



©Copyright 1999-2014 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	Introduction	4
1.2	Getting Started with Havok	4
1.2.1	Getting started with the binary-only PC version of Havok Physics 2012 and Havok Animation	4
1.2.2	Using Havok on multiple platforms	5
1.2.3	Simple Applications	5
1.2.4	Building the Demos	7
1.2.5	Demo Command Line Arguments	10
1.2.6	Demo Controls	11
1.3	StepByStep Demos	13
1.4	Building with Havok	15
1.4.1	Havok Libraries	15
1.4.2	Havok Library Link Order	17
1.4.3	Havok Header Include Policy	17
1.4.4	Integrating with Havok	18
1.4.5	Debugging Havok Libraries	19
1.4.6	File Naming Convention	20
1.5	Setting up the Visual Debugger (VDB)	20
1.5.1	Windows/Linux VDB setup	20
1.6	Evaluation Checklist	21
1.6.1	Havok Common	21
1.6.2	Havok Physics 2012	21
1.6.3	Havok Animation	22

Chapter 1

Havok PC Guide

1.1 Introduction

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.

1.2 Getting Started with Havok

1.2.1 Getting started with the binary-only PC version of Havok Physics 2012 and Havok Animation

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

- `hk[XXX]`
 - `Build`
 - `Demos`
 - `Docs`
 - `Lib`
 - `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.

1.2.1.1 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 **ShowCase** 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 **ShowCase** 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 2012** and **Havok Animation** offer.

Note:

To get the full visual and audio quality of the **ShowCase** demos, ensure that you have Microsoft DirectX installed (June 2010 or later).

1.2.2 Using Havok on multiple platforms

One of the strengths of Havok's SDK products is that they are available for a wide variety of hardware platforms. For each supported platform there may be multiple ways to build the Havok SDK. For example, for Windows, it's possible to build for Win32 using Visual Studio 2010 or Visual Studio 2012, and for other hardware it may be possible to build using makefiles or Visual Studio. Each different way to build for a given platform is referred to as a 'target'. Targets can usually be described using a single string that contains the information needed to distinguish it from other targets e.g. `win32_vs2010` and `win32_vs2012_win7`.

1.2.2.1 Windows

Havok on Windows is currently built using Microsoft Visual Studio. There are six targets available, based on different Windows API levels, different Windows OS versions, and different Visual Studio versions:

- For Win32:
 - `win32_vs2010` for Visual Studio 2010
 - `win32_vs2012_win7` for Visual Studio 2012 on Windows 7
 - `win32_vs2012_win8` for Visual Studio 2012 on Windows 8
- For Win64:
 - `x64_vs2010` for Visual Studio 2010
 - `x64_vs2012_win7` for Visual Studio 2012 on Windows 7
 - `x64_vs2012_win8` for Visual Studio 2012 on Windows 8

1.2.3 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 `Demo/StandAloneDemos` directory) are sample applications demonstrating the minimum code necessary to get Havok up and running.

To build these demos:

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:

For Windows, Linux and OS X the IP address to connect to is the IP of the machine running the application. The client can be run from any PC on the network, including the local PC running the application.

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.

