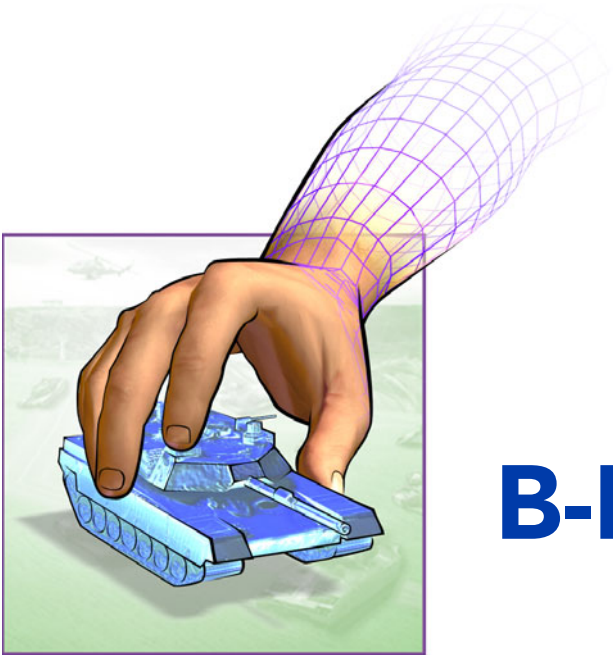
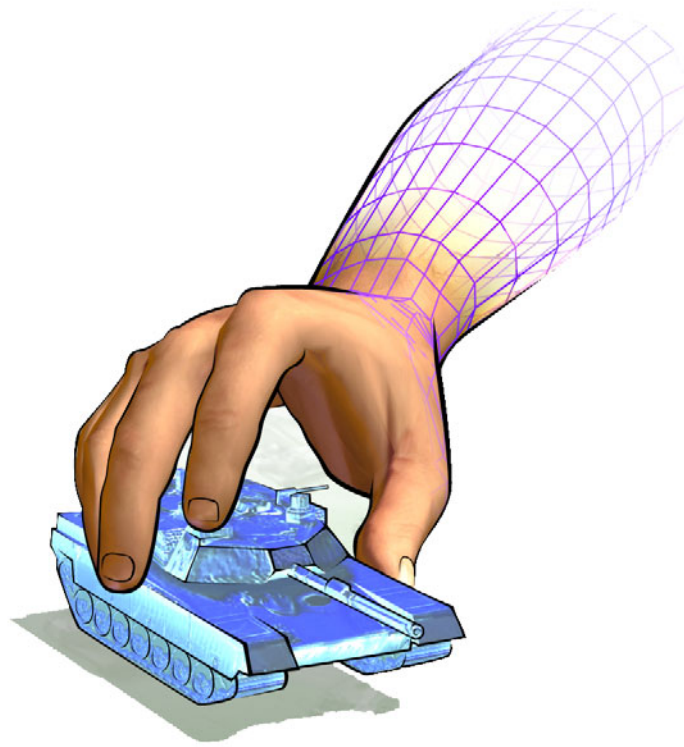




B-HAVE Module for VR-Forces

User's Guide



B-HAVE Module for VR-Forces

User's Guide

Copyright © 2007 MÄK Technologies
All rights Reserved. Printed in the United States.

Portions of this document are Copyright © 2007 Kynogon S.A.
All Rights Reserved.

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

VR-Exchange™ is a trademark of MÄK Technologies. MÄK Technologies®,
VR-Forces®, RTIsby®, and VR-Link® are registered
trademarks of MÄK Technologies, Inc.

Kynogon is a registered trademark of Kynogon S.A.
Kynapse is a trademark of Kynogon S.A.

All other trademarks are owned by their respective companies.

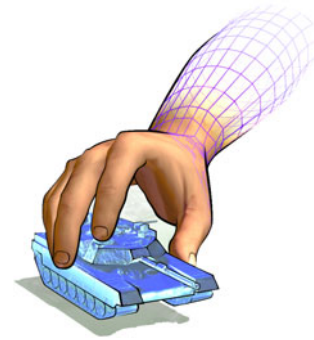
For third-party license information, please see [“Third Party Licenses,”](#) on page xii.

MÄK Technologies
68 Moulton St.
Cambridge, MA 02138 USA

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

info@mak.com
www.mak.com

Revision BHV-1.1-1-070712



Contents

Preface

How This Manual is Organized	vii
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

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 ..	1-5

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	2-6
2.2.1. Planning a Path	2-6
2.2.2. Path Following	2-7
2.2.3. Dynamic Avoidance	2-8
2.3. Spatial Reasoning	2-9
2.3.1. Dynamic Topological Analysis	2-10

2.4. B-HAVE Module Tasks	2-11
2.4.1. The Go To Task	2-11
2.4.2. The Wander Task	2-11
2.4.3. The Flee and Hide Tasks	2-11
2.5. Advanced Planning Capabilities	2-12
2.6. Glossary	2-12

Chapter 3 Generating PathData

3.1. Introduction	3-2
3.1.1. Varying the Complexity of PathData Configurations	3-3
3.2. Starting the TDB Tool	3-4
3.2.1. Loading a Terrain Database	3-5
3.3. Managing PathData Resources	3-6
3.3.1. Specifying Bounding Boxes	3-6
3.3.2. Specifying Seed Points	3-8
3.3.3. Specifying Seed Graphs	3-10
3.4. Configuring Generation of PathData	3-12
3.4.1. Creating a PathData Configuration	3-12
3.4.2. Editing PathData Configurations	3-17
3.5. Generating PathData	3-18
3.5.1. Monitoring the PathData Generation Process	3-20
3.6. Associating PathData with Entity Types	3-21
3.6.1. The BHAVEPathDataRecord	3-22
3.7. Testing PathData	3-23
3.8. Troubleshooting	3-24

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

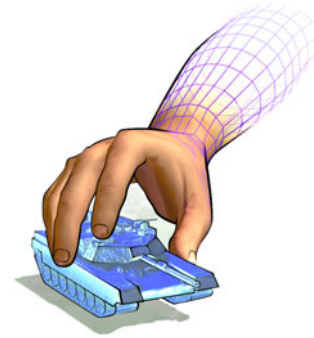
4.1. Using the B-HAVE Module in a Simulation	4-2
4.1.1. Disabling the B-HAVE Module for a Scenario	4-2
4.2. Creating Groups of Entities	4-4
4.2.1. Adding an Area for Multiple Entity Creation	4-6
4.2.2. Specifying the Entities to Create	4-6
4.3. B-HAVE Module Tasks	4-7
4.3.1. The B-HAVE Flee From Task	4-7
4.3.2. The B-HAVE Follow Entity Task	4-8
4.3.3. The B-HAVE Hide From Task	4-8
4.3.4. The B-HAVE Move to Waypoint Task	4-9
4.3.5. The B-HAVE Move to Location Task	4-9
4.3.6. The B-HAVE Move Along Route Task	4-11
4.3.7. The B-HAVE Wander Task	4-11
4.3.8. The B-HAVE Set Movement Mode Set Data Request	4-12
4.3.9. The B-HAVE Set Script Set Data Request	4-13
4.3.10. The B-HAVE View Target Set Data Request	4-13
4.4. Viewing B-HAVE Module Data in the VR-Forces GUI	4-14
4.5. Viewing B-HAVE Module Data in the MÄK Stealth	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. Creating a New Script	5-2
5.2.2. Editing a Script	5-3
5.2.3. Deleting a Script	5-3
5.3. Writing Scripts	5-3
5.3.1. Using B-HAVE Module Tasks in Scripts	5-4
5.3.2. Considerations for Writing and Using Lua Scripts	5-4
5.3.3. Example Script	5-6
5.3.4. Sending Radio Messages	5-7
5.3.5. Custom Checkpointing	5-8
5.4. Viewing B-HAVE Module Data in the MÄK Stealth	5-9
5.4.1. Drawing PathData Around Entities	5-9
5.5. B-HAVE Module Functions	5-10
5.5.1. Entity Functions	5-10
5.5.2. Behavior Functions	5-13
5.5.3. Entity List Functions	5-23
5.5.4. Perception Functions	5-24
5.5.5. Utility Functions	5-26

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.

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

MÄK Products

The B-HAVE Module for VR-Forces is a member of the MÄK Technologies® line of software products designed to streamline the process of developing and using networked simulated environments. The MÄK Technologies 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 for the MÄK RTI rtiexec.
- ♦ **VR-Forces®.** VR-Forces is a computer generated forces application and toolkit. It provides an application with a graphical user interface, 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.

- ♦ **MÄK Stealth and StealthXR.** The Stealth displays a realistic, three-dimensional view of your virtual world. You can view this world from the inside of a simulated moving vehicle, or place the eyepoint at another moving or stationary location. The Stealth lets you switch rapidly among several predefined viewpoints while the simulation is underway.

StealthXR (exaggerated reality) is a separately-licensed component of the Stealth. It combines the features of traditional 2D and 3D displays into one exaggerated reality display. It provides a big picture understanding of a battlefield along with an immersive sense of perspective.

- ♦ **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 graphical user interface 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 two-dimensional, “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.
- ♦ **MÄK HLA/DIS Toolbox for MATLAB and Simulink.** The MÄK HLA/DIS Toolbox is a wrapper around VR-Link that allows MATLAB and Simulink users to analyze data from live or recorded HLA or DIS exercises. It also allows Simulink users to build models that can participate in HLA or DIS exercises.

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

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 MÄK RTI trial version:	www.mak.com/products/rti.php
Support FAQ:	www.mak.com/support/faq.php

Post

Send postal correspondence to:	MÄK Technologies 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:

Monospaced	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.
/	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, typically 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 website 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.

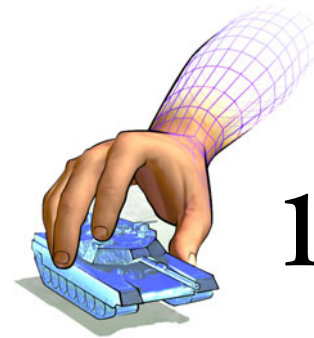
libXML and libICONV

VR-Link and all MÄK software which 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.

- ♦ 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/>



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.....	1-5

1.1. Introduction

The B-HAVE Module for VR-Forces (Brains for Human Activity in Virtual Environments) is a VR-Forces plug-in that extends the movement models for lifeforms and ground platforms, as follows:

- ♦ Full 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 AI technology solution, from Kynogon S.A. It is based on Kynapse's fundamental concepts of Topological Modeling for Path Finding, and Spatial Reasoning for Advanced Behaviors, all integrated within 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.

1.1.2. 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 two terrain databases for which PathData has already been generated: Makland 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 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-Forces3.10.1*. If you did not install VR-Forces into the default directory, it is your responsibility to change the installation directory to your VR-Forces installation directory.

