and coordinating entities to minimize collisions among entities and obstacles.

The B-HAVE Module uses a dynamic avoidance algorithm to make entities behave realistically when they are about to collide. Entities detect potential collisions in advance and adjust their trajectories smoothly using the spatial graph and the AI mesh.

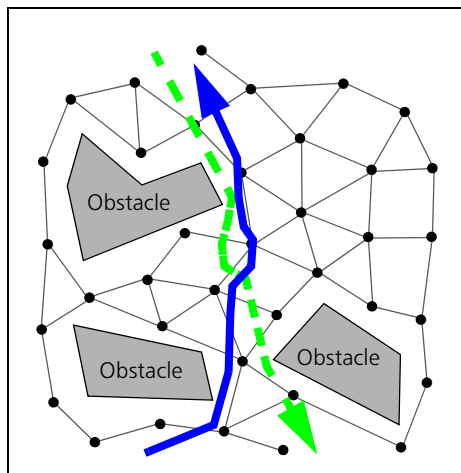


Figure 2-7. Collision avoidance

In [Figure 2-7](#), two entities traveling in opposite directions avoid each other, then resume their routes. Compare the track of the dashed line to that in [Figure 2-6](#).

2.3. Spatial Reasoning

Advanced entity behaviors require a certain level of understanding of the topology from each entity's point of view. This understanding, called spatial reasoning, allows an entity to decide where to go, to identify a hiding position, or to identify a threat point.

The B-HAVE Module can provide an analysis of the PathData in the vicinity of any point in the environment to find a collection of vertices or edges in the spatial graph with properties that are relevant for decision purposes, for example:

- ♦ The nearest vertices that are accessible by the entity and are out of sight of point A. These are the nearest positions where the entity can hide from someone standing on A. These are also the nearest destinations if an entity wants to flee from point A.
- ♦ Edges connecting one point out of sight of the entity and another point within sight of the entity (but not necessarily accessible). These are edges that a sniper could move to, shoot at the entity, and then hide from it.

This dynamic topological analysis (DTA) can be used at any moment by any behavior, and may require frequent updates (every time that a movement of an entity changes its perception point of view significantly).

2.3.1. Dynamic Topological Analysis

DTA in the B-HAVE Module runs a search-for-the-best-candidate algorithm on the existing path data. DTA is optimized for low CPU cost and memory footprint. DTA starts from any given position and propagates to the nearest vertices following the connecting edges, testing edges or vertices for specific properties.

In [Figure 2-8](#), a bot (any entity controlled by the B-HAVE Module) wants to hide from the Player (view 1). The analysis explores points in the vicinity of the bot, testing to see if they are in line-of-sight of the player (views 2 and 3), until it finds two hiding position candidates (view 4).

These tests are performed over several frames, to minimize their effect on performance.

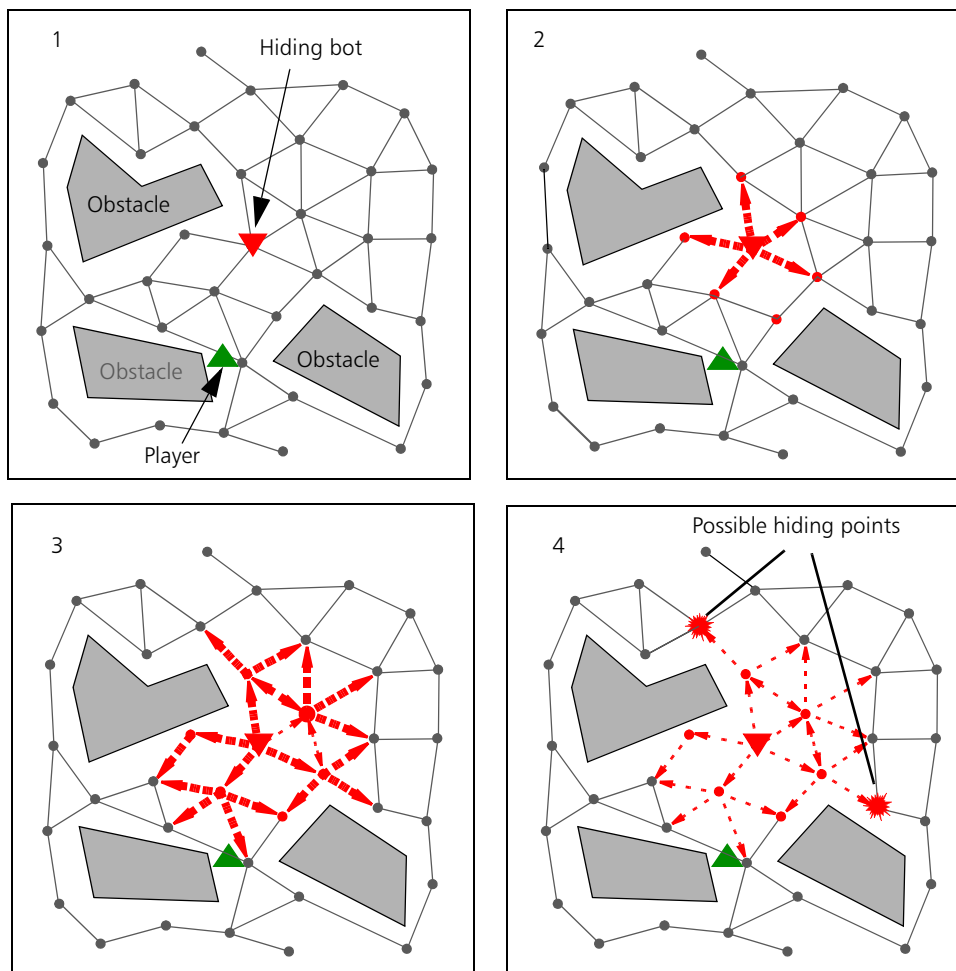


Figure 2-8. Dynamic topological analysis

2.4. B-HAVE Module Tasks

This section describes the models used for the following B-HAVE Module tasks:

- ♦ Go To
- ♦ Wander
- ♦ Flee and Hide.

All B-HAVE Module behaviors use a perception - decision - action model, as follows:

- ♦ Perception: The B-HAVE Module determines the entity state, external simulation data, such as enemies detected and sensor data, and topological information.
- ♦ Decision: The B-HAVE Module determines the appropriate behavior for the entity and translates that into tasks.
- ♦ Action: The B-HAVE Module supplies an intended speed and direction for the entity.

2.4.1. The Go To Task

For the B-HAVE Module Go To task, you supply a destination and speed. The entity computes its path (path finding), and follows the path (path following) to its destination. It avoids entities (dynamic avoidance).

A lifeform can walk on pavement, go inside buildings, and go up or down a flight of stairs or a ladder to reach its goal. A car can follow roads to its destination. A tank computes an optimized path, using roads when possible, and going off-road otherwise, to optimize time.

2.4.2. The Wander Task

An entity given the B-HAVE Module Wander task identifies destinations in its vicinity, chooses one randomly and moves towards it. When it reaches its destination, the entity finds a new destination point and repeats the process.

Dozens of lifeform entities can be tasked to wander, providing background human traffic in a simulation. As with any B-HAVE Module task, path finding takes place in the 3D topology, so entities can wander indoors or outdoors.

2.4.3. The Flee and Hide Tasks

The B-HAVE Module offers two more complex behaviors based on spatial reasoning:

- ♦ Flee: this behavior allows an entity to flee a set of danger points or hostile entities and stop only when hidden from all of them. The flight behavior is reactive; the Flee task regularly recalculates a safe destination point.
- ♦ Hide: this task allows an entity to hide from a group of dangerous entities.

2.5. Road Traffic

The B-HAVE Module enables VR-Forces ground entities to simulate road traffic with greater realism than the basic VR-Forces. Ground entities can move along roads while keeping in lanes, slowing and stopping for vehicles in front of them, giving way to pedestrians and other objects in the road, and giving way at intersections.

This capability enhances the dynamic avoidance algorithms of vehicles that know about roads. Vehicles are only affected by these road traffic algorithms while they are in Prefer Roads movement mode, and when they are moving along a road. The road traffic behaviors include the following changes to entity behavior:

- ♦ If an obstacle, such as another entity, is encountered ahead of the entity, it slows and stops, waiting for that entity to get out of the way, instead of driving around it. This includes both vehicle and pedestrian obstacles.
- ♦ At intersections, entities give right-of-way to the “driving side” (to the right if driving on the right side), or to the entity which has been waiting the longest at the intersection.
- ♦ One vehicle at a time crosses an intersection, except for “short turns” (turns that do not cross any road).
- ♦ Pedestrians avoid vehicles only when in front of them. They cross roads even if vehicles are approaching. That is why it is the vehicle’s responsibility to give way to pedestrians.
- ♦ Multiple road levels are supported. The only requirement is that in the road vector network in the GDB, at least one point of each DtNetworkEdge (a sequence of road segments with no intersection) has only one altitude candidate.

2.5.1. Terrain Database Requirements to Support Road Traffic

In order to use the Road Traffic capabilities of B-HAVE Module, the terrain database and the configuration must meet the following requirements:

- ♦ The terrain database must have road feature data. This can be from a variety of sources, such as DFD or ESRI Shape files, or created manually in the TDB Tool. The road feature data must mark the center of the roads, have a valid width attribute, and be marked with the MAK Code of MAK010.
- ♦ The PathData must be generated on this terrain using the Prefer Roads flag.
- ♦ The vehicles that are using the roads must have their movement mode set to Prefer Roads. (Most ground vehicles have this set as the default.)

2.5.2. Creating Background Road Traffic

You can use the road traffic capability, along with other B-HAVE Module and VR-Forces features, to easily create background road traffic in a scenario. Assuming the PathData has been set up as described in [“Terrain Database Requirements to Support Road Traffic,”](#) on page 2-11, follow these steps to create some background traffic in an area of the terrain.

1. Create an Area object around the area where the traffic is desired.
2. Open the Create Multiple Entities window using the B-HAVE menu.
3. Select the area.
4. Click Add. The Add Group of Entities dialog box opens.
5. Enter the number of desired vehicles to make up the traffic.
6. Specify Civilian Vehicle as the Type Of Entity (Ground - ForceNeutral).
7. Specify Wander as the Starting Action.
8. In the Wander dialog box, select Restricted To Area, and choose the same area.
9. In the Add Group of Entities dialog box, specify Prefer Roads as the Positioning method.
10. Click OK.
11. Click Create.

The entities move randomly along the roads in the designated area, stopping at intersections and waiting behind each other. They also react to any other entities that are part of the simulation, both VR-Forces entities and remote entities simulated by another federate.

2.5.3. Configuring Road Traffic

You can configure vehicles to drive on the left side of the road, instead of the right side (the default). To enable left-side driving, you must add an attribute to the PathData configuration before you generate PathData. You can set this attribute in the GUI or a configuration file.

To specify which side of the road to drive on using the GUI:

1. In the TDB Tool, select **B-HAVE** → **Configure PathData Generation**, or click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens ([Figure 3-7](#)).
2. Select the configuration that you want to edit.
3. Click Edit. The Edit PathData Configuration dialog box opens ([Figure 3-8](#)).
4. Select the Driving On Left Side check box.
5. Click OK.

To specify which side of the road to drive by editing a configuration file:

1. Open the XML file associated with the terrain
(`./data/terrain/<terrain>_PathData/<terrain>_BHave.xml`).
2. Locate the *PDGConfig* element for the configuration you want to edit.
3. Add a **drivingSide** attribute to the element next to the `generateRoadData` attribute, for example:

```
<PDGConfig name="Vehicle" generated="true" generateRoadData="true"
drivingSide="left" generateIsectRoadData="false">
```

2.6. Advanced Planning Capabilities

B-HAVE Module tasks provide a level of advanced behaviors that are accessible through the VR-Forces Task menu and in the Plan Editor and do not require any programming. Expert users can also create complex scenarios by scripting in Lua. Lua is a scripting language used to express the decision making process for an entity. For instance, a Lua script can tell an entity what to perceive and what to do based on that perception. For details, please see Chapter 5, [Writing B-HAVE Module Scripts](#).

2.7. Glossary

This section defines terms used by the B-HAVE Module.

AI mesh

An AI Mesh is a specific representation of the static physical collision world for B-HAVE. It represents the physical boundaries of the accessible world in a light format.

Vertex (vertices)

A vertex is a node of a graph. It is a 3D point expressed with its coordinates. In the PathData, it is a point that a given entity has been able to access during the exploration phase.

Edge

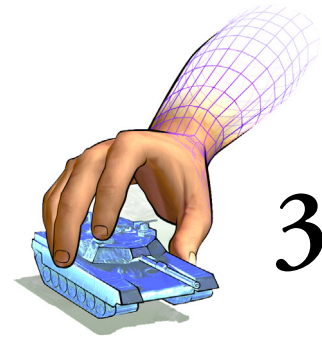
An edge is a connection between two vertices.

Seed point

A graph generation senses the physical world, to check all accessible points for a given type of entity. This process needs a first starting point that can not be guessed. This point is a seed point that must be provided by the user.

Spatial graph

A spatial graph is a collection of vertices linked by edges. This graph covers the entire accessible world for a given entity.



Generating PathData

This chapter explains how to use the MÄK Terrain Database Tool (TDB Tool) to generate PathData for use with the B-HAVE Module. It assumes that you are familiar with the TDB Tool, project files, and MTD files. For complete information about using the TDB Tool, please see *MÄK Terrain Database Tool Users Guide*.

Introduction	3-2
Varying the Complexity of PathData Configurations.....	3-3
Creating PathData from Road Vector Networks	3-3
Loading a Terrain Database.....	3-4
Managing PathData Resources.....	3-5
Specifying Bounding Boxes	3-5
Specifying Seed Points	3-7
Specifying Seed Graphs	3-9
Configuring Generation of PathData	3-11
Creating a PathData Configuration	3-11
Displaying PathData Configurations	3-16
Editing PathData Configurations	3-17
Generating PathData	3-17
Monitoring the PathData Generation Process.....	3-19
Associating PathData with Entity Types.....	3-20
The BHAVEPathDataRecord	3-21
Testing PathData.....	3-22
Troubleshooting.....	3-23

3.1. Introduction

To use the B-HAVE Module tasks in VR-Forces, you must use a terrain that has PathData generated for the areas in which you want to assign B-HAVE Module tasks. You generate PathData using the TDB Tool. The B-HAVE Module includes a plug-in to the TDB Tool that adds the functionality required to generate PathData.

When you generate PathData, the B-HAVE Module creates separate sets of PathData, at various levels of detail, for lifeforms and for ground platforms. You then specify which, if any, PathData you want to associate with the terrain for use by these two types of entities.

The chosen sets of PathData are referenced in the MÄK Terrain Descriptor (MTD) file for the terrain database that you are using.

PathData is generated and referenced as follows:

1. Specify the terrain areas for which to generate PathData.
2. Create a PathData configuration.
3. Generate the PathData.
4. Select the PathData to use and write the references to the MTD file.

There are different ways to proceed through these steps, depending on your expertise with this tool.

The typical process for generating simple PathData is as follows:

1. Create a bounding box and one seed point, valid for lifeforms and ground platforms.
2. Create two configurations, one for Lifeforms (standard lifeform and one for ground entities (standard ground platform).
3. Run a batch generation for both configurations, which generates two sets of PathData.
4. Associate lifeform PathData to lifeforms, and ground platform PathData to ground platforms.

3.1.1. Varying the Complexity of PathData Configurations

Once you are familiar with the basic process for generating PathData, described in the previous section, you may want to create spatial graphs of varying density and accuracy. The hierarchy of complexity is approximately as follows:

- ♦ Simple PathData generation. In a simple PathData generation, you typically use one bounding box and one seed point.
- ♦ Simple PathData generation with Prefer Roads enabled. The B-HAVE Module can automatically include data about roads if the terrain database has road features in the area for which you are generating PathData.
- ♦ Simple PathData generation with local zones. Local zones allow you to add greater detail in specified areas of the terrain, but maintain the same density as the overall PathData generation.
- ♦ Multi-step PathData generation using seed graphs. A seed graph allows you to generate PathData for specific, usually smaller, areas of the terrain. Then, you include the seed graph in a PathData generation for a larger area of terrain.
- ♦ Multi-step PathData generation using seed graphs and local zones. Takes advantage of all options for varying detail and density in different areas of the terrain.

Within any one of these work flows, you can vary detail and density by your choice of the templates used to generate PathData. These alternative approaches to PathData generation allow you to maximize the performance of VR-Forces and the B-HAVE Module for different terrains and entity configurations. In fact, as you continue to use the B-HAVE Module, you should plan on experimenting with the PathData generation process to create terrains that maximize your ability to achieve your simulation goals.

3.1.2. Creating PathData from Road Vector Networks

To correctly convert road vector networks to PathData, the vector networks should have the following characteristics:

- ♦ Segments should be grouped into “edges” whenever possible. An edge is a sequence of segments with no junction. This is done automatically when creating a road in the TDB Tool with several segments.
- ♦ Edges should end at junctions, and the endpoints of the connecting roads should be close to each other (less than a few meters apart, and less than the road width).
- ♦ Short segments should be avoided, even more so for segments leading to intersections. The recommended size is in the 10-100m range.
- ♦ To allow multiple road levels, each road edge should have at least one point with a single altitude candidate, so that this altitude can be propagated to other points in the edge (those points for which the altitude has to be guessed among several candidates).

3.2. Loading a Terrain Database

To generate PathData, you must load a terrain database (GDB format only) in the TDB Tool and be in edit mode. If you want to generate PathData for a terrain that is not in GDB format, you must create a project and build a GDB version of the terrain database as described in *MÄK Terrain Database Tool Users Guide*.

To load a terrain database:

1. Start the TDB Tool.
2. Choose **File** → **Open** → **Terrain Descriptor**. The Open Terrain Descriptor dialog box opens.
3. Select the MTD file for the terrain that you want to load.



If you load a terrain database directly (**File** → **Open** → **Terrain Database**), the TDB Tool loads the MTD file associated with that terrain. If the terrain does not have an MTD file, the TDB Tool creates one.

All information entered by you during the process of PathData generation is saved in an XML file that has the same name as the MTD file.

If you open a database by selecting the MTD file, the B-HAVE Module checks for the presence of an XML file in the same directory. If no file is found, it creates a new one. If you move your terrain data, it is recommended that you keep the MTD and XML files in the same directory to maintain your PathData configurations.

3.3. Managing PathData Resources

PathData is generated in areas of the terrain specified by bounding boxes. PathData generation starts from a point in the terrain called a seed point.

3.3.1. Specifying Bounding Boxes

A bounding box is a parallelepiped, defined by two points (min (Southwest corner) and max (Northeast corner)), in which a PathData generation can take place. You can specify multiple bounding boxes in a terrain, but you must specify at least one. If you plan to generate PathData for local zones, you must specify the bounding boxes for those areas.

To create a new bounding box:

1. Choose **B-HAVE → Manage PathData Resources**, or click the Step 1 icon on the B-HAVE Module toolbar (Figure 3-1). The Manage PathData Resources dialog box opens (Figure 3-2).



Figure 3-1. B-HAVE Module toolbar

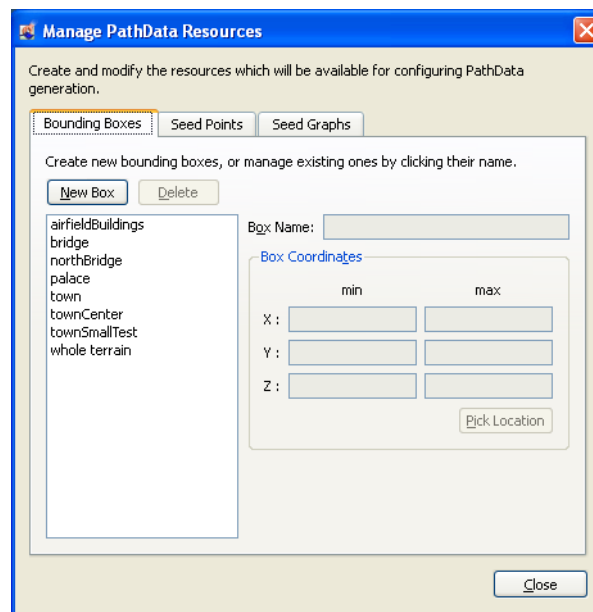


Figure 3-2. Manage PathData Resources dialog box

2. Select the Bounding Boxes tab.
3. Click New Box.

4. In the Box Name text box, type a name for the bounding box or accept the default value. (Bounding box and seed point names must be unique for the terrain.)
5. Enter the X, Y, and Z coordinates for min and max. You can enter the coordinates manually or select them on the map.

To select the min and max values on the map:

- a. Click Pick Location.
- b. Click the Northwest corner of the area you want to define and drag a box on the map.

Zmin is automatically computed as “min” observed z – 1000 m.

Zmax is automatically computed as “max” observed z + 1000 m.

Editing Bounding Boxes

After you create a bounding box, you can edit it.

To modify an existing bounding box:

1. Choose **B-HAVE** → **Manage PathData Resources**, or click the Step 1 icon on the B-HAVE Module toolbar. The Manage PathData Resources dialog box opens ([Figure 3-2](#)).
2. Select the Bounding Boxes tab.
3. In the list window, select the bounding box you want to edit.
4. Edit the min and max coordinates manually, or click Pick Locations and drag a new box on the map.

3.3.2. Specifying Seed Points

Seed points are physical points in the 3D world that can be accessed by a given type of entity. Seed points are used as a starting point for the PathData generation process (Figure 3-3).

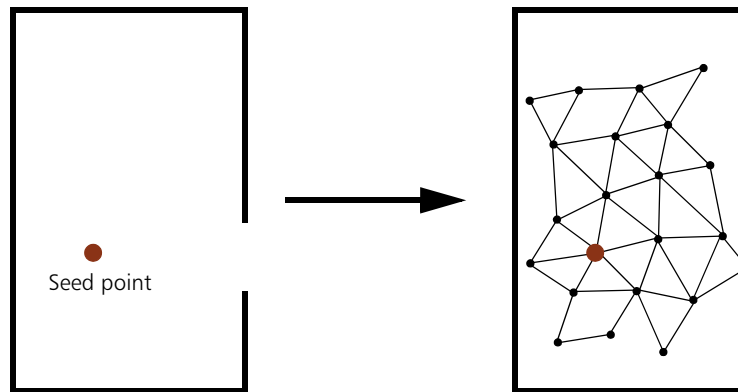


Figure 3-3. Generating a spatial graph from a seed point

You must specify at least one seed point per terrain database. If the terrain includes several zones that are not connected with a viable path for an entity type, you must specify a seed point for each zone. For example, an island that is not connected to the mainland by a bridge requires its own seed point. Or, if a roof can be accessed by a para-trooper, but is not connected to the inside of the building by an open staircase, you must place a seed point on the rooftop to generate PathData.

To create a new seed point:

1. Choose **B-HAVE** → **Manage PathData Resources**, or click the Step 1 icon on the B-HAVE Module toolbar. The Manage PathData Resources dialog box opens.
2. Select the Seed Points tab (Figure 3-4).
3. Click New Point.
4. In the Point Name text box, type a name for the seed point or accept the default value. (Bounding box and seed point names must be unique for the terrain.)
5. Enter the X, Y, and Z coordinates. You can enter the coordinates manually or select them on the map.

To select the seed point on the map:

- a. Click Pick Location.
- b. Click on the map to specify the seed point location.

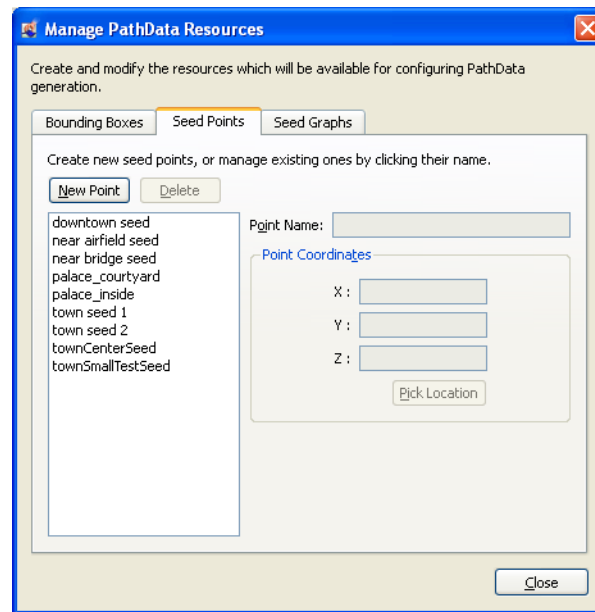


Figure 3-4. Manage PathData Resources dialog box, Seed Points tab

3.3.3. Specifying Seed Graphs

Seed graphs are previously generated graphs used as a seed for generating PathData. For instance, the inside of a building can be generated with fine detail settings, and the resulting PathData can be included as a seed graph in a configuration with street-level detail settings. This is useful when you want to create graphs with a complex geometry (non plane-parallel tiles).

A graph is a collection of edges and vertices. When presented with a seed graph, the PathData generation uses all its vertices as seeds, and generates a new graph encompassing the seed graph. Then, the simplification process can either respect the original seed graph, or merge it with the final graph.

In [Figure 3-5](#), a graph initially generated with a high density inside a room, is used as a seed to generate a much coarser graph in the outside. The resulting graph is optimized in size and accuracy.

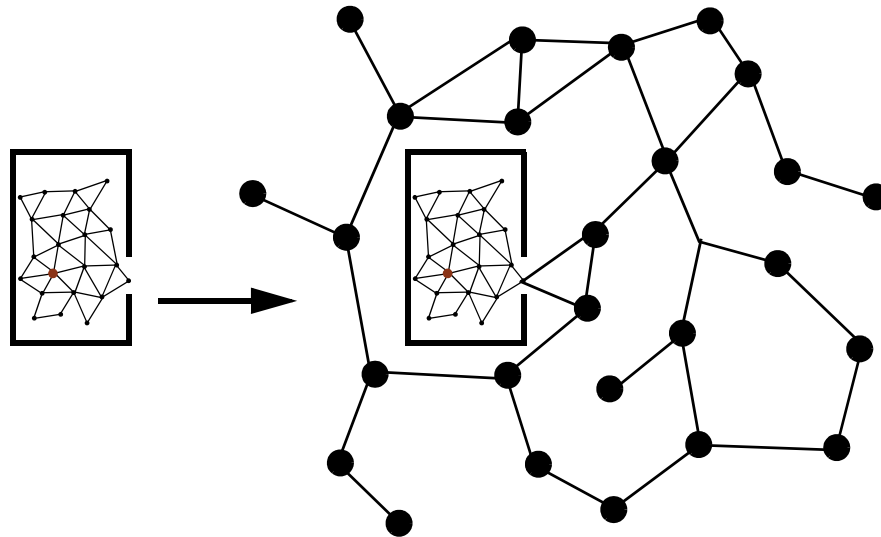


Figure 3-5. Building a spatial graph from a seed graph

To add a seed graph to the PathData resources:

1. Choose **B-HAVE → Manage PathData Resources**, or click the Step 1 icon on the B-HAVE Module toolbar. The Manage PathData Resources dialog box opens.
2. Select the Seed Graphs tab ([Figure 3-6](#)).
3. Click Load.
4. Select any file with a *.pdl* extension. You may need to navigate to other PathData directories to find the files.

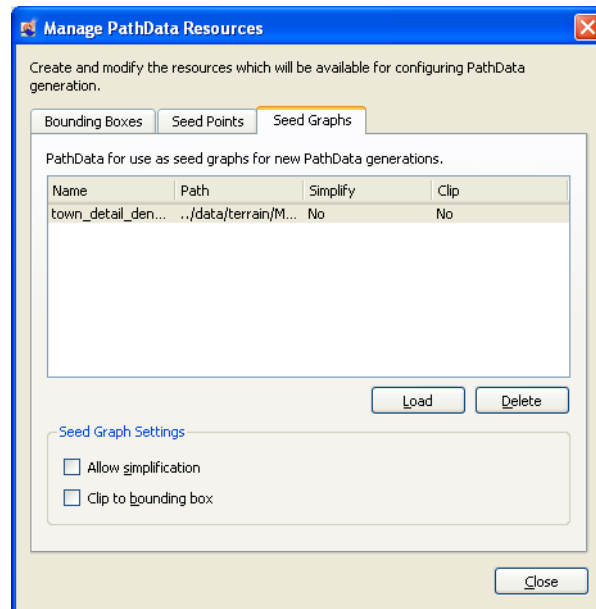


Figure 3-6. Manage PathData Resources dialog box, Seed Graphs tab

5. Select the parameters to be applied to the seed graph during the generation phase, as follows:

- Allow Simplification:

If selected, this graph is considered to be part of the dense graph, and will be simplified, if possible, during the ultimate phase of Spatial Graph generation.

If cleared, the original seed graph is left untouched. If you want to generate a graph with a high density in some places, and a low overall density, do not allow simplification.

- Clip To Bounding Box:

If the seed graph extends beyond the limit of the bounding box that you will use for generating PathData, selecting this option clips the seed graph at the border of the bounding box.

