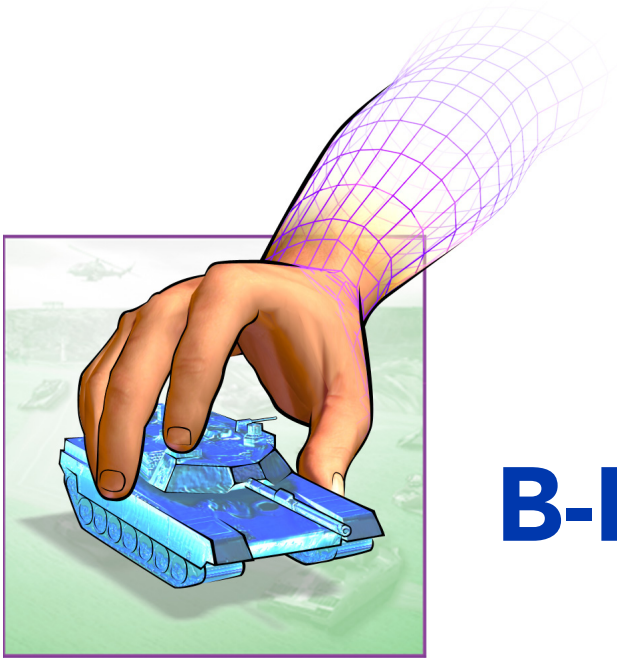
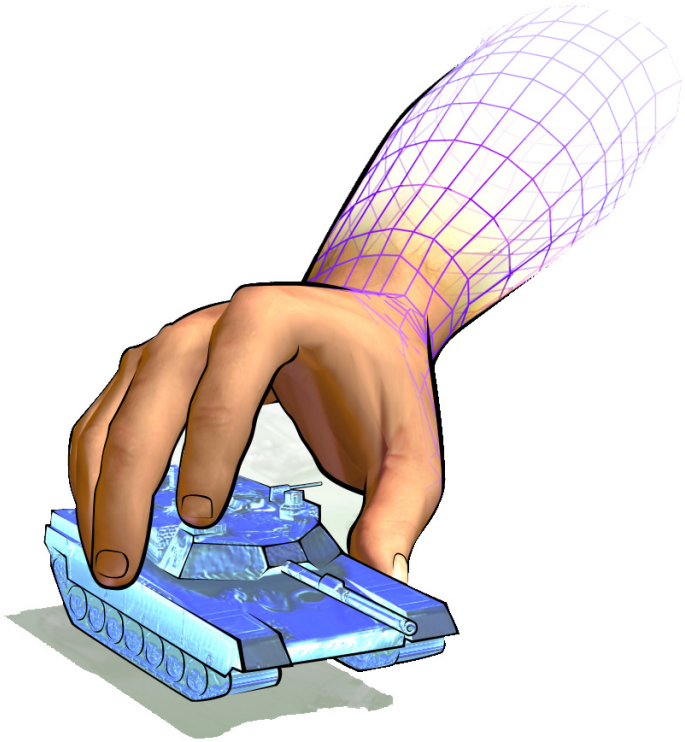


B-HAVE Module
for VR-Forces

Users Guide

VT MÄK
A company of VT Systems



B-HAVE Module for VR-Forces

Users Guide

VT MÄK
A company of VT Systems

For third-party license information, please see “Third Party Licenses,” on page xiii.

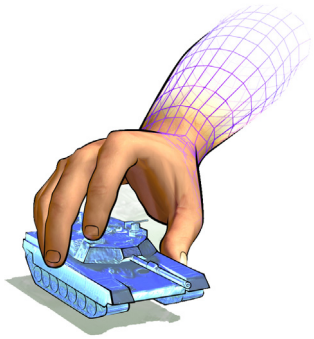
VT MÄK
68 Moulton St.
Cambridge, MA 02138 USA

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

info@mak.com

www.mak.com

Revision BHV-2.0.2-1-111107



Contents

Preface

Chapter 1 Getting Started

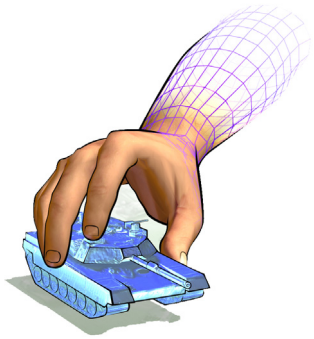
Chapter 2 B-HAVE Module Concepts

Chapter 5 Writing B-HAVE Module Scripts

Chapter 6 B-HAVE Messages API

Appendix A Lua Functions for the B-HAVE Module

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 the 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* explains how to use the tasks and set data requests that the B-HAVE Module adds to VR-Forces as well as other B-HAVE functions you can use during a simulation.

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

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

Appendix A, *Lua Functions for the B-HAVE Module* lists the Lua function for the B-HAVE Module.

B-HAVE Module Documentation

Documentation is provided in PDF format. Manuals are in the *./doc* directory. The B-HAVE Module documentation set is as follows:

- *B-HAVE Module for VR-Forces Users Guide* describes how to install and use the B-HAVE Module program.
- *B-HAVE Module for VR-Forces Release Notes*. Release documentation consists of release notes.

MÄK Products

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

- VR-Link® Network Toolkit. VR-Link is an object-oriented library of C++ functions and definitions that implement the High Level Architecture (HLA) and the Distributed Interactive Simulation (DIS) protocol. 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 (the RTI Assistant) that helps users with configuration tasks and managing federates and federations.
- VR-Forces®. VR-Forces is a computer generated forces application and toolkit. It provides an application with a GUI, that gives you a 2D and 3D views of a simulated environment.

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

- VR-Vantage™. VR-Vantage is a line of products designed to meet your simulation visualization needs. It includes four end-user applications (VR-Vantage Stealth, VR-Vantage XR, VR-Vantage PVD, and VR-Vantage IG), the VR-Vantage Toolkit, and VR-Vantage FreeView.
  - VR-Vantage Stealth displays a realistic, 3D view of your virtual world. You can view this world from the inside of a simulated moving vehicle, or place the eyepoint at another moving or stationary location. The Stealth lets you switch rapidly among several predefined viewpoints while the simulation is underway.
  - VR-Vantage IG is a configurable desktop image generator (IG) for out the window (OTW) scenes and remote camera views. It has most of the features of the Stealth, but is optimized for its IG function.

  - VR-Vantage XR provides a common operational picture using the same 3D views as VR-Vantage Stealth, a 2D plan view, and an exaggerated reality (XR) view. Together these views provide both situational awareness and the big picture of the simulated world.
  - VR-Vantage PVD provides a 2D plan view display. It gives you the big picture of the simulated world.
  - The VR-Vantage Toolkit is a 3D visual application development toolkit. Use it to customize or extend MÄK's VR-Vantage applications, or to integrate VR-Vantage capabilities into your custom applications. VR-Vantage is built on top of OpenSceneGraph (OSG). The toolkit includes the OSG version used to build VR-Vantage.
  - VR-Vantage Free View is a terrain and 3D model viewer that introduces some of the features of VR-Vantage applications in a useful, free utility.
- MÄK Data Logger. The Data Logger, also called the Logger, can record HLA and DIS exercises and play them back for after-action review. You can play a recorded file at speeds above or below normal and can quickly jump to areas of interest. The Logger has a GUI and a text interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application. The Logger editing features let you merge, trim, and offset Logger recordings.
- 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 DIS translation.
- VR-TheWorld™ Server. VR-TheWorld Server is a simple, yet powerful, web-based streaming terrain server, developed in conjunction with Pelican Mapping. Delivered with a global base map, you can also easily populate it with your own custom source data through a web-based interface. The server can be deployed on private, classified networks to provide streaming terrain data to a variety of simulation and visualization applications behind your firewall.
- VR-inTerra. VR-inTerra is a C++ API for adding terrain agility to applications. It can load, page, or stream terrain from a wide variety of formats or sources into a single, consistent run-time representation, consisting of a collision graph and vector network.

How to Contact Us

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

Telephone

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

E-mail

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

Internet

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

Post

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

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

Document Conventions

This manual uses the following typographic conventions:

`Monospaced`	Indicates commands or values you enter.
`Monospaced Bold`	Indicates a key on the keyboard.
`Monospaced Italic`	Indicates command variables that you replace with appropriate values.
`{ }`	Indicates required arguments.
`[ ]`	Indicates optional arguments.
|	Separates options in a command where only one option may be chosen at a time.
`( | )`	In command syntax, indicates equivalent alternatives for a command-line option, for example, `(-h | --help)`.
/	Indicates a directory. Since MÄK products run on both UNIX and Windows PC platforms, we use the / (slash) for generic discussions of pathnames. If you are running on a PC, substitute a \ (backslash) when you type pathnames.
Italic	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**.
	Indicates supplemental or clarifying information.
	Indicates additional information that you must observe to ensure the success of a procedure or other task.

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

Hardware and Operating System Naming Conventions

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

Mouse Button Naming Conventions

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

Third Party Licenses

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

Boost License

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

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

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

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

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

libXML and libICONV

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

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

pThreads Library

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

EHS, PCRE, and PME

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

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

Kynapse License

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

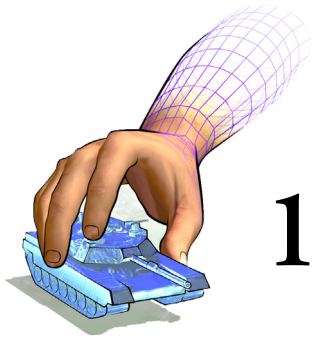
The following are registered trademarks or trademarks of Autodesk Canada Co. in the USA and/or Canada and other countries: Backburner, Multi-Master Editing, River, and Sparks.

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

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

Disclaimer

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



1

Getting Started

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

1.1. Introduction

The B-HAVE Module for VR-Forces[1] is a VR-Forces plug-in that extends the movement models for lifeforms and ground platforms. It improves:

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

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

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

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

1.1.1. The B-HAVE Module Architecture

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

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

1.1.2. Using the B-HAVE Module in a Simulation

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

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

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

1.2. Installing the B-HAVE Module

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

- You must install the B-HAVE Module into the directory in which you have installed VR-Forces. The installer defaults to the default installation directory for the associated version of VR-Forces, for example, *C:\MAK\vrforces4.0*. If you did not install VR-Forces into the default directory, it is your responsibility to change the B-HAVE Module installation directory to your VR-Forces installation directory.

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

- Each version of the B-HAVE Module is built for a specific version of VR-Forces and will work only with that version. For example, B-HAVE Module 1.5 for VR-Forces 4.0 will work only with VR-Forces 4.0. It would not work with VR-Forces 3.12.

➤ 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

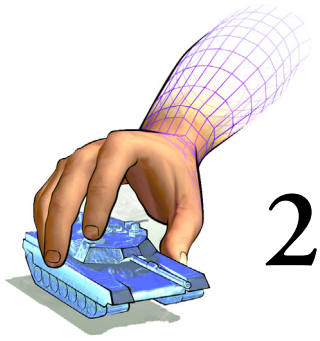
File/Directory	Description
./data/parameters/BHAVE	Sample Lua scripts, PathData generation templates, and icons.
./userData/scenarios/B-HAVE_Demos	Sample scenarios that use B-HAVE Module tasks.
./data/terrain/	MTD files and PathData for terrain databases.
./bin	B-HAVE Module DLLs are installed in the *bin* directory.
./plugins/B-HAVE	Plug-in DLLs for the B-HAVE Module.