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

- ♦ 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 1.0 for VR-Forces 3.10 will work only with VR-Forces 3.10. It would not work with VR-Forces 3.10.1.
- ♦ The B-HAVE Module installation will ask if you want to install the B-HAVE Stealth Viewer Plug-in for MÄK Stealth. This plug-in is required to view B-HAVE Module debugging data in 3D using the MÄK Stealth (for details, please see [“Viewing B-HAVE Module Data in the MÄK Stealth,”](#) on page 4-14.) If you answer yes, you will be prompted to enter the location of an existing installation of B-HAVE Module Stealth 6.0 or later.
- 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

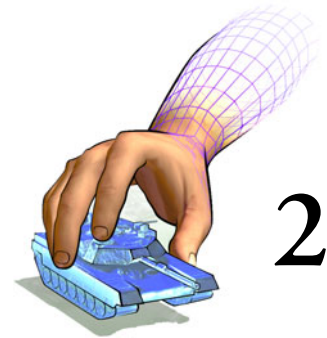
File/Directory	Description
<i>bin\vrfOpdEditor.exe</i> <i>bin\vrfOpdEditor.d.exe</i>	A version of the OPD Editor that loads plug-ins.
<i>opcodeGdbRT.dll</i> <i>opcodeGdbRTd.dll</i> <i>Opcode_D.dll</i> <i>Opcode.dll</i>	Installed in the bin, bin5, binHLA13, and binHLA1516 directories.
<i>./data/parameters/BHAVE</i>	Sample Lua scripts; PathData generation templates, and icons.
<i>./data/parameters/bhaveOpe</i>	OPE files for the B-HAVE Module-enabled entities.
<i>./data/parameters/B-HAVE.opd</i>	The B-HAVE Module object parameter database file.
<i>./data/scenarios/B-HAVE_Demos</i>	Sample scenarios that use B-HAVE Module tasks.
<i>./data/terrain/Makland.gdb</i> <i>./data/terrain/Makland.mtd</i>	Version of Makland with fixed road data. Makland MTD file that loads the B-HAVE path data.
<i>./data/terrain/TerraSim_Sample-Urban_PathData</i>	PathData files for the TerraSim_Sample_Urban terrain database.
<i>./flexlm10.8/KYNOGON.exe</i>	Executable for the B-HAVE Module license management.
<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/flexlm10.8* 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 User's Guide*.
3. Request a license using the license request procedure described in Chapter 2 of *VR-Forces User's 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.



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	2-6
Planning a Path	2-6
Path Following	2-7
Dynamic Avoidance	2-8
Spatial Reasoning	2-9
Dynamic Topological Analysis	2-10
B-HAVE Module Tasks	2-11
The Go To Task	2-11
The Wander Task	2-11
The Flee and Hide Tasks	2-11
Advanced Planning Capabilities	2-12
Glossary	2-12

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 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.

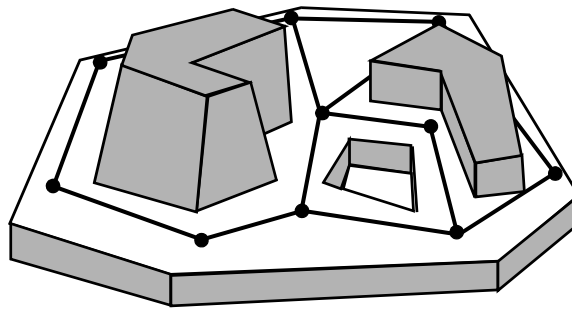


Figure 2-1. Spatial graph

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 a step (stairs, rocks, obstacles) 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.

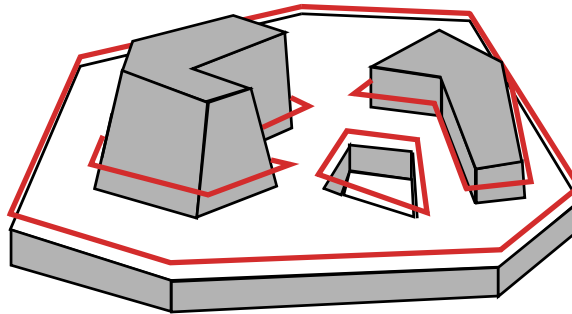


Figure 2-2. AI mesh

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.

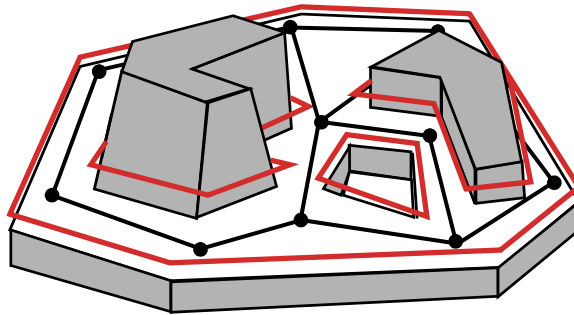


Figure 2-3. PathData (spatial graph and AI mesh)

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 does so using 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.

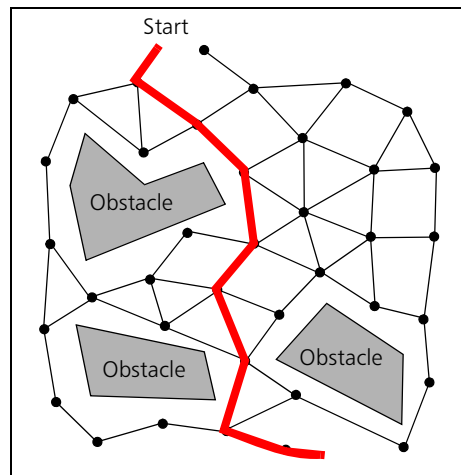


Figure 2-4. Planning a path through the spatial graph

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.

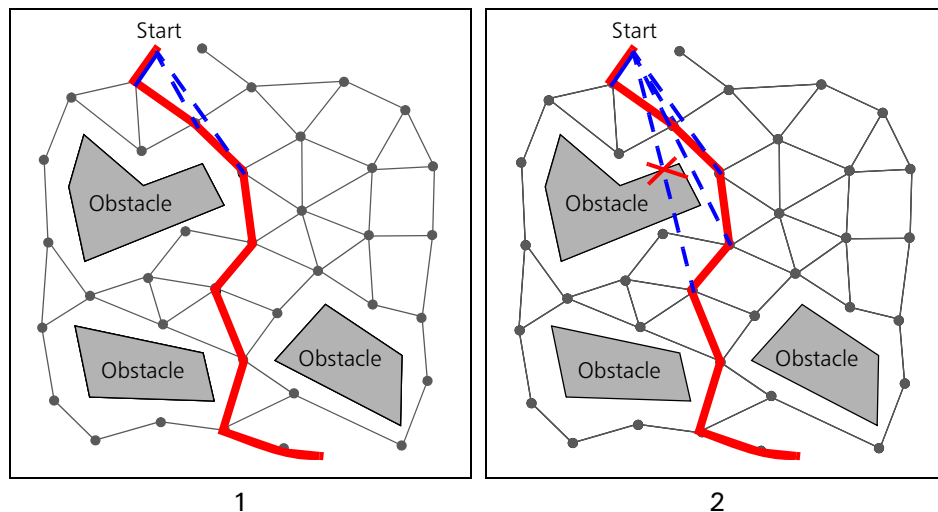


Figure 2-5. Checking next vertex

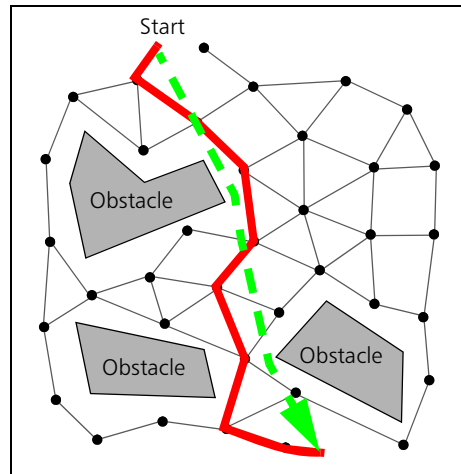


Figure 2-6. Final calculated path

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.

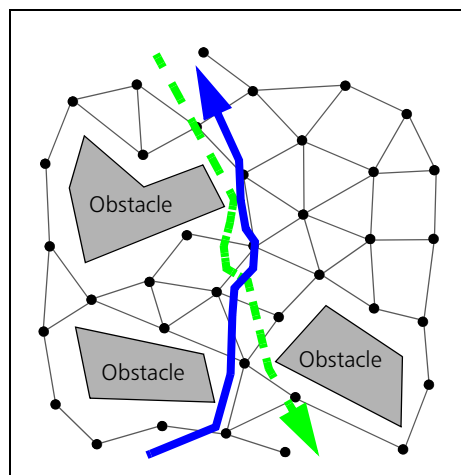


Figure 2-7. Collision avoidance

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.

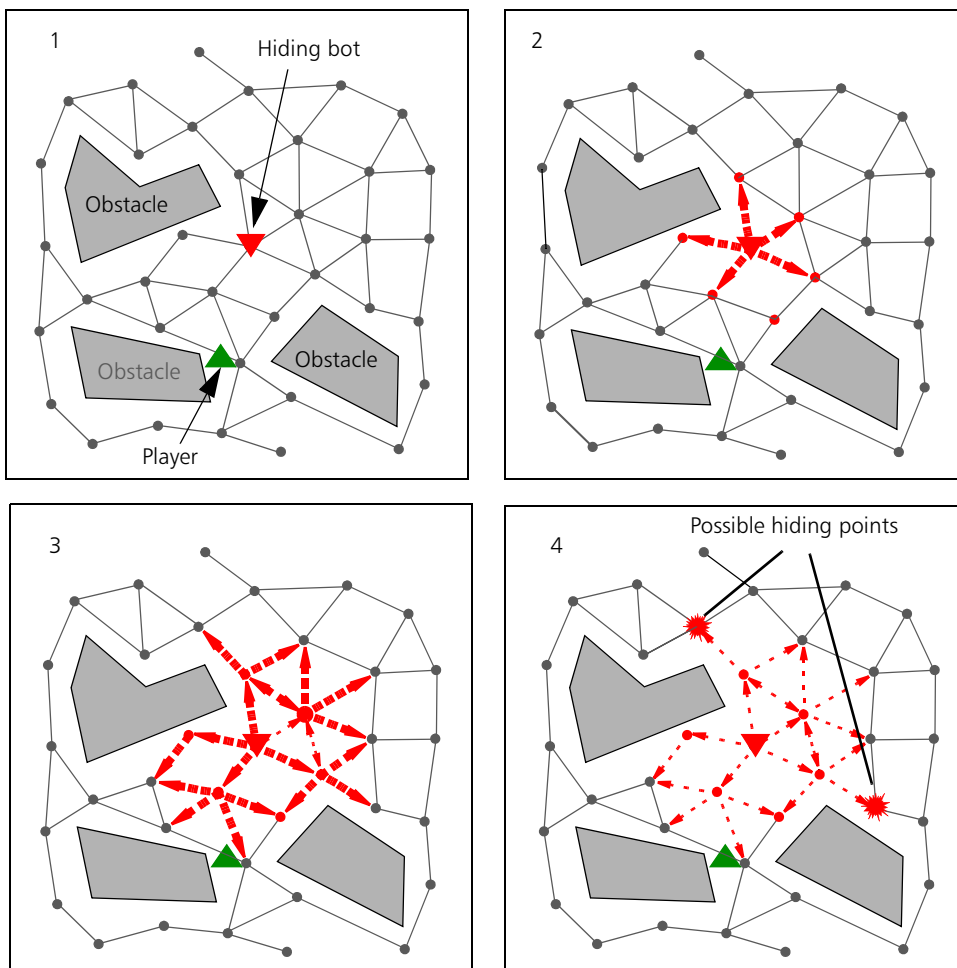


Figure 2-8. Dynamic topological analysis

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. 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*](#).