If this option is cleared, the part of the seed graph that is outside the bounding box is kept in the final graph.

These parameters are saved in an XML file generated by the B-HAVE Module. The file is named `<terrain_database>_BHave.xml`.

3.4. Configuring Generation of PathData

After you specify the terrain resources for path generation (bounding boxes, seed points, and seed graphs), you must create a configuration for the B-HAVE Module to use when it generates PathData.

3.4.1. Creating a PathData Configuration

To create a PathData configuration:

1. In the TDB Tool, select **B-HAVE** → **Configure PathData Generation**, or click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens (Figure 3-7).

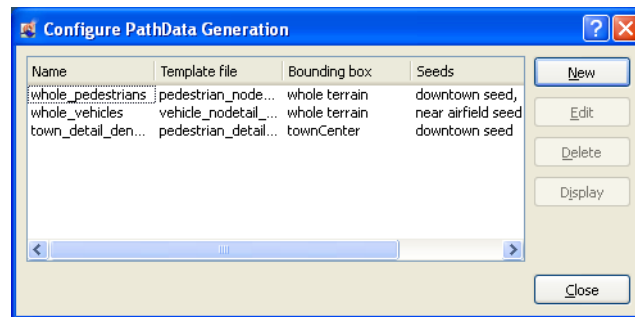


Figure 3-7. Configure PathData Generation dialog box

2. Click New. The Edit PathData Configuration dialog box opens (Figure 3-8).
3. Type a name for the configuration in the Configuration Name text box.
4. In the Template File list, select a template for this configuration. The template determines the level of detail in the PathData. If you are generating PathData for lifeforms, you must use a lifeform template. If you are generating data for ground platforms, you must use a ground platform template. For details about templates, please see “[Choosing a PathData Template,](#)” on page 3-13.
5. Optionally, specify forbidden surfaces. For details, please see “[Filtering Surfaces,](#)” on page 3-14.

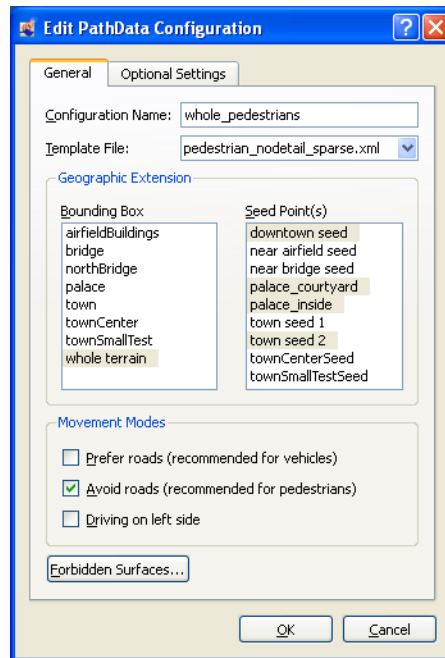


Figure 3-8. Edit PathData Configuration dialog box

6. Select a bounding box from the Bounding Box list. The bounding box defines the area that can be reached by any entity that uses this PathData.
7. Select one or more seed points in the Seed Points list. One seed point is required for each separated area in the bounding box. For more information about using multiple seed points, please see [“Specifying Seed Points,”](#) on page 3-7.
8. Optionally, in the Movement Modes group box, select one or more movement modes. If you select one of these options and the terrain contains road features, the B-HAVE Module includes information about the roads in the PathData. This allows you to set an entity’s movement mode in VR-Forces. For more information, please see [“The Set Movement Mode Set Data Request,”](#) on page 4-12. If you select The Driving on Left Side option, vehicles drive on the left side of the road instead of the right side.
9. Optionally, select one or more seed graphs. If you do not specify a seed point, you must select a seed graph. For details, please see [“Configuring Seed Graphs,”](#) on page 3-16.
10. Optionally, specify local zones. For details, please see [“Configuring Local Zones,”](#) on page 3-15.
11. Click OK. The configuration is added to the list in the Configure PathData Generation dialog box.

Choosing a PathData Template

You specify detail and density by choosing among a set of templates provided with the B-HAVE Module. Each template specifies a level of detail and a density level. For conceptual background, please see “[PathData Templates](#),” on page 2-5.

If a terrain contains no small features or corridors, you can choose the “no detail” level of detail. In most cases, choose the “detail” or “more detail” level.

Higher density provides higher quality, but longer pathfinding computations. For a large terrain, you should choose sparse or very sparse. For smaller areas (if you generate a seed graph, for instance) you can use the dense setting.

The PathData generation templates are named using the format *xxx_yyy_zzz.xml*, where:

- ♦ *xxx* is the entity type, either lifeform or ground platform. It determines the entity size and a few related internal parameters.
- ♦ *yyy* is the maximum level of detail to be taken into account by the entities.
- ♦ *zzz* is the graph density.

i

When you generate a final graph using one or more seed graphs, you should choose a template with a level of detail that is the same as the most detailed seedgraph used. Otherwise, the seedgraph's AI Mesh will be degraded, leading to inaccuracies or problems around detailed areas.

Filtering Surfaces

Filtering surfaces lets you ignore (or select exclusively) certain VR-Forces surfaces types during the terrain exploration phase of the PathData generation. No graph edges are generated over these areas, or graph edges are generated only on these surfaces.

This option is useful to prevent lifeforms from walking on water, and ground platforms from entering buildings with doors large enough to allow a path.

This option can be used only if the terrain contains the relevant information.

To specify forbidden surfaces:

1. In the Edit PathData Configuration dialog box ([Figure 3-8](#)), click Forbidden Surfaces. The Forbidden Surfaces Settings dialog box opens ([Figure 3-9](#)).

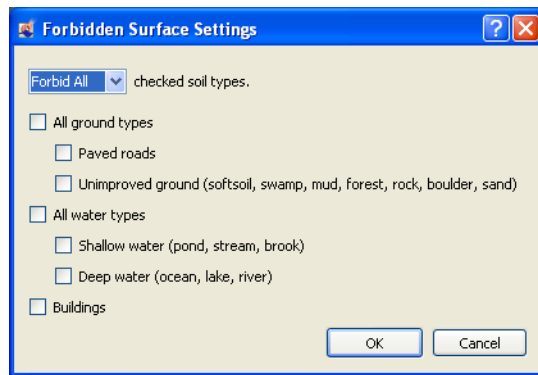


Figure 3-9. Forbidden Surface Settings dialog box

2. Select Forbid All or Allow Only from the drop-down list.
3. Select the surface types that you want to forbid or allow.
4. Click OK.

Configuring Local Zones

Local zones allow you to specify areas of greater detail within the terrain for which you are generating PathData. The bounding boxes must have already been specified as PathData resources.

To configure a local zone.

1. In the Edit PathData Configuration dialog box (Figure 3-8), select the Optional Settings tab (Figure 3-10).

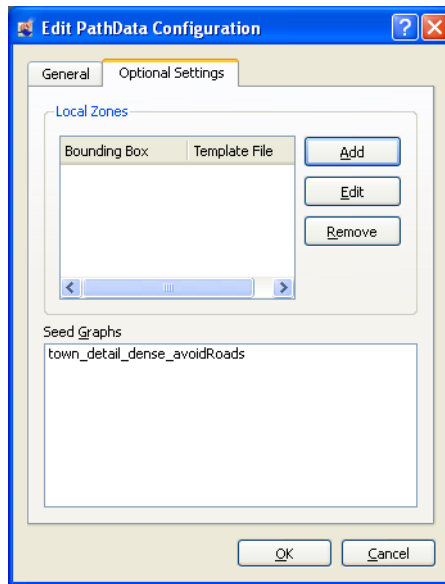


Figure 3-10. PathData configuration optional settings

2. In the Local Zones group box, click Add. The Edit Local Zone dialog box opens.
3. Select a bounding box from the drop-down list.
4. Select a template from the Template File list. Only the detail level of the template applies to local zones.
5. Click OK. The local zone is added to the Local Zones list.
6. Repeat the process for additional local zones if desired.

Configuring Seed Graphs

Adding a seed graph to a PathData configuration lets you specify areas that have different levels of detail, density, or both from the primary template.

To configure a seed graph:

1. In the Edit PathData Configuration dialog box ([Figure 3-8](#)), select the Optional Settings tab ([Figure 3-10](#)).
2. In the Seed Graphs list, select the seed graphs you want to include in the PathData configuration.

3.4.2. Displaying PathData Configurations

After you generate PathData, you can display it.

To display a PathData configuration:

1. Open a terrain for which you have generated PathData.
2. Select **B-HAVE** → **Configure PathData Generation**, or click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens ([Figure 3-7](#)).
3. In the list of PathData configurations, select the configuration that you want to display.
4. Click Display. The configuration data is displayed on the map ([Figure 3-11](#)).

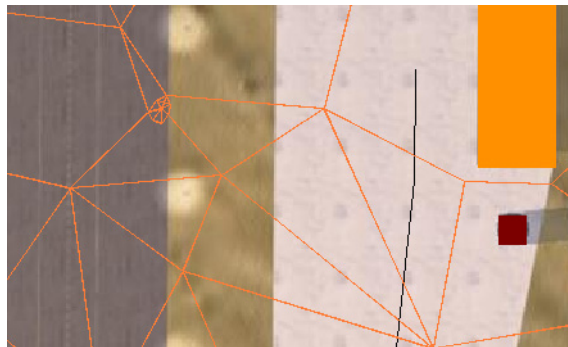


Figure 3-11. Vehicle PathData in Makland database

5. To turn off the display of configuration data, select a different configuration or click Close.

3.4.3. Editing PathData Configurations

You can edit PathData configurations.

To edit a PathData configuration:

1. In the TDB Tool, select **B-HAVE** → **Configure PathData Generation**, or click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens (Figure 3-7).
2. Select the configuration that you want to edit.
3. Click Edit. The Edit PathData Configuration dialog box opens (Figure 3-8).
4. Edit the configuration as desired.
5. Click Save.

3.5. Generating PathData

After you specify the terrain resources and create PathData configurations, you generate the PathData. The generation can last from a few minutes to hours to days depending on the terrain complexity, the level of detail, and your computer's processing power. The PathData generation process creates four files for each PathData, all named after the configuration name.

i

- ♦ If you have previously generated PathData files using the same configurations, it is recommended that you erase these files before generating new ones.
 - ♦ We recommend that you generate path data inside a small bounding box and check the generation process as it proceeds. If you are satisfied with the trial, hide the displays while you run the final generation.
 - ♦ You can edit a previously used configuration and generate a new set of PathData using the same configuration name.
 - ♦ The B-HAVE Module generates several working files. The only files that are used by the VR-Forces plug-ins are *<configuration_name>.fdl*, *.pdl*, and *.mba*.
-

To generate PathData:

1. Choose **B-HAVE** → **Generate PathData**, or click the Step 3 icon on the B-HAVE Module toolbar. The Generate PathData dialog box opens (Figure 3-12). It lists the configurations that you created for this terrain database.

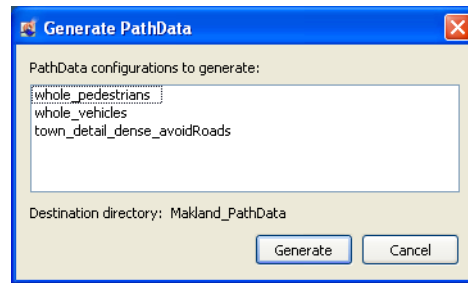


Figure 3-12. Generate PathData dialog box

2. Select the configurations for which you want to generate PathData. Remember that each configuration uses a template that is valid for only one entity type, so you will typically want to include at least two configurations, one for lifeforms and one for ground platforms.
3. Click Generate. The B-HAVE Module displays a progress dialog box (Figure 3-13).

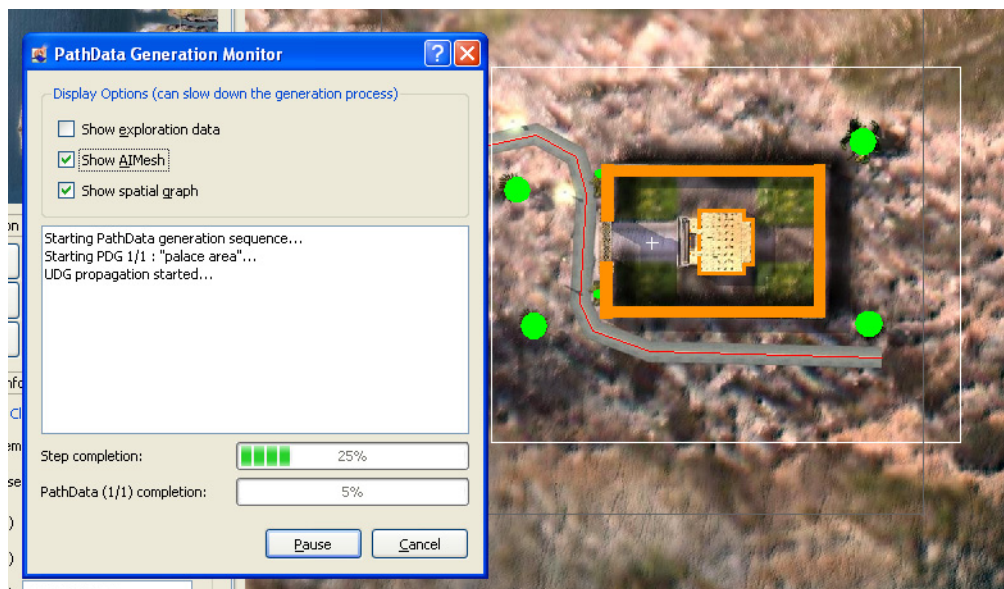


Figure 3-13. PathData generation

3.5.1. Monitoring the PathData Generation Process

The PathData Generation Monitor dialog box displays the progress of the generation process. It also provides options for viewing exploration data, the AI mesh, and the spatial graph (Figure 3-14). For an intensive generation, it is recommended that you not display this data, because it requires processing power and slows down the generation.

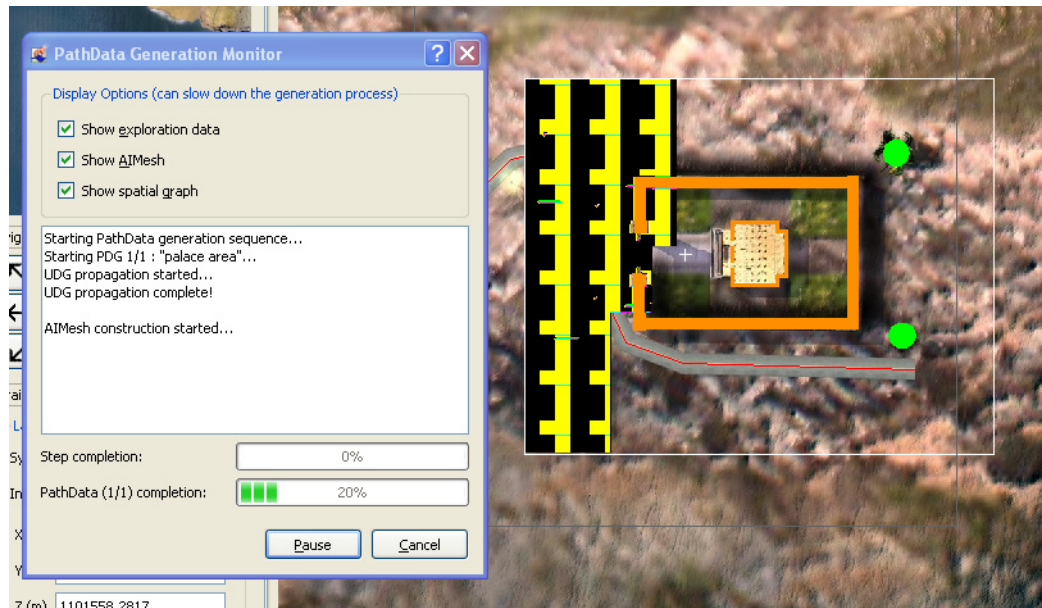


Figure 3-14. Monitoring PathData generation

The PathData files are written in a new folder in the terrain file's folder. The output folder name is the terrain file name, with the suffix *_PathData*.

- To display data about the PathData generation process, in the PathData Generation Monitor dialog box, select the check boxes for the type of data you want to view.

3.6. Associating PathData with Entity Types

After you generate PathData, you must choose which PathData to assign to lifeforms, and which to assign to ground platforms. The MTD file for the terrain is updated to reference the appropriate PathData files, as follows:

- ♦ *path_data.pdl* (graph)
- ♦ *path_data.mba* (AI Mesh)
- ♦ *path_data.fdl* (topological analysis data).

To associate PathData with entity types:

1. Choose **B-HAVE** → **Associate PathData**, or click the Step 4 icon on the B-HAVE Module toolbar. The Associate PathData with Entity Types dialog box opens.

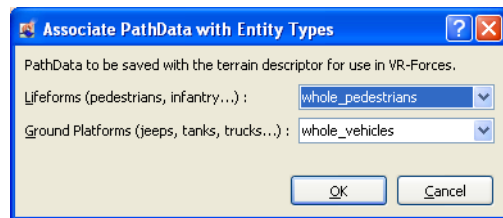


Figure 3-15. Associate PathData with Entity Types dialog box

2. In the Lifeforms list, select the PathData that you want to use for lifeforms.
3. In the Ground Platforms list, select the PathData that you want to use for ground ground platforms.



Although you do not have to assign a PathData to an entity type, if you do not, the B-HAVE Module menus will not display in the VR-Forces GUI.

3.6.1. The BHAVEPathDataRecord

This section describes the BHAVEPathDataRecord added to an MTD file for B-HAVE Module data. As with other records in an MTD file, we recommend that you use the TDB Tool as your editor. However, you can edit an MTD file manually if you want to. The syntax of a BHAVEPathDataRecord is as follows:

```
BHAVEPathDataRecord
{
  Import True
  PathDataName Path_data_for_lifeforms_or_groundplatforms
  PathDataDirectory terrain_name_PathData
  PathIsRelativeToTerrain 1
  ObjectType "entity_enumeration"
}
```

Table 3-1: BHAVEPathDataRecord attributes

Attribute	Description
Import	Specifies whether or not to include the data when the terrain is built. This attribute is not displayed in the project panel. It is set according to the state of the check box next to a record's name when a project is saved. Default: True.
PathDataName	The name of the PathData being configured, for example, <i>whole_lifeforms2</i> . The name cannot have spaces.
PathDataDirectory	The directory in which the PathData being configured is stored. Can be a complete pathname or relative path, depending on the setting for PathsRelativeToTerrain. The path cannot have spaces.
PathsRelativeToTerrain	Specifies whether the value in PathDataDirectory is absolute path (0) or is relative to the <i>./data/terrain</i> directory (1).
ObjectType	The entity enumeration for the object type configured in this record: lifeform (for example, 1 3 1 -1 -1 -1 -1 -1) or ground platform (for example, 1 1 1 -1 -1 -1 -1 -1).

3.7. Testing PathData

You can test the PathData that is generated.

To test PathData:

1. Click the Test icon on the B-HAVE Module toolbar. A list of PathData configurations is added to the toolbar.

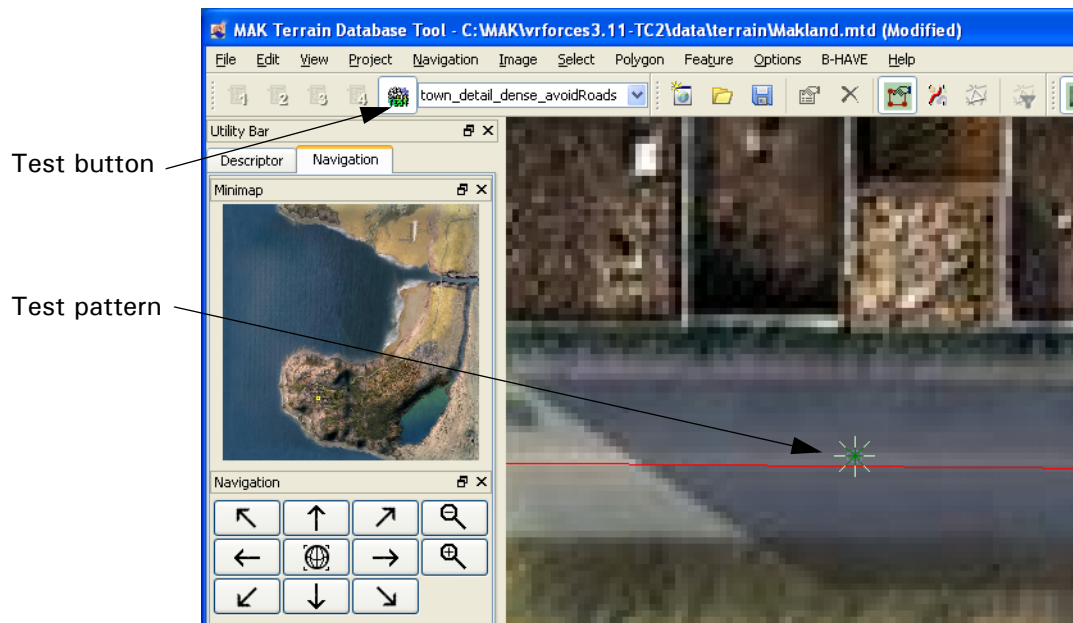


Figure 3-16. Testing PathData

2. Select a configuration.
3. Zoom in on the part of the terrain that you want to check, such as a road, until you can see an object the size of an entity.
4. Click on the map. It displays a test pattern with eight legs, either red or green. The pattern represents the entity, and the legs the possible directions it can take. A red leg indicates that the entity cannot move in this direction. If you click a location where no PathData has been generated, the pattern is a red hexagon (Figure 3-17).

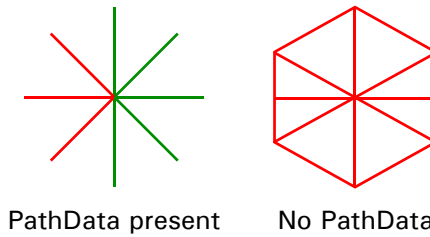
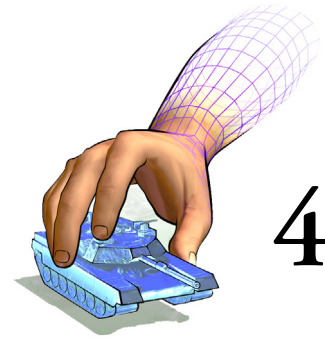


Figure 3-17. PathData test patterns

5. Click the Test icon in the B-HAVE Module toolbar to exit Test mode.

3.8. Troubleshooting

If you do not see the B-HAVE Module toolbar or menu in the TDB Tool, the plug-in was not loaded. Make sure that the file `./plugins/bhave.mtl` is present. It should contain the path in which to look for the libraries to be loaded.



Using the B-HAVE Module in VR-Forces

This chapter explains how to assign B-HAVE Module behaviors to entities. It assumes that you are familiar with VR-Forces and know how to assign independent tasks and how to assign tasks as part of plans.

Using the B-HAVE Module in a Simulation	4-2
Disabling the B-HAVE Module Plug-ins	4-2
Creating Multiple Entities.....	4-3
Adding an Area for Multiple Entity Creation.....	4-5
Specifying the Entities to Create.....	4-5
Sending Text Messages	4-6
B-HAVE Module Tasks.....	4-7
The Flee From Task.....	4-8
The Follow Entity Task	4-8
The Hide From Task.....	4-9
The Move to Waypoint Task	4-9
The Move to Location Task.....	4-10
The Move Along Route Task.....	4-11
The Wander Task	4-11
The Set Movement Mode Set Data Request	4-12
The Set Script Set Data Request	4-13
The View Target Set Data Request	4-13
Viewing B-HAVE Module Data in VR-Forces	4-14
Example Scenarios	4-15

4.1. Using the B-HAVE Module in a Simulation

When you install the B-HAVE Module, it adds a B-HAVE menu to the menu bar and new commands to the Task and Set menus. You do not have to do any additional configuration to use the B-HAVE Module, other than ensuring that use of plug-ins is enabled.



The way that B-HAVE Module tasks are organized on the Tasks menu depends on how you configure the B-HAVE Module. For details, please see [“B-HAVE Module Tasks,”](#) on page 4-7.

To use the B-HAVE Module tasks and scripts in a simulation, you must have a terrain with PathData generated for the appropriate type of entity. The procedure for generating PathData and adding them to the MTD file is explained in Chapter 3, [Generating PathData](#).

4.1.1. Disabling the B-HAVE Module Plug-ins

If, after installing the B-HAVE Module, you do not want to load the plug-ins, you can temporarily disable loading. To do so, remove or rename `./plugins/bhave.mtl`.

4.2. Creating Multiple Entities

The B-HAVE Module allows you to add multiple entities to a scenario in one operation and to assign a common task to the entities. Entities are added to the scenario as members of a selection group. Creation of multiple entities has the following constraints:

- ♦ The entities are added within a specified area. If an appropriate area does not already exist, you can create one as part of the entity creation process.
- ♦ If a terrain does not have any PathData associated with it, you can place multiple entities anyplace on the terrain. The entities are placed without regard to terrain features.
- ♦ If a terrain has PathData, you cannot place entities on portions of the terrain that do not have PathData. If the area in which you place entities is not entirely within the portion of the terrain that has PathData, the entities are placed only in that part of the area that has PathData.
- ♦ If the area you choose for placing entities is too small to contain the number of entities that you specify, some entities might not be created.

In [Figure 4-1](#), Area 1 overlaps the PathData. If, for example, you try to place 25 entities in this area, they are placed only in the portion of Area 1 that overlaps PathData, and only the number of entities that fit in this subset of Area 1 are created.



If the B-HAVE Module cannot create the number of entities specified (due to one of the reasons mentioned in this section), there is no notification. It is your responsibility to check to see how many entities were created.

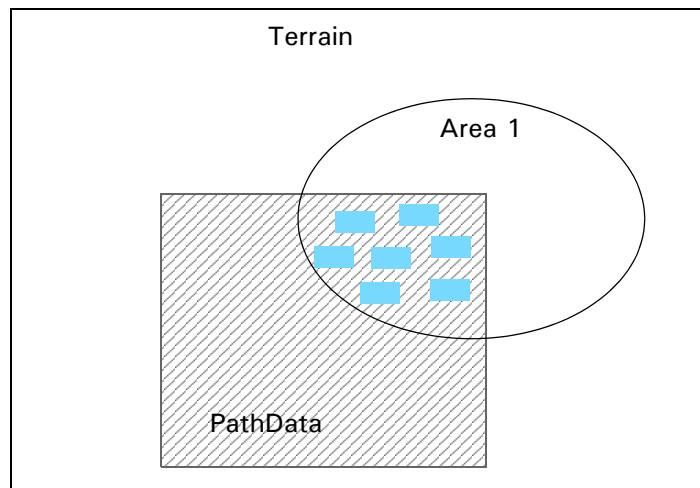


Figure 4-1. Placement of multiple entities

To add multiple entities to a scenario:

1. Choose **B-HAVE** → **Create Multiple Entities**. The Create Multiple Entities dialog box opens (Figure 4-2).

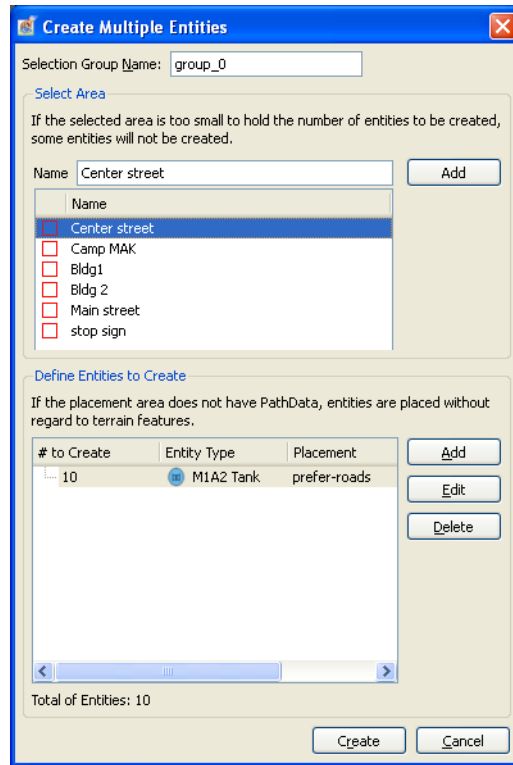


Figure 4-2. Create Multiple Entities dialog box

2. In the Selection Group Name text box, specify a name for the selection group, or accept the default name. (For information about selection groups, please see Section 7.4, “[Creating and Using Selection Groups](#)”, in *VR-Forces Users Guide*.)
3. In the Select Area group box, type the name of the area in which you want to create the entities, or select the area from the list or on the map. If the area in which you want to create the entities does not exist, add a new area as described in “[Adding an Area for Multiple Entity Creation](#),” on page 4-5.
4. In the Define Entities To Create group box, add the entities that you want to create as part of this selection group. For details about this part of the procedure, please see “[Specifying the Entities to Create](#),” on page 4-5.
5. Optionally, edit or remove any of the entity sets listed in the Define Entities To Create list.
6. Click Create. The entities are created in the specified area.