1.2.1. License Management for the B-HAVE Module

Like all MÄK products, the B-HAVE Module requires a license. For details about obtaining a license, please see Chapter 2, Installing VR-Forces, in *VR-Forces Users Guide*.

1.2.2. Localizing the B-HAVE Module

You can localize the B-HAVE Module menus using Qt Linguist, as described in Section 2.6, "Localizing the Graphical User Interface", in *VR-Forces Users Guide*. An untranslated file for B-HAVE is included in the installation.



2

B-HAVE Module Concepts

This chapter provides conceptual background for the B-HAVE Module. Understanding these concepts will help you 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.

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 are called PathData.

2.2. How PathData is Generated

PathData is generated by the B-HAVE Module from within the VR-Forces front-end. Before you can use newly generated PathData, you must close your scenario and reopen it. (This is so it can be loaded by the back-end.) If the terrain is modified, the PathData must be generated again.

The B-HAVE Module analyzes the terrain and creates a navigation mesh (NavMesh) of the terrain. The NavMesh defines areas in which an entity can move, given its dimensions. A lifeform can move in spaces that a ground vehicle cannot move in. Figure 2-1 shows a NavMesh for a terrain that has several areas in which entities cannot move.

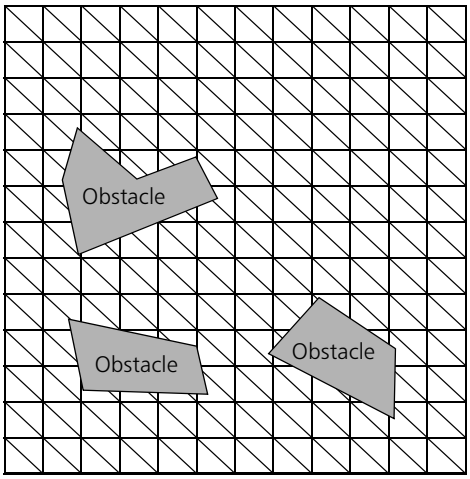


Figure 2-1. NavMesh

Based on the NavMesh, the B-HAVE Module generates a graph of locations in the terrain that the entity type can move to. Each location can be accessed from at least one other location. Figure 2-2 shows a hypothetical graph of the paths that an entity can follow in the NavMesh shown in (Figure 2-1).

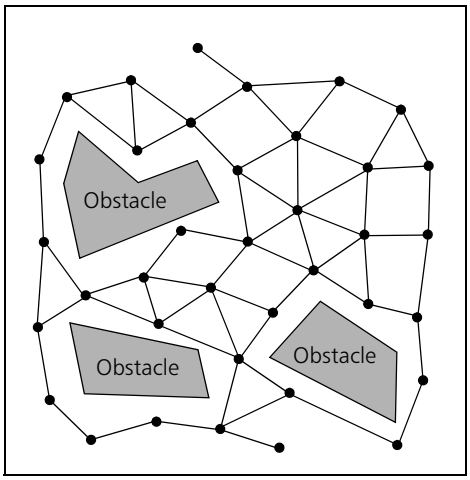


Figure 2-2. Graph

For details about generating path data, please see Chapter 3, *Generating PathData*.

2.3. Path Finding

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

2.3.1. 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 PathData.

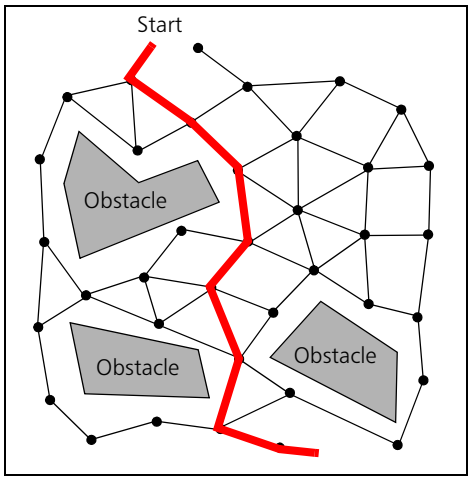


Figure 2-3. Planning a path

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.

2.3.2. Path Following

Once a path is determined for an entity, it can start moving to its final goal. As illustrated in Figure 2-3, 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 2-4 and 2-5 illustrate this process. Using the prospective path defined in Figure 2-3, the entity computes its actual path. In Figure 2-4, view 1, it tests the first three vertices (dashed lines) and finds that it can reach all of them directly. In Figure 2-4, 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 2-5 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.

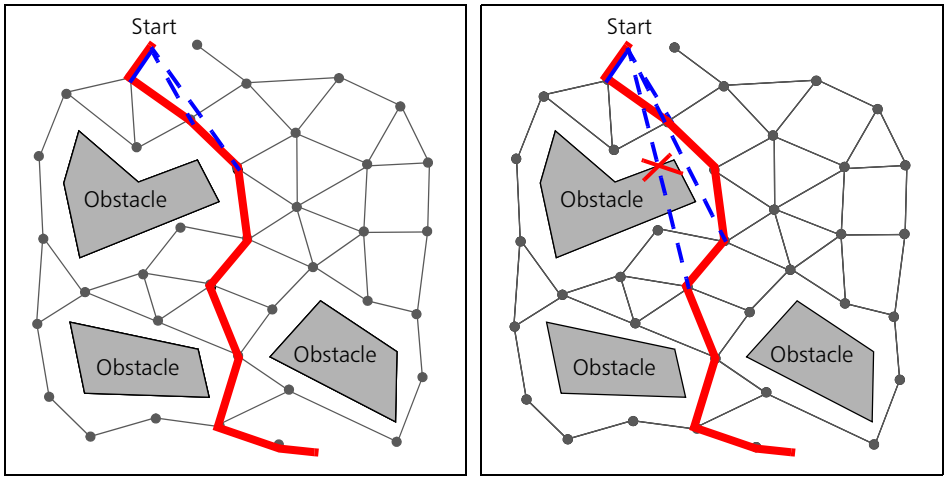


Figure 2-4. Checking next vertex

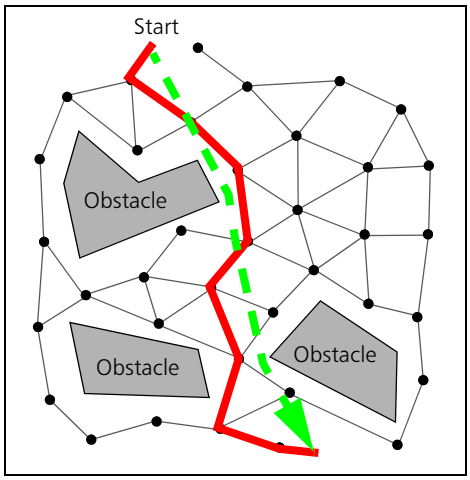


Figure 2-5. Final calculated path

2.3.3. 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 PathData.

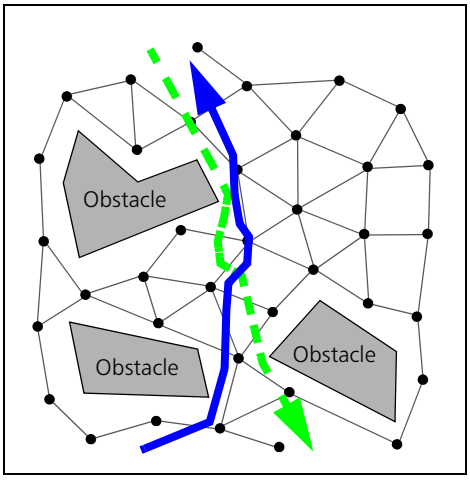


Figure 2-6. Collision avoidance

In Figure 2-6, two entities traveling in opposite directions avoid each other, then resume their routes. Compare the track of the dashed line to that in Figure 2-5.

2.4. 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 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).

2.4.1. 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 2-7, 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.

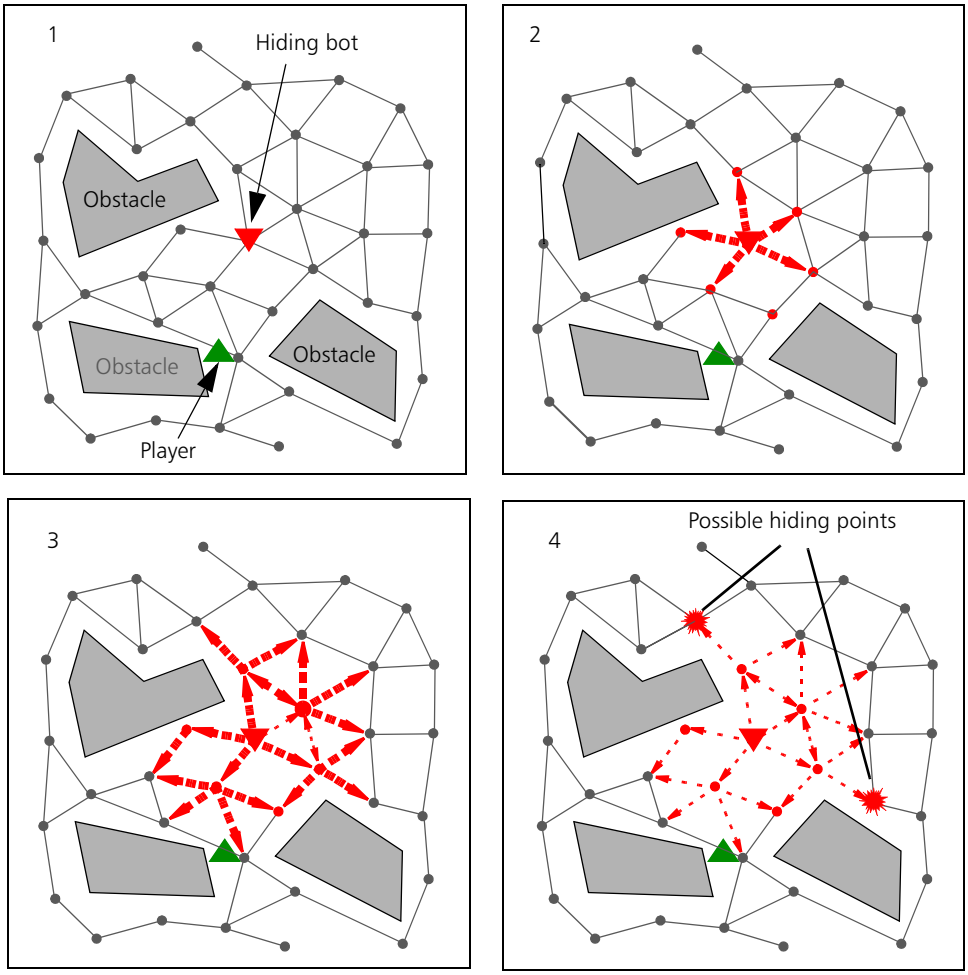


Figure 2-7. Dynamic topological analysis

2.5. 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.

2.5.1. 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.

2.5.2. 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.

2.5.3. 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.