2.6. 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

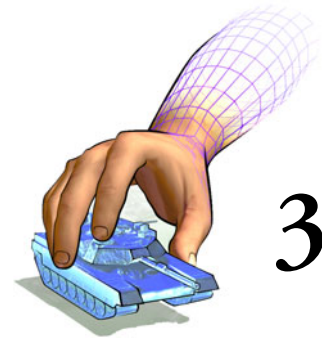
An edge is a connection between two vertices.

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. This chapter 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 User's Guide*.

Introduction	3-2
Varying the Complexity of PathData Configurations.....	3-3
Starting the TDB Tool.....	3-4
Loading a Terrain Database	3-5
Managing PathData Resources.....	3-6
Specifying Bounding Boxes	3-6
Specifying Seed Points	3-8
Specifying Seed Graphs	3-10
Configuring Generation of PathData	3-12
Creating a PathData Configuration	3-12
Editing PathData Configurations	3-17
Generating PathData	3-18
Monitoring the PathData Generation Process.....	3-20
Associating PathData with Entity Types.....	3-21
The BHAVEPathDataRecord	3-22
Testing PathData.....	3-23
Troubleshooting.....	3-24

3.1. Introduction

To use the B-HAVE Module tasks in VR-Forces, you must use a terrain that has had 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), 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 preferred movement 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. Take 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.

3.2. Starting the TDB Tool

- To start the TDB Tool in Windows, choose Start → All Programs → MAK Technologies → VR-Forces x.x → TDB Tool.
- To start the TDB Tool from the command line, run *vrforcesx.x/bin/tdbtool.exe*.

The TDB Tool window opens with a minimal menu and nothing loaded.

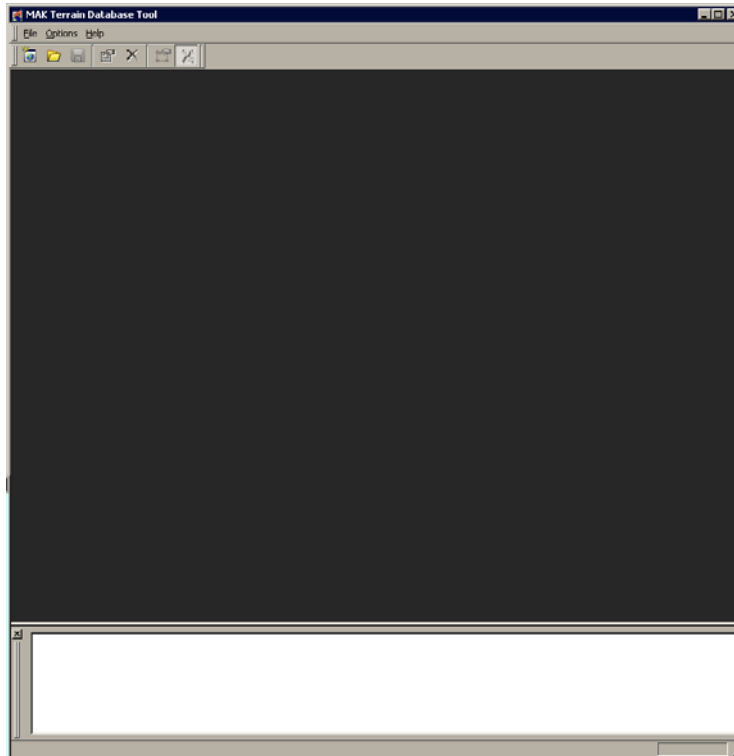


Figure 3-1. TDB Tool window at startup

3.2.1. 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 User's Guide*.

To load a terrain database:

1. Choose File → Open → Terrain Descriptor. The Open Terrain Descriptor dialog box opens.
2. 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, specify the bounding boxes for those areas in this step.

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 3-2). The Manage PathData Resources dialog box opens (Figure 3-3).



Figure 3-2. B-HAVE Module toolbar

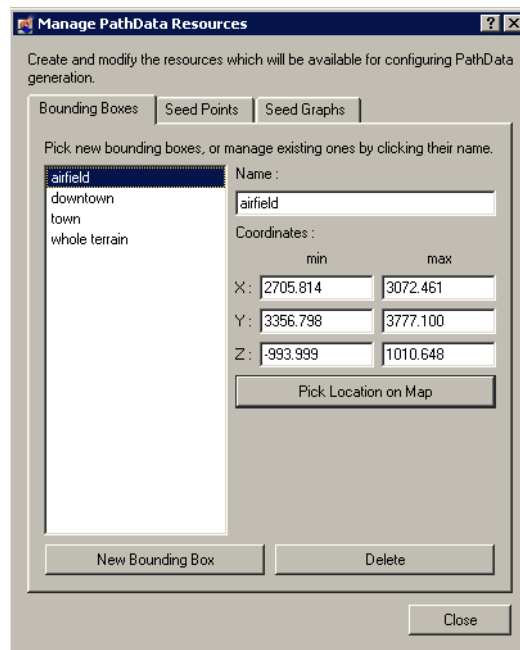


Figure 3-3. Manage PathData Resources dialog box

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

4. In the 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 on Map.
- 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 3-3](#)).
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 by hand, or 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 3-4).

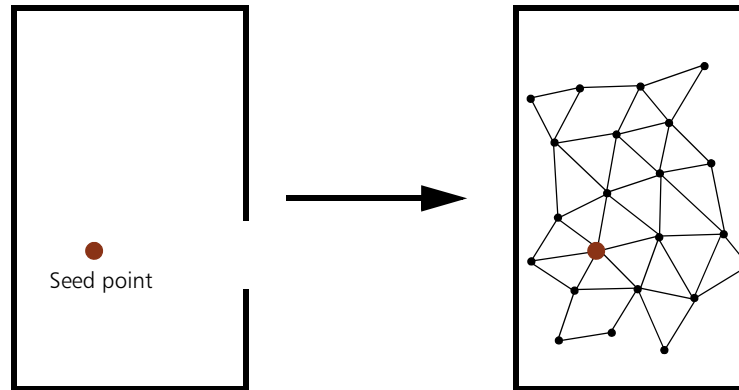


Figure 3-4. Generating a spatial graph from a seed point

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 paratrooper, 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 3-5).
3. Click New Seed Point.
4. In the 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 on Map.
- b. Click on the map to specify the seed point location.

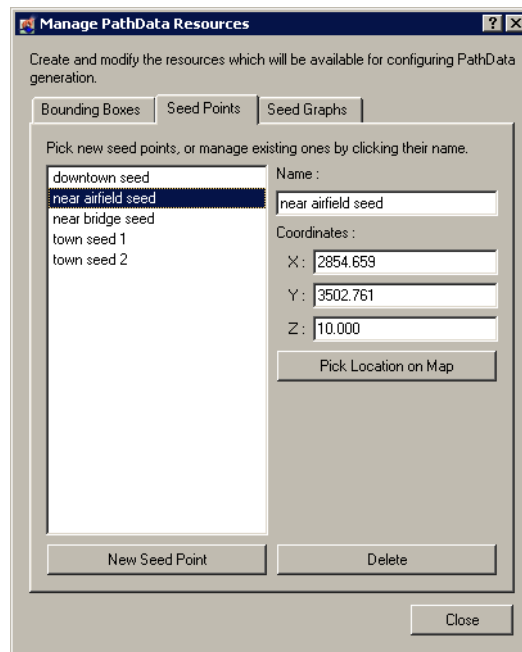


Figure 3-5. Manage PathData Resources dialog box, Seed Points tab

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 3-6](#), 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.

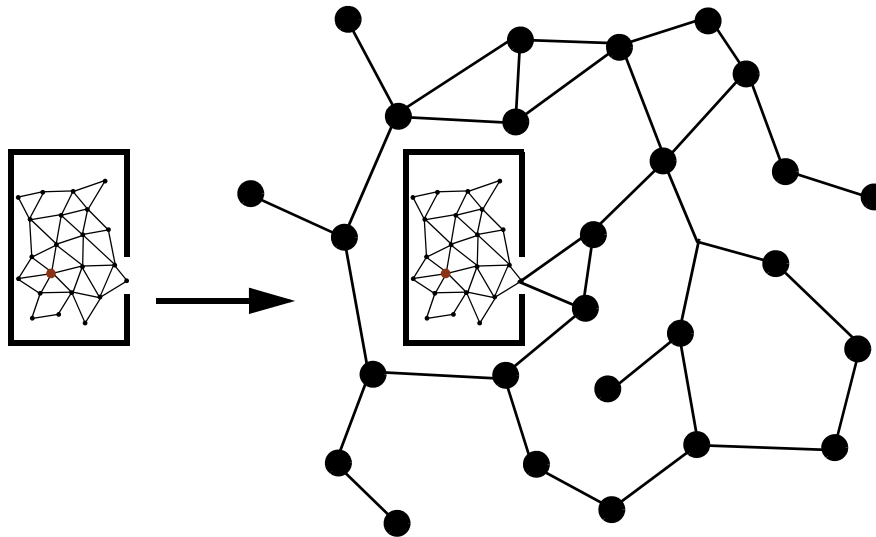


Figure 3-6. Building a spatial graph from a seed graph

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 3-7](#)).
3. Click Load.
4. Select any file with a *.pdl* extension. You may need to navigate to other PathData directories to find the files.

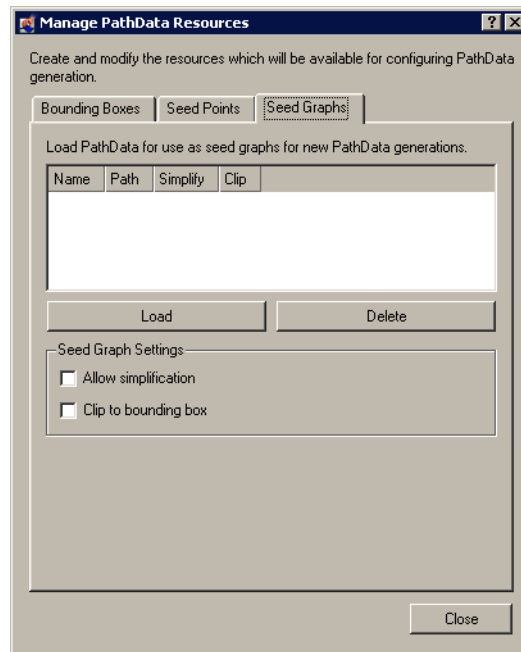


Figure 3-7. Manage PathData Resources dialog box, Seed Graphs tab

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 3-8](#)).

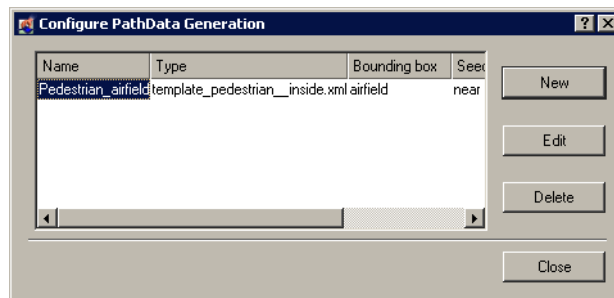


Figure 3-8. Configure PathData Generation dialog box

2. Click New. The Edit PathData Configuration dialog box opens ([Figure 3-9](#)).
3. 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 life-form 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 3-14.
4. Optionally, specify forbidden surfaces. For details, please see [“Filtering Surfaces,”](#) on page 3-15.