4.2.1. Adding an Area for Multiple Entity Creation

To add an area from the Create Multiple Entities dialog box:

1. In the Select Area group box, click Add. The Create Area dialog box opens.
2. Create an area following the standard VR-Forces procedure for creating areas. For details, please see Chapter 16, *Overlays and Graphical Objects*, in *VR-Forces Users Guide*.

4.2.2. Specifying the Entities to Create

When you add multiple entities to a scenario, you can specify any of the supported entity types and assign a common task or set data request to a specific set of entities.

To specify the entities to add in the Create Multiple Entities dialog box:

1. In the Define Entities to Create group box, click Add. The Add Group of Entities dialog box opens (Figure 4-3).



Figure 4-3. Add Group of Entities dialog box

2. In the Number of Entities box, specify the number of entities you want to add for this entity set.
3. Click the Select button that is next to the Type of Entity text box and select the entity type that you want to add. The list of entity types available is the same list of entities provided on the Create menu for ground and lifeform entities.
4. Optionally, click the Select button that is next to the Starting Action text box and select a task or set data request from the list. This list is not context sensitive. You must ensure that the selected task or set data request is appropriate to the entity type that you are adding.
5. Optionally, select an option in the Positioning group box.
6. Click OK. The entity set is added to the list in the Define Entities To Create group box (Figure 4-2).
7. Optionally, add additional entity sets.

4.3. Sending Text Messages

You can send text messages from the B-HAVE menu to entities. These messages are identical to the text messages that you can send to entities from the VR-Forces Task menu, so entities can test for them and use them as triggers to launch any appropriate behavior for the entity. Since B-HAVE Module text messages are independent of the entity task mechanisms, you can send messages directly without going through an entity.

To send a text message to an entity:

1. Choose **B-HAVE** → **Send Text Message**. The Send Text Message dialog box opens.
2. Select the entities to whom you want to send the message.
3. In the Message box, type the message.
4. Click OK.

4.4. B-HAVE Module Tasks

The B-HAVE Module has a set of tasks that you access from the Task menu or in plans, the same way that you access any other entity task. Tasks that are unique to the B-HAVE Module (such as Flee From and Wander) are always added to the VR-Forces Task menu. By default, the following B-HAVE Module tasks replace the native VR-Forces tasks when the B-HAVE Module is enabled:

- ♦ Follow Entity
- ♦ Move to Waypoint
- ♦ Move to Location
- ♦ Move Along Route.

When you assign one of these tasks, the B-HAVE Module version is used if:

- ♦ The task is applied to an entity type supported by the B-HAVE Module and for which PathData has been generated
- ♦ The task is to be executed on a portion of the terrain that contains PathData (that is, both the entity, and if applicable, the destination are on terrain for which there is PathData).

If these criteria are not met, the native VR-Forces version of the task is used. VR-Forces handles the decision making process for when to use the B-HAVE Module tasks or VR-Forces versions of these tasks. You do not have to do anything except assign the tasks that you want to use. However, you should be aware of these conditions in case your entities behave in unexpected ways.

If while a B-HAVE Module task is being executed, conditions change so that the criteria for a B-HAVE Module task no longer apply, the B-HAVE Module changes the task to a VR-Forces task. For example, if you give an entity a Move to Waypoint task and the entity and waypoint are both inside an area that has PathData for that entity type, the B-HAVE Module assigns a B-HAVE Move to Waypoint task. However, if while the entity is moving to the waypoint, you move the waypoint outside of the area with PathData, the B-HAVE Module changes the task to a VR-Forces Move to Waypoint task.

i

You can configure the B-HAVE Module so that B-HAVE Module movement tasks do not replace the standard VR-Forces tasks and all B-HAVE Module tasks are listed in a separate section of the Task menu. To make this change, in `./data/parameters/BHAVE/bhaveOptions.mtl`, change the `unify-movement-tasks` parameter to `False`.

The procedures in this chapter assume the default configuration. If you specify that B-HAVE Module tasks not replace the VR-Forces tasks, the task names in the procedures are all prefaced with B-HAVE.

4.4.1. The Flee From Task

The Flee From task causes entities to try to maintain a specified distance from other specified entities (the pursuers). If you select the Stop When Hidden option, an entity stops fleeing when it finds a hiding place, even if the hiding place is in the radius within which it is supposed to flee. This task is continuous unless it is overridden manually or by a conditional test in a plan. The entity will continue to flee any time the distance from the pursuers falls within the specified radius and is in line-of-sight.



We use the term “pursuers” for convenience. The pursuers do not have to be actively seeking the fleeing entity for this task to be in effect.

To assign a Flee From task:

1. Select the entities that you want to flee.
2. Choose **Task** → **Flee From**. The Flee From dialog box opens.
3. Optionally, filter the list of entities to flee from.
4. Select the entities (the pursuers) from which you want the entities to flee.
5. Specify the distance, in meters, the entities should try to maintain from pursuers.
6. Optionally, to have a fleeing entity stop when it is not within line-of-sight of the pursuers, select Stop When Hidden.
7. Click OK.

4.4.2. The Follow Entity Task

The Follow Entity task works like the standard VR-Forces Follow Entity task except that entities use B-HAVE Module path planning and collision avoidance. There is no option to specify a distance above because this is meaningless for ground entities.

To assign a Follow Entity task:

1. Select the entity.
2. Choose **Task** → **Follow Entity**. The cursor changes to select mode and the Follow Entity dialog box opens.
3. Optionally, filter the list of entities.
4. In the Name box, type the name of the entity to follow, or in the list, or on the map, select the entity to be followed.
5. Type an offset (in meters) in the Behind and Right text boxes. If you want an entity to follow in front or to the left put a minus sign (-) in front of the value.
6. Click OK.

4.4.3. The Hide From Task

The Hide From task causes an entity to hide from specified entities. It is considered hidden when it is not within line-of-sight of the pursuing entities. This task is continuous unless it is overridden manually or by a conditional test in a plan. The entity will continue to hide any time the pursuers are in line-of-sight.



We use the term “pursuers” for convenience. The pursuers do not have to be actively seeking the hiding entity for this task to be in effect.

To assign a Hide From task:

1. Select the entities that you want to hide.
2. Choose **Task** → **Hide From**. The Hide From dialog box opens.
3. Optionally, filter the list of entities.
4. Select the entities (pursuers) from which you want the entities to hide.
5. Click OK.

4.4.4. The Move to Waypoint Task

The Move To Waypoint task sends an entity to the specified waypoint or object using B-HAVE Module path planning and collision avoidance.

To assign a Move to Waypoint task:

1. Select the entity.
2. Choose **Task** → **Move To Waypoint**. The cursor changes to select mode and the Move To dialog box opens.
3. Optionally, filter the object selection list.
4. In the Name box, type the name of the object to move to, or in the object selection list, or on the map, select the object that you want the entity to move to.
5. Click OK.

4.4.5. The Move to Location Task

The Move to Location task sends an entity to the specified location using B-HAVE Module path planning and collision avoidance. You can specify locations that are inside buildings.

To assign a Move To Location task:

1. Select the entity to which you want to assign the task.
2. Choose **Task** → **Move To Location**. The cursor changes to select mode and the Move to Location dialog box opens ([Figure 4-4](#)).

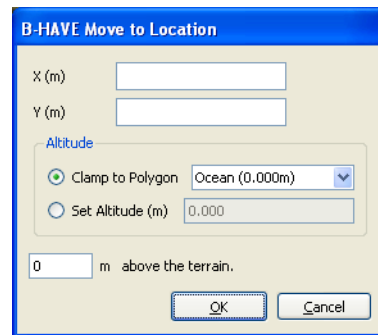


Figure 4-4. Move to Location dialog box

3. Click on the map where you want the entity to move, or enter the coordinates of the location in the Move to Location dialog box. The coordinate information required is determined by the coordinate system selected in the Display Options dialog box.
4. If there are multiple possible altitudes at the selected X, Y coordinate, you can specify the Z value manually, or select an available polygon from the list. This option is most useful for terrains that have multiple levels, such as an urban terrain in which you might be able to place an entity on the roof or several floors of a building. The default location when you click on the terrain is the uppermost polygon (the roof of a building).

To specify an available polygon, in the Altitude group box, select **Clamp to Polygon** and select a polygon from the drop-down list.

To accept the default Z value or specify the value manually, choose **Set Altitude Manually** and type a value.

5. If you want an entity to move to a location inside a building, specify a negative value in the meters above terrain box, where the top of the building is zero and the value entered is the distance to the floor on which you want the entity to be located. For more information about specifying a location in a building, please see [“Specifying a Location Inside a Building”](#), in Chapter 11, *Assigning Tasks*, in *VR-Forces Users Guide*.

6. Click OK.

4.4.6. The Move Along Route Task

The Move Along Route task is identical to the VR-Forces Move Along Route task except that entities use B-HAVE Module path planning and collision avoidance.

To assign a Move Along Route task:

1. Select the entity.
2. Choose **Task** → **Move Along Route**. The cursor changes to select mode and the Move Along Route dialog box opens.
3. In the Name box, type the name of the route, or in the list window, or on the map, select the route that you want the entity to follow.
4. If you want the entity to move along the route from the end point to the start point, select the Reverse Direction check box.
5. Click OK.

4.4.7. The Wander Task

The Wander task causes an entity to move to a series of randomly chosen locations for the specified amount of time. You can constrain wandering to a specified area. If the entity is using a preferred movement mode and there is preferred road data for the terrain, the Wander task will only choose destinations that meet the preference criteria.

To assign a Wander task:

1. Select the entity.
2. Choose **Task** → **Wander**. The Wander dialog box opens.
3. In the Duration group box, select Indefinitely or Timed. If you select Timed, specify the amount of time to wander in hours, minutes, and seconds.
4. In the Movement group box, select Free or Restricted To Area. If you select Restricted To Area, type the name of an area or select it in the list or on the map.
5. Click OK.

4.4.8. The Set Movement Mode Set Data Request

The Set Movement Mode set data request allows you to specify that when an entity plans its path it should take into account road data, if it is available. This set data request is meaningful only if one or both of the movement modes was enabled when the PathData for this terrain was configured. For details about enabling movement modes during configuration, please see [“Configuring Generation of PathData,”](#) on page 3-11.

When an entity is set to Prefer Roads, the entity gives preference to all segments along the preferred graph over those on the non-preferred movement graph. This is typically used to have vehicles prefer movement along roads, rather than cutting across open terrain, even if the distance along roads is longer.

If the movement mode is Avoid Roads, entities try not to move along roads. If they must cross a road, they take the shortest path across the road.

You can configure an entity to start up in a movement mode or in free movement mode by editing the `initial-movement-mode` parameter in the `kynapse-controller` in the system definition file for this entity type.

To set the B-HAVE Module movement mode for an entity:

1. Select the entity.
2. Choose **Set** → **B-HAVE Set Movement Mode**. The Set Movement Mode dialog box opens.
3. To have the entity take road data into account, select Prefer Roads or Avoid Roads from the Movement Mode list. To have an entity plan its path as if there were no road data, select Free.
4. Click OK.

4.4.9. The Set Script Set Data Request

The Set Script set data request applies a Lua script to the selected entity. The script is executed immediately. For details about writing and managing scripts, please see Chapter 5, *Writing B-HAVE Module Scripts*.



If you issue a Set Script set data request while a B-HAVE Module task is executing, it interrupts the task. If you issue Set Script while a VR-Forces task is executing, the script runs simultaneously with the task. It is your responsibility to know if the script conflicts with the task.



When a VR-Forces scenario is saved, each B-HAVE-enabled entity that is using a script stores the name of the script it is running and all the global variables of that script.

To execute a Lua script:

1. Select an entity.
2. Choose **Set** → **B-HAVE Set Script**. The Set Script dialog box opens.
3. Select a script from the list.
4. Click OK.

4.4.10. The View Target Set Data Request

The View Target set data request specifies that an entity face an object while it is carrying out its task.

To assign the View Target set data request:

1. Select an entity.
2. Choose **Set** → **B-HAVE View Target**. The Set View Target dialog box opens.
3. In the Name box, type the name of the object to view, or in the list window, or on the map, select the object.
4. If an entity is currently viewing an object, select Clear View Target to remove the requirement.
5. Click OK.

4.5. Viewing B-HAVE Module Data in VR-Forces

When B-HAVE-enabled entities are given tasks in VR-Forces, they publish information indicating the path that was chosen for movement. This can be useful while testing a scenario, to check the paths chosen by entities, their line of sight with enemies, and so on. This capability is enabled by default. You can also access it through Lua scripts. For details about adding this capability to scripts, please see [“Displaying B-HAVE Module Data in the 3D Front End,”](#) on page 5-13.

In the 2D front-end, B-HAVE Module paths are displayed as routes. An entity will not move directly along the indicated path because of the runtime path smoothing (described in [“Path Following,”](#) on page 2-6). If you do not want to display these routes, you can hide them.

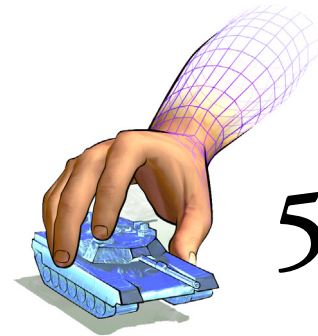
To hide B-HAVE Module path routes, use either of the following methods:

- ♦ In the VR-Forces front-end, choose **Objects** → **View Objects** → **Path Planning Routes**. This hides the debugging routes for this window only.
- ♦ Set the `publish-debug-routes` parameter to `False` in the `kynapse-controller` block in the system definition file for the entity type you want to change. This can improve performance for scenarios with large numbers of entities that are using the B-HAVE Module.

4.6. Example Scenarios

The B-HAVE Module includes the following example scenarios:

- ♦ **CarBomb:** Entity behavior is controlled through B-HAVE tasks in entity plans. Civilians wander around the business district of a small town. Enemy combatants move through tunnels and buildings to set up positions while a car drives to a location and explodes. Surviving civilians flee. Dismounted infantry search for the enemy combatants. Reinforcements arrive by helicopter, disembark, and help search for enemy combatants or take up positions to secure the town. Civilians return to the bomb site.
- ♦ **CarBomb_NoEmbarkation:** This version of the CarBomb scenario does not use embarkation, so you can use it with RPR FOM 1.0. (Embarkation requires use of RPR FOM 2.0, draft 17.)
- ♦ **Crowd:** 100 civilians wander randomly around the city in the TerraSim_SampleUrban terrain. They go in and out of buildings and to multiple floors of buildings.
- ♦ **Interior Movement:** Demonstrates the movement of lifeform entities within buildings. Some civilians wander randomly in and out of buildings, while some DI and other civilians follow pre-determined plans to reach points on different floors of some of the buildings in TerraSim_SampleUrban. DI 1 has no plan. You can task him to move to some of the waypoints to demonstrate dynamic path planning.
- ♦ **Makland B-HAVE:** Mixed vehicle and life form activity on the Makland terrain. Some civilian traffic moves along the roads. Some DI walk in and out of the palace, changing guard duty, and some military vehicles leave the palace to drive along the roads to locations in the town.
- ♦ **Rescue Convoy:** All entity activity is controlled using Lua scripts that are assigned in plans. When a Blue civilian's vehicle is destroyed, friendly vehicles move in a convoy to rescue him. Hostile DIs flee the approaching convoy.
- ♦ **Scatter:** A hostile DI follows a plan to move to a set of patrol points while twenty civilians use Lua scripts to keep hidden from the hostile entity. The civilians run behind buildings whenever they see the hostile entity.
- ♦ **Traffic:** Civilians and civilian vehicles wander around the town in Makland. The scenario was created using multiple entity create, with a Wander task assigned at creation.



Writing B-HAVE Module Scripts

This chapter explains how to write and manage scripts for use with the B-HAVE Module. It also lists B-HAVE Module functions.

Introduction	5-2
Managing Scripts for VR-Forces	5-2
Specifying a Text Editor.....	5-3
Creating a New Script	5-3
Editing a Script	5-4
Deleting a Script.....	5-4
Writing Scripts.....	5-5
Using B-HAVE Module Tasks in Scripts	5-5
Considerations for Writing and Using Lua Scripts.....	5-6
Example Script	5-7
Sending Radio Messages	5-8
Custom Checkpointing.....	5-9
Extending Lua Scripts Using Modules and Custom Libraries	5-10
Displaying B-HAVE Module Data in the 3D Front End.....	5-13
Drawing PathData Around Entities.....	5-13
B-HAVE Module Functions.....	5-14
Entity Functions.....	5-14
Behavior Functions.....	5-17
Entity List Functions.....	5-28
Perception Functions.....	5-29
Utility Functions	5-31
VR-Forces Task and Set Data Request Functions.....	5-35

5.1. Introduction

Lua is a lightweight scripting language used by the B-HAVE Module to extend functionality by enabling custom program logic to control high-level decision-making and lower level perception and behavior settings.

The B-HAVE Module includes more than 25 sample Lua scripts. Please see the comments in the script files for descriptions of what each script does.

This chapter does not explain how to program in Lua. It focuses on how to use Lua with the B-HAVE Module. For details about the Lua language, please see the Lua website (www.lua.org).

5.2. Managing Scripts for VR-Forces

This section explains how to add, edit, and delete the scripts that are available for use with the B-HAVE Module. You can edit scripts dynamically, while VR-Forces is running. You can also write and edit scripts off-line.

If you want to create or edit a script during a VR-Forces session and have the changes take effect during the session, you must create or edit the script following the procedures in this section. If you edit or create a script off-line (that is, by opening the script directly from a text editor), the changes do not take place until the next time you start VR-Forces.



- ♦ When a VR-Forces scenario is saved, each B-HAVE-enabled entity that is using a script stores the name of the script it is running and all the global variables of that script.
 - ♦ There is only one set of scripts saved in the VR-Forces simulation engine. Scripts are not specific to a scenario. If you edit or delete a script, the changes will affect all scenarios that use that script.
-

5.2.1. Specifying a Text Editor

To create or edit a Lua script from the VR-Forces GUI, you must specify the text editor that you want to use, for example, Notepad.

To specify a text editor:

1. Choose **B-HAVE** → **Options**. The B-HAVE Options dialog box opens (Figure 5-1).

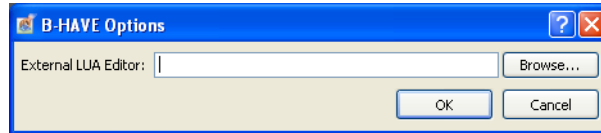


Figure 5-1. B-HAVE Options dialog box

2. Specify the path to the editor that you want to use.
3. Click OK.

5.2.2. Creating a New Script

To create a new Lua script:

1. Choose **B-HAVE** → **Scripts**. The B-HAVE Scripts window opens (Figure 5-2). This window lists all the available scripts.

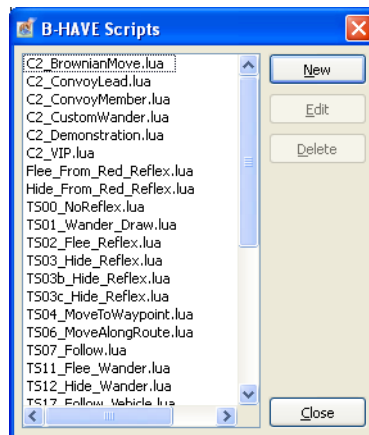


Figure 5-2. B-HAVE Scripts window

2. Click New. You are prompted to specify a name for the script.



If you have not specified a text editor, you are prompted to do so.

3. Specify a name and click OK.
4. A text editor opens a blank file with the specified name.
5. Type the script or paste in a previously written script.
6. Save the changes.
7. The new script is listed in the B-HAVE Scripts window.

5.2.3. Editing a Script

If you edit a script while VR-Forces is running, when you save the script, the changed script replaces the script that is held in memory and it is available for immediate use.

To edit a Lua script:

1. Choose **B-HAVE** → **Scripts**. The B-HAVE Scripts window opens. This window lists all the available scripts.
2. In the list window, select the script you want to edit.
3. Click Edit. The B-HAVE Script opens in a text editor.



If you have not specified a text editor, you are prompted to do so.

4. Edit the script.
5. Save your changes.
6. In the B-HAVE Script window, click Close.

5.2.4. Deleting a Script

To delete a Lua script:

1. Choose **B-HAVE** → **Scripts**. The B-HAVE Scripts window opens. This window lists all the available scripts.
2. In the list window, select the script you want to delete.
3. Click Delete.

5.3. Writing Scripts

Each script has a `Think()` function that runs concurrently with the currently assigned VR-Forces or B-HAVE task, if any. The script computes a control parameter, `AIReportReflexActivationState()`. When `AIReportReflexActivationState` is (1), the script takes control of the entity. When `AIReportReflexActivationState` is (0), the script runs (the brain continues “thinking”), but it does not cause the entity to do anything, even if the entity is not tasked.

The two general approaches to scripts are as follows:

- ♦ The reflex script: A reflex script is idle (reporting 0) by default. It takes control of the entity only when some condition is true. For instance, an entity could track visible enemies and trigger a hiding behavior under certain conditions. As soon as the condition is not true anymore, the script can report 0 again.
- ♦ The continuously active script: A continuously active script always reports 1. You might use such a script to control a lifeform with a finite state machine.

You might want to disable a script from within a plan. To do this, put a B-Have Set Script set data request in the plan, and select the *no_reflex.lua* script. This script never takes control of an entity’s actuators. (It sets `AIReportReflexActivationState()` to 0).

5.3.1. Using B-HAVE Module Tasks in Scripts

All of the B-HAVE Module tasks on the Task menu can be triggered from a Lua script. Each behavior has one “trigger” function, plus a small set of settings accessors. For instance, the Follow behavior is executed by calling `AIFollow()`. Additional settings functions include `AIFollowSetEntityToFollow(entity pointer OR entity name)`, and `AIFollowSetDistFromEntity(distance)`.

[Table 5-1](#) maps the B-HAVE Module tasks to their corresponding Lua functions:

Table 5-1: B-HAVE Module task/Lua function matrix

Task	Function
Move To	<code>AIgoto()</code>
Wander	<code>AIWander()</code>
Follow	<code>AIFollow()</code>
Move Along Route	<code>AIPathway()</code>
Hide	<code>AIHide()</code>
Flee	<code>AIFlee()</code>

5.3.2. Considerations for Writing and Using Lua Scripts

This section describes issues that you should keep in mind when writing Lua scripts for use with the B-HAVE Module.

- If the Lua interpreter detects a syntax error, for example, missing the “then” after “if” statements, missing the “end” keyword after the loop of a conditional, and misspelled (non-existing) function names, it reports the error in the VR-Forces log console. The error message displays the cause of the error and the line where the error occurred. The script is not executed. Errors can be detected during parsing (Set script operation or saving a modified script), or during execution of the Think() function.
- Most entity access-related functions accept either the entity name as a normal character string, or an entity pointer variable (called “CEntity*”). This second option is much more efficient in terms of performance, because it does not require a name-lookup in the list of entities, like the first option does.
- Whenever possible, try to use entity pointer variables, especially in loop blocks.
- The preferred way to loop through the entities list is to use an iterator, as follows:

```
local entity, iterator = AIWorldGetFirstEntity()
while entity ~= nil do
    -- do something

    -- get next entity
    entity, iterator = AIWorldGetNextEntity(iterator)
end
```

Do not use this iterator while destroying or creating entities.

- The coordinate system used internally by the B-HAVE Module is a transformation of the VR-Forces local database system, as shown in [Table 5-2](#):

Table 5-2: VR-Forces to B-HAVE Module coordinate transformation

VR-Forces	B-HAVE Module
x	x
y	-z
z	y

The B-HAVE Module has two functions for transforming coordinates: AIUtilVRF2BHAVE() and AIUtilBHAVE2VRF().

- When a Lua script releases control of an entity, the plan resumes the task that was current when the script took control. The state of the entity when released by the script must be compatible with resuming the plan task.

For instance, if a script takes control of an entity following a route, and brings it away from this route, it must reposition the entity on the route before releasing control.

- ♦ When checkpointing a scenario, the global variables of a script are serialized and saved. For more information about checkpoints, please see [“Custom Checkpointing,”](#) on page 5-9.

5.3.3. Example Script

The following sample Lua script only has a Think() function. The first time the function runs, it displays a message explaining its purpose. Thereafter, it forces the entity to wander, ignoring all other tasks, and to draw the path that is being followed.

```
-- global initialization flag
initialized = 0

function Think()
  if initialized == 0 then
    AIUtilDebug("Basic wander Lua script. It always reports a reflex,
preventing any task from being executed.")
    initialized = 1
  end

  -- simple wander behavior
  AIWander()

  -- tell the controller to apply commands from this script only,
  -- ignoring any other task
  ReportReflexActivationState(1)

  -- let's draw the path being currently followed
  AIUtilDrawPath()
end
```

The B-HAVE Module includes several example scripts. They are in *./data/parameters/BHAVE/scripts*.

5.3.4. Sending Radio Messages

You can use Lua to have entities send each other text strings (radio messages).

To send a message, use `AIVRFRadioSendMessage(string)` or `AIVRFRadioSendMessage(string, entity name)` functions anywhere in a script. For an example, please see the sample script *testRadio_Dispatcher_Waypoint_Speed.lua*, which has a list of messages to send at specific times in the simulation.

To receive a message, use the function `AIVRFRadioMessageReceivedCallback(string)`. It is called asynchronously from the `Think()` function when a message is received by the entity. For an example, please see the sample script *testRadio_Receiver_Waypoint_Speed.lua*, which shows how to listen to a message, and optionally extract a waypoint name from the message to execute an order.

The following is a simple example. It enables sending and receiving messages on this entity:

```
-- this script is a simple Lua radio test.
-- Cvln 1 will send messages
-- everyone will print received messages
counter = 0
myName = AIEntityGetName()
function Think()
    if counter == 1000 and myName == "Cvln 1" then
        AIUtilWarn("sending messages...")
        AIVRFRadioSendMessage("hello world !") -- this message is
        broadcast to all entities
        -- this message is sent to Cvln 2
        AIVRFRadioSendMessage("hello to you, Cvln 2", "Cvln 2")
        counter = 0
    end
    counter = counter + 1
end
function AIVRFRadioMessageReceivedCallback(text)
    AIUtilWarn("received radio message:" .. text)
end
```

5.3.5. Custom Checkpointing

VR-Forces checkpointing saves Lua global variables. The following aspects of B-HAVE entity behaviors are not, by default, contained in global variables:

- ♦ Current destination waypoint name
- ♦ Current point on a route
- ♦ Current list of entities to flee from, and so on.

If you want to save the entity state that is pertinent to a script, you must save it in global variables before checkpointing.

To save information to global variables, use the `AboutToCheckpoint()` function. If this function is in a script, it is called automatically before VR-Forces checkpoints a scenario.

To restore the script data when a scenario is loaded, use the `PostLoadCheckpoint()` function. It is called just after loading a checkpoint.

The following sample script shows how to save the current point in a route:

```
-- as a global variable, this will be saved/restored.
routeName = "Route 1"

initialized = false
function Think()
  if initialized == false then
    -- initialize
    AIPathwaySetVRFRoute(routeName)
    initialized = true
  end
  AIPathway()
  ReportReflexActivationState(1)
end
-- save all wanted values as globals
function AboutToCheckpoint()
  AIUtilDebug("test: aboutToCheckpoint")
  -- we need to save "manually" the current point on the route to a
  -- global variable which will then be saved automatically.
  currentRouteTargetPoint = AIPathwayGetCurrentPointIndex()
end
-- restore previously saved globals to reinitialize behaviors
function PostLoadCheckpoint()
  AIUtilDebug("test: PostLoadCheckpoint.")
  if initialized == true then
    AIPathwaySetVRFRoute(routeName)
    AIUtilDebug("test: setting current route target point " ..
currentRouteTargetPoint)
    AIPathwaySetCurrentPointIndex(currentRouteTargetPoint)
  end
end
```

Guidelines for easy checkpointing:

- To keep things simple, all working variables and tables should be declared local so they are not serialized.
- When checkpointing (AboutToCheckpoint()), copy all variables and tables that need to be serialized in one global table.
- When reloading a checkpoint, the script will have been parsed before the PostLoad-Checkpoint() is called. Therefore, you should only do the operations corresponding to what was done in AboutToCheckpoint(): copy variables from the saved table back to overwrite the local variables.

A minimal example is provided in testCustomCheckpointing.lua.

5.3.6. Extending Lua Scripts Using Modules and Custom Libraries

This section describes how to create modules of Lua functions and how to create custom libraries.

Creating Modules

It can be convenient to write additional functions in modules that can be loaded and used by any script. The following is an example of a module file:

```
-- local table = require("table") -- example of importing an external
-- variable into a script. You must retrieve external variables as local
-- variables before the module declaration.
module("MyModule1") -- this must match the module filename
                    -- for example, MyModule1.lua

local myHiddenVariable = {
    minDistance = 40.0,
    otherProperty = "someValue"
}
```

The following is a sample public function that uses the local variable

```
function getValue(key)
    return myHiddenVariable[key]
end
```

The following is the corresponding B-HAVE Lua script:

```
local myModule = require("MyModule1")

function Think()
    local propertyName = "otherProperty"
    if myModule.getValue(propertyName) == "blue" then
        -- do something...
    end
end
```

To use a module, you must set the LUA_PATH environment variable, as follows:

```
?.lua;%MAK_VRFORCESDIR%\
data\parameters\BHAVE\scripts\modules\?.lua;
%LUA_DIR%\lua\?.lua;
```

The B-HAVE Module includes two sample modules, a “pseudo-periodic activation” module and a simple finite state machine module. For details, please read the comments in the module files and in the Lua scripts that use them, for example *C2_VIP.lua*. They are both used in the RescueConvoy scenario.

