



©Copyright 1999-2009 Telekinesys Research Ltd. (t/a Havok). All rights reserved. ¹

¹ Havok.com and the Havok buzzsaw logo are trademarks of Havok. All other trademarks contained herein are the properties of their respective owners.

This document is protected under copyright law. The contents of this document may not be reproduced or transmitted in any form, in whole or in part, or by any means, mechanical or electronic, without the express written consent of Havok. This document is supplied as a manual for the Havok game dynamics software development kit. Reasonable care has been taken in preparing the information it contains. However, this document may contain omissions, technical inaccuracies, or typographical errors. Havok does not accept any responsibility of any kind for losses due to the use of this document. The information in this document is subject to change without notice.

Contents

| | | |
|----------|--|----------|
| 1 | Havok PC Guide | 3 |
| 1.1 | Getting Started with Havok | 3 |
| 1.1.1 | Getting Started with the Binary Only PC version of Havok Physics and Havok Animation | 3 |
| 1.1.2 | Simple Applications | 4 |
| 1.1.3 | Building the Demos | 6 |
| 1.1.4 | Demo Command Line Arguments | 8 |
| 1.1.5 | Demo Controls | 8 |
| 1.2 | Havok libraries | 11 |
| 1.2.1 | Havok library link order | 12 |
| 1.2.2 | Havok header include policy | 13 |
| 1.2.3 | Linking With Havok | 13 |
| 1.2.4 | File naming convention | 15 |
| 1.3 | Setting up the Visual Debugger (VDB) | 15 |
| 1.3.1 | Win32 VDB setup | 15 |
| 1.3.2 | Windows VDB Caveats | 16 |

Chapter 1

Havok PC Guide

Welcome to the Havok PC Guide. This document contains information on using Havok with the PC. It explains everything needed to get the Havok demos up and running, and also covers all the unique features of the PC, and how they're leveraged by the Havok SDK.

1.1 Getting Started with Havok

1.1.1 Getting Started with the Binary Only PC version of Havok Physics and Havok Animation

If you have successfully installed the distribution you should see the following directory structure

- hkXXX
 - Demos
 - Docs
 - Libs
 - Source
 - Tools

The version number is specified with XXX e.g. 5.5.0. In addition you can see the build number when you run the demos or at the bottom of any header file.

Running the demos

In the `Demo\Demos` folder you will find the main demo executable. This executable contains the 500+ demos that we use to demonstrate and test Havok. If you run this you will be presented with a menu screen. Controls to navigate the menu can be found in Demo controls.

Demos are categorised into Api demos, Feature demos and Show Case demos. Api demos illustrate how to use the API. They are typically very simple and straightforward. Their code is designed to be copied and

pasted into your application. Feature demos are designed to show off a particular piece of functionality e.g. the Vehicle SDK or character controller. Lastly the show case demos bring together several aspects and features to show Havok used in more complicated scenes. We suggest that you spend time running each of the demos to explore the extensive feature set that Havok Physics and Animation offers.

1.1.2 Simple Applications

The Havok demo framework contains a number of useful demos to demonstrate various parts of the SDK (see below for information how to build the demos). These demos are designed to be referenced as you read through this documentation. However, you may prefer to start by creating a simple mainline application independent of the Havok demo framework.

Included in the Havok distribution (in the **StandAloneDemos** directory) are sample applications demonstrating the minimum code necessary to get a Havok up and running.

To build these demos:

- Ensure that your Havok keycode(s) are present in **Common/Base/KeyCode.h**
- Choose a project or makefile for the platform and compiler you wish to build and execute on.
- Build and run the demo - for specific platform configuration see below

For more information on how to link and build your own project, see the sections Havok library link order and Havok header include policy.

Simple Console App

This application creates a scene with a sphere falling on a box and simulates this scene for 5 minutes. The simple application also does the minimum initialization required to set up the Havok Visual Debugger.

When you run the simple console application it will initialize the appropriate platform and then begin a simple simulation of a ball falling on a box and coming to rest. Each second the position of the ball is printed on the screen.