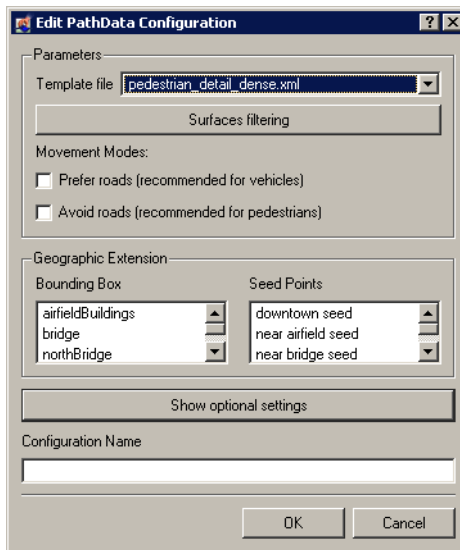


Figure 3-9. Edit PathData Configuration dialog box

5. Optionally, enable one or both movement modes. If you enable a movement mode 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 B-HAVE Set Movement Mode Set Data Request,”](#) on page 4-12.
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. 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 3-8.
8. 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 3-17.
9. Optionally, specify local zones. For details, please see [“Configuring Local Zones,”](#) on page 3-16.
10. In the Configuration Name box, type a name for this configuration.
11. Click Save. 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.

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 bugs 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, click Surfaces Filtering. The Forbidden Surfaces Settings dialog box opens ([Figure 3-10](#)).

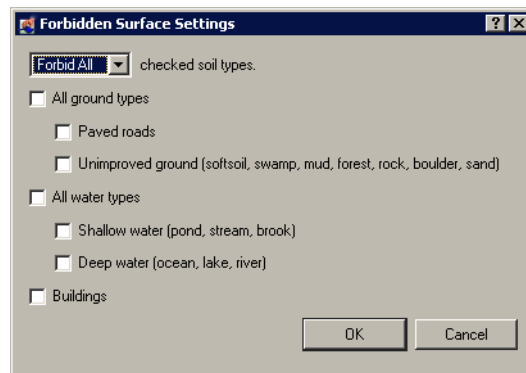


Figure 3-10. Forbidden Surface Settings dialog box

2. Select Forbid All or Allow Only from the 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 required in this step must have already been specified as PathData resources.

To configure a local zone.

1. In the Edit PathData Configuration dialog box, click Show Optional Settings. The dialog box expands (Figure 3-11).

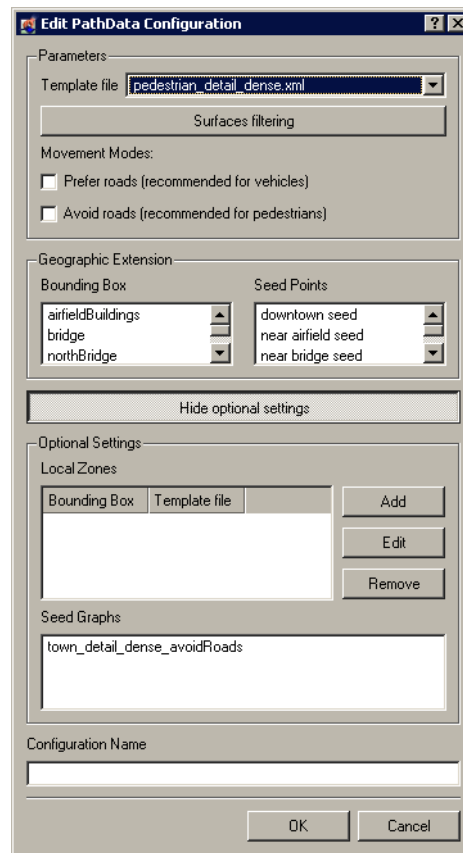


Figure 3-11. PathData configuration optional settings

2. In the Optional Settings 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 Apply. 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 seedgraph.

1. In the Edit PathData Configuration dialog box, click Show Optional Settings. The dialog box expands ([Figure 3-11](#)).
2. In the Seed Graphs list, select the seed graphs you want to include in the PathData configuration.

3.4.2. Editing PathData Configurations

You can edit PathData configurations. When a path data configuration that has already been used for a PathData generation is selected, the AI mesh and spatial graph are loaded and displayed in the TDB Tool when you select the configuration. [Figure 3-12](#) illustrates PathData for the Whole_Vehicles configuration in a portion of the Makland database.

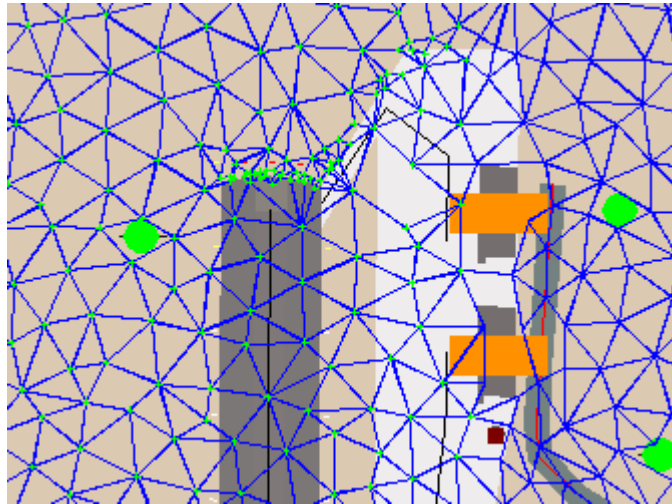


Figure 3-12. Vehicle PathData in TerraSim_SampleUrban database

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 3-8](#)).
2. Select the configuration that you want to edit.
3. Click Edit. The Edit PathData Configuration dialog box opens ([Figure 3-9](#)).
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 outputs four files for each PathData, all named after the configuration name.



- 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 `.mha`.

To generate PathData:

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

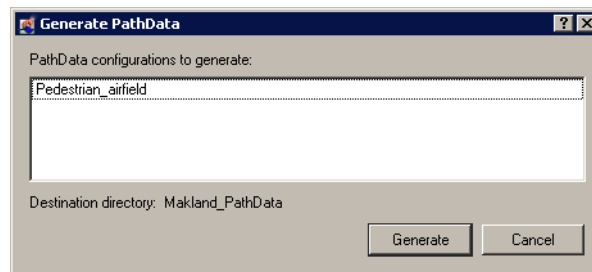


Figure 3-13. Start Batch PathData Generation dialog box

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 3-14).

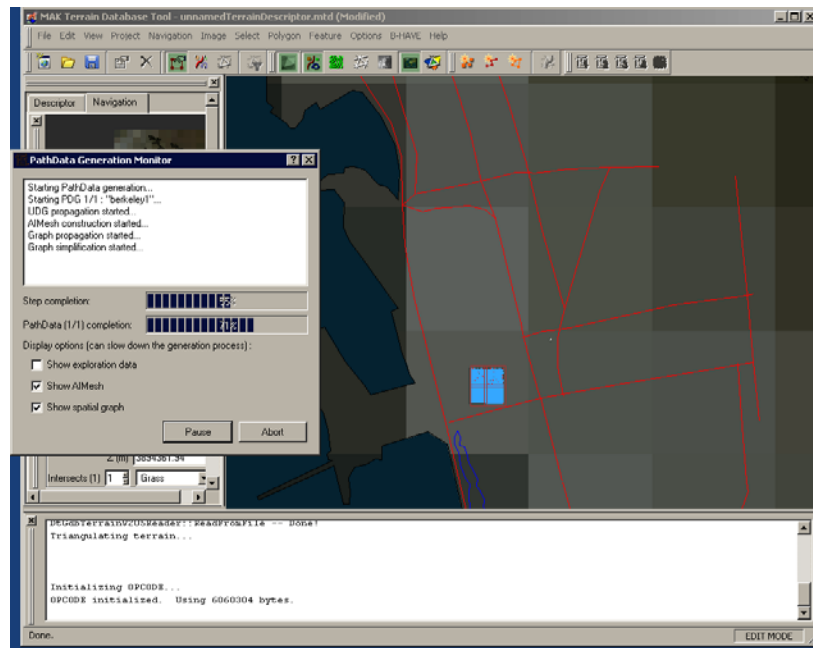


Figure 3-14. PathData generation

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 3-15). By default, exploration data are hidden, and the AI mesh and the Graph are shown. For an intensive generation, it is recommended that you not display this data, because it requires some processing power and slows down the generation.

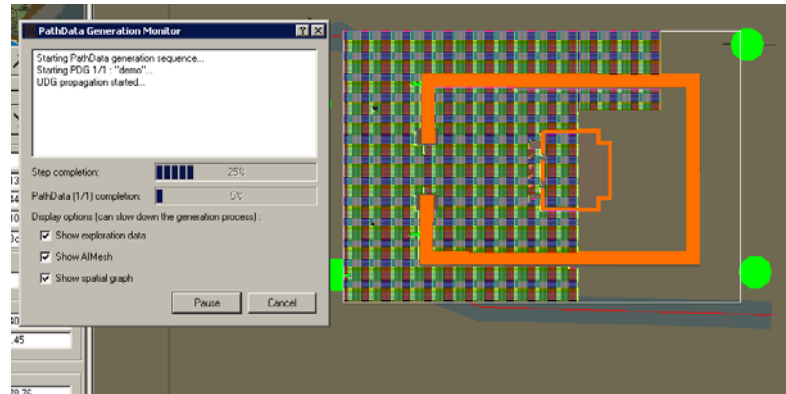


Figure 3-15. Monitoring PathData generation

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.mha* (AI Mesh)
- ♦ *path_data.fdl* (topological analysis data).

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.

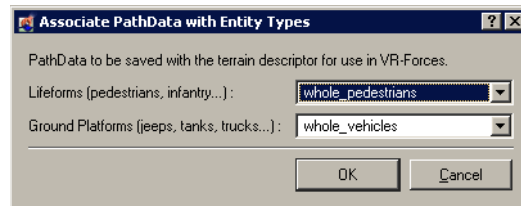


Figure 3-16. Associate PathData with Entity Types dialog box

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 be displayed 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, whole_lifeforms2.
PathDataDirectory	The directory in which the PathData being configured is stored. Can be a complete pathname or relative path, depending on the setting for PathIsRelativeToTerrain.
PathIsRelativeToTerrain	Specifies whether the value in PathDataDirectory is absolute path (0) or is relative to the <code>./data/terrain</code> directory (1).
ObjectType	The entity enumeration for the object type configured in this record: lifeform or ground platform.

3.7. Testing PathData

You can test the PathData that gets 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.

