



©Copyright 1999-2017 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	4
1.1	Introduction	5
1.2	Getting Started with Havok	6
1.2.1	Using Havok on multiple platforms	7
1.2.1.1	Windows	7
1.2.2	Simple Applications	8
1.2.2.1	Physics Example	8
1.2.2.2	Connecting the Visual Debugger	9
1.2.3	Building the Demos	10
1.2.3.1	Building for Windows	10
1.2.3.2	OpenGL Projects	12
1.2.4	Demo Command Line Arguments	13
1.2.4.1	Using the hkdemo.cfg File	13
1.2.4.2	Windows Renderer Options	14
1.2.5	Demo Controls	15
1.2.5.1	PC Demo Controls	16
1.3	StepByStep Demos	17
1.4	Building with Havok	20
1.4.1	Havok Libraries	21
1.4.1.1	Havok Library Link Order	22
1.4.2	Havok Header Include Policy	23
1.4.3	Rebuilding Havok Library Files	24
1.4.3.1	Building for Windows (32/64-bit)	24
1.4.4	Integrating with Havok	25
1.4.4.1	Havok Build Configurations	25
1.4.4.2	The Havok SDK runtime and the Visual C Runtime (CRT)	26
1.4.5	Debugging Havok Libraries	27
1.4.5.1	Finding Source Code	27
1.4.5.2	Visualizers	27
1.4.6	File Naming Convention	28
1.5	Havok Support	29
1.6	Setting up the Visual Debugger (VDB)	30
1.6.1	Windows/Linux VDB setup	31
1.7	Evaluation Checklist	32
1.7.1	Havok Common	33
1.7.2	Havok Physics 2012	34
1.7.3	Havok Animation	35
1.7.4	Havok Animation Studio	36
1.7.5	Havok Cloth	37
1.7.6	Havok Destruction 2012	38
1.8	Optimizations	39
1.8.1	Win32 SIMD	40

1.8.1.1	What happens if the Win32 SIMD setting in hkConfigSimd.h somehow ends up set to the incorrect value?	40
1.9	PC Performance Tests	42
1.9.1	Animation Timings	43
1.9.1.1	Sampling	43
1.9.1.2	Sampling and Blending	43
1.9.2	Behavior Timings	44
1.9.2.1	The Behavior Demos	44
1.9.2.2	Generate	44
1.9.2.3	Update	45

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 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 2012 or Visual Studio 2015, 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_vs2012` and `win32_vs2015`.

1.2.1.1 Windows

Havok on Windows is currently built using Microsoft Visual Studio. There are several targets available, depending on the Visual Studio version used.

- For Win32:
 - `win32_vs2012` for Visual Studio 2012
 - `win32_vs2013` for Visual Studio 2013
 - `win32_vs2015` for Visual Studio 2015
 - `win32_vs2017` for Visual Studio 2017
- For Win64:
 - `x64_vs2012` for Visual Studio 2012
 - `x64_vs2013` for Visual Studio 2013
 - `x64_vs2015` for Visual Studio 2015
 - `x64_vs2017` for Visual Studio 2017

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

To build these demos:

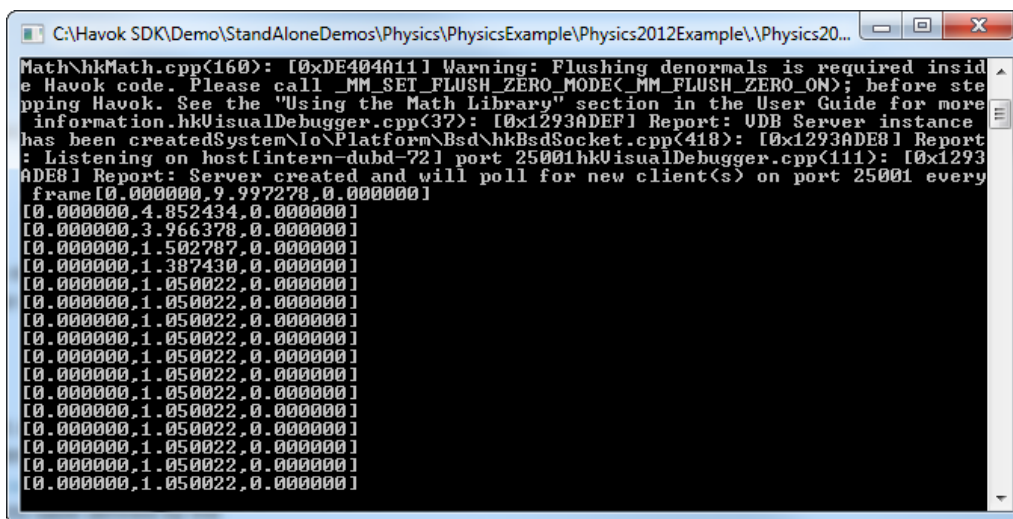
- Ensure that your Havok keycode(s) are present in Source/Common/Base/KeyCode.h.
- Choose a project or makefile for the platform and compiler you wish to build and execute on.
- Build and run the demo. For specific platform configuration see [below](#).