2.6. Traffic Generation

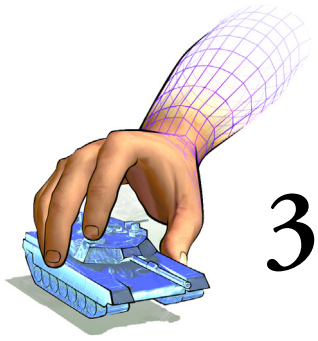
An important use for the B-HAVE Module is to populate simulations with background entity activity, such as vehicular or pedestrian traffic. The Wander task causes entities to wander randomly in an area, but this is somewhat unrealistic because traffic usually moves from one place to another, it does not wander aimlessly. The traffic feature creates more realistic traffic patterns by having entities move randomly from spawn points (where they get created), to sink points (where they are deleted). This provides the appearance of intentional traffic patterns.

You can have multiple spawn and sink points to introduce multiple traffic patterns. Vehicles choose a sink point at random. You can configure the speed of vehicles, the rate at which they are created, and other parameters.

For an example of using traffic generation, run the traffic example, which is included with the B-HAVE Module.

2.7. 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*.



3

Generating PathData

This chapter explains how to generate PathData for use with the B-HAVE Module.

3.1. Introduction

To use the B-HAVE Module tasks in VR-Forces, you must use a terrain that has PathData generated for the areas in which you want to assign B-HAVE Module tasks. You generate PathData in the VR-Forces front-end.

When you generate PathData, the B-HAVE Module creates separate sets of PathData, at various levels of detail, for lifeforms and for ground platforms.

PathData is generated as follows:

1. Load the terrain on which you want to generate PathData or load a scenario that uses that terrain.
2. Optionally, change the PathData configuration.
3. Generate the PathData.
4. Start a new scenario or save and reload the existing scenario.

3.1.1. Creating PathData from Road Vector Networks

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

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

3.2. Configuring Generation of PathData

The B-HAVE Module provides default values for generating PathData that should work for most terrains. If you want to, you can change the configuration. The B-HAVE Module remembers the most recent PathData configuration and continues to use it until you change it or revert to the default values.

To configure PathData generation:

1. Choose **B-HAVE** → **Configure Pathdata**. The Generation Options dialog box opens.
2. Select the Generation Options page (Figure 3-1).

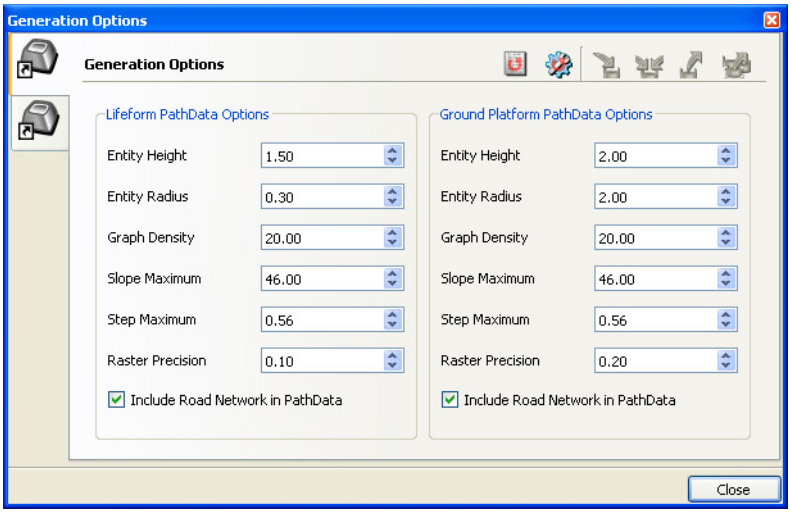


Figure 3-1. Generation Options page

3. On the Generation tab, change configuration values for lifeforms or ground platforms as desired. Table 3-1 describes the parameters.

Table 3-1: PathData configuration parameters

Parameter	Description
Entity Height	The maximum height for this type of entity. Used to calculate the ability of entities to move through doors, under bridges, and so on.
Entity Radius	The maximum radius of the entity bounding box. Used to calculate the width of spaces an entity can move through.
Graph Density	The density of the PathData. Higher density provides higher quality entity movement, but increases processing time.
Slope Maximum	The maximum slope that an entity can move on.

Table 3-1: PathData configuration parameters

Parameter	Description
Step Maximum	The maximum height of steps that an entity can move up. Also the maximum break in the terrain that an entity can cross.
Raster Precision	The precision of calculations for points in the terrain.
Generate Vector Network	When enabled, include vector features labeled as roads in the PathData.

4. On the Soil Type page (Figure 3-2), select or clear soil types as desired. If a soil type is not selected, the B-HAVE Module does not generate PathData for that soil type and B-HAVE-enabled entities will avoid locations with that soil type.

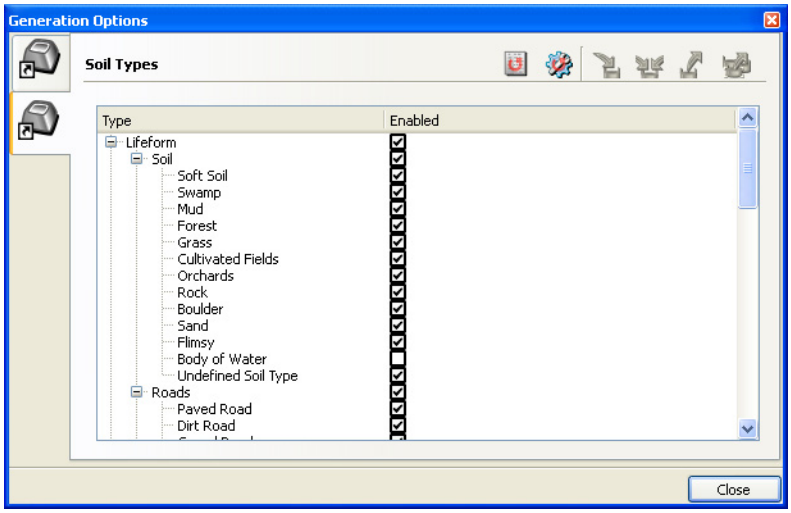


Figure 3-2. Soil Type page

5. Click Close.

3.3. Generating PathData

Generating PathData 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. You can generate PathData for the entire terrain or for a specific area of the terrain. If you generate PathData for a specific area of the terrain, you can only generate it for that one area. You cannot generate PathData for the entire area of a paged terrain.

The location of the PathData files depends on the type of terrain you generate them for. If the terrain that you loaded references a GDB terrain file (most likely if you loaded an MTD file), the PathData is placed in a directory under the directory that contains the GDB file. If you generate PathData for an MTF terrain that does not reference a GDB file, the PathData is placed in a directory under the directory where the MTF file is located (usually *./userData/terrains*).

To generate PathData:

1. Choose **B-HAVE** → **Generate PathData**. A tab is added to the side of the VR-Forces window.
2. Click the tab. The Generate PathData dialog box opens (Figure 3-3).

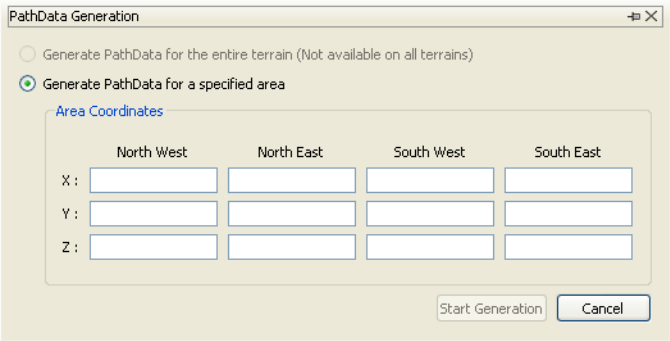


Figure 3-3. Generate PathData dialog box

3. To generate PathData for the entire terrain, select the Generate PathData for the Entire Terrain option.

   To generate PathData for an area of the terrain (the default), select the Generate PathData for a Specified Area option. Then using the mouse, drag a bounding box around the area in which you want to generate PathData. The coordinates of the area are displayed. You can edit the coordinates by hand if you want to.

4. Click Start Generation. The label on the PathData Generation tab changes to Generating and the Generate PathData command is not available on the B-HAVE menu. You can use VR-Forces for other purposes while PathData is generating. When the operation is complete, VR-Forces displays a message.

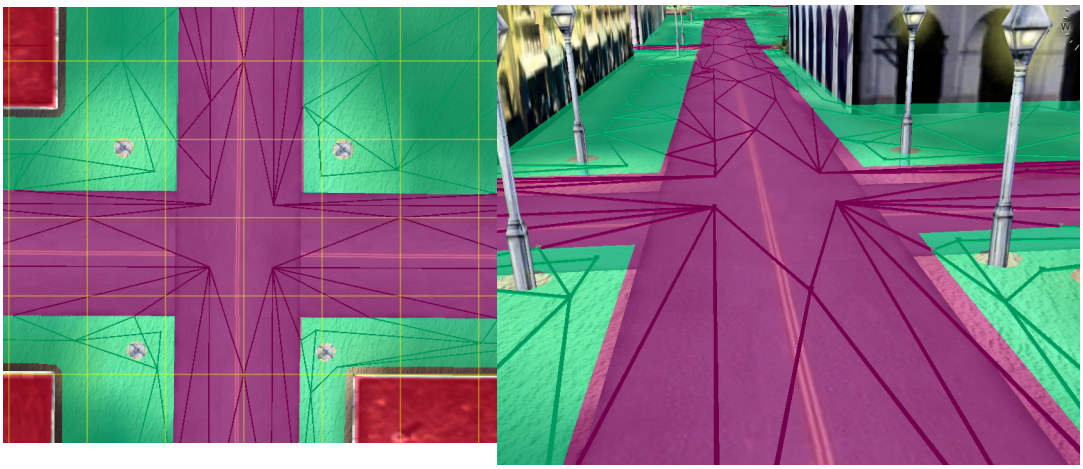
3.3.1. Using Newly Generated PathData

To use newly generated PathData, create a scenario that uses the terrain or open a scenario that uses that terrain. If you generated PathData from an open scenario, you must save the scenario, close it, and then open it again.

3.4. Displaying PathData

You can display the PathData in a terrain for lifeforms or for vehicles. Figure 3-4 illustrates PathData for lifeforms in a portion of the Makland terrain. The lines represent the paths an entity can follow. The lines will be the same for both lifeforms and ground platforms except in areas where a particular type cannot travel. The colors represent different soil types. (The colors are chosen by the underlying Kynapse software and are not configurable.)

You can display PathData in both the 2D and 3D views.



2D 3D

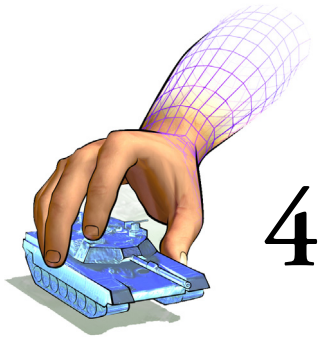
Figure 3-4. PathData display

PathData is not displayed if you are zoomed out too far from the terrain.

➤ To display or hide PathData, choose **B-HAVE** → **Show Generated PathData** → **Life-Form** (or **Ground Platform**). The display of PathData toggles to the opposite of the current setting.

3.5. Troubleshooting

If you do not see the B-HAVE menu, the plug-in was not loaded. Make sure that the file *./plugins/B-Have.xml* is present. If you do not want to load the plug-in, you can temporarily disable loading. To do so, remove *./plugins/B-Have.xml* or change its extension. You can also enable or disable the plug-in in the Plugins Manager in the VR-Forces front-end. (For details, please see Section 5.9, "Managing Plug-ins", in *VR-Forces Users Guide*.)



