



©Copyright 1999-2010 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	Evaluation Checklist	3
1.1.1	Havok Common	3
1.1.2	Havok Physics	4
1.1.3	Havok Animation	4
1.2	Getting Started with Havok	4
1.2.1	Getting Started with the Binary Only PC version of Havok Physics and Havok Animation	4
1.2.2	Simple Applications	5
1.2.3	Building the Demos	7
1.2.4	Demo Command Line Arguments	9
1.2.5	Demo Controls	10
1.3	Step By Step Demos	13
1.4	Building with Havok	15
1.4.1	Havok Libraries	15
1.4.2	Havok Library Link Order	15
1.4.3	Havok Header Include Policy	15
1.4.4	Linking with Havok	15
1.4.5	Debugging Havok Libraries	17
1.4.6	File Naming Convention	17
1.5	Setting up the Visual Debugger (VDB)	18
1.5.1	Windows/Linux VDB setup	18
1.5.2	Windows VDB Caveats	18

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 Evaluation Checklist

This checklist is intended as a basic set of things to try when first evaluating or integrating Havok runtime products. Advanced product features and console/platform specifics are discussed elsewhere in the relevant documentation and demos. All of the items listed are recommended to be tried unless otherwise indicated. Additionally it is also probably best to attempt each of the items in the order specified, because later items generally build on earlier items.

The Step By Step demos, are a set of minimal console based demos each designed to demonstrate a specific feature. Details on the demos can be found in the Step By Step demo section below.

1.1.1 Havok Common

All of the Havok products rely on the fundamental infrastructure of the common library. One of it's most important benefits is in providing a high performance platform independent machine abstraction. More specific features of the common library include reflection, serialization, threading, containers, memory management and maths support.

- Look at and compile the MemoryUtilInit Step By Step demo in `Demo/StandAloneDemos/StepByStep`.
- Route Havok memory de/allocation through your memory system. See SDK documentation on Havok Base Library and Memory Customization.
- Implement your own error handler. A small investment of effort here will save a lot of time later. See SDK documentation on Creating a Custom Error Handler.
- Look at and compile the Serialize Step By Step demo.

- Write a demo to save and load an object using the serialization system.

1.1.2 Havok Physics

- Build and run Havok Physics Demos and Console Examples.
- Look at and compile the Physics and PhysicsVdb Step by Step demos.
- Create a simple (e.g. box-shape) fixed/static rigid body in game runtime.
- Attach the Havok Visual Debugger (VDB) to view the simple fixed object.
- Create simple dynamic rigid bodies in game, colliding with each other and the simple fixed object.
- Add VDB user cameras that correspond to game cameras. (Again this may seem trivial but it will also save time later). See SDK documentation on the Visual Debugger Game Side, the Debug Display and HK.UPDATE_CAMERA.
- Synchronize dynamic objects with game renderer. Confirm consistency using game and VDB. Experiment with VDB viewers to see shapes, contact points, simulation islands, broadphase AABBs, inertia tensors, statistics etc.
- Create more complicated fixed physical geometry e.g. polygon soup or heightfield.

And optionally

- Install Havok Content Tools (HCT) for modeler(s) of choice.

1.1.3 Havok Animation

- See all of "Havok Common" above.
- Use HCT to create skeleton, animation and (optionally) skin. Preview all elements in Havok Preview Tool.
- Serialize Havok Animation data from HCT into runtime.
- Use VDB to visualize skeleton pose (without worrying about skinning/game renderer).
- Pass transforms from final character pose to game skinning.

1.2 Getting Started with Havok

1.2.1 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. 2010.1.0.r1 for the 2010.1.0 release candidate 1). 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 (such as the vehicle SDK or the 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.

Note:

To get the full visual and audio quality of the showcase demos, ensure that you have Microsoft DirectX installed (August 2009 or later).