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

1.2.2.1 Physics Example

This demo is found in Demo/StandAloneDemos/Physics/PhysicsExample. Either Havok Physics or Havok Physics 2012 is required for this demo, though other standalone demos may be run in a similar manner. The purpose of this section is merely to show how to set up and run any of the StandAloneDemos. 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 Physics Example 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:\Havok SDK\Demo\StandAloneDemos\Physics\PhysicsExample\Physics2012Example\Physics20...
Math\hkMath.cpp(160): [0xDE404011] Warning: Flushing denormals is required insid
e Havok code. Please call _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON); before ste
pping Havok. See the "Using the Math Library" section in the User Guide for more
information.hkVisualDebugger.cpp(37): [0x1293ADE8] Report: UDB Server instance
has been createdSystem\Io\Platform\Bsd\hkBsdSocket.cpp(418): [0x1293ADE8] Report
: Listening on host[intern-dubd-72] port 25001hkVisualDebugger.cpp(111): [0x1293
ADE8] Report: Server created and will poll for new client(s) on port 25001 every
frame[0.000000, 9.997278, 0.000000]
[0.000000, 4.852434, 0.000000]
[0.000000, 3.966378, 0.000000]
[0.000000, 1.502787, 0.000000]
[0.000000, 1.387430, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
[0.000000, 1.050022, 0.000000]
```

Figure 1.1: Display of output of PhysicsExample

1.2.2.2 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\VisualDebugger\HavokVisualDebugger.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 on setting up the visual debugger](#).

In this application, a visual debugger server is created and can be connected to using `HavokVisualDebugger.exe` (run it and look at the Connect panel). If supported on the platform and enabled by the server, the IP address to connect to can be determined using the Server Discovery option in the Havok Visual Debugger or you may explicitly enter the IP address 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.

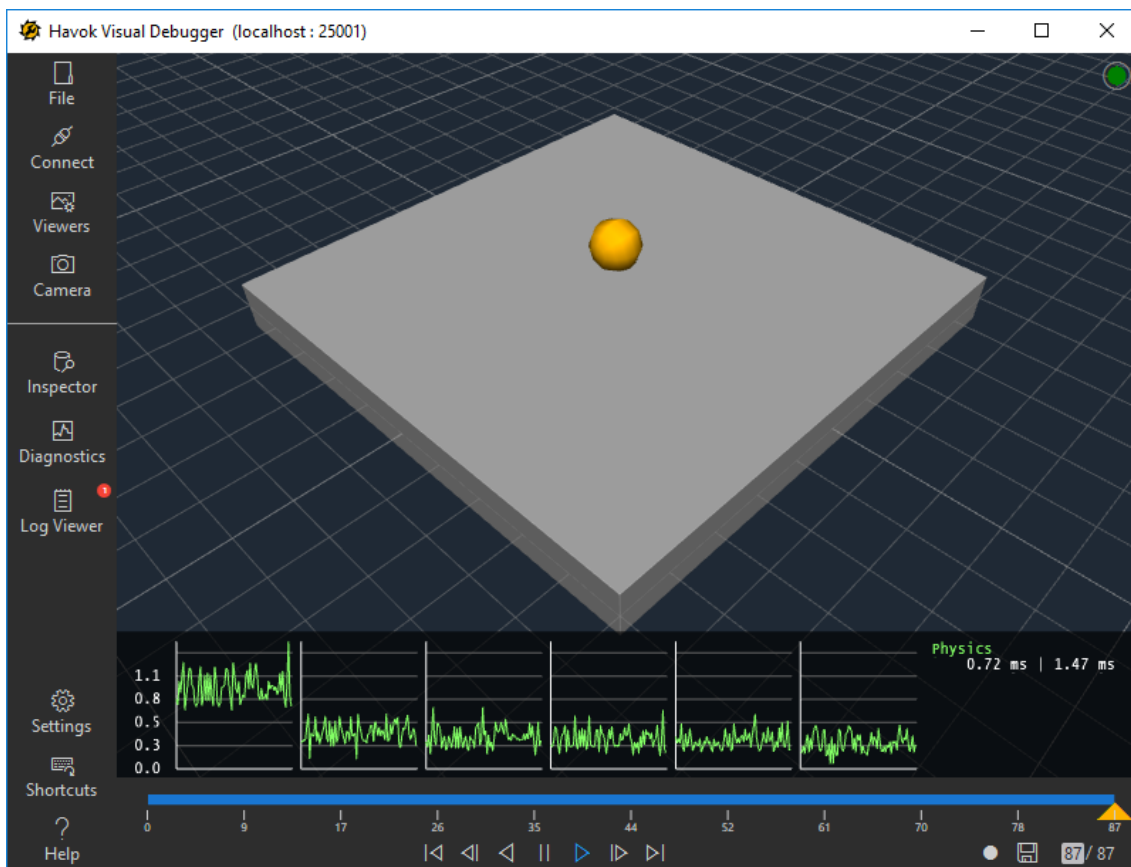


Figure 1.2: Display of PhysicsExample on the Visual Debugger

1.2.3 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 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.

1.2.3.1 Building for Windows

These steps are for building the SDK Demos on Windows 8 or later. If you want to build more than that, or are using an earlier version of Windows, please refer to the next section, [Windows and DirectX](#), for some additional DirectX-related requirements.

- Open the Demos_... 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_release.exe executable will be created in Demo/Demos (with "x64-vs2012" corresponding to Win64 and Visual Studio 2012 in this example).