4

Using the B-HAVE Module

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.

4.1. Using the B-HAVE Module in a Simulation

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

The way that B-HAVE Module tasks are organized on the Tasks menu depends on how you configure the B-HAVE Module. For details, please see "B-HAVE Module Tasks," on page 4-7.

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

4.1.1. Disabling the B-HAVE Module Plug-ins

If, after installing the B-HAVE Module, you do not want to load the plug-ins, you can temporarily disable loading. To do so, remove *./plugins/B-Have.xml* or change its extension. You can also enable or disable the plug-in in the Plugins Manager in the VR-Forces front-end. (For details, please see Section 5.9, "Managing Plug-ins", in *VR-Forces Users Guide*.)

4.2. Creating Multiple Entities

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

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

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

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

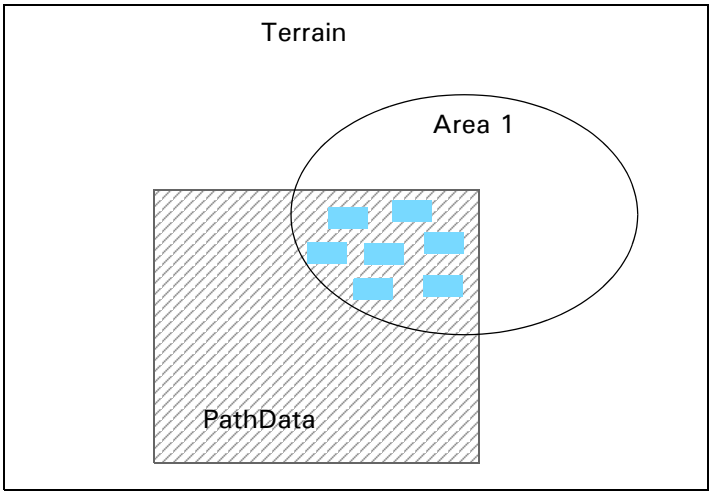


Figure 4-1. Placement of multiple entities

To add multiple entities to a scenario:

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

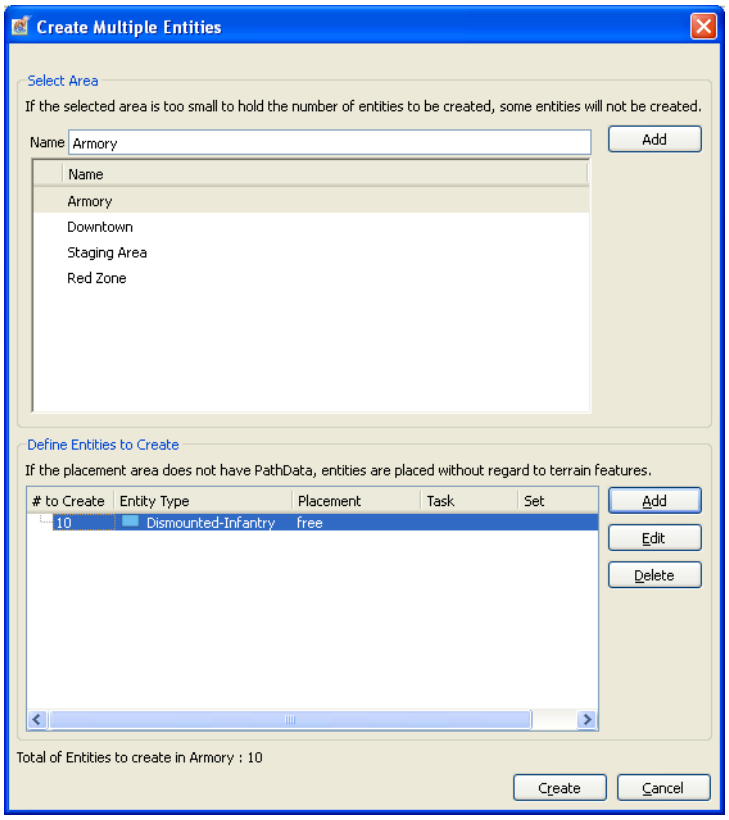


Figure 4-2. Create Multiple Entities dialog box

2. In the Select Area group box, type the name of the area in which you want to create the entities, or select the area from the list or on the map. If the area in which you want to create the entities does not exist, add a new area as described in "Adding an Area for Multiple Entity Creation," on page 4-5.
3. In the Define Entities To Create group box, add the entities that you want to create as part of this selection group. For details about this part of the procedure, please see "Specifying the Entities to Create," on page 4-5.
4. Optionally, edit or remove any of the entity sets listed in the Define Entities To Create list.
5. 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. A Create Area tab is added to the window.
2. Create an area following the standard VR-Forces procedure for creating areas. For details, please see Chapter 2, *Creating and Placing Objects*, in *VR-Forces Scenario Management Guide*.

4.2.2. Specifying the Entities to Create

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

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

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

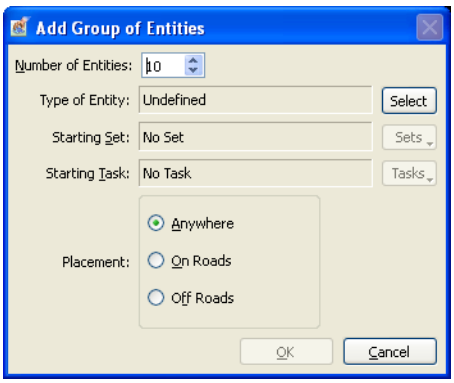


Figure 4-3. Add Group of Entities dialog box

2. In the Number of Entities box, specify the number of entities you want to add for this entity set.
3. Click the Select button that is next to the Type of Entity text box and select the entity type that you want to add. The list of entity types available is the same list of entities provided on the Create menu for ground and lifeform entities.
4. Optionally, specify a starting set data request by clicking the Sets button and selecting a set data request in the list. The list is not context sensitive. You must ensure that the selected set data request is appropriate to the entity type that you are adding.
5. Optionally, specify a starting task by clicking the Tasks button and selecting a task in the list. The list is not context sensitive. You must ensure that the selected task is appropriate to the entity type that you are adding.

6. Optionally, select an option in the Placement group box. VR-Forces will try to place the entities in the desired location.
7. Click OK. The entity set is added to the list in the Define Entities To Create group box (Figure 4-2).
8. Optionally, add additional entity sets.

4.3. Sending Text Messages

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

To send a text message to an entity:

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

4.4. B-HAVE Module Tasks

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

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

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

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

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

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

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

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

4.4.1. The Flee From Task

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

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

To assign a Flee From task:

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

4.4.2. The Follow Entity Task

The Follow Entity task works like the standard VR-Forces Follow Entity task except that entities use B-HAVE Module path planning and collision avoidance. There is no option to specify a distance above because this is meaningless for ground entities. To assign this task, follow the procedure in Section 6.6.10, "Follow Entity", in *VR-Forces Scenario Management Guide*.

4.4.3. The Hide From Task

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

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

To assign a Hide From task:

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

4.4.4. The Move to Waypoint Task

The Move To Waypoint task sends an entity to the specified waypoint or object using B-HAVE Module path planning and collision avoidance. To assign this task, follow the procedure in Section 6.6.19, "Move To Waypoint", in *VR-Forces Scenario Management Guide*.

4.4.5. The Move to Location Task

The Move to Location task sends an entity to the specified location using B-HAVE Module path planning and collision avoidance. You can specify locations that are inside buildings. To assign this task, follow the procedure in Section 6.6.18, "Move To Location", in *VR-Forces Scenario Management Guide*.

4.4.6. The Move Along Route Task

The Move Along Route task is identical to the VR-Forces Move Along Route task except that entities use B-HAVE Module path planning and collision avoidance. To assign this task, follow the procedure in Section 6.6.15, "Move Along Route", in *VR-Forces Scenario Management Guide*.

4.4.7. The Wander Task

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

To assign a Wander task:

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

4.5. B-HAVE Module Set Data Requests

The B-HAVE Module adds three set data requests that help you configure use of the B-HAVE Module.

4.5.1. The Set Movement Mode Set Data Request

The Set Movement Mode set data request allows you to specify that when an entity plans its path it should take into account road data, if it is available. This set data request is meaningful only if one or both of the movement modes was enabled when the PathData for this terrain was configured.

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

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

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

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

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

4.5.2. The Set Script Set Data Request

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

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

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

To execute a Lua script:

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

4.5.3. The View Target Set Data Request

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

To assign the View Target set data request:

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

4.6. Generating Traffic

The B-HAVE Module uses spawn points and sink points to create and delete entities. You must have at least one pair of a category and type to generate traffic. A spawn point will not spawn entities if it does not have a corresponding sink point. Spawn points and sink points must be of the same type. A vehicle cannot sink at a civilian sink point. A friendly entity cannot sink at an opposing sink point.

You can create spawn points and sink point when you create a scenario or while you are running it. The same rules apply as for making any other changes to a scenario.

To generate traffic:

1. Choose **Create** → **Traffic** → *category* → *type* **Spawn Point**, where *category* is Friendly, Opposing, or Neutral, and *type* is Civilian or Vehicle. The cursor changes to the waypoint symbol.
2. Click on the terrain where you want to place the spawn point. You can continue clicking to create additional spawn points.
3. Right-click to exit create mode.
4. Choose **Create** → **Traffic** → *category* → *type* **Sink Point**, where *category* and *type* are the same as you chose for the spawn point. The cursor changes to the waypoint symbol.
5. Click on the terrain where you want to place the sink point. You can continue clicking to create additional sink points.
6. Right-click to exit create mode.

4.6.1. Configuring Traffic Generation

The B-HAVE Module comes with spawn points and sink points configured for friendly, opposing, and neutral civilians and vehicles. You can change their parameters and you can create additional types of spawn points and sink points. Configuration is done by editing simulation model set files.

Editing Traffic Generation Parameters

Spawn points and sink points are configured in OPE files. If you want to change their behavior, you edit the OPE files for the spawn points. (Sink points use a generic OPE file.) The OPE files are in *./data/simulationModelSets/core/trafficSpawn/vrfSim*. The following code is from *vehicle-traffic-spawn-point.ope*.

```
(traffic-source-controller
  (component-descriptor-type "traffic-source-descriptor")
  (component-type  "traffic-source-controller")
  (min-tick-period  -1.000000)
  (min-tick-period-variance  -1.000000)
  (tick-period-uses-real-time False)
  (process-state-repository-name  "")
  (process-state-repository-type  "")
  (is-enabled True)
  (min-traffic-speed 15.0)
  (max-traffic-speed 20.0)
  (traffic-object-types
    (object-type 1 (1 1 225 27 0 0 0))
  )
  (creation-interval 15.0)
  (creation-interval-variance 0.8)
  (sink-point-delete-radius 15.0)
  (create-on-vector-network True)
)
```

The parameters that you would most likely want to change to configure traffic generation are:

- **min-traffic-speed** and **max-traffic-speed**. Specifies the desired speed of your entities. The actual of speed each entity is set to a random number between the two values. If you want all your entities to move at the same speed, then set both values to the same number.
- **traffic-object-types**. Specifies each type of object that this spawn point can generate. If there is more than one object in this list, each spawned entity is randomly chosen from that list.
- **creation-interval**. Specifies the amount of time, in seconds, between spawning entities. This is randomly modified by the **creation-interval-variance**.
- **creation-interval-variance**. Specifies a percentage by which to modify the **creation-interval**. In the sample code, a new entity is spawned every 12 to18 seconds (15 seconds with 80% variance).