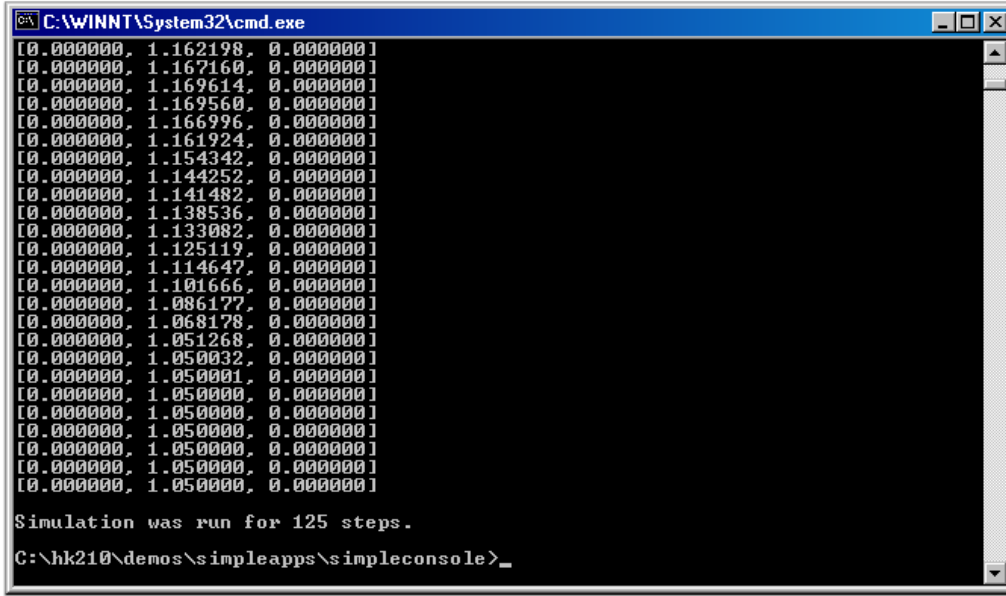


Figure 1.1: Display of output of Simple Console App

Connecting the Visual Debugger

Although this application has no graphical output it has initialized and opened a channel to communicate with the Havok Visual Debugger. The visual debugger is described in more detail in the User Guide, but basically it allows the simulation to be streamed over a network connection to a client application that can then display the simulation. The visual debugger client can be found at `tools\hkVisualDebugger\hkvisualdebugger.exe` and note that a full description of how to set up the visual debugger with your own application can be found below in the Platform Specific section.

In this application, a visual debugger server is created and can be connected to using `hkVisualDebugger.exe` (run it and click Network > Connect... > OK). The IP address to connect to should be as follows:

| | |
|------------------|--|
| Win32 / Linux | Use the IP of the machine running the Application. The client can be run from any other PC on the network, including the local PC running the application. |
|------------------|--|

Table 1.1: Visual Debugger IP address

NOTE: When you connect initially it can take up to 30 seconds for a visual representation to appear. To check if the network interface on the server side has been initialized

- Run the visual debugger enabled application.
- Open a command prompt and type `ping <IP address>` using the same IP address as above.
- You should see a reply from the target machine