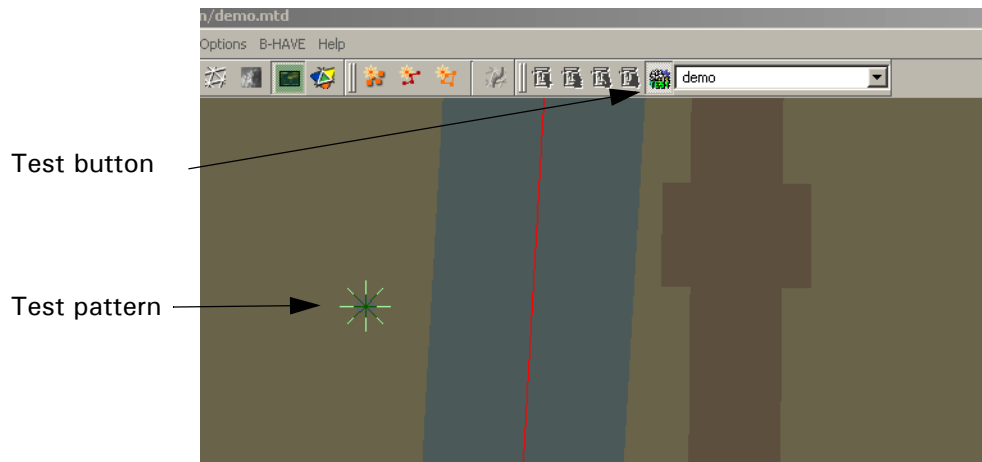
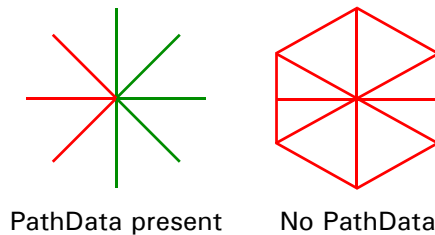


Figure 3-17. Testing PathData

2. Select a configuration.
3. Zoom in on the part of the terrain that you want to check (for instance a road), until you are able to visualize properly an object of the size of an entity.
4. Click on the screen. 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 3-18).



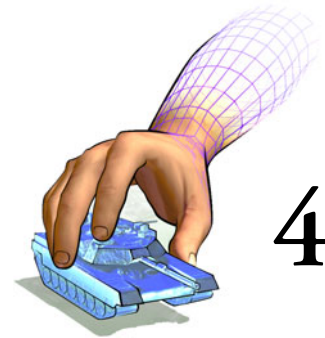
PathData present No PathData

Figure 3-18. 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 for a Scenario	4-2
Creating Groups of Entities	4-4
Adding an Area for Multiple Entity Creation.....	4-6
Specifying the Entities to Create.....	4-6
B-HAVE Module Tasks.....	4-7
The B-HAVE Flee From Task.....	4-7
The B-HAVE Follow Entity Task.....	4-8
The B-HAVE Hide From Task.....	4-8
The B-HAVE Move to Waypoint Task.....	4-9
The B-HAVE Move to Location Task.....	4-9
The B-HAVE Move Along Route Task.....	4-11
The B-HAVE Wander Task.....	4-11
The B-HAVE Set Movement Mode Set Data Request	4-12
The B-HAVE Set Script Set Data Request.....	4-13
The B-HAVE View Target Set Data Request.....	4-13
Viewing B-HAVE Module Data in the VR-Forces GUI.....	4-14
Viewing B-HAVE Module Data in the MÄK Stealth	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. It also installs a set of customized OPE files for the entities that support B-HAVE Module behaviors. 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.

In order 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](#).
- ♦ An OPD entry for the entity type in question that includes a B-HAVE Module controller (called the kynapse-local-automotive-controller.)

4.1.1. Disabling the B-HAVE Module for a Scenario

The B-HAVE Module installation includes a set of OPE files for all ground and life-form entities included with VR-Forces that have been modified to include the B-HAVE Module controller. These OPE files are referenced from the *bhave.opd* file, which is installed in *./data/parameters*. Once the B-HAVE Module is installed, all newly created scenarios will automatically have *bhave.opd* included as an additional object parameter database file in the scenario settings. Therefore, all ground and lifeform entities created in new scenarios will be ready to carry out B-HAVE tasks, if the PathData is present.

i

- ♦ If you have customized your OPE files and want those customization to be available to B-HAVE Module-enabled entities, you must make the same changes to the OPE files in *./data/parameters/bhaveOpe*, or add the B-HAVE Module controller to your customized OPE files.
 - ♦ You can use B-HAVE Module-enabled entities on terrains that do not have path data generated. In this case, the B-HAVE Module tasks are not available on the Task menu.
 - ♦ If a B-HAVE Module license is not available, VR-Forces runs normally, but B-HAVE Module tasks do not execute and are not available on the various menus.
-

If you do not want to have the B-HAVE Module-enabled entities in a particular scenario, when you create the scenario, in the Choose Terrain dialog box:

1. Click Scenario Settings.
2. Remove the *bhave.opd* file from the Additional OPD Files box ([Figure 4-1](#)).

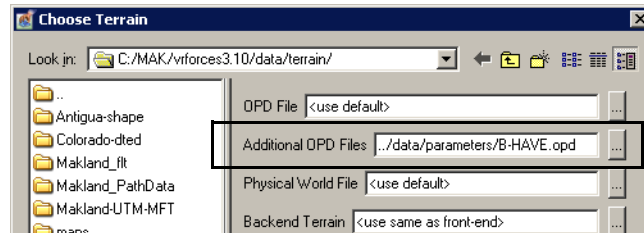


Figure 4-1. Choose Terrain dialog box

4.2. Creating Groups of 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. 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. Entities are added to the scenario as members of a selection group.



If a terrain does not have any PathData associated with it, you can place multiple entities anywhere 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 4-2](#), 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.

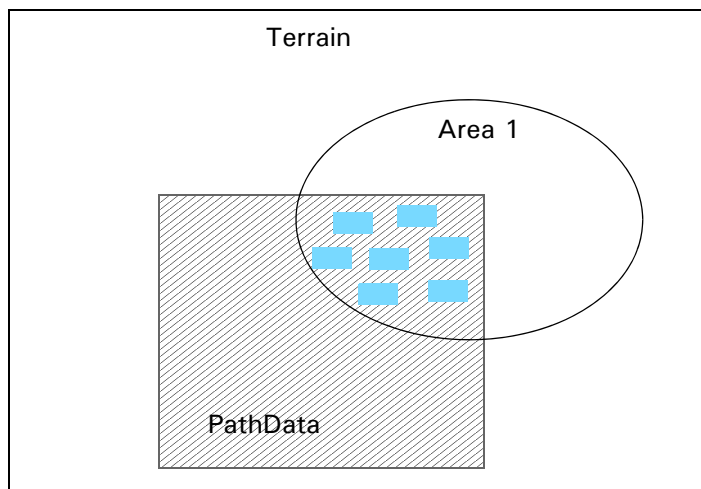


Figure 4-2. Placement of multiple entities

To add multiple entities to a scenario:

1. Choose B-HAVE → Create Multiple Entities. The Create Multiple Entities dialog box opens (Figure 4-3).

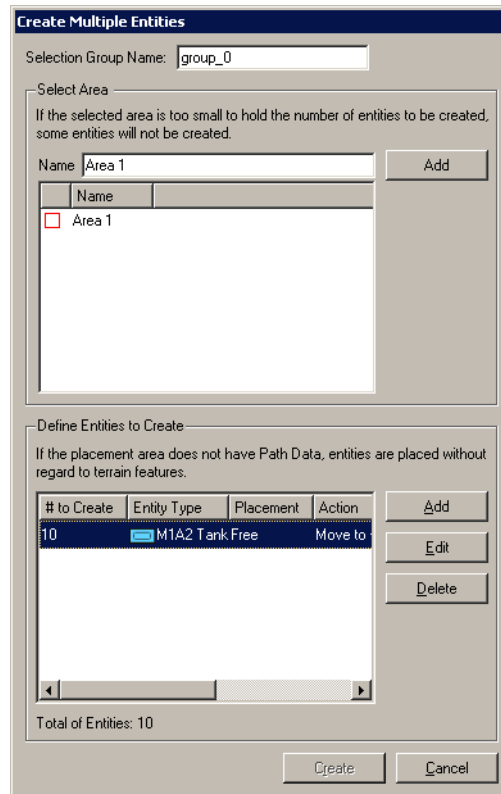


Figure 4-3. Create Multiple Entities dialog box

2. In the Selection Group Name text box, specify a name for the selection group, or accept the default name.
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 4-6.
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 4-6.
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 10, *Overlays and Graphical Objects*, in *VR-Forces User's 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 4-4).

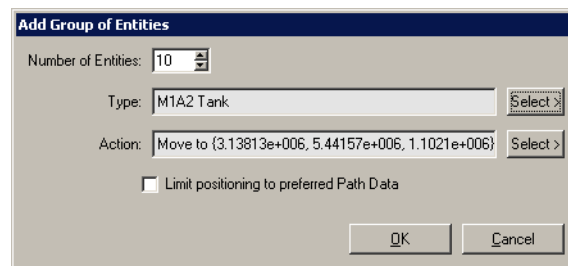


Figure 4-4. Add Group of Entities dialog box

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 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.
4. Optionally, click the Select button that is next to the 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. To restrict positioning of the created entities to locations along paths, select the Limit Positioning to Preferred Path Data check box.
6. Click OK. The entity set is added to the list in the Define Entities To Create group box.
7. Optionally, add additional entity sets.

4.3. 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.

4.3.1. The B-HAVE Flee From Task

The B-HAVE Flee From task causes an entity to try to maintain a specified distance from a specified entity (the pursuer.) If you select Stop When Hidden, the entity stops fleeing if it finds a hiding place, even if the hiding place is within 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 pursuer falls within the specified radius.



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

To assign a B-HAVE Flee From task:

1. Select the entity.
2. Choose Task → B-HAVE Flee From. The B-HAVE Flee From dialog box opens.
3. Optionally, filter the list of entities.
4. Select an entity in the list, or on the map.
5. Specify the distance, in meters, the entity should try to maintain from its pursuer.
6. Optionally, to have the fleeing entity stop when it is not within line-of-sight of the pursuer, select Stop When Hidden.
7. Click OK.

4.3.2. The B-HAVE Follow Entity Task

The B-HAVE 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 B-HAVE Follow Entity task:

1. Select the entity.
2. Choose Task → B-HAVE Follow Entity. The cursor changes to select mode and the Follow Entity dialog box opens.
3. Optionally, filter the entity list.
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.

4.3.3. The B-HAVE Hide From Task

The B-HAVE 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 pursuer is in line-of-sight.



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

To assign a B-HAVE Hide From task:

1. Select the entity.
2. Choose Task → B-HAVE Hide From. The B-HAVE Hide From dialog box opens.
3. Optionally, filter the list of entities.
4. Select the pursuers. (**Ctrl**-click for multiple selections.)
5. Click OK.

4.3.4. The B-HAVE Move to Waypoint Task

The B-HAVE 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 B-HAVE Move to Waypoint task:

1. Select the entity.
2. Choose Task → B-HAVE 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.

4.3.5. The B-HAVE Move to Location Task

The B-HAVE 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 B-HAVE Move To Location task:

1. Select the entity to which you want to assign the task.
2. Choose Task → B-HAVE Move To Location. The Move to Location dialog box opens.
3. Type the coordinates for the location or click on the map to specify the location. If you want to specify a point inside a building, you must specify an altitude. For an explanation of how to specify the altitude, please see [“Specifying a Location Inside a Building,”](#) on page 4-10.
4. Click OK.

Specifying a Location Inside a Building

When you select a location on the map for a B-HAVE Move to Location task, the selected point always has the altitude (Z) of the highest elevation of the terrain at that point. Therefore, if you click on a building, you select a point on the roof.

To specify a point inside a building, specify the location in either of the following ways:

- Type the coordinates of the location in the X, Y, and Z boxes.
- Click a point on the map, thereby entering the X, Y, and Z values of a point on the roof. Then type a negative value in the “m Above The Terrain” box. This value is the distance below the top of the building of the floor on which you want the entity to move to. The resulting value ($Z + [x]$ m above the ground) must be within 1.3 meters of the floor to be valid, as illustrated in (Figure 4-5).