Windows and DirectX

If you're on Windows 8 or later and only wish to build the Demos project then the above steps should be all you need. You **do not** need to install the Microsoft DirectX Runtime or DirectX SDK (and you can safely skip this section). If however, you're using Windows 7 or earlier, or wish to rebuild your entire Havok distribution, then some DirectX components may be required.

- If you're on Windows 7 or earlier then before compiling the SDK demos you'll need to install the DirectX June 2010 Runtime and use the special Demos_LegacyDX_... project instead of the standard version. While it is possible to build the standard version (depending on your DirectX setup) you'll see an error like *Pc\hkgSystemDX11PC.cpp(148): [0x0001E32D] Warning: Could not find proper DirectX11 DLLs, need at least version 47 DemoCommon\DemoFramework\hkDemoFrameworkRender.cpp(118): [0x3017483A] Assert: in 'initRendererAndEnv', condition: 'hkgWindow::create'. Renderer has no create(). Incorrect name specified?* when you try to run it. (Note that the Demos_LegacyDX_... project is excluded by default in the provided Visual Studio solution files, so in cases where you're building from a solution file rather than from the project file directly it will need to be enabled in the Configuration Manager instead of the default Demos project.)
- If you wish to rebuild your entire Havok distribution then you'll also need to install the DirectX SDK to successfully build the "hkg" DirectX graphics libraries. This is necessary regardless of your Windows version; also, if you're on Windows 7 or earlier you'll need to enable the "LegacyDX" variants of the Demos and hkg libraries in the Visual Studio configuration manager (found in the Demo/LegacyDX solution folder), and disable their default equivalents, in the same way as described above.

Once you have installed the DirectX SDK, you should change your IDE options to reflect this. 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).

Demos Property Pages

Configuration: Release DLL Platform: Active(x64) Configuration Manager...

- Common Properties
- Configuration Properties
 - General
 - Debugging
 - VC++ Directories
- C/C++
- Linker
- Manifest Tool
- XML Document Generator
- Browse Information
- Build Events
- Custom Build Step
- Code Analysis

General

Executable Directories	\$(VCInstallDir)bin\x86_amd64;\$(VCInstallDir)bin;\$(Windows...
Include Directories	\$(WindowsSdkDir)include;\$(FrameworkSDKDir)\include;
Reference Directories	<Edit...>
Library Directories	\$(VCInstallDir)lib\x86_amd64;\$(VCInstallDir)atlmfc\lib\x86_amd64;\$(...
Source Directories	\$(VCInstallDir)atlmfc\src\mf;\$(VCInstallDir)atlmfc\src\mf...