Building and Using Custom Libraries

You can add function libraries that can be called from Lua and interact with VR-Forces. Examples include:

- ♦ Sending task or report messages
- ♦ Interacting with a custom VR-Forces object
- ♦ Getting information about any VR-Forces object
- ♦ Creating areas or routes automatically.

You can create Lua C functions, which are special C functions in a dynamic library that can be imported and called directly from Lua.

The following is a sample function that tests if a VR-Forces object is destroyed:

```
// LUAEXT_API is the usual win32 DLL export trick for my luaExt.dll
// project
LUAEXT_API int testEntityIsDestroyed(lua_State * L)
{
    int entityDestroyed = -1; // invalid value
    // get entity vrfObject pointer that should be passed as the only
    // argument
    if (lua_gettop(L) == 1) // exactly one argument
    {
        DtVrfObject * vrfObj = (DtVrfObject * ) lua_touserdata(L, 1);
        if (vrfObj != NULL)
        {
            entityDestroyed = (vrfObj->isDestroyed() == true)? 1: 0;
        }
    }

    lua_pushboolean(L, entityDestroyed);
    return 1;
}
```

You must link your DLL to the VR-Forces libraries. You may want to create debug and release versions for each protocol (DIS, HLA13, HLA1516). The following script imports the function from the previous example:

```
testVRFFunc1 = package.loadlib("luaExt_DIS_d.dll",
    "testEntityIsDestroyed")
i = 0
function Think()
    vrfObjectPointer = AIEntityGetVrfObject()
    i = i + 1
    if i % 100 then
        objectIsDestroyed = testVRFFunc1(vrfObjectPointer)
        if objectIsDestroyed == true then
            AIUtilWarn("object destroyed")
        else
            AIUtilWarn("object NOT destroyed")
        end
    end
end
end
```

For the B-HAVE Module to find the custom DLL module, you must set the `LUA_CPATH` environment variable.

```
?.dll;%MAK_VRFORCESDIR%\
data\parameters\BHAVE\scripts\modules\?.dll;
%LUA_DIR%\lib\?.dll;
```

The `AIEntityGetVrfObject()` function is useful for retrieving a VR-Forces object pointer. This pointer can be passed to your custom functions, which can, from there, access most of the VR-Forces simulation.

5.4. Displaying B-HAVE Module Data in the 3D Front End

The B-HAVE Module plug-in outputs graphical information that can be read by the VR-Forces 3D Front End.



To display B-HAVE Module data, the 3D Front End must be generating it, or be running on the the same computer as the VR-Forces application that is generating it.

This is enabled by the following Lua functions:

- ♦ `AIUtilDrawPath()` draws the path chosen by the entity's pathfinding service.
- ♦ `AIUtilDrawLine()` and `AIUtilDrawPoint()` can be used to draw any kind of visual debugging. You can, for example, draw lines to every visible enemy entity.
- ♦ `AISetDrawPathData(name)`, where `name` is `Lifeform` or `GroundPlatform`, draws the whole `PathData` (graph + mesh). This function must be called every tick to be effective, because the drawing memory has to be reset at the beginning of every tick. If a graph is very large, the number of edges to be drawn may exceed the drawing memory size. In such cases, there will be missing edges and the drawing will become an unreliable source of information. To avoid this problem, draw `PathData` only around specific entities.

5.4.1. Drawing PathData Around Entities

You can draw `PathData` around a specific entity. This approach results in a smaller number of edges being rendered than when using `AISetDrawPathData()`, which makes it faster and more reliable since all edges are drawn.

To draw `PathData` around an entity, call `AISetDrawEntityCentric()` at every tick in the script for the entity. If several entities make this call (because they all execute the same script, for instance), only the last one to call it gets executed.

You do not have to call `AISetDrawEntityCentric()` from the script for the entity around which you want to draw `PathData`. You can call it from another entity's script in the format `AISetDrawEntityCentric(entity_name)`, where *entity_name* is the name of the entity around which you want to draw the `PathData`.

5.5. B-HAVE Module Functions

This section lists the B-HAVE Module-specific functions that you can use in Lua scripts. They are categorized as follows:

- ♦ Entity functions ([Table 5-3](#))
- ♦ Behavior functions ([Table 5-4](#))
- ♦ Entity list functions ([Table 5-5](#))
- ♦ Perception functions ([Table 5-6](#))
- ♦ Utility functions ([Table 5-7](#)).

5.5.1. Entity Functions

[Table 5-3](#) lists functions that apply to the entity that is currently evaluating the script.

Table 5-3: Entity functions (Sheet 1 of 3)

Function:	number AIEntityGetCurrentAgent()
Input parameters:	None.
Output:	The latest Agent used was: ♦ 1: Goto ♦ 2: Follow ♦ 3: Wander ♦ 4: Hide ♦ 5: Flee ♦ 6: Follow Pathway ♦ 0: No agent.
Description:	Retrieves the latest Agent used. An execution failure error (-2) is generated if the latest Agent used is unknown.
Function:	number x, number y, number z AIEntityGetDirection([string CEntity*])
Input parameters:	B-HAVE entity or name of a B-HAVE entity. If not specified, the current entity is used.
Output:	(x,y,z): The coordinates of the entity direction vector.
Description:	Gets the entity direction.
Function:	string AIEntityGetIndex([string CEntity*])
Input parameters:	B-HAVE entity or name of a B-HAVE entity. If not specified, the current entity is used.
Output:	Entity ID, if the entity is a B-HAVE entity.
Description:	Gets an entity ID.

Table 5-3: Entity functions (Sheet 2 of 3)

Function:	number AIEntityGetMaxSpeed()
Input parameters:	None.
Output:	Entity maximum speed, in meters per second.
Description:	Retrieves the entity maximum speed.
Function:	string AIEntityGetName([string CEntity*])
Input parameters:	B-HAVE entity or name of a B-HAVE entity. If not specified, the current entity is used.
Output:	Entity name, if the entity is a B-HAVE entity.
Description:	Gets the entity name.
Function:	number AIEntityGetOrderedSpeed()
Input parameters:	B-HAVE entity or name of a B-HAVE entity. If not specified, the current entity is used.
Output:	Entity ordered speed, in meters per second.
Description:	Gets the entity ordered speed.
Function:	number pitch, number yaw, number roll AIEntityGetOrientation([string CEntity*])
Input parameters:	B-HAVE entity or name of a B-HAVE entity. If not specified, the current entity is used.
Output:	Pitch, yaw, and roll: The entity Euler angles, in degrees.
Description:	Gets the entity orientation.
Function:	number x, number y, number z AIEntityGetPosition([string CEntity*])
Input parameters:	B-HAVE entity or name of a B-HAVE entity. If not specified, the current entity is used.
Output:	(x,y,z): The 3D Kynapse coordinates of the entity position.
Description:	Gets the entity position.
Function:	string AIEntityGetProfileName()
Input parameters:	Gets an entity name, pointer (CEntity*), or no argument (self).
Output:	The entity force.
Description:	Gets the entity “Profile Name”, which is Friendly, Opposing, or Neutral.
Function:	CEntity* AIEntityGetSelf()
Input parameters:	None.
Output:	Current B-HAVE entity.
Description:	Gets the B-HAVE entity executing this statement.

Table 5-3: Entity functions (Sheet 3 of 3)

Function:	number AIEntityGetSpeed([string CEntity*])
Input parameters:	B-HAVE entity or name of a B-HAVE entity. If not specified, the current entity is used.
Output:	Entity speed, in meters per second.
Description:	Gets the entity speed.
Function:	bool AIEntityIsUnderFire()
Input parameters:	None.
Output:	Returns 1 if under fire, 0 otherwise.
Description:	Finds out if the entity under fire condition is true.
Function:	AIEntitySetMaxSpeed(number)
Input parameters:	Entity maximum speed, in meters per second.
Output:	None.
Description:	Sets the entity maximum speed.
Function:	AIEntitySetRulesOfEngagement(string)
Input parameters:	The desired rules of engagement, which must be “fire-at-will”, “fire-when-fired-upon”, or “hold-fire”.
Output:	None.
Description:	Sets the entity's rules of engagement.
Function:	AIEntitySetSpeed(number)
Input parameters:	The entity's speed, in meters per second.
Output:	None.
Description:	Sets the entity's speed.
Function:	AIEntitySetSteering(number)
Input parameters:	The entity's steering, between -1 (maximum steering on the right) and 1 (maximum steering on the left).
Output:	None.
Description:	Sets the entity steering.
Function:	AIEntitySetViewTarget(string)
Input parameters:	Entity name or no value.
Output:	None.
Description:	This function is equivalent to the Set View Target set data request. If it is called without argument, it resets the entity orientation to its normal behavior (look in the direction of movement).

5.5.2. Behavior Functions

Table 5-4 describes functions that control entity behaviors.

Table 5-4: Behavior functions (Sheet 1 of 11)

Function:	AIDrivingSetCourtesy([string CEntity*], number)
Input parameters:	Optional: An entity name (a string) or a pointer to the kynapse entity. This argument must precede the courtesy level. Required: The courtesy level. Range: 0.0 through 1.0.
Output:	None.
Description:	By default, this function accesses the entity running the current Lua script. You can specify a different entity with the optional first argument. The courtesy level modifies parameters and actions in the Vehicle Dynamic Avoidance system. With a high courtesy (default 1.0), vehicles will prefer to slow down instead of overtaking other entities. A value of 0 tells the entity not to respect priorities, not to slow down, and not to avoid other vehicles.
Function:	AIDrivingGetCourtesy([string CEntity*], number)
Input parameters:	Optional: An entity name (a string) or a pointer to the kynapse entity. This argument must precede the courtesy level. Required: The courtesy level. Range: 0.0 through 1.0.
Output:	The courtesy value.
Description:	By default, this function accesses the entity running the current Lua script. You can specify a different entity with the optional first argument. The courtesy level modifies parameters and actions in the Vehicle Dynamic Avoidance system. With a high courtesy (default 1.0), vehicles will prefer to slow down instead of overtaking other entities. A value of 0 tells the entity not to respect priorities, not to slow down, and not to avoid other vehicles.
Function:	bool AIFlee()
Input parameters:	None.
Output:	Returns 1 if the entity can flee; 0 otherwise.
Description:	Makes the entity flee away from its danger points.
Function:	AIFleeAddDangerousEntity()
Input parameters:	An entity name or a B-HAVE entity pointer.
Output:	None.
Description:	Adds an entity that the entities should flee from.

Table 5-4: Behavior functions (Sheet 2 of 11)

Function:	AIFleeAddPointToFlee(number x, number y, number z)
Input parameters:	(x,y,z): The 3D coordinates of the point to flee from.
Output:	None.
Description:	Adds a point that the entity will flee away from. Generates execution failure error (-2) if the number of points to flee away from exceeds 20.
Function:	AIFleeClearDangerousEntities()
Input parameters:	None.
Output:	None.
Description:	Clears the list of entities to flee away from.
Function:	number AIFleeGetComputeDestInterval()
Input parameters:	None.
Output:	The interval, in seconds, between two destination computations.
Description:	Retrieves the interval between two destination computations.
Function:	number AIFleeGetDangerRadius()
Input parameters:	None.
Output:	A number that represents the distance to keep from danger points or entities.
Description:	Returns the distance, in meters, to keep from danger points or entities.
Function:	number x, number y, number z AIFleeGetDestination()
Input parameters:	None.
Output:	(x,y,z): The 3D Kynapse coordinates of the destination.
Description:	Retrieves the current entity destination.
Function:	number AIFleeGetPathLength()
Input parameters:	None.
Output:	The maximum number of Spatial Graph edges between the starting point and the destination.
Description:	Retrieves the maximum number of Spatial Graph edges between the starting point and the destination.
Function:	number AIFleeGetSpeed()
Input parameters:	None.
Output:	Returns the speed, in meters per second.
Description:	Reports the ordered speed for a B-HAVE Flee task.

Table 5-4: Behavior functions (Sheet 3 of 11)

Function:	bool AIFleeGetStopWhenHidden()
Input parameters:	None.
Output:	Returns 1 if the entity stops as soon as it is hidden; 0 otherwise.
Description:	Identifies whether the entity should stop as soon as it is hidden, or whether it should reach its destination.
Function:	AIFleeResetPointsToFlee()
Input parameters:	None.
Output:	None.
Description:	Resets the list of points to flee away from.
 The function AIFleeClearDangerousPoints() is an alternative function for AIFleeResetPointsToFlee().	
Function:	AIFleeSetComputeDestInterval(number)
Input parameters:	The interval, in seconds, between two destination computations.
Output:	None.
Description:	Sets the interval between two destination computations. Default: 1.0 second.
Function:	AIFleeSetDangerRadius(number)
Input parameters:	The distance, in meters, to keep from danger points or entities.
Output:	None.
Description:	Sets the distance, in meters, to keep from danger points or entities.
Function:	AIFleeSetPathLength(number)
Input parameters:	The maximum number of Spatial Graph edges between the starting point and the destination.
Output:	None.
Description:	Sets the maximum number of Spatial Graph edges between the starting point and the destination. Default: 2.
Function:	AIFleeSetSpeed(number)
Input parameters:	A number.
Output:	None.
Description:	Sets the ordered speed for a B-HAVE Flee task.

Table 5-4: Behavior functions (Sheet 4 of 11)

Function:	AI_Flee_Set_Stop_When_Hidden(bool)
Input parameters:	1 if the entity should stop as soon as it is hidden; 0 otherwise.
Output:	None.
Description:	Defines whether the entity should stop as soon as it is hidden, or whether it should reach its destination.
Function:	bool AI_Follow()
Input parameters:	None.
Output:	Returns 1 if the destination is reachable; 0 otherwise.
Description:	Makes the entity follow its target.
Function:	number AI_Follow_Get_Angle_From_Entity()
Input parameters:	None.
Output:	The angle, in degrees, between the entity's destination and the followed entity's orientation.
Description:	Retrieves the angle to the followed entity.
Function:	number x, number y, number z AI_Follow_Get_Destination()
Input parameters:	None.
Output:	(x,y,z): The 3D Kynapse coordinates of the destination.
Description:	Retrieves the destination of the follower.
Function:	number AI_Follow_Get_Dist_From_Entity()
Input parameters:	None.
Output:	The desired distance (as set by AI_Follow_Set_Dist_From_Entity()) to the followed entity.
Description:	Retrieves the desired distance (as set by AI_Follow_Set_Dist_From_Entity()) to the followed entity.
Function:	CEntity* AI_Follow_Get_Entity_To_Follow()
Input parameters:	None.
Output:	The followed entity.
Description:	Retrieves the followed entity.
Function:	number AI_Follow_Get_Speed()
Input parameters:	None.
Output:	Returns the speed of the entity, in meters per second.
Description:	Reports the ordered speed for a B-HAVE Follow task.

Table 5-4: Behavior functions (Sheet 5 of 11)

Function:	bool AIFollowHasArrived()
Input parameters:	None.
Output:	Returns 1 if the entity has arrived; 0 otherwise.
Description:	Identifies whether the destination has been reached.
Function:	AIFollowSetAngleFromEntity(number)
Input parameters:	The angle, in degrees, between the entity's destination and the followed entity's orientation.
Output:	None.
Description:	Sets the angle, in degrees, to the followed entity. Default: 180.0 degrees (that is the follower is behind its target).
Function:	AIFollowSetDistFromEntity(number)
Input parameters:	The distance to the followed entity.
Output:	None.
Description:	Sets the distance, in meters, to the followed entity. Default: 1.0 meter.
Function:	AIFollowSetEntityToFollow(CEntity* string)
Input parameters:	B-HAVE entity, or the name of the B-HAVE entity to follow.
Output:	None.
Description:	Sets the entity to follow.
Function:	AIFollowSetSpeed(number)
Input parameters:	A number.
Output:	None.
Description:	Sets the ordered speed for a B-HAVE Follow task.
Function:	bool AIGoto()
Input parameters:	None.
Output:	Returns 1 if the destination is reachable; 0 otherwise.
Description:	Moves the entity to the destination.
Function:	number x, number y, number z AIGotoGetDestination()
Input parameters:	None.
Output:	(x,y,z): The 3D Kynapse coordinates of the destination.
Description:	Retrieves the destination for the Goto Agent that will be used by AIGoto().

Table 5-4: Behavior functions (Sheet 6 of 11)

Function:	number <code>AIgotoGetDistanceLeft()</code>
Input parameters:	None.
Output:	The distance to the destination.
Description:	Retrieves the distance to the destination considering the entity path.
Function:	number <code>AIgotoGetSpeed()</code>
Input parameters:	None.
Output:	Returns a number.
Description:	Reports the ordered speed for a B-HAVE Move To task.
Function:	number x, number y, number z <code>AIgotoGetVRFDestination()</code>
Input parameters:	None.
Output:	(x,y,z): The 3D VR-Forces coordinates of the destination.
Description:	Retrieves the destination for the Goto Agent that will be used by <code>AIgoto()</code> .
Function:	bool <code>AIgotoHasArrived()</code>
Input parameters:	None.
Output:	Returns 1 if the entity has arrived; 0 otherwise.
Description:	Identifies whether the destination has been reached.
Function:	<code>AIgotoSetDestination(number x, number y, number z)</code>
Input parameters:	(x,y,z): The 3D Kynapse coordinates of the destination.
Output:	None.
Description:	Sets the destination for the Goto Agent that will be used by <code>AIgoto()</code> .
Function:	<code>AIgotoSetSpeed(number)</code>
Input parameters:	The speed, in meters per second, at which the entity should move.
Output:	None.
Description:	Sets the ordered speed for a B-HAVE Move To task.
Function:	<code>AIgotoSetVRFDestination(number x, number y, number z)</code>
Input parameters:	(x,y,z): The 3D VR-Forces coordinates of the destination.
Output:	None.
Description:	Sets the destination for the Goto Agent that will be used by <code>AIgoto()</code> .

Table 5-4: Behavior functions (Sheet 7 of 11)

Function:	AlGotoSetVRFWaypoint(string)
Input parameters:	The VR-Forces waypoint name.
Output:	Returns 1 if successful, 0 otherwise.
Description:	Sets the waypoint destination for the Goto Agent that will be used by AlGoto().
Function:	bool AlHide()
Input parameters:	None.
Output:	Returns 1 if the entity can hide; 0 otherwise.
Description:	Makes the entity hide from its dangerous entities.
Function:	AlHideAddDangerousEntity(string CEntity*)
Input parameters:	Kynapse entity, or name of the B-HAVE entity to hide from
Output:	None.
Description:	Adds an entity from which the entity must hide. Generates execution failure error (-2) if number of dangerous entities exceeds 20.
Function:	number x, number y, number z AlHideGetDestination()
Input parameters:	None.
Output:	(x,y,z): The 3D Kynapse coordinates of the hiding place.
Description:	Identifies the entity destination.
Function:	string AlHideGetGraph()
Input parameters:	None.
Output:	Name of the Spatial Graph whose vertices are possible hiding places.
Description:	Identifies the Spatial Graph of possible hiding places.
Function:	number AlHideGetSpeed()
Input parameters:	None.
Output:	Returns a number.
Description:	Reports the ordered speed for a B-HAVE Hide task.
Function:	bool AlHideHasArrived()
Input parameters:	None.
Output:	Returns 1 if the entity has reached the hiding place; 0 otherwise.
Description:	Identifies whether the entity has reached its hiding place.

Table 5-4: Behavior functions (Sheet 8 of 11)

Function:	bool AIHidelsHidden()
Input parameters:	None.
Output:	Returns 1 if the entity is hidden from dangerous entities; 0 otherwise.
Description:	Identifies whether the entity is hidden from its dangerous entities.
Function:	AIHideResetDangerousEntities()
Input parameters:	None.
Output:	None.
Description:	Resets the list of dangerous entities for the current entity.
Function:	AIHideSetGraph(string)
Input parameters:	Name of the Spatial Graph whose vertices are possible hiding places.
Output:	None.
Description:	Sets the Spatial Graph of possible hiding places.
Function:	AIHideSetSpeed(number)
Input parameters:	The speed in meters per second.
Output:	None.
Description:	Sets the at which an entity can run while executing a B-HAVE Hide task.
Function:	number AIPathfindingGetPathStatus()
Input parameters:	None.
Output:	A number describing the path status for the current entity, as follows: ♦ 1: Path in construction ♦ 2: Path found ♦ 3: Path not found.
Description:	Returns the current status of the pathfinder computation.
Function:	string AIPathfindingGetPathStatusString()
Input parameters:	None.
Output:	A string describing the path status for the current entity. One of PATH_IN_CONSTRUCTION, PATH_FOUND, or PATH_NOT_FOUND.
Description:	Returns the current status of the pathfinder computation.

Table 5-4: Behavior functions (Sheet 9 of 11)

Function:	bool AIPathway()
Input parameters:	None.
Output:	Returns 1 if the pathway can be followed; 0 otherwise.
Description:	Makes the entity follow the pathway. AIPathway has to be called at each frame (in the think() function) and requires initialization using AIPathwaySetPathway.
Function:	number AIPathwayGetCurrentPointIndex()
Input parameters:	None.
Output:	Index, in the pathway, of the next destination point.
Description:	Retrieves the index of the next pathway destination point.
Function:	number x, number y, number z AIPathwayGetDestination()
Input parameters:	None.
Output:	(x,y,z): The 3D coordinates of the current destination.
Description:	Retrieves the current destination.
Function:	number AIPathwayGetNbPoints()
Input parameters:	None.
Output:	Number of points on the pathway.
Description:	Retrieves the number of pathway points.
Function:	string AIPathwayGetPathway()
Input parameters:	None.
Output:	Name of the pathway to follow.
Description:	Retrieves the pathway to follow.
Function:	bool AIPathwayGetReverseOrder()
Input parameters:	None.
Output:	1 if the pathway is followed in reverse order; 0 otherwise.
Description:	Identifies whether the pathway is followed in normal or reverse order.
Function:	bool AIPathwayIsCurrentPointReached()
Input parameters:	None.
Output:	1 if the current destination point has been reached (the entity is executing the action related to it); 0 otherwise.
Description:	Identifies whether the current destination has been reached.

Table 5-4: Behavior functions (Sheet 10 of 11)

Function:	bool AIPathwaysIsPathwayFinished()
Input parameters:	None.
Output:	1 if the pathway has been completed; 0 otherwise.
Description:	Identifies whether the pathway has been completed.
Function:	AIPathwayResetPathway()
Input parameters:	None.
Output:	None.
Description:	Makes the entity restart the pathway from the beginning.
Function:	AIPathwaySetReverseOrder(bool)
Input parameters:	1 if the entity follows the pathway in reverse order; 0 otherwise.
Output:	None.
Description:	Makes the entity follow its pathway in normal or reverse order.
Function:	AIPathwaySetVRFRRoute(string)
Input parameters:	Name of the VR-Forces route object to follow.
Output:	Returns 1 if successful, 0 otherwise.
Description:	Sets the VR-Forces route to follow.
Function:	bool AISetMovementMode(string)
Input parameters:	Input string can be free, prefer-roads, or avoid-roads.
Output:	Returns 1 if successful, 0 if unsuccessful.
Description:	Sets the pathfinding mode with respect to PathData's preferred seed graph information.
Function:	AISetMovementModePenaltyFactor(number)
Input parameters:	A number, generally in the 1 - 100 range.
Output:	None.
Description:	The input number is the cost factor for paths that should be avoided. For example, in Prefer Roads mode, non-road terrain should be avoided, so an off-road path costs <penalty factor> times more than a road path. The same applies for edges that cross roads, when in Avoid Roads mode. The default values are 20.0 for Prefer Roads and non-road edges, and 4.0 for Avoid Roads and edges that cross the road.
Function:	AIVRFRadioSendMessage(string, string)
Input parameters:	Text string, and optionally, target entity.
Output:	None.
Description:	Sends the text string as a radio message.

Table 5-4: Behavior functions (Sheet 11 of 11)

Function:	AIVRFRadioMessageReceivedCallback(string)
Input parameters:	Text of message received.
Output:	None.
Description:	A callback to be issued when a radio message is received.
Function:	bool AIWander()
Input parameters:	None.
Output:	Returns 1 if the entity can wander; 0 otherwise.
Description:	Makes the entity wander.
Function:	number x, number y, number z AIWanderGetDestination()
Input parameters:	None.
Output:	(x,y,z): the 3D Kynapse coordinates of the destination.
Description:	Retrieves the destination of the wandering entity.
Function:	number AIWanderGetSpeed()
Input parameters:	None.
Output:	Returns a number.
Description:	Reports the ordered speed, in meters per second, for a B-HAVE Wander task.
Function:	AIWanderSelectRandomDestination()
Input parameters:	None.
Output:	None.
Description:	Instructs the entity to select a new destination for the Wander task.
Function:	AIWanderSetSpeed(number)
Input parameters:	The speed, in meters per second, to wander at.
Output:	None.
Description:	Sets the ordered speed for a B-HAVE Wander task.
Function:	AIWanderSetVRFArea(string)
Input parameters:	The name of the area in which to wander.
Output:	None.
Description:	Specifies the area that the entity should wander in.

5.5.3. Entity List Functions

Table 5-5 describes functions that you can use to manipulate entity lists.

Table 5-5: Entity list functions

Function:	CEntity* AIWorldGetEntityByIndex(number)
Input parameters:	Index of an entity.
Output:	The entity whose index is the number given in the argument, if any. Otherwise returns nil.
Description:	Retrieves an entity from its index.
Function:	CEntity* AIWorldGetEntityByName(string)
Input parameters:	Name of an entity.
Output:	The entity whose name is given in the argument, if any (not case-sensitive). Otherwise returns nil.
Description:	Retrieves an entity from its name.
Function:	number AIWorldGetEntityCount()
Input parameters:	None.
Output:	The total number of entities in the AI World.
Description:	Retrieves the total number of entities in the AI World.
Function:	CEntity*, iterator AIWorldGetFirstEntity()
Input parameters:	None.
Output:	The first entity of the AIWorld, and an iterator to pass to AIWorldGetNextEntity().
Description:	Begins an entity list traversal.
Function:	CEntity*, iterator AIWorldGetNextEntity()
Input parameters:	None.
Output:	The next entity of the AIWorld, and an iterator to pass to AIWorldGetNextEntity().
Description:	Continues an entity list traversal. This function should always be called after AIWorldGetFirstEntity has been called in the same tick.

5.5.4. Perception Functions

Table 5-6 describes functions that you can use to find out what an entity perceives about its environment.

Table 5-6: Perception functions (Sheet 1 of 2)

Function:	number AIPerceptionGetDistance(CEntity* string)
Input parameters:	B-HAVE entity, or the name of a B-HAVE entity.
Output:	The distance, in meters, to the B-HAVE entity.
Description:	Retrieves the distance from the entity to another entity.
Function:	CEntity* AIPerceptionGetNearestEnemy()
Input parameters:	None.
Output:	The nearest B-HAVE enemy, if any. Returns nil if there is none.
Description:	Retrieves the nearest entity enemy.
Function:	CEntity* AIPerceptionGetNearestEntity()
Input parameters:	None.
Output:	The nearest seen B-HAVE entity, if any. Returns nil if there is none.
Description:	Retrieves the nearest entity.
Function:	CEntity* AIPerceptionInfoGetNearestVisibleEnemy()
Input parameters:	None.
Output:	The nearest B-HAVE enemy, that is not behind an obstacle, if any. Returns nil if there is none.
Description:	Retrieves the nearest enemy that is not behind an obstacle.
Function:	CEntity* AIPerceptionInfoGetNearestVisibleEntity()
Input parameters:	None.
Output:	The nearest B-HAVE entity that is not behind an obstacle, if any. Returns nil if there is none.
Description:	Retrieves the nearest entity that is not behind an obstacle.
Function:	CEntity* AIPerceptionInfoGetNearestSeenEnemy()
Input parameters:	None.
Output:	The nearest visible Kynapse enemy, if any. Returns nil if there is none.
Description:	Retrieves the nearest seen enemy.

Table 5-6: Perception functions (Sheet 2 of 2)

Function:	CEntity* AIPerceptionGetNearestSeenEntity()
Input parameters:	None.
Output:	The nearest seen B-HAVE entity, if any. Returns nil if there is none.
Description:	Retrieves the nearest visible entity.
Function:	bool AIPerceptionIsEnemy(CEntity* string)
Input parameters:	B-HAVE entity, or the name of a B-HAVE entity.
Output:	Returns 1 if the entity is an enemy; 0 otherwise.
Description:	Retrieves whether an entity is an enemy.
Function:	bool AIPerceptionIsSeen(CEntity* string)
Input parameters:	B-HAVE entity, or the name of a B-HAVE entity
Output:	Returns 1 if an entity is seen by the entity; 0 otherwise.
Description:	Retrieves whether an entity is seen by the entity, considering its visual acuteness.
Function:	bool AIPerceptionIsVisible(CEntity* string)
Input parameters:	B-HAVE entity, or the name of a B-HAVE entity.
Output:	Returns 0 if the entity is behind an obstacle; 1 if the entity is visible.
Description:	Retrieves whether an entity is hidden from the entity by an obstacle.