- **Sink-point-delete-radius**. Specifies the distance from a sink point, in meters, at which an entity should decide it has reached it. Once an entity reaches its destination, it disappears.
- **Create-on-vector-network**. Specifies whether or not the entities being created should use the road network to get to their destination.

Creating a New Spawn Point

If you want to add a new spawn point, you need to:

1. Create a new OPE file for the spawn point.
2. Add a reference to the OPE file to the *vrfSim.opd* file.
3. Add the new spawn point type to the Traffic menu.

To create a new OPE file, such as *new-spawn-point.ope*, we recommend that you copy *vehicle-traffic-spawn-point.ope* or *civilian-spawn-point.ope*. Then, change the parameters in the traffic source controller block, as described in the previous section.

To add the new spawn point to the OPD file, open *./data/simulationModelSets/core/trafficSpawn/vrfSim.opd* and add a new entry to the parameter-files block, like the following:

```
(version  101)
(parameter-files
  (parameter-entry
    (object-type  1 (16 0 0 21 0 2 0)) ;;Note the 5th value must be 0
                                       ;; for all spawn points
    (filename  ".\vrfSim\new-spawn-point.ope")
  )
)
```

The object type can be any unused value, but we recommend that you follow our convention and start with the following numbers “16 0 0 21 0”. The last two numbers should make your spawn point unique.

To add the new type of spawn point to the Traffic menu, add the following lines to *./data/SimulationModelSets/core/trafficSpawn/gui/entity.mst*:

```
(model
  (can-create True)
  (can-aggregate False)
  (can-configure-diguy False)
  (label  "New Spawn Point")
  (unique-id  "ModelSet-151476212-New-Spawn")
  (object-type  1 (16 0 0 21 0 2 0))
  (force-types ForceAny)
  (categories "Traffic")
)
(model
  (can-create True)
  (can-aggregate False)
  (can-configure-diguy False)
  (label  "New Sink Point")
  (unique-id  "ModelSet-151476212-New-Sink")
  (object-type  1 (16 0 0 21 1 2 0))
```

```
    (force-types ForceAny)
    (categories "Traffic")
)
```

4.7. Viewing B-HAVE Module Data (Debug Lines)

When B-HAVE-enabled entities are given tasks in VR-Forces, they publish information indicating the path that was chosen for movement. This can be useful while testing a scenario, to check the paths chosen by entities, their line of sight with enemies, and so on.

! Displaying debug lines degrades performance.

Figure 4-4 illustrates 2D debug lines. Figure 4-5 illustrates 3D debug lines. An entity will not move directly along the indicated path because of the runtime path smoothing (described in "Path Following," on page 2-5).

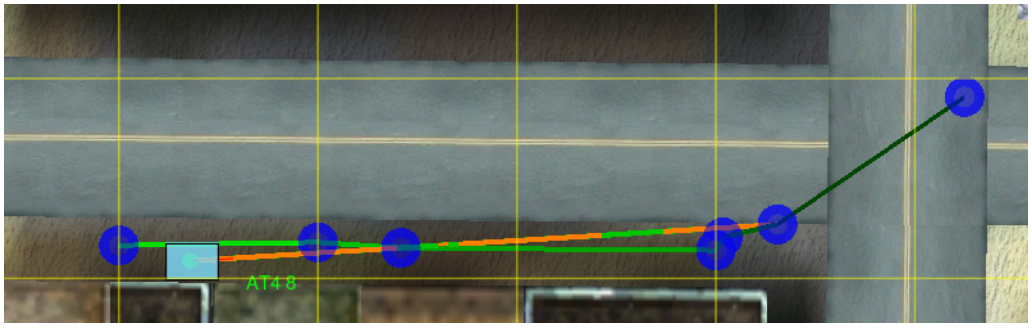


Figure 4-4. B-HAVE debug lines(2D)

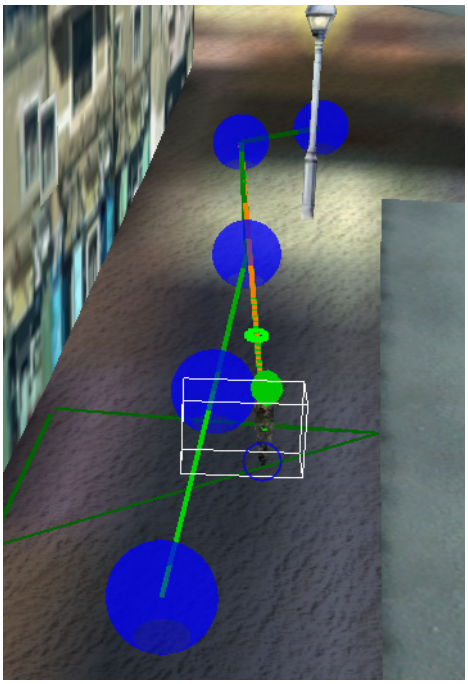


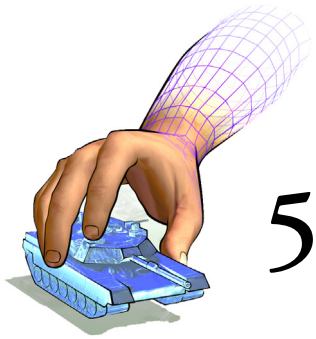
Figure 4-5. B-HAVE paths (3D)

➤ To hide or display debug lines, choose **B-HAVE → Show B-HAVE Debug Lines**.

4.8. Example Scenarios

The B-HAVE Module includes the following example scenarios:

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



5

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.

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. You can edit scripts dynamically, while VR-Forces is running. You can also write and edit scripts off-line.

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

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

5.2.1. Specifying a Text Editor

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

To specify a text editor:

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

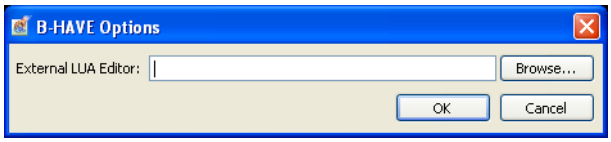


Figure 5-1. B-HAVE Options dialog box

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

5.2.2. Creating a New Script

To create a new Lua script:

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

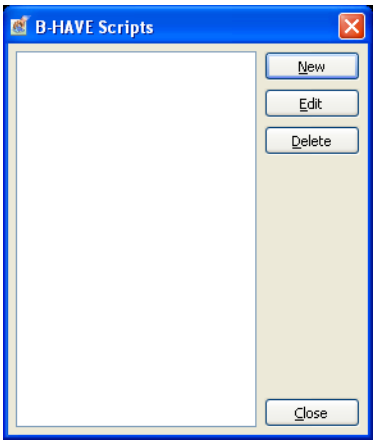


Figure 5-2. B-HAVE Scripts window

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

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

3. Specify a name and click OK.

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

5.2.3. Editing a Script

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

To edit a Lua script:

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

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

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

5.2.4. Deleting a Script

To delete a Lua script:

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

5.3. Writing Scripts

All VR-Forces Lua scripts must contain one or more of the following callback functions. This function will be called by the Lua controller at the appropriate moment. No other functions shall be called by the scripting system - if you don't have one of these functions your Lua script will simply not do anything. Most of these functions take any arguments although the default 'this' object, explained below, shall be defined globally for each script.

- onCreate(). Called on first tick. This is usually right at the beginning of a scenario or on loading of a Lua script.
- onTick(). Called once every tick when an exercise is running. It should be noted that this is called even if the entity has been destroyed. Your script should take this into account.
- onShutdown(). Called once when an entity is deleted. Note that this does not happen when an entity is destroyed.
- onSaveCheckpoint(). Called immediately before a checkpoint happens. Note that during a checkpoint the status of your script will be saved - so you may add any extra information to your script you want to keep using this call.
- onLoadCheckpoint(). Called immediately after a checkpoint has happened. This is essentially the exact opposite of the above script.
- onTeleport(). Called whenever a unit has been teleported through external means. Meaning using the set location message instead of actuator movement.
- onRadioMessage(integer type, string text). Called whenever a radio message intended for this entity is received. Radio messages can be used to communicate information between Lua scripts.
  - @param type. An integer that defines the type of Radio Message.
  - @param test. A String that defines the payload of this radio message.

5.3.1. How Scripts Interact with Plans and Independent Tasks

Almost all tasks and set data requests available to a VR-Forces entity have a corresponding Lua function. These functions act like independent tasks and override whatever task an entity is executing at the time it receives a task from a Lua script. If an entity is executing a task as part of a plan and it receives a task from a Lua script, it abandons its plan, just as it would if it receives an independent task from another source.

If a new task is assigned to an entity while it is executing a task it received from a LUA script, it overrides the LUA-assigned task. However, unlike plans, if a LUA script is still running after it sends a task, it is not abandoned when a task it has sent is overridden. The LUA script continues running and it is possible that it will send additional tasks, thereby overriding the entity's current task.

Because Lua scripts can interact with plans and user actions as described in this section, you must consider these possibilities when you write a script to be sure that it does not affect the scenario in ways that you did not intend.

Table 5-1 maps the B-HAVE Module-specific tasks to their corresponding Lua functions. For a complete list of functions, please see Appendix A, *Lua Functions for the B-HAVE Module*.

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

Task	Function
Move To	taskBHAVEMoveto()
Wander	taskBHAVEWander()
Follow	taskBHAVEFollow()
Move Along Route	taskBHAVEMoveAlongRoute()
Hide	taskBHAVEHide()
Flee	taskBHAVEFlee()

5.3.2. VR-Forces Entities

In Lua all VR-Forces entities are represented by a corresponding LuaVRFObject. This mirrors how in C++, all VR-Forces entities are represented by *DtVRFObjects*. These LuaVRFObjects are Lua Tables that contain a local reference to the corresponding C++ object. They also contain a member functions that may be used to manipulate the object. Usage is just like other Lua member functions; that is, if you have an object myEntity, you can call a function like this:
`myEntity:taskEmbarkOn(myTarget)`.

Every LUA script is attached to a specific entity that is running the script. In order to act on that entity, we provide a special global object called 'this'. The 'this' object is a LuaVRFObject and can use every function that other LuaVRFObjects do. For example, `this:taskEmbarkOn(myTarget)`. In addition to those functions, the 'this' object contains an additional set of functions that act directly on that object's state repository. These functions include setting the position, velocity, name, identifier, and a few others. For a complete list please see "Functions for the 'this' Object," on page A-15.

5.3.3. Considerations for Writing and Using Lua Scripts

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

- If the Lua interpreter detects a syntax error, for example, missing the "then" after "if" statements, missing the "end" keyword after the loop of a conditional, and misspelled (non-existing) function names, it reports the error in the VR-Forces log console. The error message displays the cause of the error and the line where the error occurred. The script is not executed. Errors can be detected during parsing (Set script operation or saving a modified script), or during execution of the onTick() function.
- The preferred way to loop through the entity list is to use the DtGetVrfObjects() global function, as follows:

```
entityList = DtGetVrfObjects()

-- iterate through array
for i,entity in ipairs(entityList ) do
  -- do something with your entity. In this case I print out the name
  DtWarn(entity:objectName())
end
```

  The DtGetVrfObjects() function is computationally intensive if you have a large list of entities. If you can, you should use DtGetVrfObjectByName() and DtGetVrfObjectByGlobalId() to grab a specific entity that you might be looking for. If you only care about entities that interact with your own, you can use DtGetVisibleObjects().