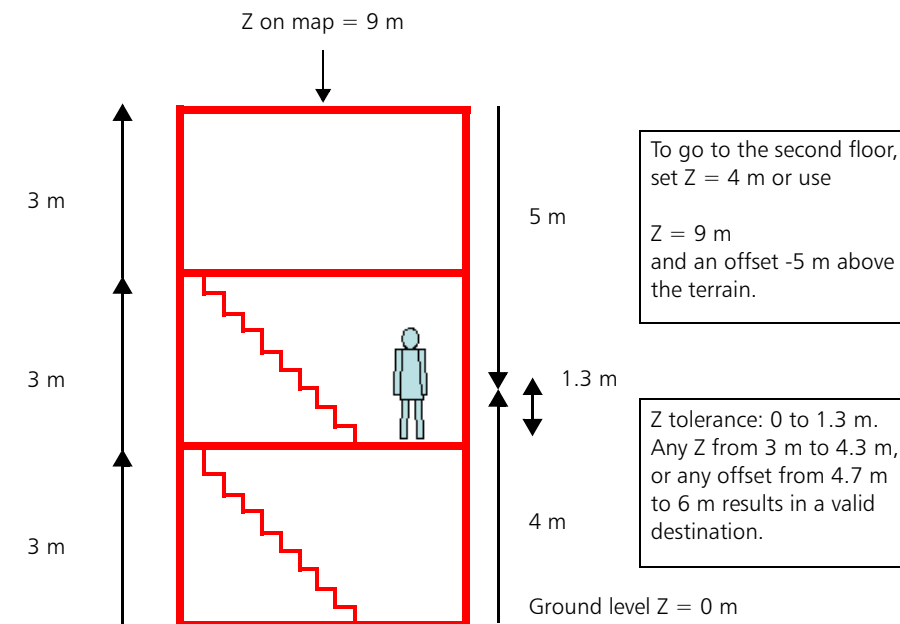


Figure 4-5. Specifying a location within a building

4.3.6. The B-HAVE Move Along Route Task

The B-HAVE 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 B-HAVE Move Along Route task:

1. Select the entity.
2. Choose Task → B-HAVE 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.

4.3.7. The B-HAVE Wander Task

The B-HAVE 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 B-HAVE Wander task:

1. Select the entity.
2. Choose Task → B-HAVE Wander. The B-HAVE 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 Coverage group box, select Unrestricted or Restricted To. If you select Restricted to, type the name of an area or select it in the list or on the map.
5. Click OK.

4.3.8. The B-HAVE Set Movement Mode Set Data Request

The B-HAVE 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 3-12.

When an entity is set to Prefer Roads and the terrain was created with preferred movement enabled, 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 its OPE file.

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

1. Select the entity.
2. Choose Set → B-HAVE Set Movement Mode. The B-HAVE 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.

4.3.9. The B-HAVE Set Script Set Data Request

The B-HAVE 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](#).



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.

4.3.10. The B-HAVE 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, check Clear View Target to remove the requirement.
5. Click OK.

4.4. Viewing B-HAVE Module Data in the VR-Forces GUI

When B-HAVE-enabled entities are given tasks in VR-Forces, they publish routes indicating the path that was chosen for movement. These routes are for easy visual feedback and debugging. An entity will not move directly along the indicated path because of the runtime path smoothing (described in “[Path Following](#),” on page 2-7). 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 GUI, 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 OPE file for each entity type you are using. This can improve performance for scenarios with large numbers of entities that are using the B-HAVE Module.

4.5. Viewing B-HAVE Module Data in the MÄK Stealth

The B-HAVE Module plug-in outputs graphical information that can be read by the MÄK Stealth if it runs on the same machine (the plug-in uses shared memory.) This is 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 “[Viewing B-HAVE Module Data in the MÄK Stealth](#),” on page 5-9.

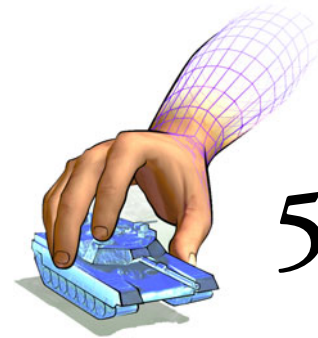


To read the B-HAVE Module output, the Stealth must load a plug-in. The plug-in is installed in the proper Stealth directory as part of the B-HAVE Module installation.

4.6. Example Scenarios

The B-HAVE Module includes the following example scenarios:

- ♦ **Crowd:** 100 civilians wander randomly around the city in the 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 SampleUrban. DI 1 has no plan. You can task him to move to some of the created waypoints to demonstrate dynamic path planning.
- ♦ **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.
- ♦ **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.



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
Creating a New Script	5-2
Editing a Script	5-3
Deleting a Script.....	5-3
Writing Scripts.....	5-3
Using B-HAVE Module Tasks in Scripts	5-4
Considerations for Writing and Using Lua Scripts.....	5-4
Example Script	5-6
Sending Radio Messages	5-7
Custom Checkpointing.....	5-8
Viewing B-HAVE Module Data in the MÄK Stealth	5-9
Drawing PathData Around Entities.....	5-9
B-HAVE Module Functions.....	5-10
Entity Functions.....	5-10
Behavior Functions.....	5-13
Entity List Functions.....	5-23
Perception Functions.....	5-24
Utility Functions	5-26

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.

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.

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. Creating a New Script

The B-HAVE Module includes a simple text editor for writing scripts. If you prefer, you can write a script in some other text editor and paste it into the B-HAVE editor.

To create a new Lua script:

1. Choose B-HAVE → Scripts. The B-HAVE Scripts window opens. This window lists all the available scripts.
2. Click New. The B-HAVE Script window, a blank text editor, opens.
3. Type the script or paste in a previously written script.
4. Click Save. The Save Script As dialog box opens.
5. Type a name for the script and click OK.
6. In the B-HAVE Script window, click Close. The new script is listed in the B-HAVE Scripts window.

5.2.2. Editing a Script

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

5.2.3. 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 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, as 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. However, global tables are not saved. For more information about checkpoint, please see [“Custom Checkpointing,”](#) on page 5-8