5.5.5. Utility Functions

Table 5-7 describes utility functions.

Table 5-7: Utility functions (Sheet 1 of 4)

Function:	AISetDrawPathData(Lifeform GroundPlatform)
Input parameters:	Name is Lifeform or GroundPlatform.
Output:	None.
Description:	Specifies that PathData be drawn in the MÄK Stealth for the specified entity type.
Function:	AISetDrawEntityCentric(string)
Input parameters:	If no parameter, draw PathData for the entity for whom the script is being executed. If entity_name is specified, draw PathData for the specified entity.
Output:	None.
Description:	Draw PathData for the entity specified (implicitly or explicitly).
Function:	number x, number y, number z AIUtilBHAVE2VRF(number x, number y, number z)
Input parameters:	B-HAVE coordinates.
Output:	VR-Forces coordinates.
Description:	Transforms B-HAVE coordinates to VR-Forces coordinates.
Function:	bool AIUtilsColliding(number x1, number y1, number z1, number x2, number y2, number z2, number mask)
Input parameters:	(x1,y1,z1): The absolute coordinates of a ray-tracing start point. (x2,y2,z2): The absolute coordinates of a ray-tracing end point. mask: The bit mask that indicates the nature of the collision test to perform, where 1 is a “can see” query, the standard VR-Forces line-of-sight test, and 2 is a “can go” query, and is about movement capability using the AIMesh.
Output:	Returns 1 if (x2,y2,z2) is visible from (x1,y1,z1); 0 otherwise.
Description:	Tests visibility between two points.
Function:	AIUtilDebug(string)
Input parameters:	Message to be written.
Output:	None.
Description:	Writes a message to the B-HAVE output when the notification level is set to Debug.

Table 5-7: Utility functions (Sheet 2 of 4)

Function:	AIUtilDrawLine(number x1, number y1, number z1, number x2, number y2, number z2, number red, number green, number blue)
Input parameters:	(x1,y1,z1): The absolute coordinates of the line's start point. (x2,y2,z2): The absolute coordinates of the line's end point. red, green, blue: The line color, between 0 and 255.
Output:	None.
Description:	Draws a line.
Function:	AIUtilDrawPath()
Input parameters:	None.
Output:	None.
Description:	Draws the entity path.
Function:	AIUtilDrawPathway(string)
Input parameters:	Name of pathway to draw.
Output:	None.
Description:	Draws a pathway.
Function:	AIUtilDrawPoint(number x, number y, number z, number red, number green, number blue)
Input parameters:	(x,y,z): The absolute coordinates of the point. red, green, blue: The point color, between 0 and 255.
Output:	None.
Description:	Draws a point.
Function:	number AIUtilGetCurrentTime()
Input parameters:	None.
Output:	The elapsed time, in seconds, since the beginning of the application.
Description:	Gets the current time.
Function:	number AIUtilGetLastError()
Input parameters:	None.
Output:	The latest error generated was: ♦ 0: no error ♦ -1: bad argument, or too few arguments ♦ -2: execution failure ♦ -3: bad entity initialization ♦ -4: Lua cannot retrieve the CLuaManager Service.
Description:	Gets the latest error generated.

Table 5-7: Utility functions (Sheet 3 of 4)

Function:	number AIUtilGetVRFEerciseStartTime()
Input parameters:	None.
Output:	Number.
Description:	Returns the start time for the current exercise.
Function:	number AIUtilGetVRFEerciseTime()
Input parameters:	None.
Output:	The exercise time, in seconds.
Description:	Returns the current VR-Forces exercise time.
Function:	number x, number y, number z AIUtilGetVRFObjectOrientation(string)
Input parameters:	The name of a VR-Forces entity.
Output:	The normalized orientation vector, in Kynapse coordinates.
Description:	Gives the orientation vector of an entity (the direction it is facing, not the movement direction).
Function:	number x, number y, number z AIUtilGetVRFObjectPosition(string)
Input parameters:	The name of a VR-Forces entity.
Output:	The position vector, in Kynapse coordinates.
Description:	Returns the position of an entity.
Function:	number x, number y, number z AIUtilMakeVectorDeg(number pitch, number yaw, number roll)
Input parameters:	Euler angles of a vector.
Output:	(x,y,z): The coordinates of the unit vector whose Euler angles are given in the argument.
Description:	Translates an angle vector into a vector.
Function:	AIUtilResetLastError()
Input parameters:	None.
Output:	None.
Description:	Resets the latest error generated.

Table 5-7: Utility functions (Sheet 4 of 4)

Function:	number fraction, number x, number y, number z, number Nx, number Ny, number Nz AIUtilGetNearestCollisionInfo(number x1, number y1, number z1, number x2, number y2, number z2, number mask)
Input parameters:	(x1,y1,z1): The absolute coordinates of a ray-tracing start point. (x2,y2,z2): The absolute coordinates of a ray-tracing end point. mask: The bit mask that indicates the nature of the collision test to perform, where 1 is a “can see” query, the standard VR-Forces line-of-sight test, and 2 is a “can go” query, and is about movement capability using the AIMesh.
Output:	fraction: The distance from the ray-tracing start point to the first obstacle identified, divided by the distance between the ray-tracing start and end points. (x,y,z): The absolute coordinates of the point where the first obstacle was reached. (Nx,Ny,Nz): The normal of the mesh where the collision occurred.
Description:	Makes a ray cast.
Function:	number pitch, number yaw, number roll AIUtilVecToAnglesDeg(number x, number y, number z)
Input parameters:	(x,y,z): The coordinates of a unit vector.
Output:	Euler angles of the vector given in the argument.
Description:	Translates a vector into an angle vector.
Function:	number x, number y, number z AIUtilVRF2BHAVE(number x, number y, number z)
Input parameters:	VR-Forces coordinates.
Output:	B-HAVE coordinates.
Description:	Transforms VR-Forces coordinates to B-HAVE coordinates.

5.5.6. VR-Forces Task and Set Data Request Functions

Table 5-8 lists Lua functions that allow you to generate a VR-Forces task in a Lua script. There are no functions for tasks that are not appropriate for ground-based entities supported by the B-HAVE Module. For example, there is no function for the Fixed-Wing Land task. Table 5-9 lists Lua functions that allow you to generate a VR-Forces set data request in a Lua script. The functions take the same inputs that are required when issuing a task or set data request in the GUI.

Table 5-8: Task functions (Sheet 1 of 6)

Function:	VRFTaskEmbarkOn(string target)
Input parameters:	The name of the entity to embark on.
Output:	None.
Description:	Tasks the entity to embark onto the specified target entity.
Function:	VRFTaskDisembark()
Input parameters:	None.
Output:	None.
Description:	Tasks the entity to disembark from its parent entity.
Function:	VRFTaskDisembarkAll()
Input parameters:	None.
Output:	None.
Description:	Tasks the entity to disembark all units currently embarked on it.
Function:	VRFTaskFireCruiseMissile(string target, string route, number proximity, number speed)
Input parameters:	♦ target: name of the target to fire at. ♦ route: name of the route for the missile to follow on the way to the target. Leave blank ("") for no route. ♦ proximity: distance (meters) from the target object within which to detonate the missile. ♦ speed: missile speed.
Output:	None.
Description:	Tasks the entity to fire a cruise missile to the specified target.
Function:	VRFTaskFireForEffectOnEntity(string target, number rounds, number height)
Input parameters:	♦ target: name of target entity ♦ rounds: number of rounds to fire ♦ height: height above the terrain to detonate.
Output:	None.
Description:	Tasks the entity to fire a specific number of rounds on the target entity.

Table 5-8: Task functions (Sheet 2 of 6)

Function:	VRFTaskFireForEffectOnLocation(number x, number y, number z, number rounds, number height)
Input parameters:	♦ (x,y,z): target location in local coordinates ♦ rounds: number of rounds to fire ♦ height: height above the terrain to detonate.
Output:	None.
Description:	Tasks the entity to fire a specific number of rounds on a specific location.
Function:	VRFTaskFireForEffectOnTarget(string target, number rounds, number height)
Input parameters:	♦ target: name of target waypoint ♦ rounds: number of rounds to fire ♦ height: height above the terrain to detonate.
Output:	None.
Description:	Tasks the entity to fire a specific number of rounds on a target.
Function:	VRFTaskFollowEntity(string entity, number behind, number right)
Input parameters:	♦ entity: name of entity to follow ♦ behind: behind offset (meters) ♦ right: right offset (meters).
Output:	None.
Description:	Tasks the entity to follow another entity.
Function:	VRFTaskInterceptAndDestroy(string target, number stand-OffDistance)
Input parameters:	♦ target: name of target entity ♦ standOffDistance: distance from target entity that attacking entity will maintain.
Output:	None.
Description:	Tasks the entity to move towards and attempt to destroy target entity.
Function:	VRFTaskMoveAlongRoute(string routeName, [bool reverse, bool startAtClosestPoint])
Input parameters:	♦ reverse: set to true to traverse the route in reverse order (optional) ♦ startAtClosestPoint: set to true to have the entity reach the route by moving to the closest point from its current position, rather than to the beginning or ending points in the route (optional).
Output:	None.
Description:	Tasks the entity to move along a route.

Table 5-8: Task functions (Sheet 3 of 6)

Function:	VRFTaskMoveToLocation(number x, number y, number z)
Input parameters:	Destination to move the entity to, in local, database coordinates.
Output:	None.
Description:	Tasks the entity to move to a location (x,y,z).
Function:	VRFTaskMoveToWaypoint(string wpName)
Input parameters:	Name of waypoint.
Output:	None.
Description:	Tasks the entity to move to a waypoint.
Function:	VRFTaskPatrolBetween(string wp1, string wp2)
Input parameters:	Names of waypoints to patrol.
Output:	None.
Description:	Tasks the entity to patrol between two waypoints.
Function:	VRFTaskPatrolAlongRoute(string routeName)
Input parameters:	Name of the route to patrol.
Output:	None.
Description:	Tasks the entity to patrol along a route.

Table 5-8: Task functions (Sheet 4 of 6)

Function:	VRFTaskSendRadioSet(string setName, string recipient, {set arguments})
Input parameters:	♦ setName: name of the set data request to send to entity. ♦ Recipient: name of the entity receiving the set data request. ♦ {set arguments}: set of arguments associated with the set data request. For a description of the arguments required for each command, please see its matching Lua function. The possible values for setName are: ♦ Destroyed ♦ Destroyed ♦ Disembarked ♦ Embarked ♦ FireNow ♦ Heading ♦ Location ♦ OrderedSpeed ♦ Posture ♦ Restore ♦ SectorOfResponsibility ♦ SpotReport ♦ Target ♦ WeaponState.
Output:	None.
Description:	Sends a specific set data request to the recipient entity.

Table 5-8: Task functions (Sheet 5 of 6)

Function:	VRFTaskSendRadioTask(string taskName, string recipient, {task arguments})
Input parameters:	taskName: name of the task to send to entity. Recipient: name of entity receiving the task. {task arguments}: set of arguments associated with the task. For a description of the arguments required for each command, please see its matching Lua function. Possible values for taskName are: ♦ Disembark ♦ DisembarkAll ♦ FireCruiseMissile ♦ FireForEffectOnEntity ♦ FireForEffectOnLocation ♦ FireForEffectOnTarget ♦ FollowEntity ♦ InterceptAndDestroy ♦ MoveAlongRoute ♦ MoveToLocation ♦ MoveToWaypoint ♦ PatrolBetween ♦ PatrolAlongRoute ♦ SendTextMessage ♦ TurnToHeading ♦ UserTask ♦ Wait ♦ WaitDuration ♦ WaitElapsed
Output:	None.
Description:	Sends a task to the recipient entity.
Function:	VRFTaskSendTextMessage(string entity, string text)
Input parameters:	♦ entity: name of recipient entity ♦ text: text message.
Output:	None.
Description:	Tasks the entity to send a text message to a target entity.
Function:	VRFTaskTurnToHeading(number heading)
Input parameters:	Heading, in degrees.
Output:	None.
Description:	Tasks the entity to turn to a heading.

Table 5-8: Task functions (Sheet 6 of 6)

Function:	VRFTaskUserTask(string taskName, string arg1, string arg2, string arg3, string arg4)
Input parameters:	The name of a user-defined task and any required arguments.
Output:	None.
Description:	Tasks the entity with a user-defined task.
Function:	VRFTaskWait()
Input parameters:	None.
Output:	None.
Description:	Tasks the entity to wait indefinitely.
Function:	VRFTaskWaitDuration(number duration)
Input parameters:	Amount of time to wait, in seconds.
Output:	None.
Description:	Tasks the entity to wait a specific amount of time (seconds).
Function:	VRFTaskWaitElapsed(number elapsed)
Input parameters:	The elapsed time, in seconds, to which the entity should wait.
Output:	None.
Description:	Tasks the entity to wait until the specified elapsed simulation time (seconds).

Table 5-9: Set data request functions (Sheet 1 of 4)

Function:	VRFSetAltitude(number altitude)
Input parameters:	Altitude to jump to.
Output:	None.
Description:	Sets the entity to a specific altitude.
Function:	VRFSetDestroyed()
Input parameters:	None.
Output:	None.
Description:	Sets the entity as destroyed.
Function:	VRFSetDIGuyAppearance(string appearance-Name)*
Input parameters:	DI-Guy appearance.
Output:	None.
Description:	Specifies a DI-Guy appearance for a lifeform.

Table 5-9: Set data request functions (Sheet 2 of 4)

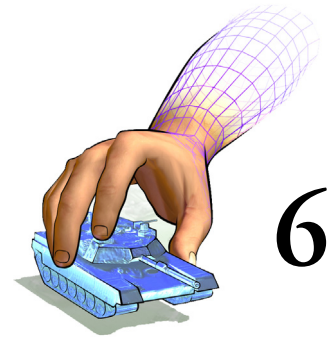
Function:	VRFSetDIGuyAnimation(string animationName, [number duration])*
Input parameters:	DI-Guy animation.
Output:	None.
Description:	Sets a DI-Guy animation for the lifeform.
Function:	VRFSetDisembarked()
Input parameters:	None.
Output:	None.
Description:	Sets the entity as disembarked.
Function:	VRFSetEmbarked(string parentName)
Input parameters:	Name of the entity on which to embark.
Output:	None.
Description:	Sets the entity to embark onto a parent entity.
Function:	VRFSetFireNow(string targetName)
Input parameters:	Entity to fire at.
Output:	None.
Description:	Sets the entity to fire at the specified target.
Function:	VRFSetHeading(number heading)
Input parameters:	Heading, in degrees.
Output:	None.
Description:	Sets the entity's heading.
Function:	VRFSetLocation(number x, number y, number z)
Input parameters:	Location, in local, database coordinates.
Output:	None.
Description:	Sets the entity's location.
Function:	VRFSetOrderedSpeed(number speed)
Input parameters:	Speed, in meters per second.
Output:	None.
Description:	Specifies the speed at which the entity is to move.

Table 5-9: Set data request functions (Sheet 3 of 4)

Function:	VRFSetPosture(string posture)
Input parameters:	Possible values for the posture are: ♦ Crawling ♦ Jumping ♦ Kneeling ♦ Parachuting ♦ Prone ♦ Running ♦ Sitting ♦ Squatting ♦ Standing ♦ Swimming ♦ Wading ♦ Walking.
Output:	None.
Description:	Sets the entity posture.
Function:	VRFSetRestore()
Input parameters:	None.
Output:	None.
Description:	Sets the entity to full health (not destroyed) and to the resource levels specified for that entity type in the object parameter database.
Function:	VRFSetSectorOfResponsibility(number scanCenter, number scanAmount)
Input parameters:	scanCenter: center, in degrees, of the sector of responsibility scanAmount: size, in degrees, of the sector of responsibility.
Output:	None.
Description:	Sets the sector of responsibility for the entity.
Function:	VRFSetSpotReports(bool flag)
Input parameters:	Specify 1 for true (on); 0 for false.
Output:	None.
Description:	Sets the entity spot reporting on or off.
Function:	VRFSetTarget(string targetName)
Input parameters:	Name of the target entity.
Output:	None.
Description:	Specifies a target for the entity.

Table 5-9: Set data request functions (Sheet 4 of 4)

Function:	VRFSetWeaponState(string state)
Input parameters:	Possible states are: ♦ Stowed ♦ Deployed ♦ In Fire Position.
Output:	None.
Description:	Sets the (lifeform) entity weapon state.



B-HAVE Messages API

This chapter explains how to use the B-HAVE Messages API to send tasks and set data requests to VR-Forces.

Introduction	6-2
Sending B-HAVE Tasks	6-2
Sending B-HAVE Set Data Requests.....	6-3
Set Script.....	6-3
Set Movement Mode	6-3
Set View Target	6-3
Requesting the Name of an Entity's Lua Script.....	6-4
Sending Interface Messages	6-5
Using the API	6-5
Building a Project	6-6

6.1. Introduction

The B-HAVE Module provides an extension to the VR-Forces Remote Control API that lets you access B-HAVE Module functionality. The API allows you to programmatically send messages to create entities and query or modify their behavior during a simulation. For example, through the API, you can assign a new B-HAVE Module task or assign a Lua script to a specific entity. The API comes with a set of routines to control the set of Lua scripts available to the simulation. For information about the VR-Forces Remote Control API, please see *VR-Forces Developers Guide*.

6.2. Sending B-HAVE Tasks

You can use the API to send the following tasks to entities:

- ♦ Move to Location
- ♦ Move to Waypoint
- ♦ Move Along Route
- ♦ Flee
- ♦ Hide
- ♦ Wander
- ♦ Follow Entity.

To assign a task, you create a *DtBhaveTask* object and specify the task name and one or more arguments. The file *bhaveTask.h* contains the definitions of the all the task names available and the task-specific arguments.

The way that you send a task to the B-HAVE Module depends on where the code is executing. If you are issuing the task through the VR-Forces front-end, either by rebuilding the front-end or building a plug-in, you would use code similar to the following example and then emit a signal. This example code tells an entity to hide from two other entities:

```
// Hide from two entities (ent1 and ent2)
DtBhaveTask task;
task.init();
task.setArg1Text(BHAVE_HIDE_TASK);
task.setArg4Text("ent1, ent2");
```

If you are sending the task from an external application, that is, one that is not part of the VR-Forces front-end, you would use the code in the previous example to specify the task and then you would have to send a *DtTaskCommand*, as in the following example:

```
DtTaskCommand cmd;
cmd.setTask(task);
cmd.setObjectName(...);
remoteController->sendTaskMsg(...);
```

6.3. Sending B-HAVE Set Data Requests

This section describes the B-HAVE remote control API set data requests. As with tasks, you can send a set data request from within the VR-Forces front-end or from an external application. If you send a set data request from the front-end, use the command syntax show in the examples in this section. If you send the set data request from an external application, specify the set data request as shown in the examples, and then send a *DtSetDataCommand*. For example, if you wanted to send the example Set Script set data request, you would do the following:

```
DtSetDataCommand cmd;  
cmd.setSetRequest(setScript);  
cmd.setObjectName(...);  
remoteController->sendSetDataMsg(...);
```

6.3.1. Set Script

Use a *DtSetBhaveScript* command to assign a Lua script to an entity. To assign the *specificScript.lua* script to an entity:

```
DtSetBhaveScript setScript;  
setScript.setScriptName("specificScript.lua");
```

6.3.2. Set Movement Mode

To set the movement mode of an entity, use a *DtSetBhaveMovementMode* command. The different movement modes are defined *setBhaveMovementMode.h*. To set the movement mode of a given entity to Prefer Roads:

```
DtSetBhaveMovementMode setMovementMode;  
setMovementMode.setMovementType(DtBhaveMovementModePreferRoads);
```

6.3.3. Set View Target

Use a *DtSetViewTarget* command to set an entity to always keep its orientation facing another entity. To set an entity to always face an entity called "M1A2 1":

```
DtSetViewTarget set;  
set.setViewTargetName("M1A2 1");
```

6.4. Requesting the Name of an Entity's Lua Script

Use a *DtAdminRequestScriptName* command to ask the simulation engine for the name of the Lua script associated with an entity. To send the command, set the *DtAdminRequestScriptName* command as content of a *DtAdminMessage*. For example, to request the Lua script associated with an entity named "M1A2 1", do the following:

```
DtAdminMessage msg;  
DtAdminRequestScriptName content;  
msg.setReceiver(...);  
msg.setAdminContent(&content);  
remoteController()->sendMessageToObject(&msg, <backend>);
```

The simulation engine returns a *DtAdminMessage* with a *DtAdminScriptName* object as its content. You must register a callback to the *DtScriptNameType* message type in the remote controller to process such message, for example:

```
remoteController()->objectMessageExecutive()->  
addMessageCallback(DtScriptNameType, processScriptNameCallback, ...);
```

In the example, `processScriptNameCallback()` is the callback function that will be invoked when a message of *DtScriptNameType* is received.

6.5. Sending Interface Messages

The API offers different kinds of messages to handle Lua scripts as well as entity creation.

The B-HAVE Module API lets you send the following messages to the simulation engine:

- ♦ To request the list of Lua scripts available, send a `DtIfRequestLuaScriptList` message.
- ♦ To request the Lua code for a file, send a `DtIfRequestLuaScriptText` message.
- ♦ To specify the Lua code for a file, send a `DtIfSetLuaScriptText` message.
- ♦ To delete a Lua script file, send a `DtIfDeleteLuaScript` message.
- ♦ To create a group of entities, send a `DtIfCreateMultipleEntities` message.
- ♦ To create a selection group, send a `DtIfCreateSelectionGroup` message.

The following example requests the list of Lua scripts available:

```
DtIfRequestLuaScriptList request;
remoteController()->vrfMessageInterface()->
    createAndDeliverMessage(simEngineAddr, request);
```

The simulation engine returns a list of Lua scripts available inside a `DtIfLuaScriptList` message. The content of a given Lua script file is sent inside a `DtIfLuaScriptText` message.

To process a `DtIfLuaScriptList` message in the GUI, you must register a callback to the `DtLuaScriptListType` message type in the remote controller.

```
remoteController()->vrfMessageInterface()->addMessageCallback(
    DtLuaScriptListType, luaScriptListCallback, ...);
```

In the example, `luaScriptListCallback` is the callback function that will be invoked when a message of `DtLuaScriptListType` is received.

6.6. Using the API

To use the API, you must first register the different kinds of messages with the remote controller. The *bhaveMsgs.h* file has the list of calls available to automatically register the various kinds of messages supported by the API.

For example, to register the B-HAVE tasks available, use:

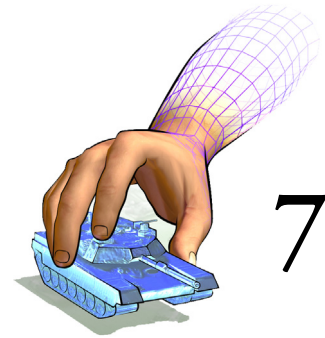
```
registerBhaveTasks(controller()->taskFactory());
```

where `controller()` returns the *DtVrfRemoteController* object associated with your application.

Once the tasks are registered in the controller, you can create *DtBhaveTask* commands and deliver them through the controller. Check the *./examples/bhaveRemoteControl* project for an example illustrating the use of the B-HAVE API inside VR-Forces.

6.7. Building a Project

The B-HAVE Module Remote Control API is based on the VR-Forces Remote Control API. Therefore, if you want to build a project that uses the B-HAVE Module Remote Control API, you must add the libraries from the VR-Forces Remote Control API and the *bhaveMsgsRT[d].lib* file to the link line of your project. The B-HAVE Module API header files are located in the *./include/bhave*. For information about how to set up a project that uses the VR-Forces Remote Control API, please see *VR-Forces Developers Guide*.



A B-HAVE Module Tutorial

This chapter is a tutorial that shows you how to add PathData to the Makland terrain database and how different settings affect the density and amount of detail in the spatial graph. It assumes that you are familiar with the TDB Tool, project files, and MTD files. For complete information about using the TDB Tool, please see *MAK Terrain Database Tool Users Guide*.

Introduction	7-3
Create a Basic Terrain Database and MTD File.....	7-3
Start the TDB Tool	7-3
Create and Save a New Project	7-4
Simple PathData Generation.....	7-5
Set Up PathData Resources	7-5
Configure Generation of PathData.....	7-10
Generate PathData	7-13
Associate PathData with Entity Types.....	7-14
Save the MTD File	7-15
View the Spatial Graph for the PathData.....	7-15
Simple PathData with Preferred Movement Enabled.....	7-17
Set Up PathData Resources	7-17
Configure Generation of PathData.....	7-20
Generate PathData	7-22
Associate PathData with Entity Types.....	7-22
Save the MTD File	7-23
Simple PathData with Local Zones	7-23
Set Up PathData Resources	7-23
Configure Generation of PathData.....	7-23
Generate PathData	7-26

Associate PathData with Entity Types	7-26
Save the MTD File.....	7-26
Generating PathData Using a Seed Graph.....	7-27
Set Up PathData Resources	7-27
Configure Generation of PathData.....	7-28
Generate PathData	7-29
Associate PathData with Entity Types	7-30
Save the MTD File.....	7-30
Examine the Spatial Graph.....	7-31
Generating PathData Using a Seed Graph and Local Zones	7-31
Conclusion	7-31

7.1. Introduction

In “[Varying the Complexity of PathData Configurations](#),” on page 3-3, we describe the following hierarchy of PathData complexity.

- ♦ Simple PathData generation
- ♦ Simple PathData generation with preferred movement enabled
- ♦ Simple PathData generation with local zones
- ♦ Multi-step PathData generation using seed graphs
- ♦ Multi-step PathData generation using seed graphs and local zones.

This tutorial shows you how to generate PathData at each of these levels of complexity. (The examples only generate pedestrian PathData. The process for generating ground vehicle PathData is identical.)



To see short video tutorials that illustrate portions of this chapter, please go to: http://www.mak.com/products/bhave_tutorials.php

7.2. Create a Basic Terrain Database and MTD File

To start the tutorial, we will generate a basic terrain database and MTD file to use as the basis for creating various PathData configurations.

7.2.1. Start the TDB Tool

- To start the TDB Tool in Windows, choose **Start** → **All Programs** → **MAK Technologies** → **VR-Forces x.x** → **Tools** → **TDB Tool**.

The TDB Tool window opens with a minimal menu and nothing loaded ([Figure 7-1](#)).

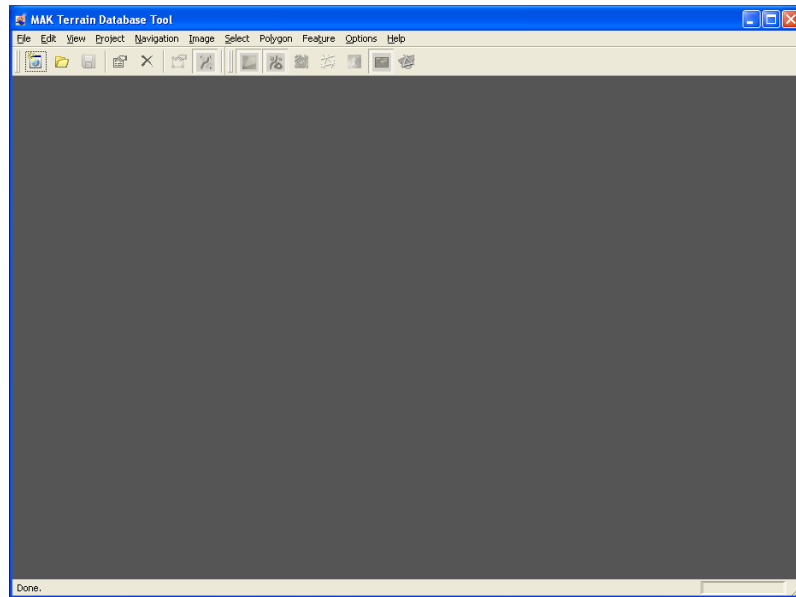


Figure 7-1. TDB Tool window at startup

7.2.2. Create and Save a New Project

The tutorial is based on the Makland terrain. However, we do not want to overwrite the installed terrain, so we will create a new terrain based on Makland and use the new terrain to experiment. To create the new terrain, we need to create a project and build the terrain. This results in a new GDB file and an MTD file. The MTD file stores information about the terrain, and eventually, information about the PathData that we want to add.