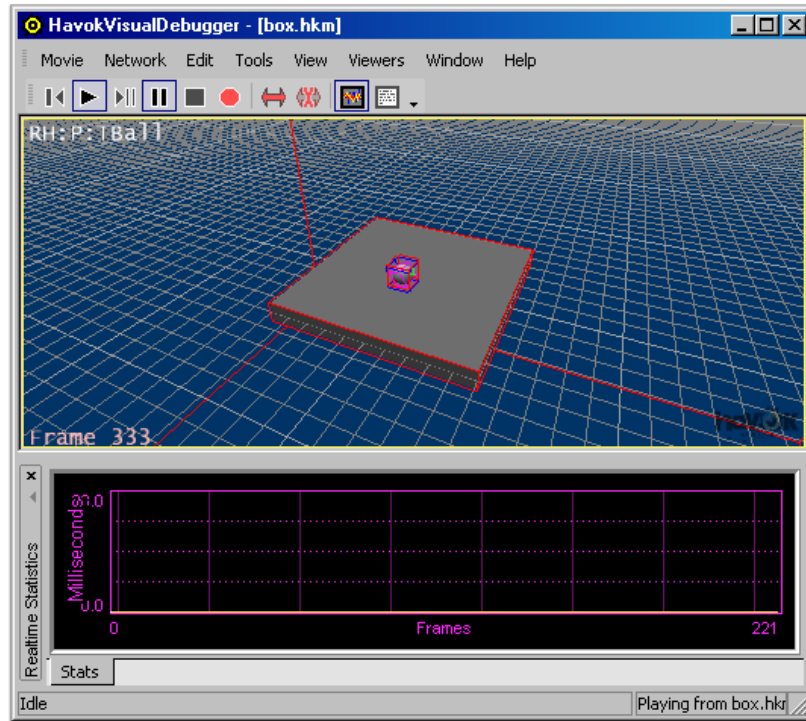


Figure 1.2: Display of Simple Visual Debugger App on the hkVisualDebugger2.exe

1.1.3 Building the Demos

Note that before the Havok demos can be compiled and run they must first be configured to match the Havok products that you have installed. For more information please refer to the Havok 6.0 Setup Guide.pdf document in the Docs directory.

Building for Win32

- Open the demo project in the `demo/demos` subdirectory of your distribution using Microsoft Visual Studio.
- Ensure that your Havok keycode(s) are present in `Common/Base/KeyCode.h`
- Select `Win32 Release` as the active configuration.
- Build and run.

The `demos_win32-vc_release.exe` or `demos_win32-net_release.exe` executable should be created in `demo/demos`.

Note:

If using Visual Studio 6.0 you need Microsoft Visual Studio 6.0 Enterprise Edition with Service Pack 5 and Visual C++ 6.0 Processor Pack installed. Both Packs are available from <http://msdn.microsoft.com/>
Errors produced when they are not installed include: `error 'fatal error C1600 : unsupported data type`

Note:

The Havok demo framework uses Microsoft's DirectX 9 Software Development Kit. If you already have this you should change your IDE options (see below). If you do not have a DirectX SDK, download and install the latest version (version 9.0 at time of writing). When the SDK is installed you may need to modify your IDE options. e.g. for Visual C++ 6.0 go to Tools/Options.../Directories(tab)/Show Directories for: There you need to add the paths to your DirectX SDK install e.g. Include files/"C:\DXSDK\include" and Library files/"C:\DXSDK\lib" (default DirectX 9.0 install paths)

Below are some screenshots of the relevant options for Visual C++ 6.0 on Windows XP

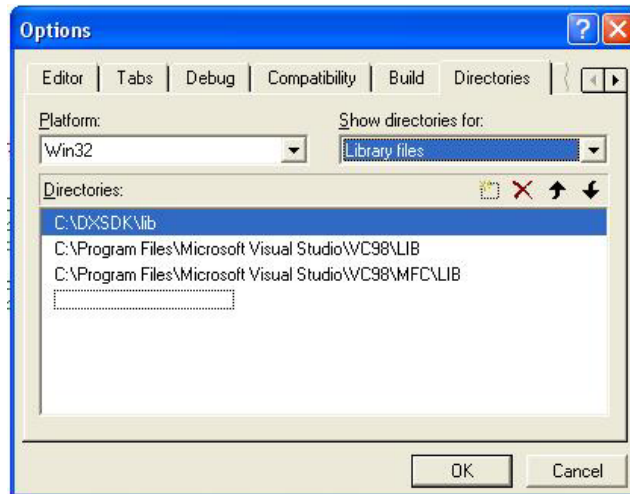


Figure 1.3: Setting the library options for DirectX9

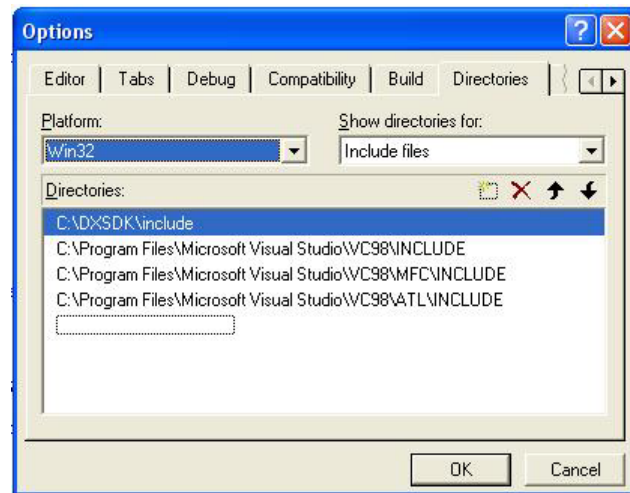


Figure 1.4: Setting the include options for DirectX9

To quickly change the build configuration from the command line you may alter an environmental variable named "HK_LINK_TYPE". Simply set this variable to the desired build specification and run **make** to begin compilation. E.g. from DOS type "set HK_LINK_TYPE=debug" to change the current specification to debug.