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
    AIUtilDebugMessage("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 *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 *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()
    AIUtilDebugMessage("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()
    AIUtilDebugMessage("test: PostLoadCheckpoint.")
    if initialized == true then
        AIPathwaySetVRFRoute(routeName)
        AIUtilDebugMessage("test: setting current route target point " ..
            currentRouteTargetPoint)
        AIPathwaySetCurrentPointIndex(currentRouteTargetPoint)
    end
end
```

5.4. Viewing B-HAVE Module Data in the MÄK Stealth

The B-HAVE Module plug-in outputs graphical information that can be read by the MÄK Stealth if it runs on the same machine.

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 ♦ 7: Shoot ♦ 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:	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 coordinates of the entity position.
Description:	Gets the entity position.
Function:	CEntity* AIEntityGetSelf()
Input parameters:	None.
Output:	Current entity.
Description:	Gets the current entity.
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.

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

Function:	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.

5.5.2. Behavior Functions

Table 5-4 describes functions that control entity behaviors.

Table 5-4: Behavior functions (Sheet 1 of 10)

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:	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:	number AIFleeGetComputeDestInterval()
Input parameters:	None.
Output:	The interval, in seconds, between two destination computations.
Description:	Retrieves the interval between two destination computations.
Function:	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 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:	AIFleeGetSpeed()
Input parameters:	None.
Output:	Returns a number.
Description:	Reports the ordered speed for a B-HAVE Flee task.

Table 5-4: Behavior functions (Sheet 2 of 10)

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.
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()
Input parameters:	A number.
Output:	None.
Description:	Sets the ordered speed for a B-HAVE Flee task.
Function:	AIFleeSetStopWhenHidden(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.

Table 5-4: Behavior functions (Sheet 3 of 10)

Function:	bool AIFollow()
Input parameters:	None.
Output:	Returns 1 if the destination is reachable; 0 otherwise.
Description:	Makes the entity follow its target.
Function:	number AIFollowGetAngleFromEntity()
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 AIFollowGetDestination()
Input parameters:	None.
Output:	(x,y,z): The 3D coordinates of the destination.
Description:	Retrieves the destination of the follower.
Function:	number AIFollowGetDistFromEntity()
Input parameters:	None.
Output:	The distance to the followed entity.
Description:	Retrieves the distance to the followed entity.
Function:	CEntity* AIFollowGetEntityToFollow()
Input parameters:	None.
Output:	The followed entity.
Description:	Retrieves the followed entity.
Function:	AIFollowGetSpeed()
Input parameters:	None.
Output:	Returns a number.
Description:	Reports the ordered speed for a B-HAVE Follow task.
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 to the followed entity. Default: 180.0 degrees (that is the follower is behind its target).

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

Function:	AIFollowSetDistFromEntity(number)
Input parameters:	The distance to the followed entity.
Output:	None.
Description:	Sets the distance 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()
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().
Function:	number AIGotoGetDistanceLeft()
Input parameters:	None.
Output:	The distance to the destination.
Description:	Retrieves the distance to the destination considering the entity path.
Function:	AIGotoGetSpeed()
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 AIGotoGetVRFDestination()
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 AIGoto().

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

Function:	bool AIGotoHasArrived()
Input parameters:	None.
Output:	Returns 1 if the entity has arrived; 0 otherwise.
Description:	Identifies whether the destination has been reached.
Function:	AIGotoSetDestination(number x, number y, number z)
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 AIGoto().
Function:	AIGotoSetSpeed(number)
Input parameters:	A number.
Output:	None.
Description:	Sets the ordered speed for a B-HAVE Move To task.
Function:	AIGotoSetVRFDestination(number x, number y, number z)
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 AIGoto().
Function:	AIGotoSetVRFWaypoint(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 AIGoto().
Function:	bool AIHide()
Input parameters:	None.
Output:	Returns 1 if the entity can hide; 0 otherwise.
Description:	Makes the entity hide from its dangerous entities.
Function:	AIHideAddDangerousEntity(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.

Table 5-4: Behavior functions (Sheet 6 of 10)

Function:	number x, number y, number z AIHideGetDestination()
Input parameters:	None.
Output:	(x,y,z): The 3D coordinates of the hiding place.
Description:	Identifies the entity destination.
Function:	string AIHideGetGraph()
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 AIHideGetResearchType()
Input parameters:	None.
Output:	Returns 1 if the entity maximizes the distance between the hiding place and the dangerous entities. 2 if the entity minimizes the distance from its current position to the hiding place. 3 if the entity minimizes the length of the path to the hiding place.
Description:	Identifies the entity hiding place selection mode.
Function:	AIHideGetSpeed()
Input parameters:	None.
Output:	Returns a number.
Description:	Reports the ordered speed for a B-HAVE Hide task.
Function:	bool AIHideHasArrived()
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.
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.

Table 5-4: Behavior functions (Sheet 7 of 10)

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:	AIHideSetResearchType(number)
Input parameters:	1 if the entity maximizes the distance between the hiding place and the dangerous entities. 2 if the entity minimizes the distance from its current position to the hiding place. 3 if the entity minimizes the length of the path to the hiding place.
Output:	None.
Description:	Sets the entity hiding place selection mode.
Function:	AIHideSetSpeed()
Input parameters:	A number.
Output:	None.
Description:	Sets the ordered speed for a B-HAVE Hide task.
Function:	bool AIPathway()
Input parameters:	None.
Output:	Returns 1 if the pathway can be followed. 0 otherwise.
Description:	Makes the entity follow the pathway.
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.

Table 5-4: Behavior functions (Sheet 8 of 10)

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.
Function:	bool AIPathwayIsPathwayFinished()
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 object to follow.

Table 5-4: Behavior functions (Sheet 9 of 10)

Function:	AISetMovementMode(string)
Input parameters:	None.
Output:	Returns “free” if the entity can move anywhere. Returns “preferred” if the entity should try to stay on preferred graph areas.
Description:	Sets the pathfinding mode with respect to PathData's preferred seed graph information.
Function:	AIVRFRadioSendMessage(string, entity name)
Input parameters:	Text string, and optionally, target entity.
Output:	None.
Description:	Sends the text string as a radio message.
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 coordinates of the destination.
Description:	Retrieves the destination of the wandering entity.
Function:	AIWanderGetSpeed()
Input parameters:	None.
Output:	Returns a number.
Description:	Reports the ordered speed 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.

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

Function:	AIWanderSetSpeed()
Input parameters:	A number.
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 Kynapse 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* AIPerceptionGetNearestObstacleSeenEnemy()
Input parameters:	None.
Output:	The nearest Kynapse 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* AIPerceptionGetNearestObstacleSeenEntity()
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* AIPerceptionGetNearestSeenEnemy()
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 1 if the entity is behind an obstacle; 0 otherwise.
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 3)

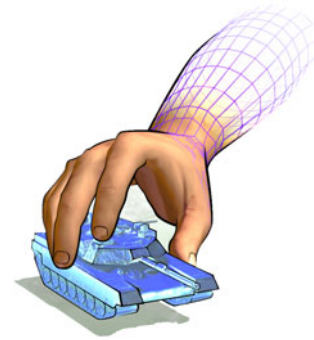
Function:	AISetDrawPathData(name)
Input parameters:	Name is Lifeform or GroundPlatform.
Output:	None.
Description:	Specifies that PathData be drawn in the MÄK Stealth for the specified entity type.
Function:	AISetDrawEntityCentric(entity_name)
Input parameters:	If no parameter, draw PathData for the 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:	AIUtilBHAVE2VRF(x, y, z)
Input parameters:	B-HAVE coordinates.
Output:	VR-Forces coordinates.
Description:	Transforms B-HAVE coordinates to VR-Forces coordinates.
Function:	bool AIUtilCheckVisible (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.
Output:	Returns 1 if (x2,y2,z2) is visible from (x1,y1,z1); 0 otherwise.
Description:	Tests visibility between two points.
Function:	AIUtilDebugMessage(string)
Input parameters:	Message to be written.
Output:	None.
Description:	Writes a message to the B-HAVE output.
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.

Table 5-7: Utility functions (Sheet 2 of 3)

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.
Function:	AIUtilGetVRForceExerciseStartTime()
Input parameters:	None.
Output:	Number.
Description:	Returns the start time for the current exercise.
Function:	AIUtilGetVRForceExerciseTime()
Input parameters:	None.
Output:	Number.
Description:	Returns the current VR-Forces exercise time.

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

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.
Function:	number fraction, number x, number y, number z, number Nx, number Ny, number Nz AIUtilTraceLine (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.
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:	AIUtilVRF2BHAVE(x, y, z)
Input parameters:	VR-Forces coordinates.
Output:	B-HAVE coordinates.
Description:	Transforms VR-Forces coordinates to B-HAVE coordinates.



Index

A

- adding, seed graph 3-10
- advanced planning 2-12
- AI Mesh 2-12
- AI mesh 2-2, 2-3, 2-4, 3-20
- AIEntityGetCurrentAgent() function 5-10
- AIEntityGetDirection() function 5-10
- AIEntityGetIndex() function 5-10
- AIEntityGetMaxSpeed() function 5-11
- AIEntityGetName() function 5-11
- AIEntityGetOrderedSpeed() function 5-11
- AIEntityGetOrientation() function 5-11
- AIEntityGetPosition() function 5-11
- AIEntityGetSelf() function 5-11
- AIEntityGetSpeed() function 5-11
- AIEntityIsUnderFire() function 5-12
- AIEntitySetMaxSpeed() function 5-12
- AIEntitySetRulesOfEngagement() function 5-12
- AIEntitySetSpeed() function 5-12
- AIEntitySetSteering() function 5-12
- AI flee() function 5-4, 5-13
- AI fleeAddPointToFlee() function 5-13
- AI fleeGetComputeDestInterval() function 5-13
- AI fleeGetDangerRadius() function 5-13
- AI fleeGetDestination() function 5-13
- AI fleeGetPathLength() function 5-13
- AI fleeGetStopWhenHidden() function 5-14
- AI fleeResetPointsToFlee() function 5-14
- AI fleeSetComputeDestInterval() function 5-14
- AI fleeSetDangerRadius() function 5-14
- AI fleeSetPathLength() function 5-14
- AI fleeSetStopWhenHidden() function 5-14
- AI follow() function 5-4, 5-15
- AI followGetAngleFromEntity() function 5-15
- AI followGetDestination() function 5-15
- AI followGetDistFromEntity() function 5-15
- AI followGetEntityToFollow() function 5-15
- AI followHasArrived() function 5-15
- AI followSetAngleFromEntity() function 5-15
- AI followSetDistFromEntity() function 5-4, 5-16
- AI followSetEntityToFollow() function 5-4, 5-16
- AI goto() function 5-4, 5-16
- AI gotoGetDestination() function 5-16
- AI gotoGetDistanceLeft() function 5-16
- AI gotoGetVRFDestination() function 5-16
- AI gotoHasArrived() function 5-17
- AI gotoSetDestination() function 5-17
- AI gotoSetSpeed() function 5-17
- AI gotoSetVRFDestination() function 5-17
- AI gotoSetVRFWaypoint() function 5-17
- AI hide() function 5-4, 5-17
- AI hideAddDangerousEntity() function 5-17
- AI hideGetDestination() function 5-18
- AI hideGetGraph() function 5-18
- AI hideGetResearchType() function 5-18
- AI hideHasArrived() function 5-18