- The text output functions, such as DtWarn() and DtInfo() do not output until the line is completed. Therefore, you should make sure you do not forget to add "\n" to the end of the output or you may not see anything at all. For example:

```
DtWarn("Hello\n")
```

or

```
DtInfo(this, "This will not print until it is a complete line. ")
DtInfo(this, "Now it is complete.\n")
```

- Unless specified in the description of the function, all LUA functions use the terrain local coordinate system as its method for passing in locations. If your location is defined in a different coordinate system, you must convert it to local before passing it.
- When checkpointing a scenario, the global variables of a script are serialized and saved. For more information about checkpoints, please see "Custom Checkpointing," on page 5-10.

5.3.4. A Simple Script

The following sample Lua script (*defaultBehaviour_Wander.lua*) gives the entity a wander task. Thereafter, on each tick, it tests to see if the entity has a task. If it does not, it reassigns the wander task.

```
function onCreate ()
  this:taskBHAVEWander(0,0)
end

function onTick ()
  if this:isCurrentlyTasked () == false then
    this:taskBHAVEWander(0,0)
  end
end

function onShutdown ()
  DtWarn("Goodbye World - From the Lua Scripting Service")
end
```

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

5.3.5. Sending Radio Messages

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

To send a message, use DtRadioSendTextMessage(). For an example, please see the sample script *testRadio_Dispatcher_Waypoint_Speed.lua*.

To receive a message, use the function onRadioMessage(string). It is called asynchronously when a message is received by the entity. For an example, please see the sample script *testRadio_Receiver_Waypoint_Speed.lua* script, 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 (*testRadio_Simple.lua*). The entity Cvln 1 sends a message. All of the other entities print the message.

```
function onCreate()
  counter = 0
  myName = this:objectName()
end
function onTick()
  if counter >= 1000 and myName == "Cvln 1" then
    DtWarn(this, "sending messages...\n")

    -- this message is broadcast to all entities
    DtRadioSendTextMessage("hello world !")

    -- this message is sent to Cvln 2
    DtRadioSendTextMessage("hello to you, Cvln 2", "Cvln 2")

    counter = 0
 end
 counter = counter + 1
end
function onRadioMessage(radioType, text)
  DtWarn(this, "received radio message:" .. text .. "\n")
end
```

5.3.6. 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 onSaveCheckpoint() 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 onLoadCheckpoint() function. It is called just after loading a checkpoint.

Guidelines for easy checkpointing:

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

The following example (*testCheckpoint_mortalWound.lua*) show how checkpointing works. It kills the scripted entity exactly 60 seconds after the script was created. If the scenario is saved, it notifies what the time left to death is.

```
function onCreate()
  refTime = DtGetExerciseTime()
  done = false;
end
function onTick()
  if done == false then
    curTime = DtGetExerciseTime()
    elapsedTime = curTime - refTime
    if elapsedTime > 60 then
      this:setDestroyed()
      done = true;
    end
  end
end
function onSaveCheckpoint()
  curTime = DtGetExerciseTime()
  timeLeft = 60 - curTime + refTime
  DtWarn(this, "Entity Saved with only " .. timeLeft .. " seconds left
    to live\n")
end
function onLoadCheckpoint()
```

```
  curTime = DtGetExerciseTime()
  timeLeft = 60 - curTime + refTime
  DtWarn("Entity Loaded with only " .. timeLeft .. " seconds left
    to live\n")
end
```

5.3.7. Extending Lua Scripts Using Modules and Custom Libraries

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

Creating Modules

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

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

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

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

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

The following is the corresponding B-HAVE Lua script:

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

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

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

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

Building and Using Custom Libraries

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

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

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

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

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

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

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

```
testVRFFunc1 = package.loadlib("luaExt_DIS_d.dll",
  "testEntityIsDestroyed")
i = 0
function onTick()
  i = i + 1
  if i % 100 then
    objectIsDestroyed = testVRFFunc1(this)
    if objectIsDestroyed == true then
      DtWarn("object destroyed")
    else
      DtWarn("object NOT estroyed")
    end
  end
end
```

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

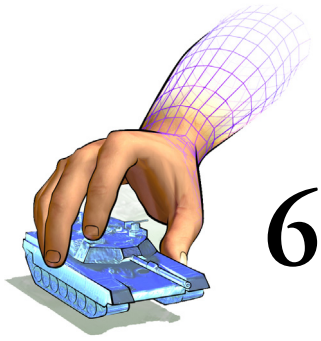
```
?.dll;%MAK_VRFORCESDIR%\
  data\parameters\BHAVE\scripts\modules\?.dll;
  %LUA_DIR%\lib\?.dll;
```

5.4. Displaying B-HAVE Module Data in the VR-Forces Front-End

The B-HAVE Module plug-in outputs graphical information that can be read by the VR-Forces front-end.

Debug lines are created with the functions DtDrawLine() and DtDrawPoint(). You can use these functions to draw any kind of visuals for debugging. For example, you could draw lines to every visible enemy entity.

To display B-HAVE Module data, the front-end must be running on the same computer as the VR-Forces back-end that is generating it and you must enable Show B-HAVE Debug Lines.



6

B-HAVE Messages API

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

6.1. Introduction

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

6.2. Sending B-HAVE Tasks

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

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

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

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

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

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

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

6.3. Sending B-HAVE Set Data Requests

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

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

6.3.1. Set Script

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

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

6.3.2. Set Movement Mode

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

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

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

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

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

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

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

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

6.5. Sending Interface Messages

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

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

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

The following example requests the list of Lua scripts available:

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

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

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

```
remoteController()->vrfMessageInterface()->addMessageCallback(
  DtLuaScriptListType, LuaScriptListCallback, …);
```

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

6.6. Using the API

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

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

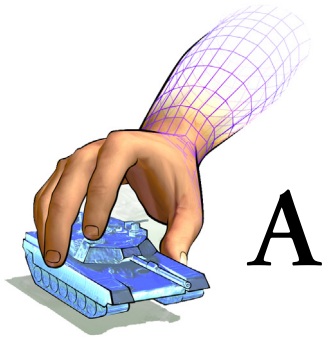
```
registerBhaveTasks(controller()->taskFactory());
```

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

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

6.7. Building a Project

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



A

Lua Functions for the B-HAVE Module

This appendix describes the functions that you can use in Lua scripts with the B-HAVE Module.

A.1. Callback Functions

All VR-Forces Lua scripts must contain one or more of the following callback functions. The functions will be called by the Lua controller at the appropriate moment. No other functions shall be called by the scripting system. If you do not have one of these functions, your Lua script will simply not do anything. Most of these functions take any arguments although the default 'this' object, explained below, shall be defined globally for each script.

Table A-1: Callback functions

Function	Description
`onCreate()`	Called on first tick. This is usually right at the beginning of a scenario or on loading of a Lua script.
`onLoadCheck-point()`	Called immediately after a checkpoint has happened. This is essentially the exact opposite of the above script.
`onRadioMes-sage(inte-ger type, string text)`	Called whenever a radio message intended for this entity is received. Radio messages can be used to communicate information between Lua scripts. @param type. An integer that defines the type of Radio Message. @param test. A String that defines the payload of this radio message.
`onSaveCheck-point()`	Called immediately before a checkpoint happens. During a checkpoint the status of your script will be saved, so you may add any extra information to your script you want to keep using this call.
`onShutdown()`	Called once when an entity is deleted. This does not happen when an entity is destroyed.
`onTeleport()`	Called whenever a unit has been teleported through external means. Meaning using the set location message instead of actuator movement.
`onTick()`	Called once every tick when an exercise is running. It should be noted that this is called even if the entity has been destroyed. Your script should take this into account.

A.2. Global Functions

The following functions are located in the global Lua table and can be called by any Lua script at any time.

Table A-2: Global functions (Sheet 1 of 4)

Function	Description
`DtAddSubordinate(LuaVRFObject object, LuaVRFObject parent)`	Turns the object into a subordinate of the parent.
`DtCreateAggregate(string name, string type, number force, number positionX, number positionY, number positionZ, number heading, bool createSubordinates)`	Create an aggregate with the specified parameters. The entity shall be created in the backend of the calling function. Due to the fact that this function is not blocking, and the object does not necessarily immediately get created, there is no return value. @param name A String identifying the name of the entity to be created @param type. A String identifying the entity type in the form: "2:3:4:5:6:7:8". @param force. A number identifying the force. That is, 0 for other, 1 for friendly, 2 for opposing, and 3 for neutral, or any other defined force. @param positionX, positionY, positionZ. A number identifying the position to place the entity in the local terrain coordinate system. @param createSubordinates. If this is true, all subordinate entities shall be created also.
`DtCreateAggregate(string name, string type, number force, number positionX, number positionY, number positionZ, number heading, List<LuaVRFObjects> subordinates)`	Create an aggregate with the specified parameters. The entity shall be created in the backend of the calling function. Due to the fact that this function is not blocking, and the object does not necessarily immediately get created, there is no return value. This function differs from the above in that this one takes pre-created subordinates to create the aggregate. @param name A String identifying the name of the entity to be created @param type. A String identifying the entity type in the form: "2:3:4:5:6:7:8". @param force. A number identifying the force. That is, 0 for other, 1 for friendly, 2 for opposing, and 3 for neutral, or any other defined force. @param positionX, positionY, positionZ. A number identifying the position to place the entity in the local terrain coordinate system. @param subordinates. This is a Lua array containing a list of all subordinates that should be attached to this entity.

Table A-2: Global functions (Sheet 2 of 4)

Function	Description
`DtCreateControlArea(List points, string name)`	Creates a control area using the points specified. @param points. A list of numbers. Every 3 numbers specifies a location (pointX,pointY,pointZ,pointX2,pointY2,pointZ2 ...). @param name. Optional, but when filled will name your object the specified name.
`DtCreateDisaggregation-Area(List points, string name)`	Creates a disaggregation area using the points specified. @param points. A list of numbers. Every 3 numbers specifies a location (pointX,pointY,pointZ,pointX2,pointY2,pointZ2 and so on). @param name. Optional, but when filled will name your object the specified name.
`DtCreateEntity(string name, string type, number force, number positionX, number positionY, number positionZ, number heading)`	Create an entity with the specified parameters. The entity shall be created in the backend of the calling function. Due to the fact that this function is not blocking, and the object does not necessarily immediately get created, there is no return value. @param name. A String identifying the name of the entity to be created. @param type. A String identifying the entity type in the form: "2:3:4:5:6:7:8". @param force. A number identifying the force. That is, 0 for other, 1 for friendly, 2 for opposing, and 3 for neutral, or any other defined force. @param positionX, positionY, positionZ. A number identifying the position to place the entity in the local terrain coordinate system. @param heading. The heading in radians measured clockwise from north.
`DtCreateObstacle(List points, string name)`	Creates an obstacle using the points specified. @param points. A list of numbers. Every 3 numbers specifies a location (pointX,pointY,pointZ,pointX2,pointY2,pointZ2 and so on). @param name. Optional, but when filled will name your object the specified name.
`DtCreatePhaseLine(number originX, number originY, number originZ, number endX, number endY, number endZ, string name)`	Creates a phase line from the origin to the end point. The name parameter is optional, but when filled will name your object the specified name.