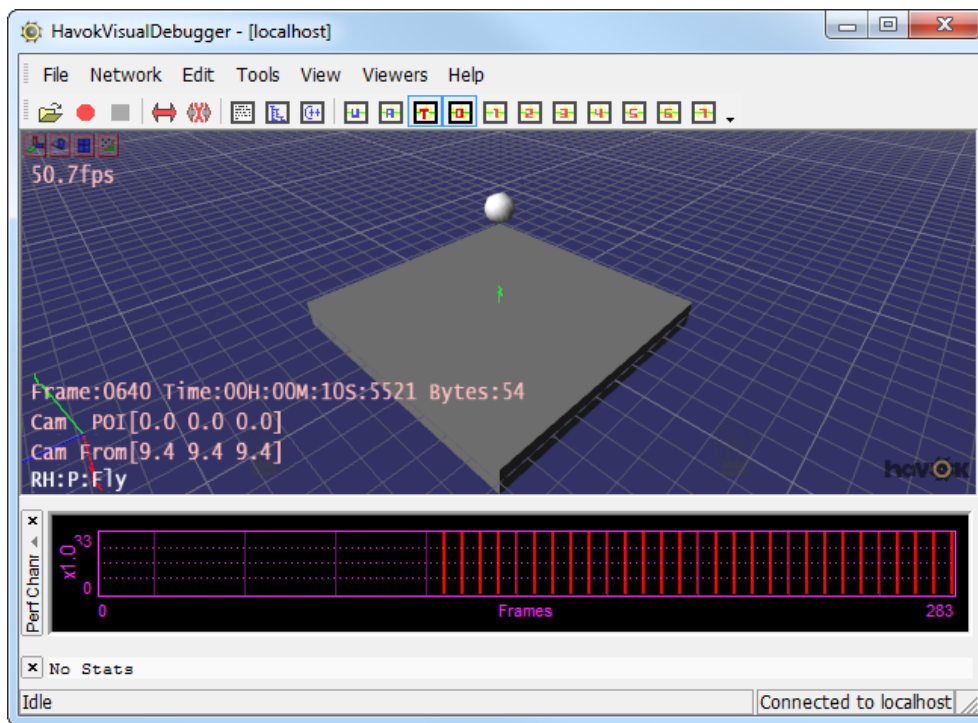


Figure 1.2: Display of PhysicsExample on the Visual Debugger

1.2.4 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 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.

1.2.4.1 Building for Windows

- 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 **Source/Common/Base/KeyCode.h**.
- Select **Release** and **Win32** or **x64** as the active configuration.
- Build and run.

The **Demos_x64-vs2012_win7_release.exe** executable will be created in **Demo/Demos** (with "x64-vs2012_win7" corresponding to Win64 and the Windows 7 version of Visual Studio 2012 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 TrollDemo** and the **Destruction DestructableBridgeDemo**).

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 **Project > Properties > VC++ Directories** . There you need to add the paths to your DirectX SDK install, e.g. "C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Include" under **Include files** , and "C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Lib\x64" under **Library files** (default DirectX install paths).

Below are some screenshots of the relevant options for Visual Studio 2010 on Windows 7:

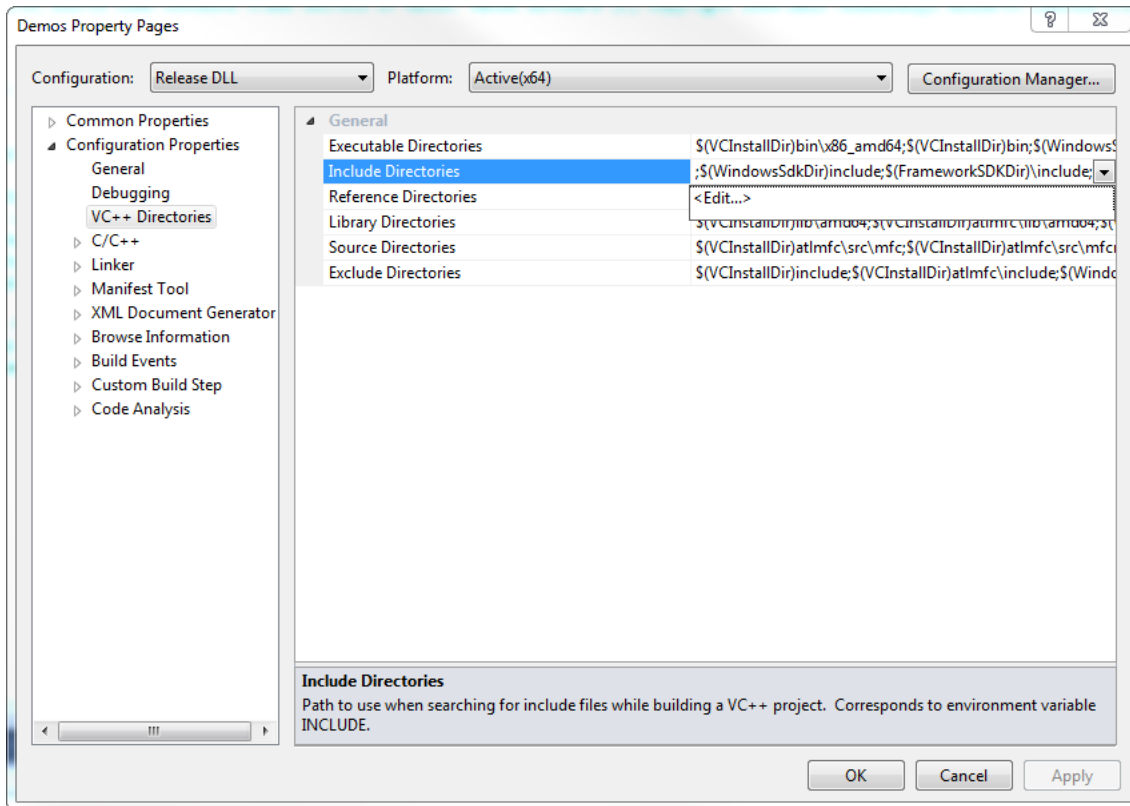


Figure 1.3: Setting the directory options for DirectX

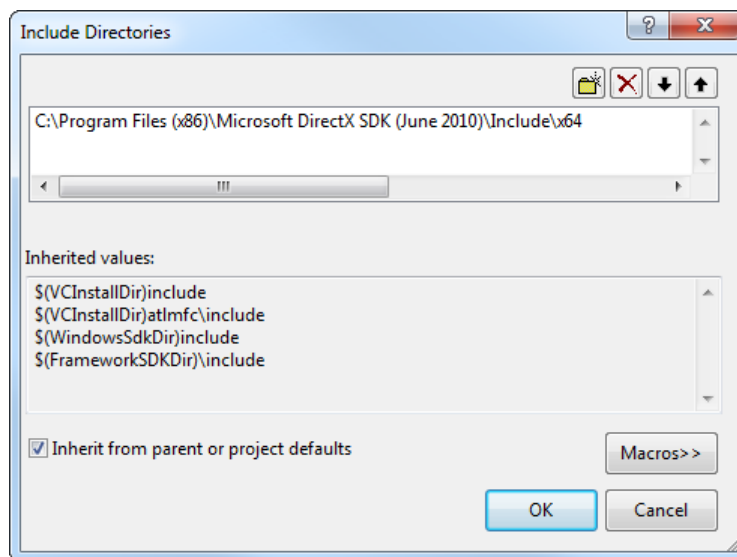


Figure 1.4: Setting the include options for DirectX; library options are set similarly

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.

1.2.5 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>Windows</i>
-c	Use checking 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 <i>[name of demo]</i>	Automatically run a specific demo, e.g. '-g Pyramid'. The default demo is the 'Menu'.	Yes
-l <i>[fps]</i>	Lock the frame rate to the specified frequency, e.g. '-l60'.	Yes
-pi,d,l,lp,f	Disable or enable the various performance stats (<i>i</i> : instruction cache misses; <i>d</i> : dcache misses; <i>l</i> : load-hit-store; <i>lp</i> : load-hit-store penalties; <i>f</i> : instructions flushed). Multiple stats can be enabled at the same time by combining them. The performance counters are disabled by default.	No
-r <i>[none]</i>	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.
-st	Force the demos to run in single-threaded simulation mode.	Yes
-mt <i>[num threads]</i>	Force the demos to run in multithreaded simulation mode. You can optionally specify the number of threads (-mt4 for 4 threads).	Yes
-lastdemo	If running one of the bootstrap demos, this will start from the last demo run by the bootstrapper.	Yes
-nextdemo	If running one of the bootstrap demos, this will start from the demo after the last demo run by the bootstrapper.	Yes
-novfs	Disable virtual file server.	Not relevant