1.2.2 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.

```

C:\WINNT\System32\cmd.exe
[0.000000, 1.162198, 0.000000]
[0.000000, 1.167160, 0.000000]
[0.000000, 1.169614, 0.000000]
[0.000000, 1.169560, 0.000000]
[0.000000, 1.166996, 0.000000]
[0.000000, 1.161924, 0.000000]
[0.000000, 1.154342, 0.000000]
[0.000000, 1.144252, 0.000000]
[0.000000, 1.141482, 0.000000]
[0.000000, 1.138536, 0.000000]
[0.000000, 1.133082, 0.000000]
[0.000000, 1.125119, 0.000000]
[0.000000, 1.114647, 0.000000]
[0.000000, 1.101666, 0.000000]
[0.000000, 1.086177, 0.000000]
[0.000000, 1.068178, 0.000000]
[0.000000, 1.051268, 0.000000]
[0.000000, 1.050032, 0.000000]
[0.000000, 1.050001, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
[0.000000, 1.050000, 0.000000]
Simulation was run for 125 steps.
C:\hk210\demos\simpleapps\simpleconsole>_

```

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

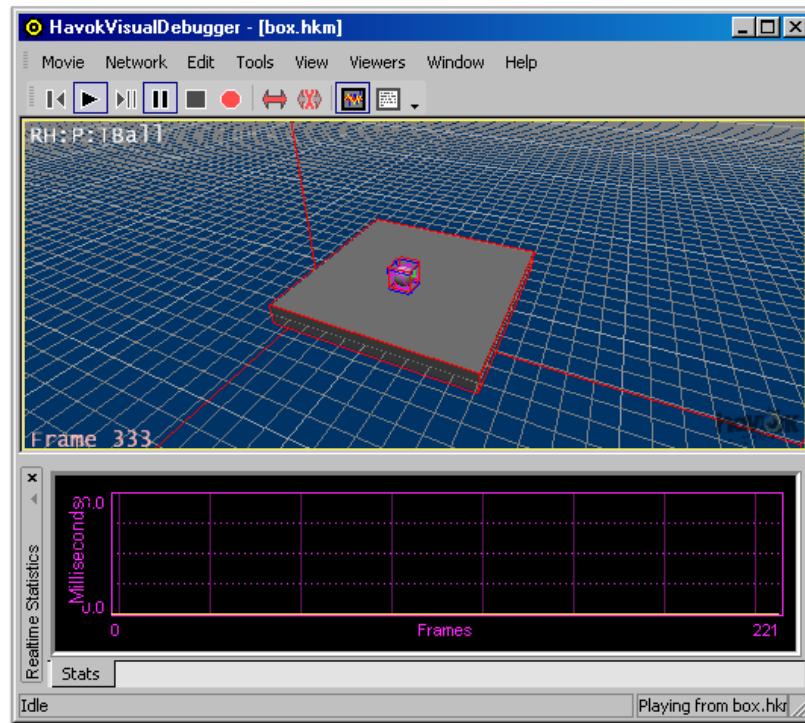


Figure 1.2: Display of SimpleConsoleApp on the Visual Debugger

1.2.3 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 Setup Guide.pdf document** in the Docs directory.

After you've finished the configuration process check that the compiler you plan to use is the same as the one used to compile the libraries in your Havok distribution. This information can be found in the Docs/PackageDetails directory, which contains a text file for each Havok SDK package describing the compiler, compiler version, etc. used to build it.

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 Multithreaded DLL** as the active configuration.
- Build and run.

The `Demos_win32-net_9-0-debug_multithreaded_dll.exe` executable will be created in `demo/demos` (with "net_9-0" corresponding to Visual Studio 2008 in this example).

DirectX

If you are only building the Demos project, then you **do not** need to install the Microsoft DirectX SDK (and you can safely skip this section). If however, you need to rebuild the entire Havok distribution, you will need to install the DirectX SDK (in order to build the `hkgGraphics` library).

By default, the Havok demos use Microsoft's DirectX for rendering. DirectX is also used for sound in certain showcase demos (e.g. the Cloth Troll Demo and the Destruction Train Demo).

If you already have installed the DirectX SDK, you should change your IDE options to reflect this (see below). If you do not have a DirectX SDK, download and install the latest version (version 11.0 at time of writing). When the SDK is installed you may need to modify your IDE options. For example, in Visual Studio (2005 and up) go to Tools->Options...->Projects and Solutions->VC++ Directories->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 install paths).