1.1.4 Demo Command Line Arguments

Command line arguments can be passed to the Havok Demo Framework using the standard command line method and by using a hkdemo.cfg file. The complete list of command line arguments is as follows:

| Option | Description | Win32 |
|-------------------|---|--------------------------|
| -b | Don't boot IOP, IOP is booted by default. | No |
| -c | Use debug memory manager, and report memory leaks. [Debug only] | Yes |
| -d | Run the visual debugger server. The visual debugger server is disabled by default. | Yes |
| -f | Run full screen. The demos run in windowed mode by default. | Yes |
| -g [name of demo] | Automatically run a specific demo, e.g. '-g Pyramid'. The default demo is the 'Menu' | Yes |
| -l[fps] | Lock the frame rate to that specified, e.g.-l60 | Yes |
| -p | Disable or enable the performance stats. The performance counters are enabled by default. | Yes |
| -r [none] | Enable or disable the renderer. The renderer is enabled by default. Disabling the renderer is useful for collecting performance stats unaffected by display code execution. | Yes, see below for more. |
| -w | Run the demos in windowed mode. The demos run in windowed mode by default on Win32. | Yes |
| -wiifw | Enable Wii-to-PC file writing (if -d is not also specified). | No |
| -lastdemo | If running one of the bootstrap demos start from the last demo run by the bootstrapper. | Yes |
| -nextdemo | If running one of the bootstrap demos start from the demo after the last demo run by the bootstrapper. | Yes |

Using the hkdemo.cfg file

The hkdemo.cfg is of most use for platforms that do not facilitate command line arguments or running from CD on a console. The hkdemo.cfg file must reside in the same directory as the demos executable. The file format is very simple, essentially a replica of the command line parameters, e.g:

```
; --- hkdemo.cfg ---  
  
; this is a comment  
  
; run the visual debugger server and the pyramid demo  
  
-d -g Pyramid  
  
; run in full screen mode and lock to sixty FPS  
  
-f -l60
```

Win32 renderer options

The "-r" argument specifies which renderer to use. For consoles the only available option is "-r none", which disables the renderer. However on Win32 you may choose any of the following DirectX variants: "d3d8", "d3d9", "d3d9s", or "ogl" for OpenGL mode. By default Win32 uses DirectX 9. "d3d9s" is the shader-only version of the DirectX 9 renderer.

1.1.5 Demo Controls

To control the demos we use an abstract user interface class. This class returns information about a virtual mouse, directional pad, analog joystick and a set of digital buttons. Picking is not implemented for console controllers.

The following tables show how each platform specific set of controls maps to our virtual user input device.

You can also see a menu of options with their corresponding controls for your platform at any time while playing the demos by pressing the Pause control. This pauses the demo while the information is displayed, and allows you to select one of the options. Press the key again to resume the demo.

If you are not sure which control or key on your input device corresponds to a control icon shown on-screen in the menu, you can select **common - gamepad** from the main demo menu. This displays the control icons. Using the appropriate control will highlight the corresponding icon.

PC Demo Controls



| Function | Role | Platform Specific Mapping |
|--|---|---|
| Select Demo | Selects a demo or submenu from the menu system. | Enter |
| Look | Allows you to rotate the camera used to view the scene. | Left Mouse Button |
| Fly | Allows you to move and rotate the camera used to view the scene. | Left Mouse Button + W, A, S, D, Q, Z keys |
| Pan | Allows you to pan the camera used to view the scene. | Right Mouse Button |
| Zoom | Allows you to zoom the camera used to view the scene. | Mouse Wheel or Alt + Left Mouse Button |
| Mouse Cursor | Moves the mouse cursor and point to dynamic object. | Mouse |
| Mouse Pick | Lets you interact with dynamic objects in the scene using the mouse. | Mouse with Space Bar held |
| Menu navigation | Lets you navigate through the menu system.. | Arrow Keys |
| Pause/Options/Resume | Use this control to pause the scene and access the available options. Press it again to resume. | Enter |
| Step (available while Paused) | Steps the scene forward one frame at a time. | Space Bar |
| Toggle Help (available while Paused) | When this option is selected, help information - such as the name of the demo - is displayed while the demo runs. | Del |
| Settings (available while Paused) | Where relevant, allows you to enter parameters to tweak the behavior of a demo. | End |
| Toggle Statistics (available while Paused) | When this option is selected, statistical information is displayed while the demo runs. | 1 |
| Quit (available while Paused) | Returns to the menu. | 2 |
| Restart Demo (available while Paused) | Restarts the demo. | 3 |