1.2.5.1 Using the hkdemo.cfg File

The `hkdemo.cfg` is of most use for platforms that do not facilitate command line arguments or when 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
```

Note:

1.2.5.2 Windows 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 and Win64 you may choose any of the following DirectX variants: "d3d8", "d3d9", "d3d9s", or "ogl" for OpenGL mode. By default Win32 and Win64 use the shader version of DirectX 9. "d3d9s" is the shader-only version of the DirectX 9 renderer.

1.2.6 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.

1.2.6.1 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.	Insert
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 StepByStep 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 ‘StepByStep’ 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 (i.e. they have no graphics component) and contain all the code needed in a single file `main.cpp` file with the entry point ‘`-main`’.

The StepByStep demos can be found in the directory `Demo/StandAloneDemos/StepByStep`. Each demo has its own directory, containing a project file or makefile for each supported platform and the `main.cpp` file containing the demo code. These demos are compiled in the same way as the main Havok Demos application.

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 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 <Common/Base/Fwd/hkcstdio.h>
```

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 platform’s standard memory allocator, generally the C standard library’s `malloc`. This is needed because Havok’s 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.

```

#if defined(HK_PLATFORM_WINRT) || defined(HK_PLATFORM_DURANGO)
[Platform::MTAThread]
int main(Platform::Array<Platform::String^>^ args)
#elif defined( HK_PLATFORM_TIZEN )
int hkMain(int argc, const char** argv)
#else
int HK_CALL main(int argc, const char** argv)
#endif
{
    // Perform platform specific initialization for this demo - you should already have something similar
    // in your own code.
    PlatformInit();

    hkMemoryRouter* memoryRouter = hkMemoryInitUtil::initDefault( hkMallocAllocator::
        m_defaultMallocAllocator, 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_2012
#undef HK_FEATURE_PRODUCT_DESTRUCTION
#undef HK_FEATURE_PRODUCT_BEHAVIOR
#undef HK_FEATURE_PRODUCT_PHYSICS_2012
#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. The tables below 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, as well as math core and other utilities. This is always required. You can rebuild or replace parts of this library to suit your specific configuration.	System libraries (e.g. libc, libm)		
hkCompat	Version compatibility library. Allows loading of assets from previous Havok versions.	hkBase , hkSerialize		
hkGeometryUtilities	Contains utilities to manipulate, transform and convert Havok meshes and hkGeometry structures.	hkBase , hkSceneData		
hkInternal	A public interface to Havok's internal common classes and data structures.	hkBase		
hkSerialize	The Havok import/export library. You can import/export Havok data into your own format and serialize custom created objects.	hkBase		
hkSceneData	This library contains the data descriptions that are extracted from the animation toolchain.	hkBase		
hkVisualize	This module provides the interface and visualization helper classes for the debug viewers and the Visual Debugger.	hkBase , hkSerialize		
hkaiPathfinding	All	The central library for AI, including streaming, dynamic cutting, and character control	hkaiInternal	Product
hkaiInternal	All	Contains nav mesh and nav volume generation functions, as well as utility methods for other AI libraries	hkaiPathfinding	Internal
hkaiSpatialAnalysis	All	Contains high-level traversal analysis, as well as utility functions to access data produced by traversal analysis	hkaiPathfinding , hkaiInternal	Base
hkaiVisualize	All	Provides debug visualization for other AI libraries, including AI-specific VDB viewers	hkaiPathfinding	Base
hkaiPhysicsBridge	All	Provides integration between AI simulation and Physics simulation	hkaiPathfinding	Base
hkaiPhysics2012Bridge	All	Provides integration between AI simulation and Physics 2012 simulation	hkaiPathfinding , hkpDynamics , hkpCollide	Base
hkaiSpatialAnalysis	All	Contains high-level traversal analysis, as well as utility functions to access data produced by traversal analysis	hkaiPathfinding , hkaiInternal	Base
hkcdCollide	This library contains core collision detection functions, used across multiple products.	hkBase		
hkcdInternal	This library contains internal core collision detection functions, used across multiple products.	hkBase		
hkgpConvexDecomposition	This library contains the Havok convex decomposition tools.	hkBase		
hkpDynamics	Core dynamics library. This links the constraint solver library with the collision detection library. This is always required.	hkpCollide , hkpConstraintSolver , hkBase		
hkpCollide	This module contains all the collision detection functionality used by Havok.	hkBase		
hkpConstraintSolver	The Havok constraint solver. This is always required.	hkBase		
hkpInternal	A public interface to Havok's internal collision and dynamics classes and data structures.	hkpDynamics , hkpCollide , hkBase		

<i>Library</i>	<i>Description</i>	<i>Dependencies</i>	
hkgCommon	Contains a cross-platform core graphics engine.	hkBase	
hkgBridge	Provides the bridge between physics and graphics.	hkgCommon, hkpUtilities, hkpDynamics, hkpCollide, hkpConstraintSolver, hkBase	
hkgOgl	OpenGL implementation of hkGraphics.	hkgCommon, hkBase	
hkgOglS	OpenGL implementation with shader support of hkGraphics (PSGL on PlayStation®3).	hkgCommon, hkBase	
hkgDx9	DirectX 9 implementation of hkGraphics.	hkgCommon, hkBase	
hkgDx9s	DirectX 9s implementation of hkGraphics.	hkgCommon, hkBase	
hkgDx10	DirectX 10 implementation of hkGraphics.	hkgCommon, hkBase	
hkgDx11	DirectX 11 implementation of hkGraphics	hkgCommon, hkBase	Base
hksCommon	This contains the interface to the Havok sound system.	hkBase	
hksXAudio2	DirectX implementation of hksCommon.	hkBase	

Table 1.2: Havok Demo Libraries

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
```