Below are some screenshots of the relevant options for Visual Studio 2008 on Windows Vista:

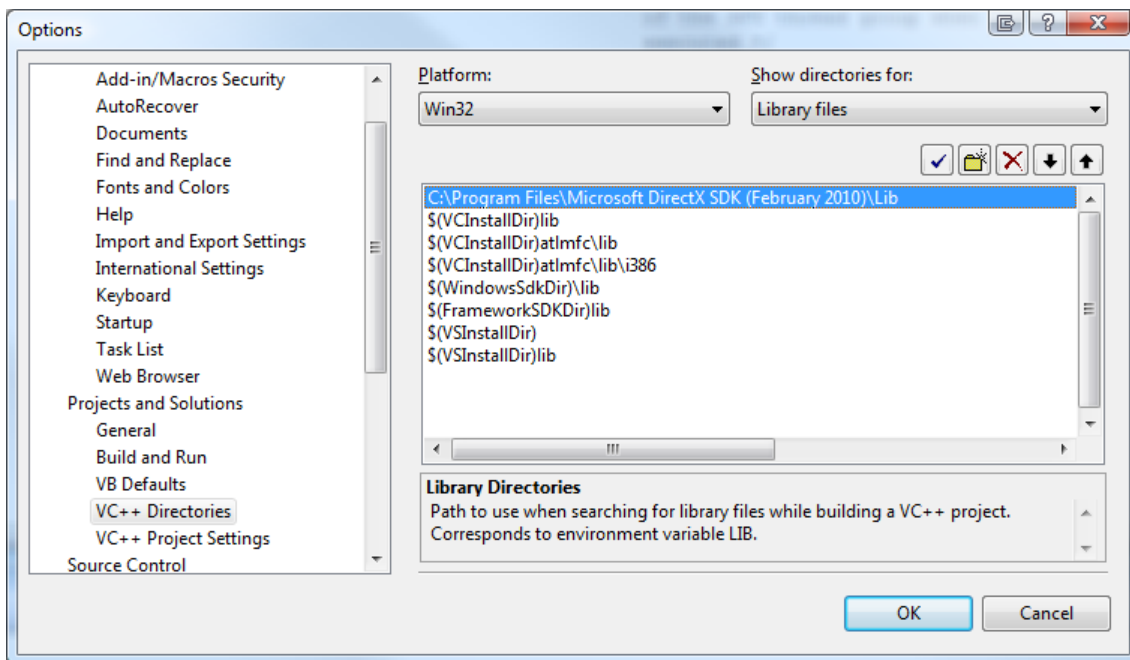


Figure 1.3: Setting the library options for DirectX

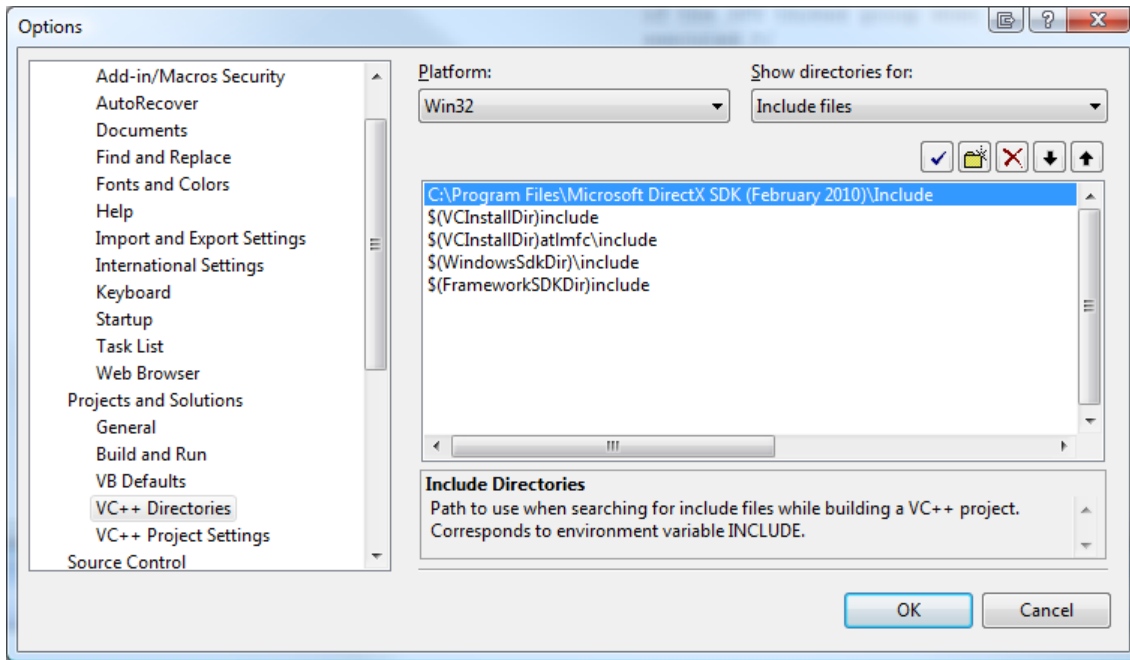


Figure 1.4: Setting the include options for DirectX

Note:

Alternatively, you can run the demos in OpenGL by passing the "-ogl" as a command line parameter to the demo executable. In this situation, you can also strip out the DirectX dependencies from the Demo source code to avoid having to install DirectX. We do however strongly recommend the DirectX route as it is the simplest.

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. For example, in a DOS window type "set HK_LINK_TYPE=debug" to change the current specification to debug.

1.2.4 Demo Command Line Arguments

Command line arguments can be passed to the Havok Demo Framework using the standard command line method or by using a hkdemo.cfg file (see below). The complete list of command line arguments can be displayed by adding "-?" to the command line. A partial list of the most useful options is as follows:

<i>Option</i>	<i>Description</i>	<i>Win32</i>
<code>-c</code>	Use checking memory manager, and report memory leaks. [Debug only]	Yes
<code>-d</code>	Run the visual debugger server. The visual debugger server is disabled by default.	Yes
<code>-f</code>	Run full screen. The demos run in windowed mode by default.	Yes
<code>-g [name of demo]</code>	Automatically run a specific demo, e.g. '-g Pyramid'. The default demo is the 'Menu'	Yes
<code>-l[fps]</code>	Lock the frame rate to the specified frequency, e.g.-l60	Yes
<code>-pi,d,l,lp,f</code>	Disable or enable the various performance stats (i: instruction cache misses, d: dcache misses, l: load-hit-store, lp: load-hit-store penalties, f: instructions flushed). Multiple stats can be enabled at the same time by combining them. The performance counters are disabled by default.	No