1. Choose **File** → **New** → **Terrain Project**. The Project and Navigation toolbars are added to the TDB Tool window.
2. Choose **Project** → **Add Data**. The Add Project Data dialog box opens.
3. Select *makland.gdb*.
4. Click Open. Records for the *makland.gdb* database are added to the project tab under Terrain Data.
5. Choose **Project** → **Build Terrain**. The Build Terrain dialog box opens.
6. Type a name for the project, for example, myMakland, and click Save. The TDB Tool reports the results of the build process and prompts you to open the new GDB and MTD files.
7. Click Yes. You are prompted to save the project files. Since this is a very simple project, it is not really necessary to do so.
8. Click No. The new GDB file is displayed and the MTD data is listed in the Descriptor tab (Figure 7-2).

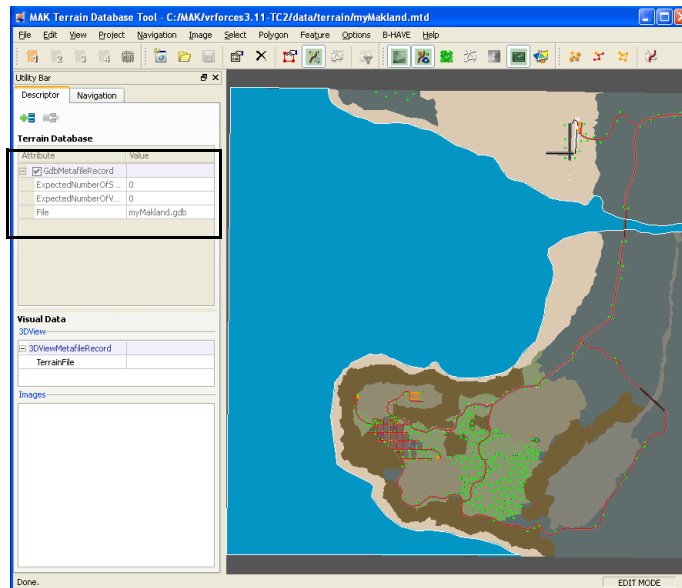


Figure 7-2. myMakland terrain

7.3. Simple PathData Generation

In this example, we generate PathData for a small part of the terrain. This is the most detailed part of the tutorial. In later examples, we assume you can repeat the detailed procedures you have learned here. The PathData you generate for this example also gets used as a seed graph in a later example.

7.3.1. Set Up PathData Resources

PathData is generated in areas of the terrain specified by bounding boxes. PathData generation starts from a point in the terrain called a seed point.

Specify a Bounding Box

A bounding box is an parallelepiped in which a PathData generation can take place. You can specify multiple bounding boxes in a terrain, but you must specify at least one.

To create a new bounding box:

1. If the TDB Tool is not already running, open it. Load the *myMakland.mtd* file.
2. Choose **B-HAVE** → **Manage PathData Resources**, or click the Step 1 icon on the B-HAVE Module toolbar (Figure 7-3). The Manage PathData Resources dialog box opens (Figure 7-4).



Figure 7-3. B-HAVE Module toolbar

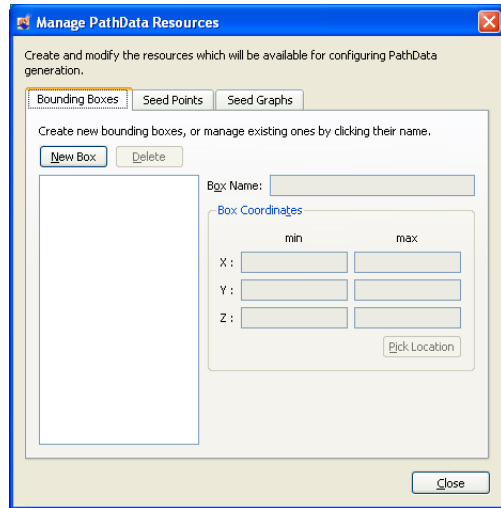


Figure 7-4. Manage PathData Resources dialog box

3. For this tutorial, we want to create PathData in and around the palace in Makland. Zoom in on the palace ([Figure 7-5](#)).

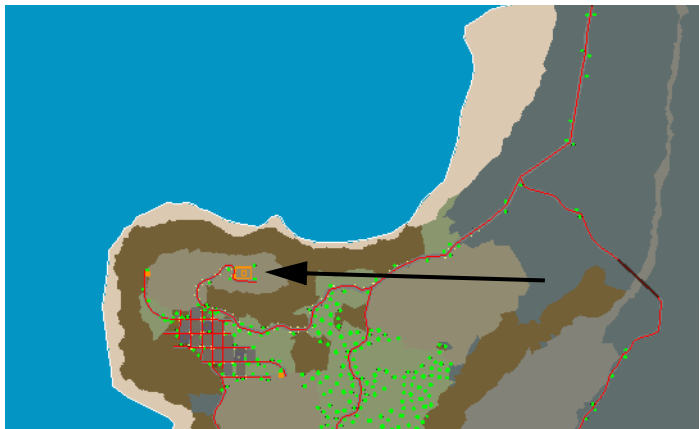


Figure 7-5. Palace

4. In the Manage PathData Resources dialog box, select the Bounding Boxes tab.
5. Click New Box.
6. In the Box Name text box, type palace. (You could also just accept the default name supplied by the B-HAVE Module. Bounding box and seed point names must be unique for the terrain.)

7. Click Pick Location.
8. Drag a rectangle around the palace similar to the white rectangle in [Figure 7-6](#). When you release the mouse button, the TDB Tool draws a white box on the terrain.

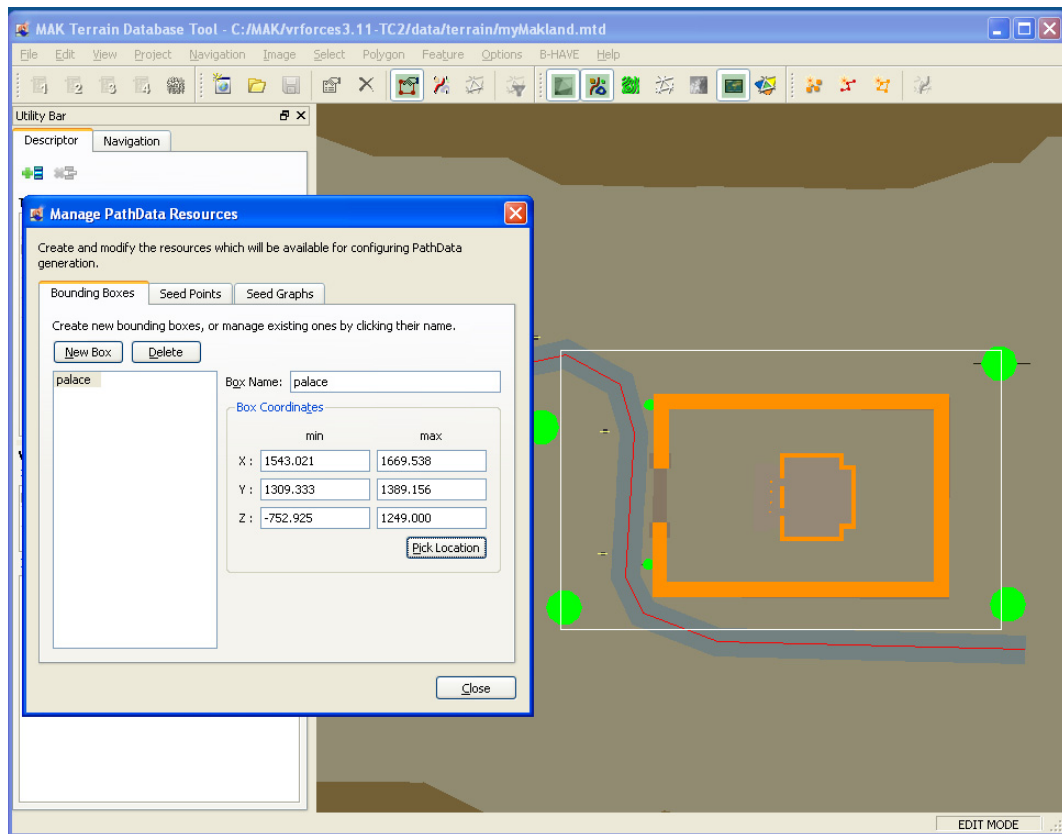


Figure 7-6. Bounding box around palace

Specify a Seed Point

A seed point is the starting point for the B-HAVE Module to find all the accessible locations in the bounding box. For more details, please see “[Specifying Seed Points](#),” on page 3-7.

1. In the Manage PathData Resources dialog box, select the Seed Points tab ([Figure 7-7](#)).

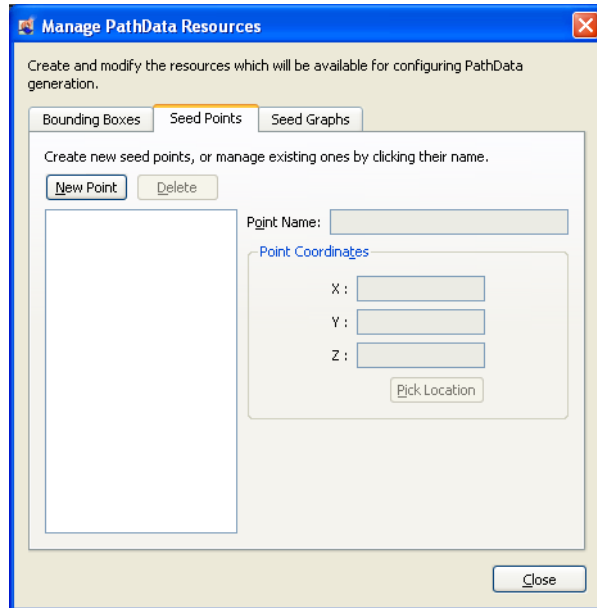


Figure 7-7. Manage PathData Resources dialog box, Seed Points tab

2. Click New Point.
3. In the Point Name text box, type courtyard.
4. Click Pick Location.
5. Click on the map in the courtyard of the palace about midway between the entrance and the building. A white dot is added to the map where you clicked (in the black circle added to [Figure 7-8](#)).

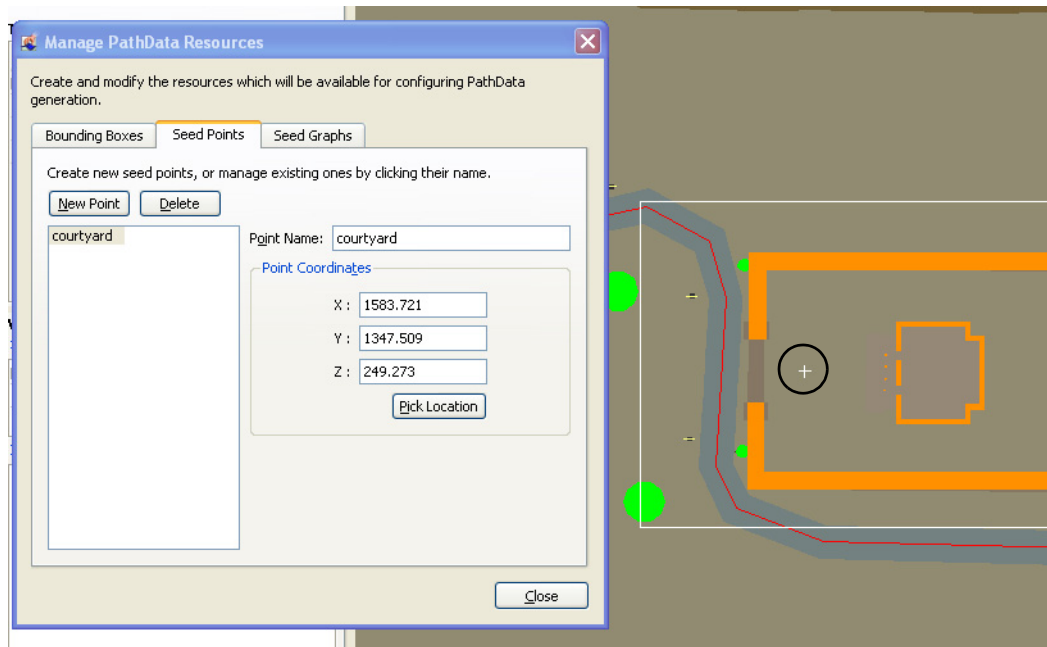


Figure 7-8. Seed point

6. Close the Manage PathData Resources dialog box.

7.3.2. Configure Generation of PathData

After you specify the terrain resources for path generation (bounding boxes, seed points, and, optionally, seed graphs), you must create a configuration for the B-HAVE Module to use when it generates PathData. The PathData configuration specifies:

- ♦ A template file - template files are provided with the B-HAVE Module. They are specific to entity type and define a level of detail and density.
- ♦ The bounding box and seed point to use
- ♦ Movement modes to enable
- ♦ Optionally, filtering criteria
- ♦ Optionally, local zones
- ♦ Optionally, seed graphs.

In this first part of the tutorial, we do not use movement modes or any of the optional settings.

The PathData generation templates are named using the format *entity-type_detail_density.xml*, where:

- ♦ Entity type is either lifeform or ground platform. It determines the entity size and a few related internal parameters.
- ♦ Detail specifies the maximum level of detail that has to be taken into account by the entities. If a terrain contains no small features or corridors, you can choose the “no detail” setting. Otherwise (in most cases), choose the “detail” setting or the “more detail” setting (for building interiors).
- ♦ Density specifies the graph density. Higher density provides higher quality, but longer pathfinding computations. For a large terrain, you should choose sparse or very sparse. For smaller areas (if you generate a seed graph, for instance) you can use the dense setting.



The templates chosen for the tutorial examples have not necessarily been chosen for optimal use in VR-Forces scenarios. They have been chosen at least partly to minimize the time spent generating PathData, and to illustrate varying detail in the resulting spatial graphs.

1. Select **B-HAVE → Configure PathData Generation**, or click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens (Figure 7-9).

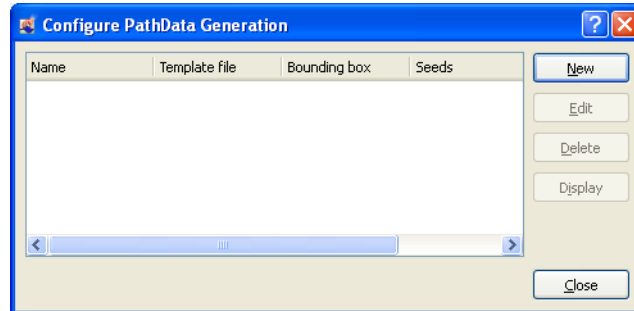


Figure 7-9. Configure PathData Generation dialog box

2. Click New. The Edit PathData Configuration dialog box opens (Figure 7-11).

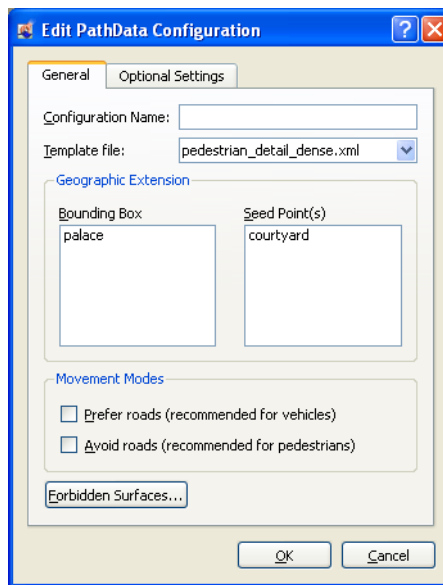


Figure 7-10. Edit PathData Configuration dialog box

3. In the Configuration Name box, type a meaningful name for this configuration, for example tutorial_pedestrian_detail_dense.
4. In the Template File drop-down list, select pedestrian_detail_dense.xml.
5. Select palace in the Bounding Box list. This is the bounding box that you created in “Specify a Bounding Box,” on page 7-5.
6. Select courtyard in the Seed Points list. This is the seed point that you created in “Specify a Seed Point,” on page 7-8 (Figure 7-11).

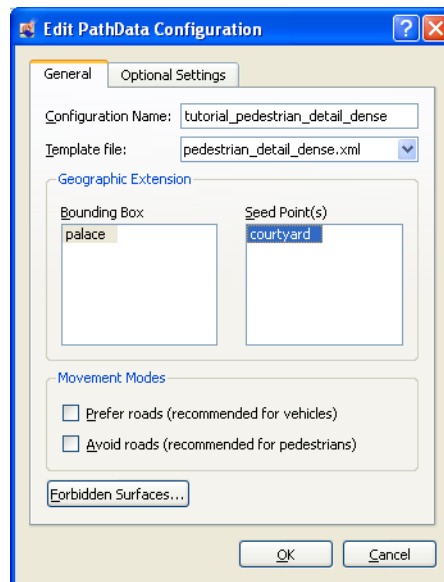


Figure 7-11. Completed Edit PathData Configuration dialog box

7. Click OK. The configuration is added to the list in the Configure PathData Generation dialog box (Figure 7-12).

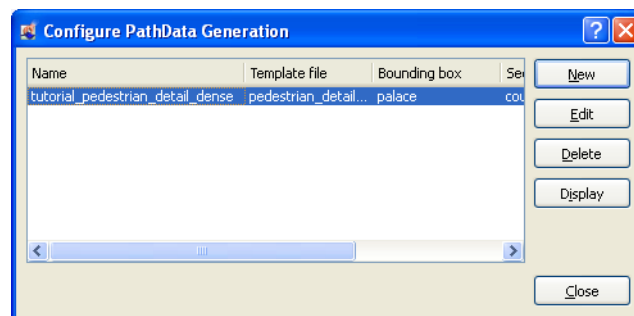


Figure 7-12. PathData configuration

8. Click Close.

7.3.3. Generate PathData

After you specify the terrain resources and create PathData configurations, you generate the PathData. The generation can last from a few minutes to a few hours to a few days depending on the terrain complexity, the level of detail, and your computer's processing power. The generation process for this example should take three to four minutes.

1. Choose **B-HAVE** → **Generate PathData**, or click the Step 3 icon on the B-HAVE Module toolbar. The Generate PathData dialog box opens (Figure 7-13). It lists the configuration that you created for this terrain database.

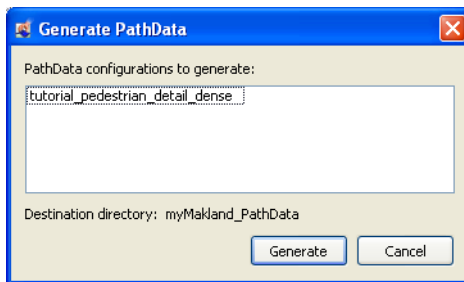


Figure 7-13. Start Batch PathData Generation dialog box

2. Select tutorial_pedestrian_detail_dense (or whatever you called your configuration).
3. Click Generate. The B-HAVE Module displays a progress dialog box (Figure 7-14). If you enable the display options, you will be able to view a colorful visual display, but it may slow down the process.

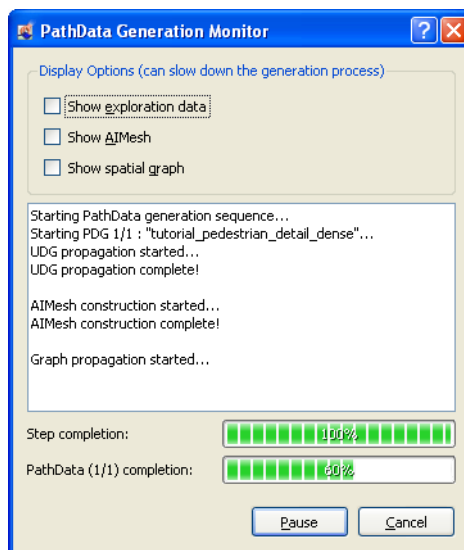


Figure 7-14. PathData generation

- When the PathData is finished being generated, the Cancel button changes to Close. Click Close.

7.3.4. Associate PathData with Entity Types

After you generate PathData, you must choose which PathData to assign to lifeforms, and which to assign to ground platforms. For this tutorial, we only generated PathData for lifeforms.

To associate PathData with an entity type:

- Choose **B-HAVE** → **Associate PathData**, or click the Step 4 icon on the B-HAVE Module toolbar. The Associate PathData with Entity Types dialog box opens.
- In the Lifeforms list, select tutorial_pedestrian_detail_dense (Figure 7-15).

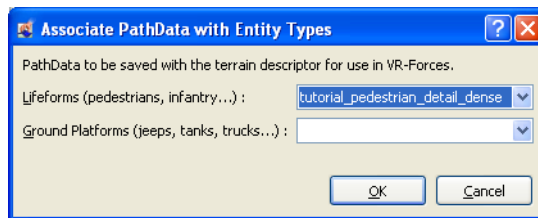


Figure 7-15. Associate PathData with Entity Types dialog box

- Click OK. The records in the Descriptor panel are updated with this information (Figure 7-16).

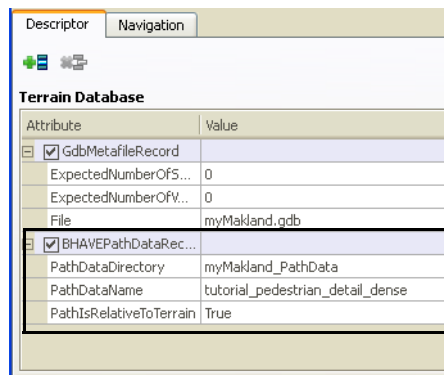


Figure 7-16. MTD file records

7.3.5. Save the MTD File

1. Choose **File** → **Save**. This saves *myMakland.mtd* with the new data. We will use this as the basis for additional examples.
2. Choose **File** → **Save As**. Save the data to a different name, such as *myMakland_simple*. This will allow you to load the terrain with the PathData you generated in this example separately from other tutorial examples.

7.3.6. View the Spatial Graph for the PathData

As mentioned in “[Configure Generation of PathData](#),” on page 7-10, your choice of template determines how dense the spatial graph of the PathData will be. You can view the results of PathData generation in the TDB Tool.



The spatial graph screen captures in this section have been edited to improve visibility.

To view the spatial graph for the tutorial:

1. If *myMakland.mtd* is not loaded in the TDB Tool, open it.
2. Select **B-HAVE** → **Configure PathData Generation**, or click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens ([Figure 7-17](#)).

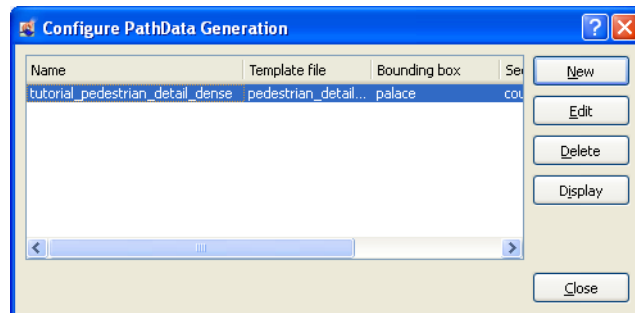


Figure 7-17. Configure PathData Generation dialog box

3. Select the *tutorial_pedestrian_detail_dense* configuration.
4. Click Display. The spatial graph is displayed in the area in which PathData was generated (the palace) ([Figure 7-18](#)).

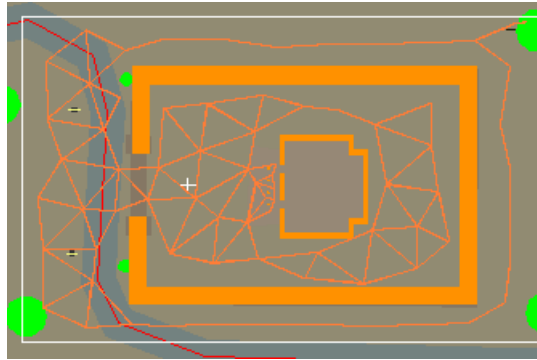


Figure 7-18. Spatial graph for tutorial_pedestrian_detail_dense configuration

5. To hide the display, close the dialog box.

Compare Spatial Graphs for Different Templates

To see the effect of using different templates, repeat the steps of the tutorial, but instead of using the *pedestrian_detail_dense.xml* template, use the *pedestrian_evenmoredetail_dense.xml* template. Then repeat the process using the *pedestrian_nodetail_sparse.xml* template. Take a look at the spatial graphs. They should look like the graphics in [Figure 7-19](#).

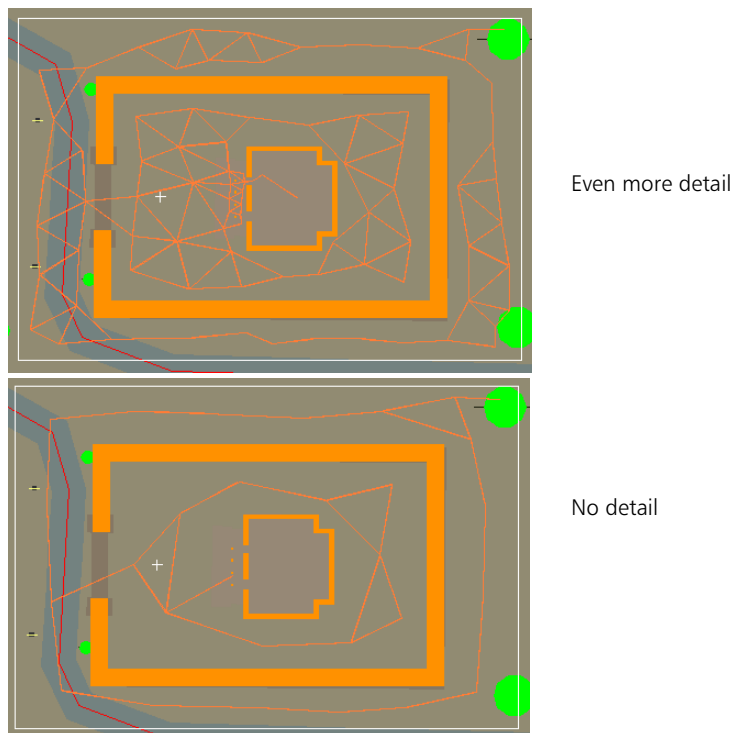


Figure 7-19. Spatial graphs with more detail and no detail

Viewing the spatial graph can help you understand the behavior of entities executing B-HAVE Module tasks. For example, if you want a DI entity to be able to walk into the building in the palace, these spatial graphs show that there is no PathData in the palace using the no detail template, so you would want to use a template with detail or even more detail.

7.4. Simple PathData with Preferred Movement Enabled

In the VR-Forces GUI, the B-HAVE Module lets you specify movement modes for B-HAVE entities. The options are Free, Prefer Roads, and Avoid Roads. To use the Prefer Roads or Avoid Roads movement modes, the PathData for the terrain database must have been configured with one or both of these movement modes enabled. This example shows how to enable the movement modes. For more information about movement modes, please see [“The Set Movement Mode Set Data Request,”](#) on page 4-12.

7.4.1. Set Up PathData Resources

We will use *myMakland.mtd* again as the base terrain descriptor.

1. If the TDB Tool is not already running, open it. Load the *myMakland.mtd* file. Note that all of the buttons on the B-HAVE Module toolbar are enabled. This is because all of the data that they require was saved when you saved the previous example.
2. Choose **B-HAVE → Manage PathData Resources**, or click the Step 1 icon on the B-HAVE Module toolbar ([Figure 7-3](#)). The Manage PathData Resources dialog box opens ([Figure 7-4](#)).
3. Zoom in on the peninsula where the town is located ([Figure 7-20](#)).

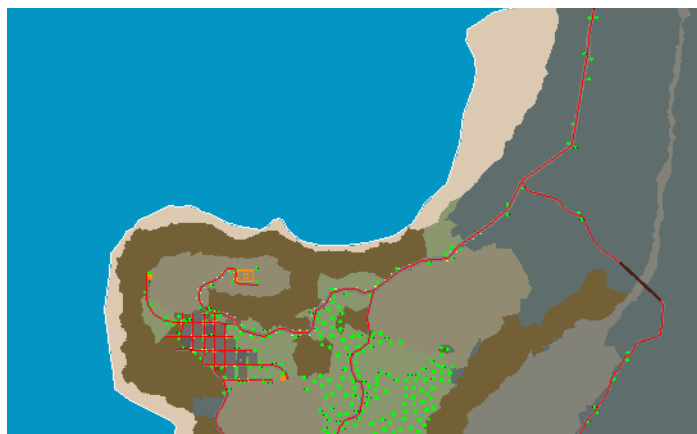


Figure 7-20. Palace

4. In the Manage PathData Resources dialog box, select the Bounding Boxes tab.

5. Click New Box.
6. In the Box Name text box, type Town.
7. Click Pick Location.
8. Drag a rectangle around the town and the palace similar to the white rectangle in [Figure 7-21](#).