1.2 Havok libraries

Not all Havok libraries are required when building your application. These tables show each library in the distribution and detail any dependencies or requirements it has.

| <i>Library</i> | <i>Description</i> | <i>Dependencies</i> | |
|--------------------|--|--|--|
| hkbase | Contains platform and system dependant functionality for the other libraries. This is always required. You can rebuild or replace parts of this library to suit your specific configuration. | System libraries (e.g. libc, libm) | |
| hkmath | Core math library. This is always required. | hkbase | |
| hkcompat | Version compatibility library. Allows loading of assets from previous havok versions. | hkbase,hkserialize | |
| hkphysics | Core physics library. This links the constraint solver library with the collision detection library. This is always required. | hkcollide, hkconstraint-solver, hkmath, hkbase | |
| hkcollide | This module contains all the collision detection functionality used by Havok. | hkmath, hkbase | |
| hkserialize | The Havok import/export library. You can import/export Havok data into your own format and serialize custom created objects. | hkmath, hkbase | |
| hkvehicle | Provides the abstraction layer and default vehicle implementations for the Havok Vehicle Kit. This library is only required if you wish to use the Vehicle Kit. | hkphysics, hkcollide, hkconstraintsolver, hkmath, hkbase | |
| hkutilities | Contains a number of handy utilities, including example actions and the Visual Debugger. | hkphysics, hkcollide, hkconstraint-solver, hkserialize, hkmath, hkbase | |
| hkcompression | Utility methods for data compression, including wavelet compression, delta compression, block encoding and quantization. | hkmath, hkbase | |
| hkanimation | The central library for animation, including storage and playback, blending, motion extraction and skinning. | hkcompression, hkmath, hkbase | |
| hkconstraintsolver | The Havok constraint solver. This is always required. | hkmath, hkbase | |
| hkinternal | A public interface to Havok's internal classes and data structures. | hkphysics, hkcollide, hkmath, hkbase | |
| hkscenedata | This library contains the data descriptions that are extracted from the animation toolchain. | hkbase | |
| hkragdoll | Contains classes to setup and manipulate ragdolls. Physics and Animation required. | hkbase, hkmath, hkanimation, hkphysics | |

Table 1.2: Havok Sdk Libraries

| <i>Library</i> | <i>Description</i> | <i>Dependencies</i> |
|------------------|---|---|
| hkgraphics | Contains a cross-platform core graphics engine. | hkbase |
| hkgraphicsbridge | Provides the bridge between physics and graphics. | hkgraphics, hkutilities, hkphysics, hkcollide, hkconstraintsolver, hkmath, hkbase |
| hkgraphicsogl | OpenGL implementation of hkgraphics | hkgraphics, hkbase |
| hkgraphicsdx8 | DirectX 8 implementation of hkgraphics | hkgraphics, hkbase |
| hkgraphicsdx9 | DirectX 9 implementation of hkgraphics | hkgraphics, hkbase |
| hkgraphicsdx9s | DirectX 9s implementation of hkgraphics | hkgraphics, hkbase |
| hkdemoframework | Provides a consistent cross-platform framework for Havok demos. | hkgraphicsbridge, hkgraphics, hkutilities, hkphysics, hkcollide, hkinternal, hkconstraintsolver, hkmath, hkbase |