Table A-2: Global functions (Sheet 3 of 4)

Function	Description
`DtCreateRoute(List routePoints, string name)`	Creates a route using the points specified. @param routePoints. A list of numbers. Every 3 numbers specifies a location (pointX,pointY,pointZ,pointX2,pointY2,pointZ2 etc).
`DtCreateWaypoint(number positionX, number positionY, number positionZ, string name)`	Creates a waypoint at the position indicated. the name parameter is optional, but when filled will name your waypoint the specified name.
`DtGetExerciseTime()`	@return the current exercise clock in seconds (start time + elapsed time).
`DtGetExerciseStartTime()`	@return the time the exercise started, expressed in seconds.
`DtGetTerrainElevation(number posX, number posY, number posZ)`	Get the current terrain elevation at a specific point. posZ is only required on geocentric terrains. @return a number containing the terrain height above sea level.
`DtGetVisibleObjects(LuaVRFObject obj)`	@return A Lua table filled with all LuaVRFObjects in your scenario that the passed object can see. A seen object is an object that can trace an uninterrupted line of sight line to the target.
`DtGetVrfObjects()`	@return A Lua table filled with all LuaVRFObjects in your scenario. If you have a large number of objects in your scenario, this is a fairly computationally heavy function so you should only use it sparingly. A LuaVRFObject is a table with a number of functions described below.
`DtGetVrfObjectByGlobalId(string globalID)`	@return a single LuaVRFObject that is identified by the included globalID. All Lua Global IDs are strings. In the case of DIS the string is defined as the triple divided by a ", ". For example: "2, 5, 2005". A LuaVRFObject is a table with a number of functions described below.
`DtGetVrfObjectByName(string name)`	@return a single LuaVRFObject that is identified by the included name. A LuaVRFObject is a table with a number of functions described below.

`DtInfo(Variant text)`
`DtVerbose(Variant text)`
`DtWarn(Variant text)`
`DtError(Variant text)`
`DtDebug(Variant text)`
	All of these functions do the same thing: output the variant text much in the same way as the similarly named C++ functions do. The output could be the console or a file depending on the configuration options selected at startup.
	@param text. The output to console. This could be a number, a string, any other Lua construct, even an entire Lua table.

Table A-2: Global functions (Sheet 4 of 4)

Function	Description

`DtInfo(LuaVRFObject obj, Variant text)`
`DtVerbose(LuaVRFObject obj, Variant text)`
`DtWarn(LuaVRFObject obj, Variant text)`
`DtError(LuaVRFObject obj, Variant text)`
`DtDebug(LuaVRFObject obj, Variant text)`
	In addition to outputting to console, if the above functions are called with a LuaVRFObject as the first parameter, the output shall be the object console window instead of the standard output.

`DtQueryLineOfSight (number posX, number posY, number posZ, number targetX, number targetY, number targetZ)`	Queries a line-of-sight test between the posXYZ point and the target XYZ point. @return 4 numbers, The first one is 1 if there points can see each other. 0 otherwise. If 0, the next three points are the XYZ coordinates of the terrain intersection point between the two locations.
`DtRadioSendTextMessage (string text, string receiver)`	Send a text message using the radio. @param text - The radio message to send. @param receiver - The name of the entity that shall receive the message.
`DtRemoveEntity(LuaVRFObject object)`	Delete the entity specified by the passed object. After this function the object is no longer valid.
`DtRemoveObject(string name)`	Delete an object specified by it's name. This could be an entity as well as any other type of object.
`DtRemoveSubordinate(LuaVRFObject object, LuaVRFObject parent)`	Stops the object from being a subordinate of the parent. Note that the object needs to be attached to a different parent or it might be lost in limbo. See DtCgf->removeSubordinate for more information.

A.3. LuaVRFObject Functions

In Lua a DtVRFObject is represented by a corresponding LuaVRFObject. Some of the global functions use and return LuaVRFObjects. These LuaVRFObjects are Lua Tables that contain a local reference to the corresponding C++ object. They also contain member functions that may be used to manipulate the object. Usage is just like other Lua member functions; that is, if you have an object myEntity, you can call a function like this: myEntity:taskEmbarkOn(myTarget).

Table A-3 describes functions for assigning tasks to entities. Table A-4 describes functions for assigning set data requests to entities. Table A-5 describes functions for getting the state of an entity.

Table A-3: VR-Forces Lua task object functions

Function	Description
`taskBHAVEMoveToLocation(number positionX, number positionY, number positionZ)`	Task your LuaVRFObject to move towards a location using BHAVE pathdata. @param positionX,Y,Z - The target location in local coordinates
`taskBHAVEMoveToWaypoint(string waypointName)`	Task your LuaVRFObject to move towards a waypoint using BHAVE pathdata. @param waypointName - The name of the waypoint to traverse
`taskBHAVEMoveAlongRoute (string routeName, boolean reverseRoute)`	Task your LuaVRFObject to move along a route using BHAVE pathdata. @param routeName - The name of the route to traverse @param reverse - If set to true your entity will traverse the route in reverse, otherwise it will traverse normally
`taskBHAVEPatrolRoute (string routeName, boolean reverseRoute)`	Task your LuaVRFObject to patrol along a route using BHAVE pathdata. @param routeName - The name of the route to traverse @param reverse - If set to true your entity will traverse the route in reverse, otherwise it will traverse normally
`taskBHAVEPatrolBetween (string waypointName1, string waypointName2)`	Task your LuaVRFObject to patrol between two waypoint using BHAVE pathdata. @param waypointName1 - The name of the first waypoint @param waypointName2 - The name of the second waypoint
`taskBHAVEFollow (string entityName)`	Task your LuaVRFObject to follow another entity. @param entityName - The name of the entity to follow

Table A-3: VR-Forces Lua task object functions

Function	Description
`taskBHAVEHide (string entities)`	Task your LuaVRFObject to hide from a list of entities @param entities - The name of the entities to hide from, divided by ','.
`taskBHAVEFlee (string entities)`	Task your LuaVRFObject to flee from a list of entities. @param entities - The name of the entities to flee from, divided by ','.
`taskBHAVEWander (number time, string area)`	Task your entity to wander. @param time - the time in seconds to wander. If this is set to 0, your entity wanders indefinitely. @param area - the area to wonder in. If set to null or "", your entity will wanders the entire terrain.
`taskDisembark()`	Task your LuaVRFObject to disembark from whichever entity it is currently embarked in.
`taskDisembarkAll()`	Task all objects embarked on your LuaVRFObject to disembark.
`taskEmbarkOn(string` *`name`*`)`	Task your LuaVRFObject to embark on the entity specified by 'name'.
`taskFireCruiseMissile(string` *`targetName`*`, string` *`routeName`*`, number` *`detonationProximity`*`, number` *`speed`*`)`	@param targetName. The name of the target @param routeName. The name of the route to use to reach the target or "" if you want to fire directly. @param detonationProximity. Distance in meters from the target before the cruise missile explodes. @param speed. The speed to travel towards it's target.
`taskFireForEffectOnEntity(string` *`entityName`*`, integer` *`numberOfRounds`*`, integer` *`heightAboveTerrain`*`)`	Task your LuaVRFObject to fire for effect. @param entityName. The name of the target entity. @param numberOfRounds. Number of rounds to fire/ @param heightAboveTerrain. Set target to number of meters to fire above the terrain.
`taskFireForEffectOnLocation(number` *`positionX`*`, number` *`positionY`*`, number` *`positionZ`*`, integer` *`numberOfRounds`*`, integer` *`heightAboveTerrain`*`)`	Task your LuaVRFObject to fire for effect. @param positionX,Y,Z. The target location in local coordinates. @param numberOfRounds. Number of rounds to fire. @param heightAboveTerrain. Set target to number of meters to fire above the terrain.

Table A-3: VR-Forces Lua task object functions

Function	Description
`taskFireForEffectOnTarget(string name, integer numberOfRounds, integer heightAboveTerrain)`	Task your LuaVRFObject to fire for effect. @param name. The name of the target. @param numberOfRounds. Number of rounds to fire. @param heightAboveTerrain. Set target to number of meters to fire above the terrain.
`taskFollowEntity(string entityName, number behindDistance, number rightDistance)`	Task your LuaVRFObject to follow an entity. @param entityName. The name of the target entity. @param behindDistance. The number of meters to stay behind the target. A negative value would put you in front of the target. @param rightDistance. The number of meters to stay to the right of the target. A negative value would put you to the left of the target.
`taskInterceptAndDestroy(string entityName, number standOffDistance)`	Task your LuaVRFObject to Intercept and destroy a target. @param entityName. The name of the target entity. @param standOffDistance. The distance in meters from the target that this LuaVRFObject should stop
`taskMoveAlongRoute(string routeName, boolean reverseRoute, boolean startAtClosestLoc)`	Task your LuaVRFObject to move along a route. @param routeName. The name of the route to traverse. @param reverse. If set to true your entity will traverse the route in reverse, otherwise it will traverse normally. @param startAtClosestLoc. If set to true your entity will start traversing the route from its closest point, otherwise it will traverse from the first point.
`taskMoveToLocation(number positionX, number positionY, number positionZ)`	Task your LuaVRFObject to move towards a location. @param positionX,Y,Z. The target location in local coordinates.
`taskMoveToaWaypoint(string waypointName)`	Task your LuaVRFObject to move towards a waypoint. @param waypointName. The name of the waypoint to traverse.
`taskPatrolAlongRoute(string routeName)`	Task your LuaVRFObject to patrol along a route. @param routeName. The name of the route to patrol.

Table A-3: VR-Forces Lua task object functions

Function	Description
`taskPatrolBetween(string` *`waypointName1`*`, string` *`waypointName2`*`)`	Task your LuaVRFObject to patrol between two waypoint. @param waypointName1. The name of the first waypoint. @param waypointName2. The name of the second waypoint.
`taskSendRadioSet(string` *`setName`*`, string` *`recipient`*`, ...)`	Task your LuaVRFObject to send a set message to another entity through the radio Set messages are described later in this section and can be identified by the fact that they start with the words "set", such as "setDestroyed". @param setName the string representation of the set to create. The following sets are supported: Destroyed, Disembarked, Embarked, FireNow, Heading, Location, OrderedSpeed, Posture, Restore, SectorOfResponsibility, SpotReport, Target, WeaponState. @param *`recipient`*. The name of the target entity to receive the message. @param .... Any extra parameters that the set requires. See the set functions below to verify what your parameter needs.
`taskSendRadioTask(string` *`taskName`*`, string` *`recipient`*`, ...)`	Task your LuaVRFObject to send a task message to another entity through the radio. task messages are described in this section and can be identified by the fact that they start with the words "task", such as "taskDisembark". @param taskName the string representation of the task to create. The following tasks are supported: Disembark, DisembarkAll, EmbarkOn, FireCruiseMissile, FireForEffectOnEntity, FireForEffectOnLocation, FireForEffectOnTarget, FollowEntity, InterceptAndDestroy, MoveAlongRoute, MoveToLocation, MoveToWaypoint, PatrolBetween, PatrolAlongRoute, SendTextMessage, TurnToHeading, UserTask, Wait, WaitDuration, WaitElapsed. @param *`recipient`*. The name of the target entity to receive the message. @param .... Any extra parameters that the task requires. See the task functions below to verify what your parameter needs.