- AI hideIsHidden() function 5-18
- AI hideResetDangerousEntities() function 5-18
- AI hideSetGraph() function 5-19
- AI hideSetResearchType() function 5-19
- AI pathway() function 5-4, 5-19
- AI pathwayGetCurrentPointIndex() function 5-19
- AI pathwayGetDestination() function 5-19
- AI pathwayGetNbPoints() function 5-19
- AI pathwayGetPathway() function 5-20
- AI pathwayGetReverseOrder() function 5-20
- AI pathwayIsCurrentPointReached() function 5-20
- AI pathwayIsPathwayFinished() function 5-20
- AI pathwayResetPathway() function 5-20
- AI pathwaySetReverseOrder() function 5-20

AIPathwaySetVRFRRoute() function 5-20
AIPerceptionGetDistance() function 5-24
AIPerceptionGetNearestEnemy() function 5-24
AIPerceptionGetNearestEntity() function 5-24
AIPerceptionGetNearestObstacleSeenEntity()
function 5-24
AIPerceptionGetNearestSeenEnemy() function 5-
24
AIPerceptionGetNearestSeenEntity() function 5-
25
AIPerceptionIsEnemy() function 5-25
AIPerceptionIsSeen() function 5-25
AIPerceptionIsVisible() function 5-25
AIReportReflexActivationState() 5-3
AISetMovementMode() function 5-21
AIUtilBHAVE2VRF() function 5-5, 5-26
AIUtilCheckVisible() function 5-26
AIUtilDebugMessage() function 5-26
AIUtilDrawLine() function 5-9, 5-26
AIUtilDrawPath() function 5-9, 5-27
AIUtilDrawPathway() function 5-27
AIUtilDrawPoint() function 5-9, 5-27
AIUtilGetCurrentTime() function 5-27
AIUtilGetLastError() function 5-27
AIUtilGetVRFXerciseStartTime() function 5-27
AIUtilGetVRFXerciseTime() function 5-27
AIUtilMakeVectorDeg() function 5-28
AIUtilResetLastError() function 5-28
AIUtilTraceLine() function 5-28
AIUtilVecToAnglesDeg() function 5-28
AIUtilVRF2BHAVE() function 5-5, 5-28
AIWander() function 5-4, 5-21
AIWanderGetDestination() function 5-21
AIWanderSelectRandomDestination() function 5-
21
AIWanderSetVRFArea() function 5-22
AIWorldGetEntityByIndex() function 5-23
AIWorldGetEntityByName() function 5-23
altitude, specifying for B-HAVE Move to Location
task 4-10
architecture 1-2
area, restricting wandering to 4-11
Associate PathData with Entity Types dialog box
3-21
Avoid Roads 4-12
avoidance, collision 2-8

B

B-HAVE Flee From task 4-7

B-HAVE Follow Entity task 4-8
B-HAVE Hide From task 4-8
B-HAVE Move Along Route task 4-11
B-HAVE Move to Location task 4-9
B-HAVE Move To Waypoint task 4-9
B-HAVE Scripts window 5-2
B-HAVE Set Movement Mode set data request 4-
12
B-HAVE Set Script set data request 4-13
B-HAVE Wander task 4-11
bhave.opd 4-2
BHAVEPathDataRecord 3-22
bounding box
clipping seed graph to 3-11
creating 3-6
editing 3-7
selecting for PathData generation 3-13
Brains for Human Activity in Virtual Environ-
ments 1-2

C

CEntity 5-4
checkpointing, custom 5-8
choosing, PathData template 3-14
clipping, seed graph to bounding box 3-11
collision avoidance 2-8
Configure PathData Generation dialog box 3-12,
3-17
configuring
generation of PathData 3-12
local zone 3-16
seed graph 3-17
continuously active script 5-3
coordinate, transforming 5-28
coordinate system, transforming VR-Forces to B-
HAVE 5-5
creating
bounding box 3-6
multiple entities 4-4
PathData configuration 3-12
script 5-2
seed point 3-8

D

data, path 2-4
deleting, script 5-3
dense graph density 3-14
density 3-3

- graph 3-14
 - spatial graph 2-5, 3-14
- detail 3-3
 - PathData 2-5, 3-14
 - template 3-14
- dialog box
 - Associate PathData with Entity Types 3-21
 - Configure PathData Generation 3-12, 3-17
 - Edit PathData Configuration 3-12, 3-17
 - Manage PathData Resources 3-6, 3-8, 3-10
 - PathData Generation Monitor 3-20
 - Set Movement Mode 4-12
 - Start Batch PathData Generation 3-18
- directory, installed 1-3
- disabling
 - B-HAVE Module tasks 4-2
 - script in plan 5-3
- displaying, path planning routes 4-14
- drawing, PathData in Stealth 5-9
- DTA 2-10
- dynamic avoidance 2-8, 2-11
- dynamic topological analysis 2-9, 2-10

E

- edge 2-2, 2-12, 3-10
 - length of 2-5, 3-14
- Edit PathData Configuration dialog box 3-12, 3-17
- editing
 - bounding box 3-7
 - PathData configuration 3-17
 - script 5-3
- entity
 - assigning tasks to multiple 4-4
 - configuring movement mode 4-12
 - controlling from script 5-5
 - creating multiple 4-4
 - drawing PathData around 5-9
 - looping through lists of 5-4
 - sending radio messages to 5-7
- entity type, lifeform and ground platform 3-14
- example
 - scenarios 4-15
 - scripts 5-6
 - terrain with PathData 1-3
- executing, script 4-13

F

- facing object 4-13
- FDL file extension 3-18
- file
 - extension, PDL 3-10
 - installed 1-3
- filtering
 - surface types in PathData configuration 3-12
 - surfaces 3-15
- Flee From task 2-11, 4-7
- FLEXlm 1-5
- Follow Entity task 4-8
- following, paths 2-7
- forbidden surface 3-12
- function
 - AIEntityGetCurrentAgent() 5-10
 - AIEntityGetDirection() 5-10
 - AIEntityGetIndex() 5-10
 - AIEntityGetMaxSpeed() 5-11
 - AIEntityGetName() 5-11
 - AIEntityGetOrderedSpeed() 5-11
 - AIEntityGetOrientation() 5-11
 - AIEntityGetPosition() 5-11
 - AIEntityGetSelf() 5-11
 - AIEntityGetSpeed() 5-11
 - AIEntityIsUnderFire() 5-12
 - AIEntitySetMaxSpeed() 5-12
 - AIEntitySetRulesOfEngagement() 5-12
 - AIEntitySetSpeed() 5-12
 - AIEntitySetSteering() 5-12
 - AIxFlee() 5-4, 5-13
 - AIxFleeAddPointToFlee() 5-13
 - AIxFleeGetComputeDestInterval() 5-13
 - AIxFleeGetDangerRadius() 5-13
 - AIxFleeGetDestination() 5-13
 - AIxFleeGetPathLength() 5-13
 - AIxFleeGetStopWhenHidden() 5-14
 - AIxFleeResetPointsToFlee() 5-14
 - AIxFleeSetComputeDestInterval() 5-14
 - AIxFleeSetDangerRadius() 5-14
 - AIxFleeSetPathLength() 5-14
 - AIxFleeSetStopWhenHidden() 5-14
 - AIFollow() 5-4, 5-15
 - AIFollowGetAngleFromEntity() 5-15
 - AIFollowGetDestination() 5-15
 - AIFollowGetDistFromEntity() 5-15
 - AIFollowGetEntityToFollow() 5-15
 - AIFollowHasArrived() 5-15
 - AIFollowSetAngleFromEntity() 5-15

AIFollowSetDistFromEntity() 5-4, 5-16
AIFollowSetEntityToFollow() 5-4, 5-16
AIGoto() 5-4, 5-16
AIGotoGetDestination() 5-16
AIGotoGetDistanceLeft() 5-16
AIGotoGetVRFDestination() 5-16
AIGotoHasArrived() 5-17
AIGotoSetDestination() 5-17
AIGotoSetSpeed() 5-17
AIGotoSetVRFDestination() 5-17
AIGotoSetVRFWaypoint() 5-17
AIHide() 5-4, 5-17
AIHideAddDangerousEntity() 5-17
AIHideGetDestination() 5-18
AIHideGetGraph() 5-18
AIHideGetResearchType() 5-18
AIHideHasArrived() 5-18
AIHideIsHidden() 5-18
AIHideResetDangerousEntities() 5-18
AIHideSetGraph() 5-19
AIHideSetResearchType() 5-19
AIPathway() 5-4, 5-19
AIPathwayGetCurrentPointIndex() 5-19
AIPathwayGetDestination() 5-19
AIPathwayGetNbPoints() 5-19
AIPathwayGetPathway() 5-20
AIPathwayGetReverseOrder() 5-20
AIPathwayIsCurrentPointReached() 5-20
AIPathwayIsPathwayFinished() 5-20
AIPathwayResetPathway() 5-20
AIPathwaySetReverseOrder() 5-20
AIPathwaySetVRFRoute() 5-20
AIPerceptionGetDistance() 5-24
AIPerceptionGetNearestEnemy() 5-24
AIPerceptionGetNearestEntity() 5-24
AIPerceptionGetNearestObstacleSeenEntity()
5-24
AIPerceptionGetNearestSeenEnemy() 5-24
AIPerceptionGetNearestSeenEntity() 5-25
AIPerceptionIsEnemy() 5-25
AIPerceptionIsSeen() 5-25
AIPerceptionIsVisible() 5-25
AISetMovementMode() 5-21
AIUtilBHAVE2VRF() 5-5, 5-26
AIUtilCheckVisible() 5-26
AIUtilDebugMessage() 5-26
AIUtilDrawLine() 5-9, 5-26
AIUtilDrawPath() 5-9, 5-27
AIUtilDrawPathway() 5-27
AIUtilDrawPoint() 5-9, 5-27

AIUtilGetCurrentTime() 5-27
AIUtilGetLastError() 5-27
AIUtilGetVRFExerciseStartTime() 5-27
AIUtilGetVRFExerciseTime() 5-27
AIUtilMakeVectorDeg() 5-28
AIUtilResetLastError() 5-28
AIUtilTraceLine() 5-28
AIUtilVecToAnglesDeg() 5-28
AIUtilVRF2BHAVE() 5-5, 5-28
AIWander() 5-4, 5-21
AIWanderGetDestination() 5-21
AIWanderSelectRandomDestination() 5-21
AIWanderSetVRFArea() 5-22
AIWorldGetEntityByIndex() 5-23
AIWorldGetEntityByName() 5-23
Think() 5-3, 5-6

G

GDB format 3-5
generating
 PathData 3-18
 complex 3-3
Go To task 2-11
graph density 3-14
graph edge, limiting 3-15
ground platform entity type 3-14

H

heuristic, path planning 2-6
Hide From task 2-11, 4-8
hiding, path planning routes 4-14

I

ignoring, surface types 3-15
Import parameter 3-22
indoors, setting location in 4-10
initial-movement-mode parameter 4-12
installing
 B-HAVE module 1-3
 license 1-5
interior, setting location in 4-10
iterator 5-4

K

Kynapse 1-2

kynapse-local-automotive-controller 4-2

L

level of detail, terrain 3-14
 license
 management 1-5
 unavailability of B-HAVE 4-2
 lifeform entity type 3-14
 loading, terrain database in TDB Tool 3-5
 local zone 3-3, 3-6
 configuring 3-16
 location, interior 4-10
 looping, through entity list 5-4
 Lua 2-12, 5-2, 5-6
 interpreter 5-4
 sending radio messages 5-7

M

MÄK Stealth, viewing B-HAVE data in 4-14, 5-9
 MÄK Terrain Database Tool 1-2
 Manage PathData Resources dialog box 3-6, 3-8, 3-10
 message, sending radio 5-7
 MHA file extension 3-18
 model, topological 2-2
 modifying, bounding box 3-7
 monitoring, PathData generation 3-20
 more detail level of detail 3-14
 Move Along Route task 4-11
 Move to Location task 4-9
 Move to Waypoint task 4-9
 movement, smoothness of 2-5, 3-14
 movement mode 4-12
 configuring for entity 4-12
 MTD file 1-3, 3-2, 3-5, 3-21, 3-22

N

no detail level of detail 3-14
 no_reflex.lua script 5-3

O

object, viewing 4-13
 object parameters database 4-2
ObjectType parameter 3-22
 OPE file 4-2

P

parameter
 Import 3-22
 initial-movement-mode 4-12
 ObjectType 3-22
 PathDataDirectory 3-22
 PathDataName 3-22
 PathsRelativeToTerrain 3-22
 publish-debug-routes 4-14
 path
 data 2-4
 following 2-7
 planning 2-6
 path finding 2-2, 2-6, 2-11
 path following 2-11
 path planning route, hiding and displaying 4-14
 path smoothing 4-14
 PathData 1-2, 1-3, 2-4, 2-9, 2-12, 3-2
 associating with terrain 3-21
 configuration 3-12
 configuring, optional settings 3-16
 detail and density 2-5, 3-14
 drawing around entity 5-9
 editing configuration 3-17
 filtering surface types 3-12
 generating 3-18
 generating complex 3-3
 level of detail 3-12
 monitoring generation of 3-20
 resources 3-6
 template 3-12
 templates 2-5, 3-14
 testing 3-23
 PathData Generation Monitor dialog box 3-20
PathDataDirectory parameter 3-22
PathDataName parameter 3-22
PathsRelativeToTerrain parameter 3-22
 PDL file extension 3-10, 3-18
 performance 4-14
 physical world 2-12
 Pick Location on Map 3-7
 plan, disabling script in 5-3
 planning
 advanced 2-12
 path 2-6
 plug-in 1-2
 point, on roof 4-10
 pointer variable 5-4
 Prefer Roads 4-12

Preferred Movement mode 4-11
publish-debug-routes parameter 4-14

R

radio message, sending using Lua 5-7
reflex script 5-3
resource, PathData 3-6
roof, points on 4-10

S

sample terrain 1-3
scenario
 example 4-15
 settings for new 4-2
script
 creating 5-2
 deleting 5-3
 disabling from plan 5-3
 editing 5-3
 entity control 5-5
 example 5-6
 executing 4-13
 issues for writing 5-4
 Lua 2-12, 5-6
 setting 4-13
 types 5-3
 using B-HAVE tasks in 5-4
 writing 5-3
seed graph 3-3
 adding 3-10
 clipping to bounding box 3-11
 configuring 3-17
 selecting for PathData configuration 3-13
 simplification 3-11
seed point 2-4, 2-12, 3-6
 creating 3-8
 selecting for PathData configuration 3-13
set data request
 B-HAVE Set Movement Mode 4-12
 B-HAVE Set Script 4-13
 disabling B-HAVE 4-2
 View Target 4-13
Set Movement Mode dialog box 4-12
Set Script 4-13
setting
 movement mode 4-12
 script 4-13
simplification 3-11

sparse graph density 3-14
spatial graph 2-2, 2-4, 2-12, 3-20
spatial reasoning 2-9
specifying, PathData resources 3-6
Start Batch PathData Generation dialog box 3-18
starting, TDB Tool 3-4
Stealth
 drawing PathData in 5-9
 viewing B-HAVE data in 4-14, 5-9
Stop When Hidden 4-7
subgoal 2-7
surface
 filtering 3-15
 forbidden 3-12
syntax error 5-4

T

target, viewing 4-13
task 2-11
 assigning to multiple entities 4-4
 B-HAVE Flee From 4-7
 B-HAVE Follow Entity 4-8
 B-HAVE Hide From 4-8
 B-HAVE Move Along Route 4-11
 B-HAVE Move to Location 4-9
 B-HAVE Move To Waypoint 4-9
 B-HAVE Wander 4-11
 disabling B-HAVE 4-2
 Flee From 2-11
 Go To 2-11
 Hide From 2-11
 using in script 5-4
 wander 2-11
TDB Tool 1-2
 loading terrain database 3-5
 starting 3-4
template 2-5, 3-14
 choosing 3-14
 PathData configuration 3-12
terrain
 associating PathData with 3-21
 level of detail 3-14
 loading in TDB Tool 3-5
 provided with PathData 1-3
Terrain Descriptor File 3-2
testing, PathData 3-23
text, sending 5-7
Think() function 5-3, 5-6
topological model 2-2

- transforming
 - coordinates 5-28
 - VR-Forces coordinates to B-HAVE 5-5

V

- vertex 2-2, 2-12, 3-10
- very sparse graph density 3-14
- View Target set data request 4-13
- viewing
 - B-HAVE data in Stealth 4-14, 5-9
 - object 4-13

W

- Wander task 2-11, 4-11
 - restricting to area 4-11
- writing, script 5-2

X-Y-Z

- XML file, B-HAVE configuration 3-11
- Z value, indoors 4-10



Link - Simulate - Visualize

BHV-1.1-1-070712