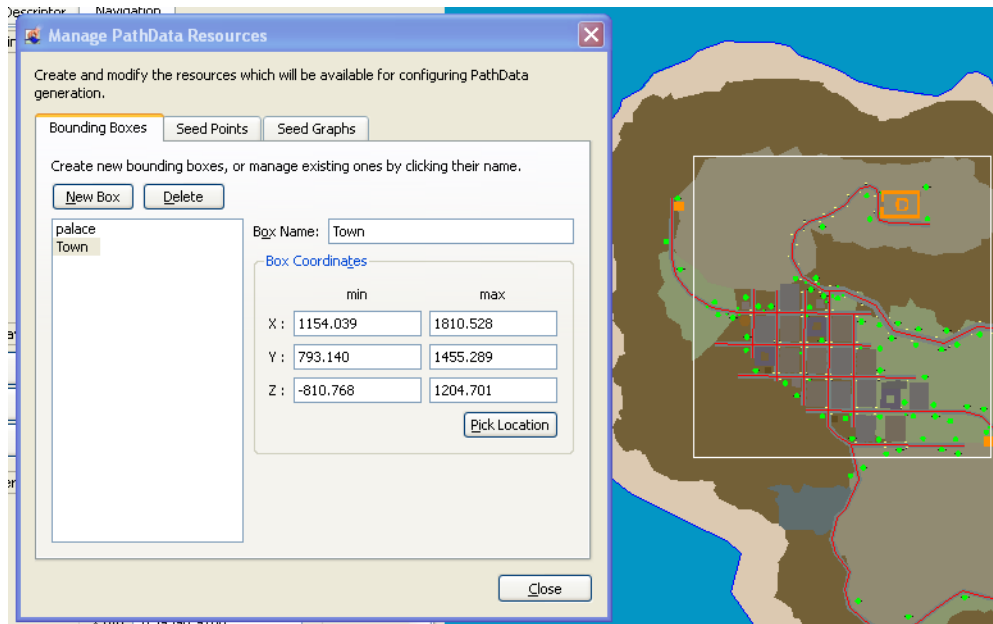


Figure 7-21. Town bounding box

Specify a Seed Point

1. In the Manage PathData Resources dialog box, select the Seed Points tab (Figure 7-22).

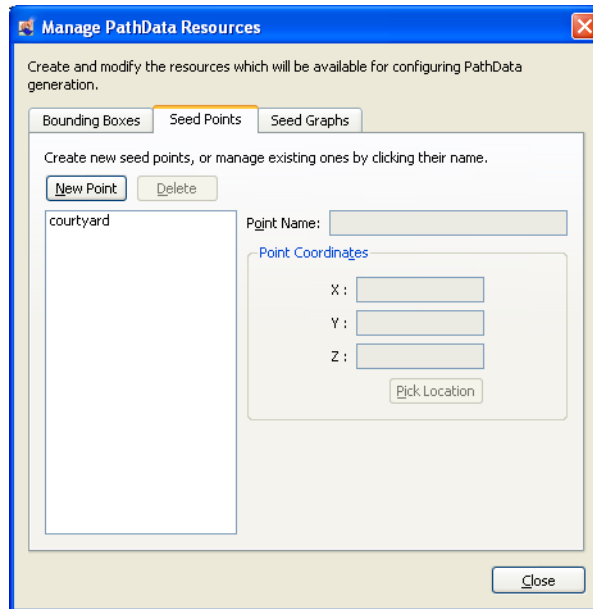


Figure 7-22. Manage PathData Resources dialog box, Seed Points tab

2. Click New Point.
3. In the Point Name text box, type Town_Center.
4. Click Pick Location.
5. Click on the map where the road leaves the town to go towards the palace (Figure 7-23).
6. Close the Manage PathData Resources dialog box.

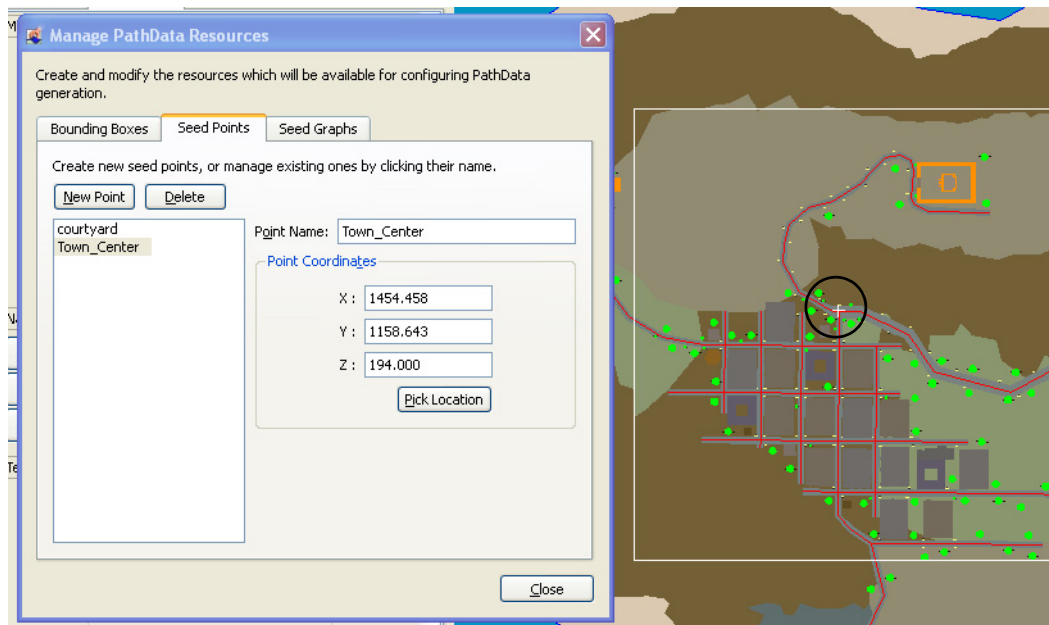


Figure 7-23. Seed point

7.4.2. Configure Generation of PathData

1. Select **B-HAVE** → **Configure PathData Generation**, or click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens (Figure 7-24).

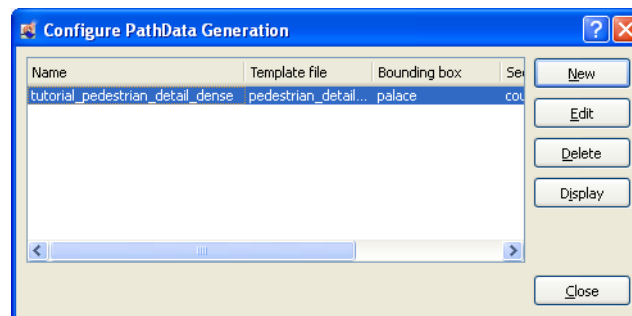


Figure 7-24. Configure PathData Generation dialog box

2. Click New. The Edit PathData Configuration dialog box opens.
3. In the Configuration Name box, type a meaningful name for this configuration, for example tutorial_pedestrian_movement_mode (Figure 7-25).
4. In the Template File drop-down list, select pedestrian_detail_dense.xml. We need to be sure that there is enough detail and density for the movement modes to be effective.

5. Select Town in the Bounding Box list.
6. Select Town_Center in the Seed Points list.
7. Select the Prefer Roads and Avoid Roads check boxes. The B-HAVE Module will include information about the road features in the makland terrain database when it generates PathData (Figure 7-25).

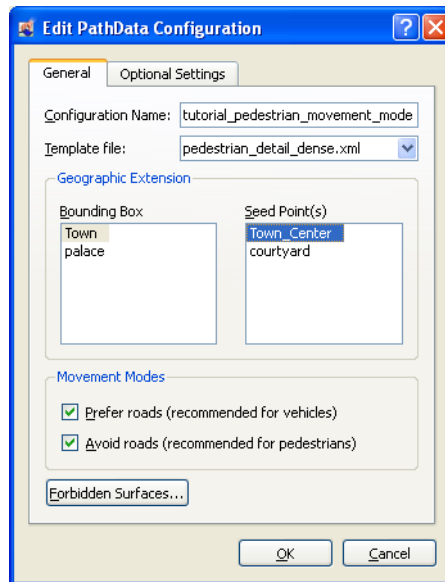


Figure 7-25. Completed PathData configuration

8. Click OK. The configuration is added to the list in the Configure PathData Generation dialog box (Figure 7-26).

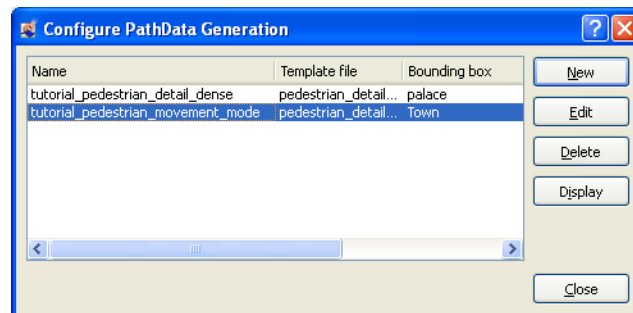


Figure 7-26. PathData configuration

9. Close the Configure PathData Generation dialog box.

7.4.3. Generate PathData

1. Choose **B-HAVE** → **Generate PathData**, or click the Step 3 icon on the B-HAVE Module toolbar. The Generate PathData dialog box opens (Figure 7-27). It lists the configurations that you have created for this terrain database.

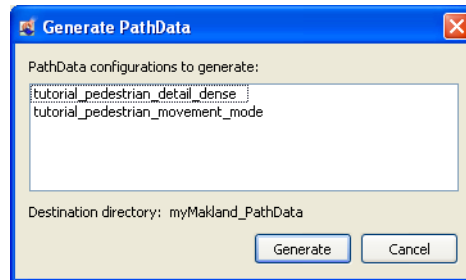


Figure 7-27. Start Batch PathData Generation dialog box

2. Select tutorial_pedestrian_movement_mode.
3. Click Generate. Generation of PathData for this configuration may take more than one hour.
4. When the PathData is finished being generated, click Close.

7.4.4. Associate PathData with Entity Types

To export PathData:

1. Choose **B-HAVE** → **Associate PathData**, or click the Step 4 icon on the B-HAVE Module toolbar. The Associate PathData with Entity Types dialog box opens.
2. In the Lifeforms list, select tutorial_pedestrian_movement_mode (Figure 7-28).

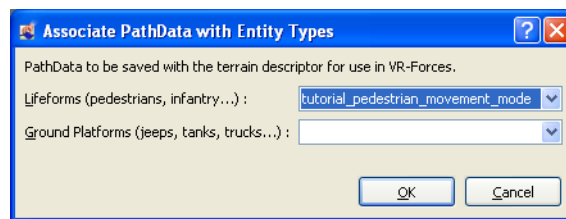


Figure 7-28. Associate PathData with Entity Types dialog box

3. Click OK. The records in the Descriptor panel are updated with this information.

7.4.5. Save the MTD File

1. Choose **File** → **Save**.
2. Choose **File** → **Save As**. Choose an appropriate name, such as `myMakland_movementMode`.

7.5. Simple PathData with Local Zones

Local zones allow you to include areas of increased detail in PathData. Local zones do not allow you to vary density in the PathData. To vary density, you must use seed graphs. In this example, we will generate PathData for the palace and town and create a local zone around the palace.

7.5.1. Set Up PathData Resources

We will use *myMakland.mtd* again as the base terrain descriptor. We will use the PathData resources we created in the previous examples.

- If the TDB Tool is not already running, open it. Load the *myMakland.mtd* file.

7.5.2. Configure Generation of PathData

In the first tutorial example, we did not use any options for configuring PathData generation. In this example, we will configure a local zone.

1. Click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens (Figure 7-29).

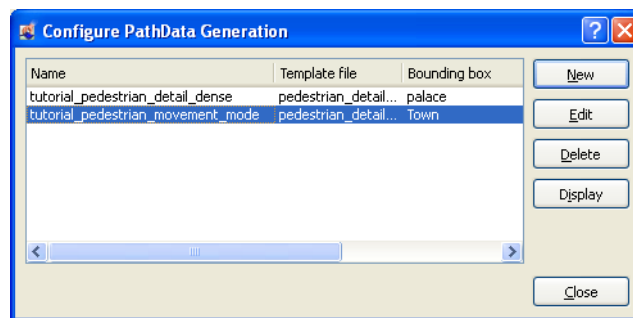


Figure 7-29. Configure PathData Generation dialog box

2. Click **New**. The Edit PathData Configuration dialog box opens.
3. In the Configuration Name box, type a meaningful name for this configuration, for example `tutorial_pedestrian_local_zone`.
4. In the Template File drop-down list, select `pedestrian_nodetail_sparse.xml`.
5. Select **Town** in the Bounding Box list.

6. Select Town_Center in the Seed Points list (Figure 7-30).

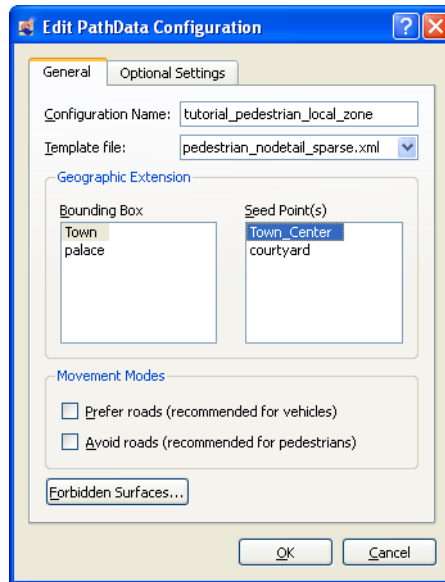


Figure 7-30. Edit PathData Configuration dialog box

7. Select the Optional Settings tab (Figure 7-31).

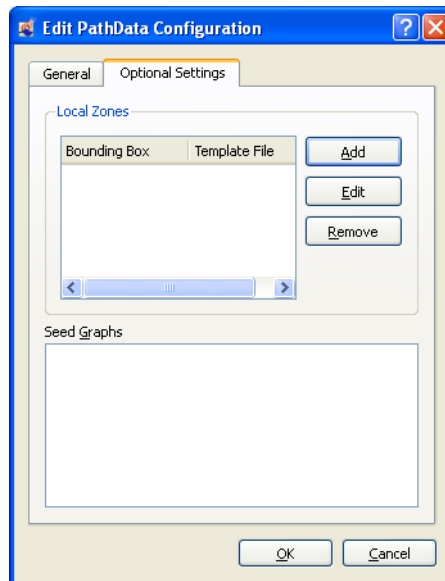


Figure 7-31. Edit PathData Configuration dialog box showing optional settings

8. In the Local Zones group box, click Add. The Edit Local Zone dialog box opens (Figure 7-32).

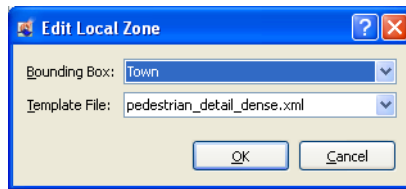


Figure 7-32. Edit Local Zone dialog box

9. In the Bounding Box list, select palace.
10. In the Template File list, select pedestrian_detail_dense.xml.
11. Click Apply. The dialog box closes and an entry is added to the Local Zones list (Figure 7-33).

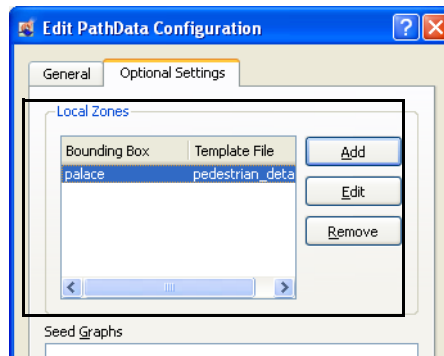


Figure 7-33. Completed PathData configuration

12. Click OK. The configuration is added to the list in the Configure PathData Generation dialog box (Figure 7-34).

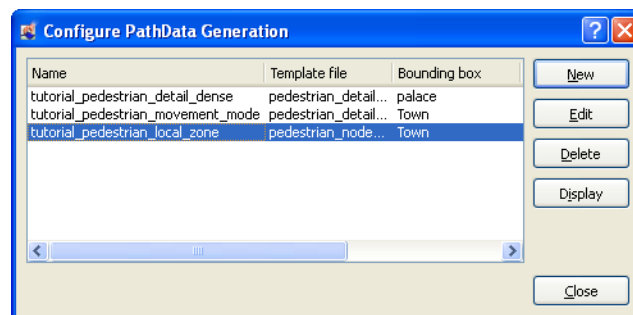


Figure 7-34. PathData configuration

13. Close the Configure PathData Generation dialog box.

7.5.3. Generate PathData

1. Click the Step 3 icon on the B-HAVE Module toolbar. The Generate PathData dialog box opens (Figure 7-35).

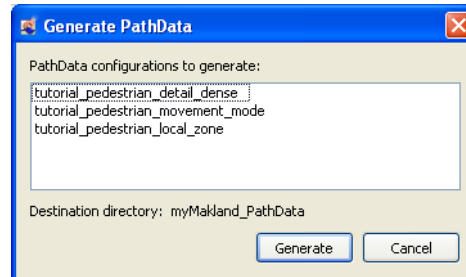


Figure 7-35. Start Batch PathData Generation dialog box

2. Select tutorial_pedestrian_local_zone.
3. Click Generate.
4. When the PathData is finished being generated, click Close.

7.5.4. Associate PathData with Entity Types

To associate PathData with entity types:

1. Click the Step 4 icon on the B-HAVE Module toolbar. The Associate PathData with Entity Types dialog box opens.
2. In the Lifeforms list, select tutorial_pedestrian_local_zone (Figure 7-36).

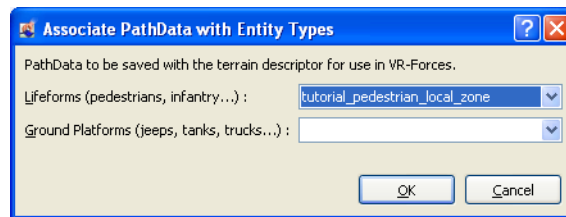


Figure 7-36. Associate PathData with Entity Types dialog box

3. Click OK. The records in the Descriptor panel are updated with this information.

7.5.5. Save the MTD File

1. Choose **File** → **Save**.
2. Choose **File** → **Save As**. Give the terrain a name such as myMakland_localzone.

7.6. Generating PathData Using a Seed Graph

Using seed graphs to build PathData allows you to mix more detailed, dense PathData in relatively smaller parts of the terrain with less detailed, less dense PathData in larger areas of the terrain. For example, you might want to have dense PathData in a building, but be satisfied with sparse PathData in surrounding streets. Or you might want to have dense PathData in a town, but sparse PathData in the surrounding countryside.

In this part of the tutorial, we will use the detailed PathData for the palace that we created previously as a seed graph for a larger area that encompasses the city in Makland.

7.6.1. Set Up PathData Resources

1. If the TDB Tool is not already running, open it. Load the *myMakland.mtd* file.
2. Click the Step 1 icon on the B-HAVE Module toolbar (Figure 7-3). The Manage PathData Resources dialog box opens (Figure 7-4).

Specify a Seed Graph

1. Select the Seed Graphs tab.
2. Click Load. An Open dialog box opens.
3. Select the *.pdl* file for the first PathData the you generated (*tutorial_pedestrian_detail_dense.pdl*). If you had generated PathData for this terrain in other projects, you could navigate to that PathData directories for those projects and select appropriate *.pdl* files.
4. Click Open. The PathData is listed in the seed graphs window (Figure 7-37).

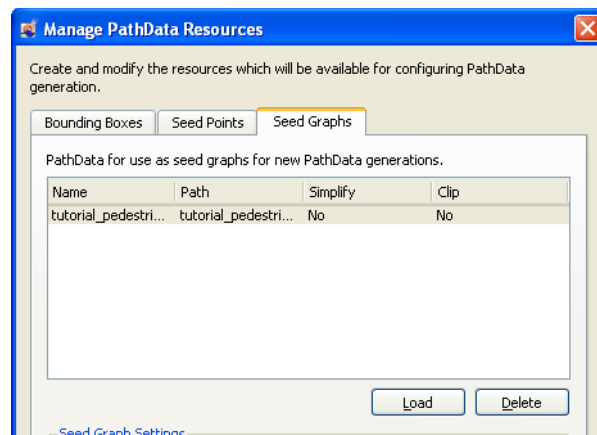


Figure 7-37. Seed Graphs tab

5. Close the Manage PathData Resources dialog box.

7.6.2. Configure Generation of PathData

1. Click the Step 2 icon on the B-HAVE Module toolbar. The Configure PathData Generation dialog box opens (Figure 7-38).

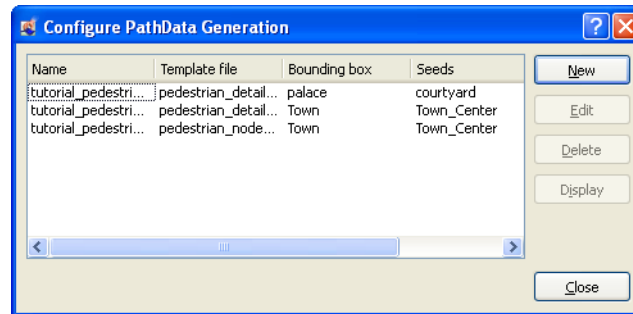


Figure 7-38. Configure PathData Generation dialog box

2. Click New. The Edit PathData Configuration dialog box opens.
3. In the Configuration Name box, type a meaningful name for this configuration, for example tutorial_pedestrian_palaceseedgraph.
4. In the Template File drop-down list, select pedestrian_nodetail_sparse.xml.
5. Select Town in the Bounding Box list.
6. Select Town_Center in the Seed Points list.
7. Select the Optional Settings tab.
8. In the Seed Graphs list, select tutorial_pedestrian_detail_dense (Figure 7-39).

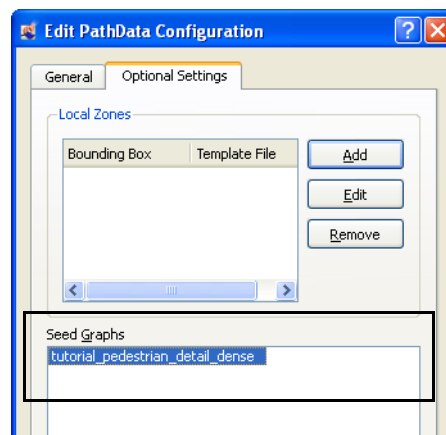


Figure 7-39. Edit PathData Configuration dialog box

9. Click OK. The configuration is added to the list in the Configure PathData Generation dialog box (Figure 7-40).

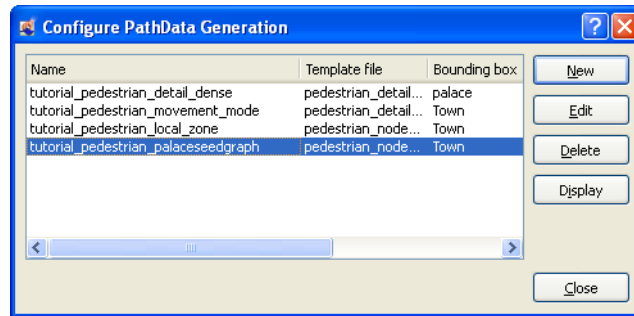


Figure 7-40. PathData configuration

10. Close the Configure PathData Generation dialog box.

7.6.3. Generate PathData

1. Click the Step 3 icon on the B-HAVE Module toolbar. The Generate PathData dialog box opens (Figure 7-41).

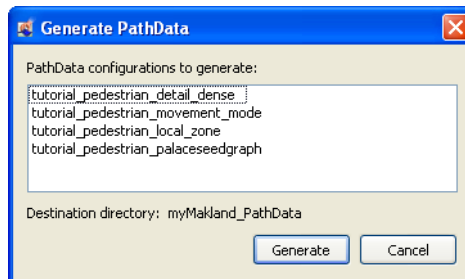


Figure 7-41. Start Batch PathData Generation dialog box

2. Select tutorial_pedestrian_palaceseedgraph.
3. Click Generate.
4. When the PathData is finished being generated, click Close.

7.6.4. Associate PathData with Entity Types

To associate PathData with entity types:

1. Click the Step 4 icon on the B-HAVE Module toolbar. The Associate PathData with Entity Types dialog box opens.
2. In the Lifeforms list, select `tutorial_pedestrian_palaceseedgraph` (Figure 7-42).

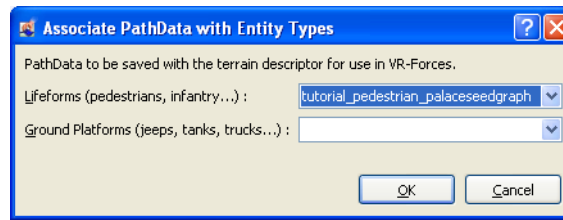


Figure 7-42. Associate PathData with Entity Types dialog box

3. Click OK. The records in the Descriptor panel are updated with this information.

7.6.5. Save the MTD File

1. Choose **File** → **Save**.
2. Choose **File** → **Save As**.
3. Type a name for this terrain and PathData configuration, such as `myMakland_simple_town_palaceseedgraph`.

7.6.6. Examine the Spatial Graph

Take a look at the spatial graph for the new PathData. It should look something like [Figure 7-43](#). Notice how the palace area has dense PathData from the seed graph, while the rest of the PathData is much less dense.

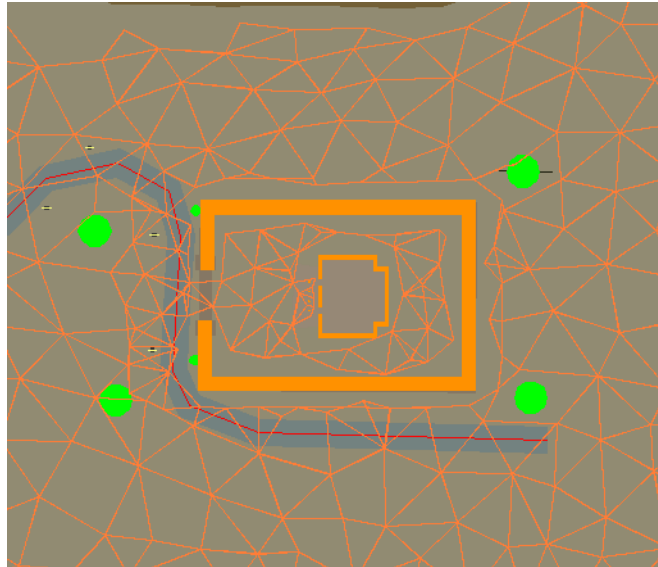


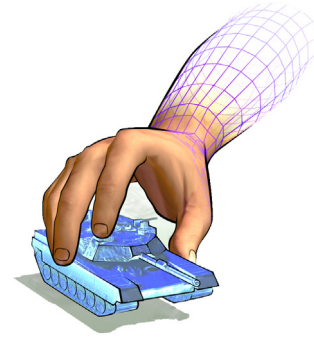
Figure 7-43. Mixture of dense and sparse PathData

7.7. Generating PathData Using a Seed Graph and Local Zones

To create PathData that has both local zones and seed graphs, combine the individual procedures that you have practiced in the previous examples in this tutorial. If you are using a seed graph built around the palace, do not create a local zone in the same area.

7.8. Conclusion

Each of the tutorial examples that you worked through ended by creating a new MTD file. You now have several MTD files that you can load in VR-Forces. They all use *Makland.gdb* as the terrain database, but each has different PathData, which will produce different results when you use them with B-HAVE Module tasks.