Table A-3: VR-Forces Lua task object functions

Function	Description
`taskSendTextMes-sage(string recipient, string message)`	Task your LuaVRFObject to send a text message to another entity through the radio. @param *recipient*. The name of the target entity to receive the message. @param message. The string payload of the message being sent
`taskTurnToHeading(number heading)`	Task your LuaVRFObject to turn to a specified heading. @param heading. The specified heading in degrees.
`taskUserTask(string taskName, string arg1, string arg2, string arg3, string arg4)`	Task your LuaVRFObject to perform a custom user specified task. The argument are dependent on the defined user task.
`taskWait()`	Task your LuaVRFObject to stop and wait.
`taskWaitDuration(number duration)`	Task your LuaVRFObject to stop and wait for the specified number of seconds
`taskWaitElapsed(number duration)`	Task your LuaVRFObject to stop and wait until the given amount of simulation time has elapsed, relative to the start time of the simulation.

Table A-4: VR-Forces Lua set data request object functions

Function	Description
`setAltitude(number altitude)`	Set your LuaVRFObject's altitude to the specified number in meters
`setDestroyed()`	Set your LuaVRFObject's health state to destroyed.
`setDIGuyAppear-ance(string appearanceName)`	Set your LuaVRFObject's DI Guy Appearance to the specified value. See the C++ API's DtDiGuySetAp-pearanceRequest for more details on this value. Not that this only with DIs.
`setDIGuyAnimation(string animationName)`	Set your LuaVRFObject's DI Guy Animation to the specified value. See the C++ API's DtDiGuySetAni-mationRequest for more details on this value. Not that this only with DIs.
`setDisembarked()`	Set your LuaVRFObject to unmount itself from it's parent entity.
`setEmbarked(string parentName)`	Set your LuaVRFObject to mount itself to it's parent entity.
`setFireNow(string targetName)`	Set your LuaVRFObject to fire immediately towards the specified target.

Table A-4: VR-Forces Lua set data request object functions

Function	Description
`setHeading(number` *`heading`*`)`	Set your LuaVRFObject to change it's heading. Unlike the task command this will cause the object to immediately clamp to the new heading as opposed to moving towards it. @param heading. The specified heading in degrees.
`setLocation(number` *`positionX`*`, number` *`positionY`*`, number` *`positionZ`*`)`	Set your LuaVRFObject to change it's location. Unlike the task command this will cause the object to immediately clamp to the new location as opposed to moving towards it. @param positionX,Y,Z. The target location in local coordinates.
`setOrderedSpeed(number` *`orderedSpeed`*`)`	Set your LuaVRFObject's ordered speed. The object will do its best to move at this speed but may have to move slower if circumstances do not let it move faster. @param orderedSpeed. Speed in meters per second.
`setPosture(string` *`posture`*`)`	Set your LuaVRFObject's posture to the specified value. This only works with lifeforms. @param posture. A string representing the posture. The following posture options are available: Standing, Walking, Running, Kneeling, Prone, Crawling, Swimming, Parachuting, Jumping, Sitting, Squatting, Crouching, Wading.
`setRestore()`	Set your LuaVRFobject to restore back to life. This is the opposite of setDestroyed()
`setsectorOfResponsibil-ity(number` *`center`*`, number` *`size`*`)`	Set your LuaVRFObject's sector of responsibility. See C++ API's DtSetScanSectorRequest. @param center. The scan center in degrees @param size. The scan amount in degrees.
`setSpotReports(boolean` *`reporting`*`)`	Set your LuaVRFObject to send spot reports. @param reporting. True if you want to enable reporting, false otherwise.
`setTarget(string` *`objectName`*`)`	Set your LuaVRFObject's target to the specified named object.
`setWeaponState(string` *`weaponState`*`)`	Set your LuaVRFObject's weapon state. @param weaponState. A String that describes your new weapon state. The following options are available: Stowed, Deployed, In Fire Position.

Table A-5: VR-Forces Lua entity status object functions (Sheet 1 of 2)

Function	Description
`isCurrentlyTasked()`	@return true if the entity is currently tasked, false otherwise
`isLocal()`	@return true if the entity is local to the backend that is currently running this script, false otherwise
`isVrfSimulatedObject()`	@return true if the entity is a VR-Forces object that is currently being simulated, false otherwise.
`isExecutingPlan()`	@return true if the entity has an active plan that is currently running, false otherwise.
`isEmbarked()`	@return true if the entity is currently embarked, false otherwise
`isDestroyed()`	@return true if the entity is destroyed, false otherwise.
`location()`	@return 3 numbers: x,y,z representing the local position of the entity
`heading()`	@return the heading in radians.
`orientation()`	@return 3 numbers: psi, theta, phi representing the local orientation of the entity.
`objectName()`	@return A string representing the name of this object.
`objectLabel()`	@return A string representing the display label of this object.
`overlayParent()`	@return A string representing the name of the overlay parent.
`entityIdentifier()`	@return A string representing the entity identifier. In the case of DIS, the tuple identifier is turned into a string.
`entityType()`	@return A string representing the entity type. This is the 7 tuple number that should look like this: "1:2:3:4:5:6:7"
`extent()`	@return 6 numbers that identify the box extent of your entity: minX, minY, minZ, maxX, maxY, maxZ
`velocity()`	@return 3 numbers, x,y,z, that represent the velocity vector in meters per second in local coordinate.
`orderedSpeed()`	@return a number representing the ordered speed of the entity. Note that ordered speed and actual speed are not necessarily exactly the same.

Table A-5: VR-Forces Lua entity status object functions (Sheet 2 of 2)

Function	Description
`rotationalVelocity()`	@return 3 numbers, x,y,z, that represent the rotational velocity vector in meters per second in local coordinate.
`acceleration()`	@return 3 numbers, x,y,z, that represent the acceleration vector in meters per second squared in local coordinate.
`getVertexes()`	@return a table filled with numbers. Each 3 numbers represent a single vertex in your object.

A.4. Functions for the 'this' Object

Every Lua script automatically includes a "this" object. "this" is an extended LuaVRFObject that describes the object that is running the Lua Script. Or more accurately the object that contains the Lua Controller that is running the script. In addition to containing all of the functions that a normal LuaVRFObject contains, the "this" object contains the following extra functions:

Table A-6: Object functions for the 'this' object

Function	Description
`addVertex(number x, number y, number z)`	Adds a vertex to your entity in the included location in local coordinates.
`setAcceleration(number x, number y, number z)`	Sets the acceleration of your entity in local coordinates.
`setDestroyed(boolean destroyed)`	Sets your entity to be destroyed if the boolean is true, or alive if the boolean is false
`setEntityIdentifier(string id)`	Sets the entity identifier of your entity. This is a string, so in DIS the value that must be passed should be the string representation of your tuple.
`setEntityType(string type)`	Sets the entity type of your entity. This is a string, so the value that must be passed should be the string representation of your tuple of the form "1:2:3:4:5:6:7".
`setObjectLabel(string label)`	Sets the label of your entity.
`setObjectName(string name)`	Sets the name of your entity.
`setOverlayParent(string parent)`	Sets the overlay parent of your entity.
`removeVertex(number loc)`	Remove the 'loc' vertex of your entity. **i** loc does not specify the location, but the Nth vertex in your entity.
`setRotationalVelocity(number x, number y, number z)`	Sets the rotational velocity of your entity in local coordinates.
`setVelocity(number x, number y, number z)`	Sets the velocity of your entity in local coordinates.

A.5. B-HAVE-only Global Functions

Table A-7 describes global functions that are only included in the B-HAVE LUA module and not the regular VR-Forces module:

Table A-7: B-HAVE module global functions

Function	Description
`DtDrawLine (number originX, number originY, number originZ, number endX, number endY, number endZ, integer red, integer green, integer blue)`	Draw a line from the origin to the endpoint. The color of the line is determined by the last three parameters whose value may vary from 0 to 255.
`DtDrawPoint (number originX, number originY, number originZ, integer red, integer green, integer blue)`	Draw a point from the origin to the endpoint. The color of the point is determined by the last three parameters whose value may vary from 0 to 255.

A.6. B-HAVE-Only VR-Forces Lua Object Functions

This section describes object functions that are part of the B-HAVE Lua module. They are not in the VR-Forces Lua module.

Table A-8 describes functions for assigning tasks to entities. Table A-9 describes functions for assigning set data requests to entities.

Table A-8: B-HAVE module task functions

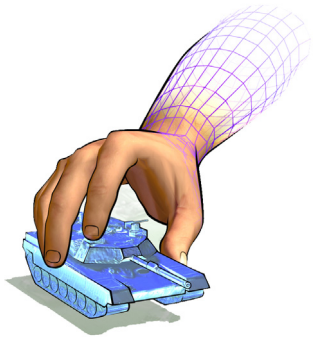
Function	Description
`taskBHAVEMoveToLocation (number positionX, number positionY, number positionZ)`	Task your LuaVRFObject to move towards a location using BHAVE pathdata. @param positionX,Y,Z - The target location in local coordinates.
`taskBHAVEMoveToWaypoint (string waypointName)`	Task your LuaVRFObject to move towards a waypoint using BHAVE pathdata. @param waypointName - The name of the waypoint to traverse.
`taskBHAVEMoveAlongRoute (string routeName, boolean reverseRoute)`	Task your LuaVRFObject to move along a route using BHAVE pathdata. @param routeName - The name of the route to traverse. @param reverse - If set to true your entity will traverse the route in reverse, otherwise it will traverse normally.
`taskBHAVEPatrolRoute (string routeName, boolean reverseRoute)`	Task your LuaVRFObject to patrol along a route using BHAVE pathdata. @param routeName - The name of the route to traverse. @param reverse - If set to true your entity will traverse the route in reverse, otherwise it will traverse normally.
`taskBHAVEPatrolBetween (string waypointName1, string waypointName2)`	Task your LuaVRFObject to patrol between two waypoint using BHAVE pathdata. @param waypointName1 - The name of the first waypoint. @param waypointName2 - The name of the second waypoint.
`taskBHAVEFollow (string entityName)`	Task your LuaVRFObject to follow another entity. @param entityName - The name of the entity to follow.
`taskBHAVEHide (string entities)`	Task your LuaVRFObject to hide from a list of entities @param entities - The name of the entities to hide from, divided by ','.

Table A-8: B-HAVE module task functions

Function	Description
`taskBHAVEFlee (string` *`entities`*`)`	Task your LuaVRFObject to flee from a list of entities. @param entities - The name of the entities to flee from, divided by ','.
`taskBHAVEWander (number` *`time`*`, string` *`area`*`)`	Task your entity to wander. @param time - the time in seconds to wander. If this is set to 0, your entity wanders indefinitely. @param area - the area to wonder in. If set to null or "", your entity will wanders the entire terrain.

Table A-9: B-HAVE module set data request functions

Function	Description
`setBHAVEMovement-Type(string` *`movementType`*`)`	Set the BHAVE movement type for your entity. Possible options are "free", "prefer-roads", or "avoid-roads".
`setLuaScript(string` *`scriptName`*`)`	Set a different Lua Script for your entity. Note that OnCreate will be called for this script during the next tick.



Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Link - Simulate - Visualize

BHV-2.0.2-1-111107

68 MOULTON STREET CAMBRIDGE, MA 02138 617.876.8085 www.mak.com