Similarly:

```
// Include the hkpDynamics header before any other hkpDynamics includes.
#include <Physics2012/Dynamics/hkpDynamics.h>
#include <Physics2012/Dynamics/World/hkpWorld.h>
// more hkpDynamics headers
```

Note that many of the library headers include other libraries' headers. For instance `hkpDynamics.h` includes `hkBase.h`. So for instance if a file included both the above blocks, it could be simplified to:

```
#include <Physics2012/Dynamics/hkpDynamics.h> // hkBase.h included
#include <Common/Base/System/IO/IStream/hkIstream.h>
#include <Physics2012/Dynamics/World/hkpWorld.h>
```

1.4.4 Integrating with Havok

There are a few things which are required when integrating Havok code and libraries with your own code.

1.4.4.1 Havok Build Configurations

The Havok SDK has three build configurations: **Release**, **Dev** and **Debug**. 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 **Dev** or **Debug** configuration of Havok. If you need to link the **Release** configuration of Havok while your application is built in **Dev**, you must ensure that `HK_DISABLE_DEBUG` is defined in your build. Symbols are included in **Release** builds on platforms where there is no associated performance overhead. The **Release** build is linked against the dynamic release C runtime library.

- **Dev**

The **Dev** 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 callstack. Single stepping through Havok code may be error-prone with **Dev** libraries, as the compiler has fully optimized the code. It is recommended that developers working on systems related to Havok products link with Havok **Dev** 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. The **Dev** build is linked against the dynamic release C runtime library.

- **Debug**

The **Debug** 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 **Debug** libraries on a day-to-day basis. The **Debug** build is linked against the dynamic debug C runtime library.

- **Hybrid**

The **Hybrid** build configuration is the same as the **Debug** build configuration, but linked against the dynamic release C runtime library. This makes it possible to debug Havok code in contexts where the debug C runtime library cannot be used. In particular, **Hybrid** should be used instead of **Debug** when building HavokAssembly or the Havok Content Tools.

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 built 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 **Dev** (development) and **Debug** 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.

1.4.4.2 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:

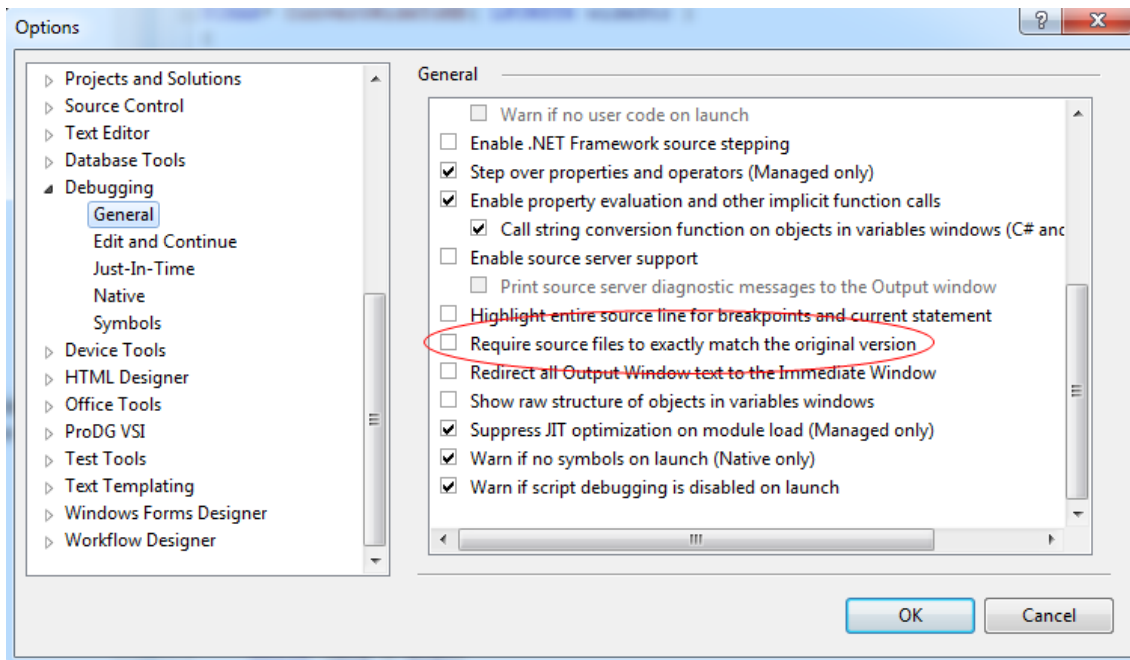
```
// Register Havok classes.
#include <Common/Base/KeyCode.h>
#define HK_CLASSES_FILE <Common/Serialize/Classlist/hkKeyCodeClasses.h>
#include <Common/Serialize/Util/hkBuiltinTypeRegistry.cxx>
```

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 boilerplates 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 the following extensions:

.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 Common manual for more information.

1.6 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 for each product you have licensed, because later items generally build on earlier items.

The **StepByStep** 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 **StepByStep** demo section below.

1.6.1 Havok Common

All of the Havok products rely on the fundamental infrastructure of the Common library. One of its 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 math support.

- Look at and compile the **MemoryUtilInit StepByStep** demo in **Demo/StandAloneDemos/StepByStep**.
- Route Havok memory (de)allocations through your memory system. See SDK documentation on Havok Base Library and Memory Customization in the Havok Common manual.
- 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 in the Havok Common manual.
- Look at and compile the **Serialize StepByStep** demo.
- Write a demo to save and load an object using the serialization system.

1.6.2 Havok Physics 2012

- Build and run **Havok Physics 2012 Demos** and Console Examples.
- Look at and compile the **Physics 2012** and **Physics2012Vdb StepByStep** demos.
- Create a simple (e.g. box-shaped) fixed/static rigid body in the game runtime.
- **Attach the Havok Visual Debugger** (VDB) to view the simple fixed object. See the Havok Common manual for information on the VDB.
- Create simple dynamic rigid bodies in-game, that collide with each other and the simple fixed object.
- Add VDB user cameras that correspond to game cameras (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** in the Havok Common manual.

- Synchronize dynamic objects with game renderer. Confirm consistency using game and VDB. Experiment with VDB viewers to see shapes, contact points, simulation islands, broad phase AABBs, inertia tensors, statistics, etc.
- Create more complicated fixed physical geometry, e.g. polygon soup or height field.

And optionally:

- Install Havok Content Tools (HCT) for your modeler(s) of choice. See the Havok Content Tools manual for information on the HCT.

1.6.3 Havok Animation

- See all of "**Havok Common**" above.
- Use HCT to create skeleton, animation and (optionally) skin. Preview all elements in Havok Preview Tool. See the Havok Content Tools manual for information about the 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.