<code>-r [none]</code>	Enable, disable or specify the renderer. The renderer is enabled by default. The renderer can be disabled using the option "-r none" Disabling the renderer is useful for collecting performance stats unaffected by display code execution.	Yes, and can specify the preferred renderer to use. See below for more.
<code>-st</code>	Force the demos to run in singlethreaded simulation mode.	Yes
<code>-mt[num threads]</code>	Force the demos to run in multithreaded simulation mode. You can optionally specify the number of threads (-mt4 for 4 threads).	Yes
<code>-lastdemo</code>	If running one of the bootstrap demos, this will start from the last demo run by the bootstrapper.	Yes
<code>-nextdemo</code>	If running one of the bootstrap demos, this will 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 the shader version of DirectX 9. "d3d9s" is the shader-only version of the DirectX 9 renderer.

1.2.5 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

1.3 Step By Step Demos

It can be difficult sometimes to see what 'support' code is needed to get Havok to work within an application. In order to try and demonstrate concretely what is needed to get basic Havok integration, the demos suite now contains 'Step By Step' demos. These demos are designed to be as simple as possible and to show step by step what is needed to add a specific feature to your application. The demos are all console demos (ie they have no graphics component) and contain all the code needed in a single file `main.cpp` file with the entry point 'main'.

The Step By Step demos can be found in the directory `Demo/StandAloneDemos/StepByStep`. Each demo has its own directory - and its own `settings.build` file so that, via the Havok build system, each demo will produce a separate stand alone project. Within Visual Studio the Havok demo workspace contains the demos in a group 'StepByStep' which contains each demo as a separate project.