Table 1.3: Havok Demo Libraries

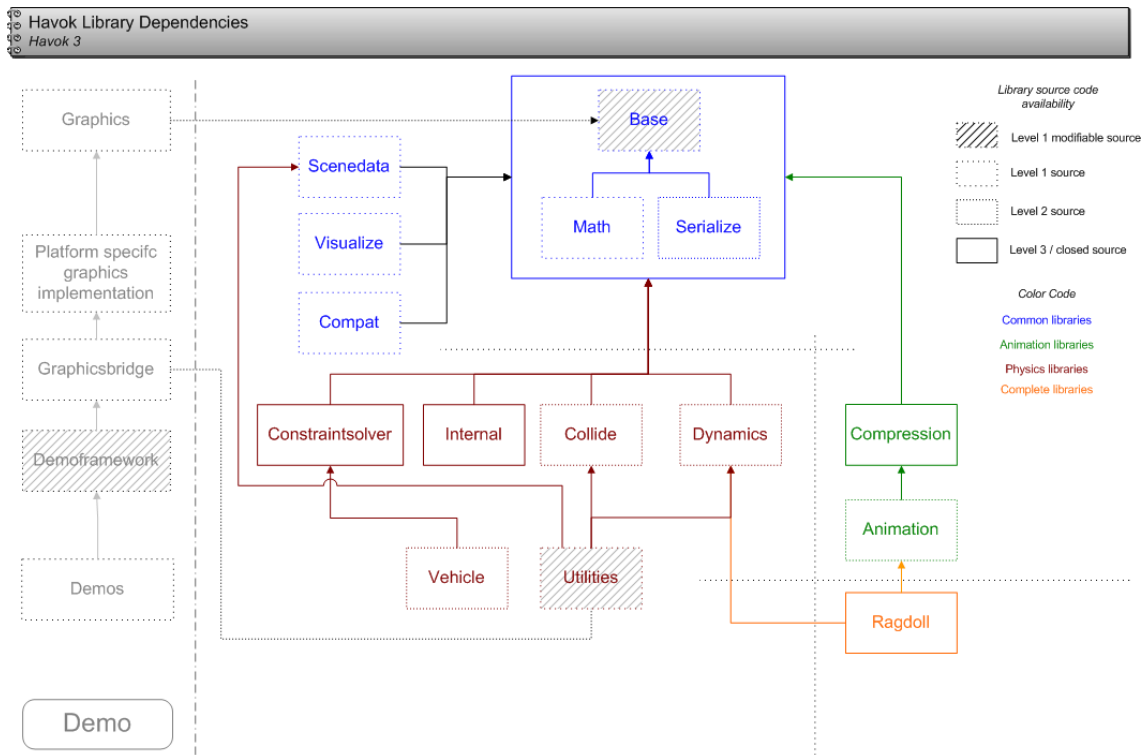


Figure 1.5: Havok library dependencies

1.2.1 Havok library link order

Using the above information we can extract the correct link order for the Havok libraries, which is (dependency increases from top to bottom):

Common libraries:

- hkbase
- hkmath
- hkserialize
- hkcompat

Havok Physics libraries:

- hkconstraintsolver
- hkcollide
- hkdynamics
- hkinternal
- hkutilities

- hkvehicle

Havok Animation libraries:

- hkcompression
- hkanimation
- hkragdoll (requires Havok Physics also)

When using the demo framework:

- hkgraphics
- hkgraphicsbridge
- hkgraphicsogl / hkgraphicsdx8
- hkdemoframework

If you encounter linker issues, ensure that you are grouping Havok library references using the `-Wl,-start-group` and `-Wl,-end-group` tags.

1.2.2 Havok header include policy

For any given library, there are a small set of headers which need to be included from almost every file. By convention this file is named identically to the library name. The general rule is to include this library header before including any other headers from that library. For example:

```
// Include the hkbase header before any other hkbase includes.
#include <Common/Base/hkBase.h>
#include <Common/Base/System/IO/IStream/hkIstream.h>
// more hkbase headers

// Include the hkdynamics header before any other hkdynamics includes.
#include <Physics/Dynamics/hkpDynamics.h>
#include <Physics/Dynamics/World/hkpWorld.h>
// more hkdynamics headers
```

NOTE: First time Havok users may choose to include the file `hkCompletePublicInclude.h` found in the havok directory `\sdk\include\`. This file contains all of Havok's public header files to simplify building a project; however, this may be the slowest to compile.

1.2.3 Linking With Havok

The Havok SDK has 3 build configurations: *Release*, *Debug* and *Fulldebug*. The libraries can be found in the Havok SDK folder under `Lib/Platform/Configuration/`.

- **Release**

The Release build configuration is a fully optimized build of Havok. It contains no debug assertions. Since there are no assertions, Havok may crash if used incorrectly - there is no error checking. For this reason it is recommended that all development takes place with the Debug or Fulldebug configuration of Havok. Symbols are included in Release builds on platforms where there is no associated performance overhead.