Index

Numerics

3D Front End, viewing B-HAVE data in 5-13

A

AboutToCheckpoint() function 5-9

Add Group of Entities, dialog box 4-5

adding

group of entities 4-5

seed graph 3-9

advanced planning 2-13

AI Mesh 2-14

AI mesh 2-2, 2-3, 2-4, 3-19

AIDrivingGetCourtesy(), function 5-17

AIDrivingSetCourtesy(), function 5-17

AIEntityGetCurrentAgent() function 5-14

AIEntityGetDirection() function 5-14

AIEntityGetIndex() function 5-14

AIEntityGetMaxSpeed() function 5-15

AIEntityGetName() function 5-15

AIEntityGetOrderedSpeed() function 5-15

AIEntityGetOrientation() function 5-15

AIEntityGetPosition() function 5-15

AIEntityGetProfileName() function 5-15

AIEntityGetSelf() function 5-15

AIEntityGetSpeed() function 5-16

AIEntityIsUnderFire() function 5-16

AIEntitySetMaxSpeed() function 5-16

AIEntitySetRulesOfEngagement() function 5-16

AIEntitySetSpeed() function 5-16

AIEntitySetSteering() function 5-16

AIEntitySetViewTarget(), function 5-16

AI_Flee() function 5-17

AI_FleeAddDangerousEntity() function 5-17

AI_FleeAddPointToFlee() function 5-18

AI_FleeClearDangerousEntities() function 5-18

AI_FleeGetComputeDestInterval() function 5-18

AI_FleeGetDangerRadius() function 5-18

AI_FleeGetDestination() function 5-18

AI_FleeGetPathLength() function 5-18

AI_FleeGetSpeed() function 5-18

AI_FleeGetStopWhenHidden() function 5-19

AI_FleeResetPointsToFlee() function 5-19

AI_FleeSetComputeDestInterval() function 5-19

AI_FleeSetDangerRadius() function 5-19

AI_FleeSetPathLength() function 5-19

AI_FleeSetSpeed() function 5-19

AI_FleeSetStopWhenHidden() function 5-20

AI_Follow() function 5-5, 5-20

AI_FollowGetAngleFromEntity() function 5-20

AI_FollowGetDestination() function 5-20

AI_FollowGetDistFromEntity() function 5-20

AI_FollowGetEntityToFollow() function 5-20

AI_FollowGetSpeed() function 5-20

AI_FollowHasArrived() function 5-21

AI_FollowSetAngleFromEntity() function 5-21

AI_FollowSetDistFromEntity() function 5-5, 5-21

AI_FollowSetEntityToFollow() function 5-5, 5-21

AI_FollowSetSpeed() function 5-21

AI_Goto() function 5-21

AI_GotoGetDestination() function 5-21

AI_GotoGetDistanceLeft() function 5-22

AI_GotoGetSpeed() function 5-22

AI_GotoGetVRFDestination() function 5-22

AI_GotoHasArrived() function 5-22

AI_GotoSetDestination() function 5-22

AI_GotoSetSpeed() function 5-22

AI_GotoSetVRFDestination() function 5-22

AI_GotoSetVRFWaypoint() function 5-23

- AIHide() function 5-23
- AIHideAddDangerousEntity() function 5-23
- AIHideGetDestination() function 5-23
- AIHideGetGraph() function 5-23
- AIHideGetSpeed() function 5-23
- AIHideHasArrived() function 5-23
- AIHideIsHidden() function 5-24
- AIHideResetDangerousEntities() function 5-24
- AIHideSetGraph() function 5-24
- AIHideSetSpeed() function 5-24
- AIPathfindingGetPathStatusString function 5-24
- AIPathway() function 5-25
- AIPathwayGetCurrentPointIndex() function 5-25
- AIPathwayGetDestination() function 5-25
- AIPathwayGetNbPoints() function 5-25
- AIPathwayGetPathway() function 5-25
- AIPathwayGetReverseOrder() function 5-25
- AIPathwayIsCurrentPointReached() function 5-25
- AIPathwayIsPathwayFinished() function 5-26
- AIPathwayResetPathway() function 5-26
- AIPathwaySetReverseOrder() function 5-26
- AIPathwaySetVRFRRoute() function 5-26
- AIPerceptionGetDistance() function 5-29
- AIPerceptionGetNearestEnemy() function 5-29
- AIPerceptionGetNearestEntity() function 5-29
- AIPerceptionGetNearestSeenEntity() function 5-30
- AIPerceptionInfoGetNearestSeenEnemy() function 5-29
- AIPerceptionInfoGetNearestVisibleEnemy() function 5-29
- AIPerceptionInfoGetNearestVisibleEntity() function 5-29
- AIPerceptionIsEnemy() function 5-30
- AIPerceptionIsSeen() function 5-30
- AIPerceptionIsVisible() function 5-30
- AIReportReflexActivationState() 5-5
- AISetDrawEntityCentric() function 5-13, 5-31
- AISetDrawPathData() function 5-13, 5-31
- AISetMovementMode() function 5-26
- AISetMovementModePenaltyFactor() function 5-26
- AIUtilIsColliding() function 5-31
- AIUtilBHAVE2VRF() function 5-6, 5-31
- AIUtilDebug() function 5-31
- AIUtilDrawLine() function 5-13, 5-32
- AIUtilDrawPath() function 5-13, 5-32
- AIUtilDrawPathway() function 5-32
- AIUtilDrawPoint() function 5-13, 5-32
- AIUtilGetCurrentTime() function 5-32
- AIUtilGetLastError() function 5-32
- AIUtilGetNearestCollisionInfo() function 5-34
- AIUtilGetVRFExerciseStartTime() function 5-33
- AIUtilGetVRFExerciseTime() function 5-33
- AIUtilGetVRFObjectOrientation function 5-33
- AIUtilGetVRFObjectPosition function 5-33
- AIUtilMakeVectorDeg() function 5-33
- AIUtilResetLastError() function 5-33
- AIUtilVecToAnglesDeg() function 5-34
- AIUtilVRF2BHAVE() function 5-6, 5-34
- AIVRFRadioMessageReceivedCallback() function 5-8, 5-27
- AIVRFRadioSendMessage() function 5-8, 5-26
- AIWander() function 5-27
- AIWanderGetDestination() function 5-27
- AIWanderGetSpeed() function 5-27
- AIWanderSelectRandomDestination() function 5-27
- AIWanderSetSpeed() function 5-27
- AIWanderSetVRFArea() function 5-27
- AIWorldGetEntityByIndex() function 5-28
- AIWorldGetEntityByName() function 5-28
- AIWorldGetEntityCount() function 5-28
- AIWorldGetFirstEntity() function 5-28
- AIWorldGetNextEntity() function 5-28
- API
 - building project 6-6
 - remote control 6-2
 - VR-Forces remote control 6-2
- architecture 1-2
- area, creating for multiple entity creation 4-5
- Associate PathData with Entity Types dialog box 3-20, 7-14
- attribute, drivingSide 2-13
- Avoid Roads 4-12
- avoidance, collision 2-7

B

- B-HAVE Hide From, dialog box 4-9
- B-HAVE menu 4-2
- B-HAVE Move to Location, dialog box 4-10
- B-HAVE Options dialog box
 - dialog box, B-HAVE Options 5-3
- B-HAVE Scripts window 5-3
- bhave.Msgs.h* header file 6-5
- BHAVEPathDataRecord 3-21
- bhave.Task.h* header file 6-2

- bounding box
 - clipping seed graph to 3-10
 - creating 3-5, 7-5
 - editing 3-6
 - selecting for PathData generation 3-12
- Brains for Human Activity in Virtual Environments 1-2
- building, project 6-6

C

- callback, registering for B-HAVE messages 6-5
- CarBomb scenario 4-15
- CarBomb_NoEmbarkation 4-15
- CEntity 5-6
- checkpointing, custom 5-9
- choosing, PathData template 3-13
- class
 - DtAdminMessage* 6-4
 - DtAdminRequestScriptName* 6-4
 - DtAdminScriptName* 6-4
 - DtBhaveTask* 6-2
 - DtSetDataCommand* 6-3
 - DtTaskCommand* 6-2
 - DtVrfrRemoteController* 6-5
- clipping, seed graph to bounding box 3-10
- code
 - requesting 6-5
 - setting 6-5
- collision avoidance 2-7, 4-8, 4-9, 4-10, 4-11
- command
 - DtBhaveTask* 6-5
 - DtSetBhaveScript* 6-3
 - DtSetViewTarget* 6-3
- Configure PathData Generation dialog box 2-13, 3-11, 3-16, 3-17
- configuring
 - generation of PathData 3-11, 7-10
 - local zone 3-15
 - Lua text editor 5-3
 - road traffic 2-13
 - seed graph 3-16
- continuously active script 5-5
- controller() function 6-5
- coordinate system, transforming VR-Forces to B-HAVE 5-6
- Create Multiple Entities, dialog box 4-4
- creating
 - area for multiple entities 4-5
 - bounding box 3-5, 7-5

- creating (continued)
 - entity, group of 4-3
 - libraries 5-11
 - module 5-10
 - multiple entities, remote control API 6-5
 - PathData configuration 3-11
 - script 5-3
 - seed point 3-7
 - selection group, using API 6-5
 - terrain project 7-4
- Crowd scenario 4-15

D

- bhaveMsgsRT 6-6
- library, bhaveMsgsRT 6-6
- data, path 2-4
- database, requirements for road traffic 2-11
- deleting, script 5-4
- dense graph density 3-13
- density 3-3, 7-10
 - graph 3-13
 - spatial graph 2-5, 3-13
- detail 3-3, 7-10
 - PathData 2-5, 3-13
 - template 3-13
- dialog box
 - Add Group of Entities 4-5
 - Associate PathData with Entity Types 3-20, 7-14
 - B-HAVE Hide From 4-9
 - B-HAVE Move to Location 4-10
 - Configure PathData Generation 2-13, 3-11, 3-16, 3-17
 - Create Multiple Entities 4-4
 - Edit PathData Configuration 2-13, 3-11, 3-17, 7-11
 - Flee From 4-8
 - Generate PathData 3-17, 3-18, 7-13
 - Manage PathData Resources 3-5, 3-7, 3-9, 7-5
 - PathData Generation Monitor 3-19
- directory, installed 1-4
- disabling, script in plan 5-5
- displaying
 - B-HAVE route 4-14
 - PathData configuration 3-16
 - spatial graph 7-15
- drawing, PathData in Stealth 5-13
- drivingSide attribute 2-13
- DTA 2-9

DtAdminMessage class 6-4
DtAdminRequestScriptName class 6-4
DtAdminScriptName class 6-4
DtBhaveTask class 6-2
DtBhaveTask command 6-5
DtIfCreateMultipleEntities 6-5
DtIfCreateSelectionGroup 6-5
DtIfDeleteLuaScript 6-5
DtIfRequestLuaScriptList 6-5
DtIfRequestLuaScriptText 6-5
DtIfSetLuaScriptText 6-5
DtSetBhaveScript command 6-3
DtSetDataCommand class 6-3
DtSetViewTarget command 6-3
DtTaskCommand class 6-2
DtVrfRemoteController class 6-5
dynamic avoidance 2-7, 2-10
dynamic topological analysis 2-8, 2-9

E

edge 2-2, 2-14, 3-3, 3-9
 length of 2-5, 3-13
Edit PathData Configuration dialog box 2-13, 3-11, 3-17, 7-11
editing
 bounding box 3-6
 PathData configuration 3-17
 script 5-4
editor, external 5-2
entity
 controlling from script 5-6
 creating group of 4-3
 creating multiple, remote control API 6-5
 drawing PathData around 5-13
 looping through lists of 5-6
 sending radio messages to 5-8
 specifying for multiple entity creation 4-5
entity type, lifeform and ground platform 3-13
example 7-3
 scripts 5-7
 terrain with PathData 1-3

F

file
 deleting 6-5
 extension, PDL 3-9
 installed 1-4

filtering
 surface types in PathData configuration 3-11
 surfaces 3-14
Flee From, dialog box 4-8
Flee From task 2-10, 4-8
FLEXIm 1-4
Follow Entity task 4-7, 4-8
following, paths 2-6
forbidden surface 3-11, 3-14
function 5-24
 AboutToCheckpoint() 5-9
 AIDrivingGetCourtesy() 5-17
 AIDrivingSetCourtesy() 5-17
 AIEntityGetCurrentAgent() 5-14
 AIEntityGetDirection() 5-14
 AIEntityGetIndex() 5-14
 AIEntityGetMaxSpeed() 5-15
 AIEntityGetName() 5-15
 AIEntityGetOrderedSpeed() 5-15
 AIEntityGetOrientation() 5-15
 AIEntityGetPosition() 5-15
 AIEntityGetProfileName() 5-15
 AIEntityGetSelf() 5-15
 AIEntityGetSpeed() 5-16
 AIEntityIsUnderFire() 5-16
 AIEntitySetMaxSpeed() 5-16
 AIEntitySetRulesOfEngagement() 5-16
 AIEntitySetSpeed() 5-16
 AIEntitySetSteering() 5-16
 AIEntitySetViewTarget() 5-16
 AIFlee() 5-17
 AIFleeAddDangerousEntity() 5-17
 AIFleeAddPointToFlee() 5-18
 AIFleeClearDangerousEntities() 5-18
 AIFleeGetComputeDestInterval() 5-18
 AIFleeGetDangerRadius() 5-18
 AIFleeGetDestination() 5-18
 AIFleeGetPathLength() 5-18
 AIFleeGetSpeed() 5-18
 AIFleeGetStopWhenHidden() 5-19
 AIFleeResetPointsToFlee() 5-19
 AIFleeSetComputeDestInterval() 5-19
 AIFleeSetDangerRadius() 5-19
 AIFleeSetPathLength() 5-19
 AIFleeSetSpeed() 5-19
 AIFleeSetStopWhenHidden() 5-20
 AIFollow() 5-5, 5-20
 AIFollowGetAngleFromEntity() 5-20
 AIFollowGetDestination() 5-20
 AIFollowGetDistFromEntity() 5-20

function 5-24 (continued)

AIFollowGetEntityToFollow() 5-20
 AIFollowGetSpeed() 5-20
 AIFollowHasArrived() 5-21
 AIFollowSetAngleFromEntity() 5-21
 AIFollowSetDistFromEntity() 5-5, 5-21
 AIFollowSetEntityToFollow() 5-5, 5-21
 AIFollowSetSpeed() 5-21
 AIGoto() 5-21
 AIGotoGetDestination() 5-21
 AIGotoGetDistanceLeft() 5-22
 AIGotoGetSpeed() 5-22
 AIGotoGetVRFDestination() 5-22
 AIGotoHasArrived() 5-22
 AIGotoSetDestination() 5-22
 AIGotoSetSpeed() 5-22
 AIGotoSetVRFDestination() 5-22
 AIGotoSetVRFWaypoint() 5-23
 AIHide() 5-23
 AIHideAddDangerousEntity() 5-23
 AIHideGetDestination() 5-23
 AIHideGetGraph() 5-23
 AIHideGetSpeed() 5-23
 AIHideHasArrived() 5-23
 AIHideIsHidden() 5-24
 AIHideResetDangerousEntities() 5-24
 AIHideSetGraph() 5-24
 AIHideSetSpeed() 5-24
 AIPathfindingGetPathStatus 5-24
 AIPathfindingGetPathStatusString 5-24
 AIPathway() 5-25
 AIPathwayGetCurrentPointIndex() 5-25
 AIPathwayGetDestination() 5-25
 AIPathwayGetNbPoints() 5-25
 AIPathwayGetPathway() 5-25
 AIPathwayGetReverseOrder() 5-25
 AIPathwayIsCurrentPointReached() 5-25
 AIPathwayIsPathwayFinished() 5-26
 AIPathwayResetPathway() 5-26
 AIPathwaySetReverseOrder() 5-26
 AIPathwaySetVRFRoute() 5-26
 AIPerceptionGetDistance() 5-29
 AIPerceptionGetNearestEnemy() 5-29
 AIPerceptionGetNearestEntity() 5-29
 AIPerceptionGetNearestSeenEntity() 5-30
 AIPerceptionInfoGetNearestSeenEnemy() 5-29
 AIPerceptionInfoGetNearestVisibleEnemy() 5-29
 AIPerceptionInfoGetNearestVisibleEntity() 5-29

function 5-24 (continued)

AIPerceptionIsEnemy() 5-30
 AIPerceptionIsSeen() 5-30
 AIPerceptionIsVisible() 5-30
 AISetDrawEntityCentric() 5-13, 5-31
 AISetDrawPathData() 5-13, 5-31
 AISetMovementMode() 5-26
 AISetMovementModePenaltyFactor() 5-26
 AIUtilsColliding() 5-31
 AIUtilBHAVE2VRF() 5-6, 5-31
 AIUtilDebug() 5-31
 AIUtilDrawLine() 5-13, 5-32
 AIUtilDrawPath() 5-13, 5-32
 AIUtilDrawPathway() 5-32
 AIUtilDrawPoint() 5-13, 5-32
 AIUtilGetCurrentTime() 5-32
 AIUtilGetLastError() 5-32
 AIUtilGetNearestCollisionInfo() 5-34
 AIUtilGetVRFExerciseStartTime() 5-33
 AIUtilGetVRFExerciseTime() 5-33
 AIUtilGetVRFObjectOrientation 5-33
 AIUtilGetVRFObjectPosition 5-33
 AIUtilMakeVectorDeg() 5-33
 AIUtilResetLastError() 5-33
 AIUtilVecToAnglesDeg() 5-34
 AIUtilVRF2BHAVE() 5-6, 5-34
 AIVRFRadioMessageReceivedCallback() 5-8, 5-27
 AIVRFRadioSendMessage() 5-8, 5-26
 AIWander() 5-27
 AIWanderGetDestination() 5-27
 AIWanderGetSpeed() 5-27
 AIWanderSelectRandomDestination() 5-27
 AIWanderSetSpeed() 5-27
 AIWanderSetVRFArea() 5-27
 AIWorldGetEntityByIndex() 5-28
 AIWorldGetEntityByName() 5-28
 AIWorldGetEntityCount() 5-28
 AIWorldGetFirstEntity() 5-28
 AIWorldGetNextEntity() 5-28
 controller() 6-5
 PostLoadCheckpoint() 5-9
 processScriptNameCallback() 6-4
 set data request 5-35
 task 5-35
 Think() 5-5, 5-7, 5-8
 VRFSetsAltitude() 5-40
 VRFSetsDestroyed() 5-40
 VRFSetsDIGuyAnimation() 5-41
 VRFSetsDIGuyAppearance() 5-40

function 5-24 (continued)
 VRFSetsDisembarked() 5-41
 VRFSetsEmbarked() 5-41
 VRFSetsFireNow() 5-41
 VRFSetsHeading() 5-41
 VRFSetsLocation() 5-41
 VRFSetsOrderedSpeed() 5-41
 VRFSetsPosture() 5-42
 VRFSetsRestore() 5-42
 VRFSetsSectorOfResponsibility() 5-42
 VRFSetsSpotReports() 5-42
 VRFSetsTarget() 5-42
 VRFSetsWeaponState() 5-43
 VRFTaskDisembark() 5-35
 VRFTaskDisembarkAll() 5-35
 VRFTaskEmbarkOn() 5-35
 VRFTaskFireCruiseMissile() 5-35
 VRFTaskFireForEffectOnEntity() 5-35
 VRFTaskFireForEffectOnLocation() 5-36
 VRFTaskFireForEffectOnTarget() 5-36
 VRFTaskFollowEntity() 5-36
 VRFTaskInterceptAndDestroy() 5-36
 VRFTaskMoveAlongRoute() 5-36
 VRFTaskMoveToWaypoint() 5-37
 VRFTaskMoveToLocation() 5-37
 VRFTaskPatrolAlongRoute() 5-37
 VRFTaskPatrolBetween() 5-37
 VRFTaskSendRadioSet() 5-38
 VRFTaskSendRadioTask() 5-39
 VRFTaskSendTextMessage() 5-39
 VRFTaskTurnToHeading() 5-39
 VRFTaskUserTask() 5-40
 VRFTaskWait() 5-40
 VRFTaskWaitDuration() 5-40
 VRFTaskWaitElapsed() 5-40

G

GDB format 3-4
 Generate PathData dialog box 3-17, 3-18, 7-13
 generating
 PathData 3-17, 7-13
 complex 3-3
 Go To task 2-10
 graph density 3-13
 graph edge, limiting 3-14
 ground platform entity type 3-13
 group
 entities, adding 4-5

H

header file
 bhaveMsgs.h 6-5
 bhaveTask.h 6-2
 heuristic, path planning 2-5
 Hide From task 2-10
 task, Hide From 4-9

I

ignoring, surface types 3-14
Import parameter 3-21
initial-movement-mode parameter 4-12
 installing
 B-HAVE module 1-3
 license 1-4
 interface message 6-5
 Interior Movement scenario 4-15
 intersection, traffic behavior 2-11
 iterator 5-6

K

Kynapse 1-2
 kynapse-controller 4-12

L

level of detail, terrain 3-13
 .lib 6-6
 library 6-6
 library, Lua 5-11
 license, management 1-4
 lifeform
 traffic behavior 2-11
 vehicle avoidance 2-11
 lifeform entity type 3-13
 line-of-sight 4-9
 loading, terrain database in TDB Tool 3-4
 local zone 3-3, 3-5
 configuring 3-15
 specifying 7-23
 looping, through entity list 5-6
 Lua 2-13, 5-7
 assigning script remotely 6-3
 deleting script file 6-5
 getting script name 6-4
 interpreter 5-6
 library 5-11

Lua 2-13, 5-7 (continued)

- modules 5-10
- requesting list of scripts 6-5
- requesting script code 6-5
- script, creating and editing 5-2
- sending radio messages 5-8
- setting script 4-13
- setting script code 6-5

M

- MÄK Terrain Database Tool 1-2
- Makland B-HAVE scenario 4-15
- Manage PathData Resources dialog box 3-5, 3-7, 3-9, 7-5
- menu, B-HAVE 4-2
- message
 - registering callbacks for 6-5
 - remote control API 6-5
 - sending radio 5-8
 - sending text 4-6
- model, topological 2-2
- modifying, bounding box 3-6
- module, creating 5-10
- monitoring, PathData generation 3-19
- more detail level of detail 3-13
- Move Along Route task 4-7, 4-11
- Move to Location task 4-7, 4-10
- Move To Waypoint task 4-9
- Move to Waypoint task 4-7
- movement, smoothness of 2-5, 3-13
- movement mode 2-11, 4-12
- MTD file 1-3, 3-2, 3-4, 3-20, 3-21, 7-4, 7-14
- multiple entity creation 4-3
 - specifying entity type 4-5

N

- network
 - road, PathData 3-3
 - road vector 2-11
- no detail level of detail 3-13
- no_reflex.lua script 5-5

O

- ObjectType parameter 3-21

P

- parameter
 - Import 3-21
 - initial-movement-mode 4-12
 - ObjectType 3-21
 - PathDataDirectory 3-21
 - PathDataName 3-21
 - PathsRelativeToTerrain 3-21
 - publish-debug-routes 4-14
 - unify-movement-tasks 4-7
- path
 - data 2-4
 - displaying B-HAVE 4-14
 - following 2-6
 - planning 2-5
- path finding 2-2, 2-5, 2-10
- path following 2-10
- path planning 4-8, 4-9
 - using road data 4-12
- path smoothing 4-14
- PathData 1-2, 1-3, 2-4, 2-8, 2-14, 3-2
 - associating with terrain 3-20, 7-14
 - configuration 3-11
 - displaying 3-16
 - configuring, optional settings 3-15
 - configuring generation of 7-10
 - detail and density 2-5, 3-13
 - drawing around entity 5-13
 - editing configuration 3-17
 - filtering surface types 3-11
 - generating 3-17, 7-13
 - generating complex 3-3
 - level of detail 3-11
 - monitoring generation of 3-19
 - resources 3-5, 7-5
 - road network 3-3
 - simple process 7-5
 - template 3-11, 7-10
 - templates 2-5, 3-13
 - testing 3-22
- PathData Generation Monitor dialog box 3-19
- PathDataDirectory parameter 3-21
- PathDataName parameter 3-21
- PathsRelativeToTerrain parameter 3-21
- PDL file extension 3-9
- pedestrian, traffic behavior 2-11
- physical world 2-14
- Pick Location 3-6, 7-7
- plan, disabling script in 5-5

- planning
 - advanced 2-13
 - path 2-5
- plug-in 1-2
- pointer variable 5-6
- PostLoadCheckpoint() function 5-9
- Prefer Roads 2-11, 4-12
- preferred movement mode 4-11
- preferred road data 4-11
- processScriptNameCallback() function 6-4
- project
 - API, building 6-6
 - creating terrain 7-4
- publish-debug-routes parameter 4-14

R

- radio message, sending using Lua 5-8
- reflex script 5-5
- remote control, VR-Forces 6-2
- remote control API
 - assigning script 6-3
 - assigning task 6-2
 - building project 6-6
 - creating, selection group 6-5
 - creating entities 6-5
 - getting script name 6-4
 - messages 6-5
 - registering callbacks 6-5
 - sending set view target set data request 6-3
 - setting movement mode 6-3
- requesting, list of Lua scripts 6-5
- Rescue Convoy scenario 4-15
- resource, PathData 3-5, 7-5
- right-of-way 2-11
- road data, using in path planning 4-12
- road traffic 2-11
 - configuring 2-13
- road vector network 2-11
 - creating PathData from 3-3
- route
 - B-HAVE, displaying 4-14

S

- sample terrain 1-3
- Scatter scenario 4-15
- script
 - assigning using remote control API 6-3
 - creating 5-2, 5-3

- script (continued)
 - deleting 5-4
 - deleting file 6-5
 - disabling from plan 5-5
 - editing 5-2, 5-4
 - entity control 5-6
 - example 5-7
 - getting name of 6-4
 - issues for writing 5-6
 - Lua 2-13, 5-7
 - requesting code 6-5
 - requesting list of available 6-5
 - setting 4-13
 - setting code 6-5
 - types 5-5
 - using B-HAVE tasks in 5-5
 - writing 5-5
- seed graph 3-3, 7-27
 - adding 3-9
 - clipping to bounding box 3-10
 - configuring 3-16
 - selecting for PathData configuration 3-12
 - simplification 3-10
- seed point 2-4, 2-14, 3-5, 7-5
 - creating 3-7
 - selecting for PathData configuration 3-12
 - specifying 7-8
- Seed Points tab 7-8
- selection group 4-3
 - creating programmatically 6-5
- Send Text Message dialog box
 - dialog box, Send Text Message 4-6
- sending, text message 4-6
- set data request
 - functions 5-35
 - sending remotely 6-3
 - Set Movement Mode 4-12
 - Set Script 4-13
 - View Target 4-13
- set movement mode, assigning remotely 6-3
- Set Movement Mode set data request 4-12
- Set Script set data request 4-13
- set view target, sending remotely 6-3
- simplification 3-10
- sparse graph density 3-13
- spatial graph 2-2, 2-4, 2-14, 3-19
 - comparing different 7-16
 - displaying 7-15
- spatial reasoning 2-8

- specifying
 - entity to create 4-5
 - local zone 7-23
 - PathData resources 3-5, 7-5
 - seed point 7-8
 - text editor 5-3
 - starting, TDB Tool 7-3
 - Stealth, drawing PathData in 5-13
 - Stop When Hidden 4-8
 - subgoal 2-6
 - surface
 - filtering 3-14
 - forbidden 3-11, 3-14
 - syntax error 5-6
- ## T
- task 2-10
 - assigning remotely 6-2
 - B-HAVE, integration with VR-Forces 4-7
 - Flee From 2-10, 4-8
 - Follow Entity 4-8
 - functions 5-35
 - Go To 2-10
 - Hide From 2-10
 - Move Along Route 4-11
 - Move to Location 4-10
 - Move To Waypoint 4-9
 - using in script 5-5
 - Wander 4-11
 - wander 2-10
 - TDB Tool 1-2
 - loading terrain database 3-4
 - starting 7-3
 - template 2-5, 3-13
 - choosing 3-13
 - PathData 7-10
 - PathData configuration 3-11
 - terrain
 - associating PathData with 3-20, 7-14
 - creating project 7-4
 - level of detail 3-13
 - loading in TDB Tool 3-4
 - provided with PathData 1-3
 - requirements for road traffic 2-11
 - Terrain Descriptor File 3-2
 - testing, PathData 3-22
 - text, sending 5-8
 - text editor, specifying 5-3
 - text message, sending 4-6
 - Think() function 5-5, 5-7, 5-8
 - topological model 2-2
 - traffic
 - behavior 2-11
 - configuring 2-13
 - right-of-way 2-11
 - terrain database requirements 2-11
 - Traffic scenario 4-15
 - transforming, VR-Forces coordinates to B-HAVE 5-6
 - tutorial 7-3
- ## U
- unify-movement-tasks parameter 4-7
- ## V
- vector, road 2-11
 - vector network, PathData 3-3
 - vehicle, traffic behavior 2-11
 - vertex 2-2, 2-14, 3-9
 - very sparse graph density 3-13
 - View Target set data request 4-13
 - viewing, B-HAVE data in 3D Front End 5-13
 - VR-Forces Remote Control API 6-2
 - VRFSetAltitude() function 5-40
 - VRFSetDestroyed() function 5-40
 - VRFSetDIGuyAnimation() function 5-41
 - VRFSetDIGuyAppearance() function 5-40
 - VRFSetDisembarked() function 5-41
 - VRFSetEmbarked() function 5-41
 - VRFSetFireNow() function 5-41
 - VRFSetHeading() function 5-41
 - VRFSetLocation() function 5-41
 - VRFSetOrderedSpeed() function 5-41
 - VRFSetPosture() function 5-42
 - VRFSetRestore() function 5-42
 - VRFSetSectorOfResponsibility() function 5-42
 - VRFSetSpotReports() function 5-42
 - VRFSetTarget() function 5-42
 - VRFSetWeaponState() function 5-43
 - VRFTaskDisembark() function 5-35
 - VRFTaskDisembarkAll() function 5-35
 - VRFTaskEmbarkOn() function 5-35
 - VRFTaskFireCruiseMissile() function 5-35
 - VRFTaskFireForEffectOnEntity() function 5-35
 - VRFTaskFireForEffectOnLocation() function 5-36
 - VRFTaskFireForEffectOnTarget() function 5-36

VRFTaskFollowEntity() function 5-36
VRFTaskInterceptAndDestroy() function 5-36
VRFTaskMoveAlongRoute() function 5-36
VRFTaskMoveToWaypoint() function 5-37
VRFTaskMoveToLocation() function 5-37
VRFTaskPatrolAlongRoute() function 5-37
VRFTaskPatrolBetween() function 5-37
VRFTaskSendRadioSet() function 5-38
VRFTaskSendRadioTask() function 5-39
VRFTaskSendTextMessage() function 5-39
VRFTaskTurnToHeading() function 5-39
VRFTaskUserTask() function 5-40
VRFTaskWait() function 5-40
VRFTaskWaitDuration() function 5-40
VRFTaskWaitElapsed() function 5-40

W

Wander task 2-10, 4-11
writing, script 5-3

X-Y-Z

XML file, B-HAVE configuration 3-10



Link - Simulate - Visualize

BHV-1.4-1-090710