A description of each of the demos can be found in the `readme.txt` file found in the `Demo/StandAloneDemos/StepByStep` directory.

Here is a breakdown of one of the 'MemoryUtilInit' demo to show the basic structure of what is needed to integrate Havok into your application.

```
#include <Common/Base/hkBase.h>
#include <Common/Base/Memory/System/Util/hkMemoryInitUtil.h>
#include <Common/Base/Memory/Allocator/Malloc/hkMallocAllocator.h>
#include <cstdio>
```

The `hkBase.h` include is needed to access the basic Havok common library features. `hkMemoryInitUtil` provides methods that make it easier to set up memory allocation. `hkMallocAllocator` is an implementation of the `hkMallocAllocator` interface using a platforms standard memory allocator - generally the C standard libraries `malloc`. This is needed because Havoks memory allocator implementations, which are optimized for great Havok performance - still need to allocate their own pools of memory from somewhere.

The final include of '`cstdio`' isn't needed by Havok, but for the demo which needs to access standard C functions such as `printf`.

```
static void HK_CALL errorReport(const char* msg, void* userContext)
{
    using namespace std;
    printf("%s", msg);
}
```

When Havok is started up via `hkBaseSystem::init` it needs a function that can be used to report any errors. Here since the example is just a console application - any errors can just be printed to the terminal via `printf`. Depending on the application and platform something else may be more appropriate such as writing the message to a log file.

```

int HK_CALL main(int argc, const char** argv)
{
    hkMallocAllocator baseMalloc;
    hkMemoryRouter* memoryRouter = hkMemoryInitUtil::initDefault( &baseMalloc, hkMemorySystem::FrameInfo(0) );
    hkBaseSystem::init( memoryRouter, errorReport );
    {
        HK_WARN_ALWAYS(0x417ffd72, "Hello world!");
    }
    hkBaseSystem::quit();
    hkMemoryInitUtil::quit();

    return 0;
}

```

This is the main entry point for the application. To use the `hkMemoryInitUtil` generally you need an allocator that is going to provide the low level memory allocations. `hkMemoryInitUtil::initDefault` will wrap this allocator with other allocators which have good performance characteristics and other features. In this example an instantiation of a `hkMallocAllocator` is used as the underlying allocator.

Once the memory system is set up, Havok as a whole can be started up. The `hkBaseSystem::init` does this, using the `memoryRouter` that was previously created, and the previously described `errorReport` function.

```

#include <Common/Base/keycode.cxx>

```

The `keycode.cxx` file is included and it exposes the Havok key information for the licences to products that you have purchased. These are used internally to verify valid licences - if a licence is not valid for a product, the product will not operate.

```

// we're not using anything product specific yet. We undef these so we don't get the usual
// product initialization for the products.
#undef HK_FEATURE_PRODUCT_AI
#undef HK_FEATURE_PRODUCT_ANIMATION
#undef HK_FEATURE_PRODUCT_CLOTH
#undef HK_FEATURE_PRODUCT_DESTRUCTION
#undef HK_FEATURE_PRODUCT_BEHAVIOR
#undef HK_FEATURE_PRODUCT_PHYSICS

// Also we're not using any serialization/versioning so we don't need any of these.
#define HK_EXCLUDE_FEATURE_SerializeDeprecatedPre700
#define HK_EXCLUDE_FEATURE_RegisterVersionPatches
#define HK_EXCLUDE_FEATURE_RegisterReflectedClasses
#define HK_EXCLUDE_FEATURE_MemoryTracker

// This include generates an initialization function based on the products
// and the excluded features.

#include <Common/Base/Config/hkProductFeatures.cxx>

```

This section turns off products that are not going to be used in this application by undefining the associated `HK_FEATURE_PRODUCT` macro. Next specific features can be excluded using the `HK_EXCLUDE_FEATURE` macros. Finally including `hkProductFeatures.cxx` uses the previously defined/undefined symbols in order to link in the appropriate libraries.

1.4 Building with Havok

1.4.1 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.

1.4.2 Havok Library Link Order

For a full list of Havok libraries and their product dependencies please see `Docs/SourceCodeLevels.txt` in your Havok distribution.