- **Debug**

The Debug build configuration is a fully optimized build of Havok, which also contains debug assertions and symbols. These assertions will catch errors in Havok, that would otherwise cause a crash. The assertions provide the user with the file and line number on which the problem occurred. A quick description of what went wrong is also provided, and the program execution is halted. This allows the user to see a full call-stack. Single stepping through Havok code may be error prone in with Debug libraries, as the compiler has fully optimized the code. It is recommended that developers working on systems related to the Havok products should link with Havok Debug libraries on a day to day basis. Any errors introduced / exposed by these developers will be caught by clear assertions. Such errors would cause a subsequent crash in Release libraries, which could be harder to diagnose.

- **Fulldebug**

The Fulldebug build configuration is not optimized, and contains assertions and debug symbols. This build configuration is useful if the user wants to debug into the Havok code. It is recommended that developers who are writing code that directly interfaces with the Havok SDK should link with Fulldebug libraries on a day to day basis.

On some platforms, the compiler will hard-code the path to the Havok source files within the Havok libraries. On these platforms, the debugger may not be able to locate the source code that corresponds to the current callstack when execution breaks within Havok code. Some debuggers will allow the user to manually locate the corresponding source file on disk, others will not. It is often beneficial to rebuild the Havok libraries locally, and check the newly build libraries into your source control system. This will ensure that the path to the source file is correct, and the debugger will open the corresponding file when execution breaks within Havok libraries. Note: some Havok internal libraries cannot be rebuilt.

The *Debug* and *Fulldebug* libraries will print debug information / logging / warnings to the TTY, and may have external dependencies on platform-specific debug libraries. The *Release* libraries will not depend on any debug libraries.

On PC there are also DLL variants of the Release, Debug and Fulldebug libraries. These libraries are designed to link with DLL version of any dependencies.

Havok Class Registration

Many of the classes in the Havok SDK store data that is used at runtime for simulation, e.g. physical objects, character animations, etc. The *hkSerialize* library contains functionality for saving this data and loading it later on.

Havok's serializable classes must be registered with the serialization infrastructure before they can be used. To do this add the following code, which checks the keycodes in *KeyCode.h* to determine which Havok products are in use and registers Havok classes accordingly:

```
// Register Havok classes.
#include <Common/Base/KeyCode.h>
#define HK_CLASSES_FILE <Common/Serialize/Classlist/hkKeyCodeClasses.h>
#include <Common/Serialize/Util/hkBuiltinTypeRegistry.cxx>
```

This code snippet is the default method for getting Havok's serialization up and running. It's also possible to add custom-created classes and only register some of the Havok classes instead of all of them. For more information refer to the Type Registration section in the Serialization chapter of the Havok Common manual.

1.2.4 File naming convention

Havok uses extensions to

.h: C++ declarations.

.inl: C++ inline function and method definitions to be included from a .h file.

.cpp: C++ definitions.

.cxx: Source file meant to be included from a .cpp file. e.g. super macro files (hkserialize/util/hkBuiltinTypeRegistry.cxx), and implementations which are selected at compile time (hkmath/linear/impl/hkFpuVector4.cxx).

[**class name** Class.cpp:]

Havok class metadata, usually extracted from the corresponding .h file.

1.3 Setting up the Visual Debugger (VDB)

1.3.1 Win32 VDB setup

The visual debugger uses WinSock version 2.02 to communicate over the network. It has been tested on Windows 2000 and Windows XP. The IP address of your game will be the IP address of the machine the visual debugger game side is running on. It is possible to run your game and the visual debugger application on the same machine though this not recommended unless you have a multiprocessor machine. Use 'localhost' or '127.0.0.1' as the IP Address to connect to your game running on the same machine. If you wish to connect to a remote PC the procedure is the same, you may use the name of the machine or use the IP address directly. You can obtain this information by running the 'ipconfig /all' command at the command prompt on Windows for example.

Please note that the Windows implementation of the visual debugger game side uses a #pragma directive to link with wsock32.lib. Some Havok libraries are also required, please see the section entitled 'The Visual Debugger Game Side' below for more information.

1.3.2 Windows VDB Caveats

There are no current special issues for this platform.