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

1.4.4 Linking with Havok

The Havok SDK has 4 build configurations: *Release*, *Debug*, *FullDebug* and *Hybrid*. 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.

- **Hybrid**

The Hybrid build is a FullDebug build linked against the release C runtime library. This is primarily useful when debugging Havok plugins written for the modeler tools (like 3dsmax or Maya) as the modelers use the release C runtime library.

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:

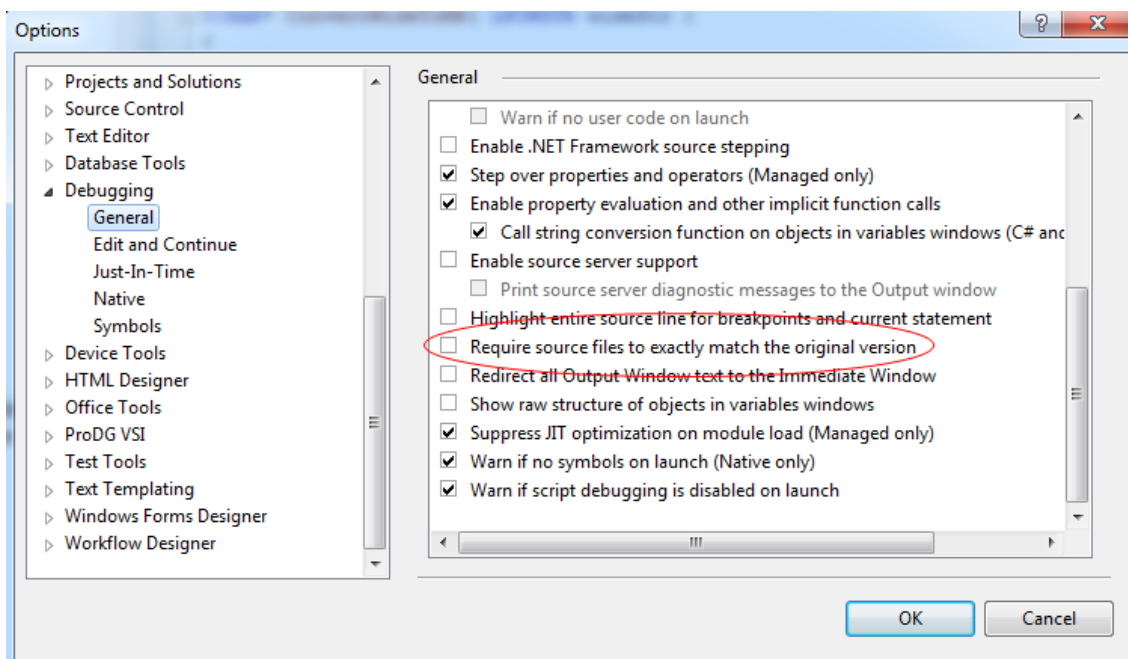
```
// Register Havok classes.
#include <Common/Base/KeyCode.h>
#define HK_CLASSES_FILE <Common/Serialize/Classlist/hkKeyCodeClasses.h>
#include <Common/Serialize/Util/hkBuiltinTypeRegistry.cxx>
```

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.

1.4.5 Debugging Havok Libraries

Depending on your license, Havok provides much of the source code necessary to debug its libraries.

However, when debugging, Microsoft Visual Studio will often prompt you to locate the original source instead of automatically opening the relevant source. This is because the source files used to compile the Havok libraries are not binary-identical to those distributed to customers. Dates, build numbers, and legal boiler plates are added to distributed source, though the line numbers still match. To avoid the need to navigate to each source file in Microsoft Visual Studio, go to Tools > Options... > Debugging > General and uncheck the box "Require source files to exactly match the original version":



1.4.6 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 (hkBuiltinTypeRegistry.cxx), and implementations which are selected at compile time (hkStackTracerWin32.cxx).

1.5 Setting up the Visual Debugger (VDB)

1.5.1 Windows/Linux VDB setup

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 is 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' in the Havok Physics user guide for more information.

1.5.2 Windows VDB Caveats

There are no current special issues for this platform.