Exclude Directories	\$(VCInstallDir)include;\$(VCInstallDir)atlmfc\include;\$(Windo...

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

OK Cancel Apply

Confidential Information of Havok. ©Copyright 1999-2017 Havok. All Rights Reserved.

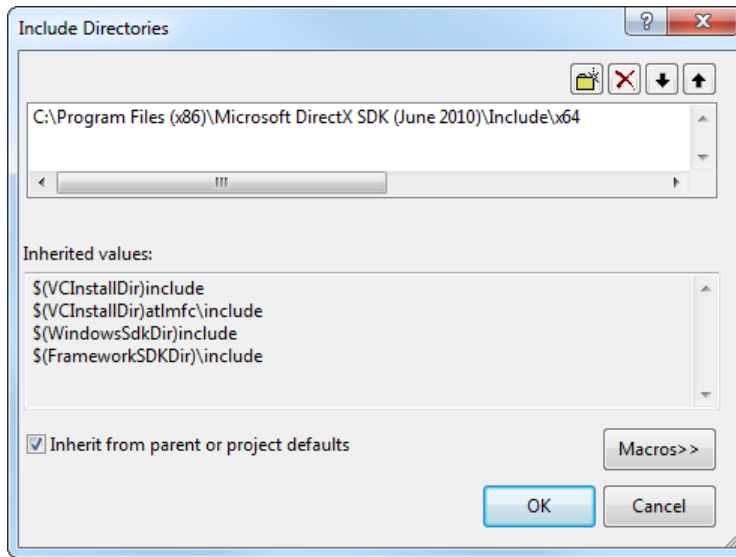


Figure 1.4: Setting the include options for DirectX; library options are set similarly

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.

1.2.3.2 OpenGL Projects

In order to build OpenGL graphics projects, hkgOgl, hkgOgls, hkgOglES or hkgOglES2 you may need to download certain opengl files. If they are not present you will see missing file errors for GL/gltext.h and GL/wgltext.h. The two files can be found at the links below. They should be placed in a folder called GL and after downloading you will need to add the path to the additional include directories of the projects in order to compile them.

- <https://www.khronos.org/registry/OpenGL/api/GL/gltext.h>
- <https://www.khronos.org/registry/OpenGL/api/GL/wgltext.h>

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:

Option	Description	Windows
-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. <code>'-g Pyramid'</code> . The default demo is the 'Menu'.	Yes
-l <i>[fps]</i>	Lock the frame rate to the specified frequency, e.g. <code>'-l60'</code> .	Yes
-p <i>[i,d,l,lp,f]</i>	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 <code>"-r none"</code> . Disabling the renderer is useful for collecting performance stats unaffected by display code execution.	Yes, and can specify the preferred renderer to use. See Windows Renderer Options 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 (<code>-mt4</code> 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

1.2.4.1 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
```

1.2.4.2 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.

1.2.5 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.

1.2.5.1 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/System/Error/hkErrorReport.cxx>
```

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.

```

int hkMain(int argc, const char** argv)
{
    // 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 );
    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
#undef HK_FEATURE_PRODUCT_FX
#undef HK_FEATURE_PRODUCT_NEW_TOOLS

// Also we're not using any serialization/versioning so we don't need any of these.
#define HK_EXCLUDE_FEATURE_RegisterReflectedClasses
#include <Common/Base/Config/hkProductFeaturesNoPatchesOrCompat.h>

// 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>	<i>Source Level</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)	Base
hkCompat	Version compatibility library. Allows loading of assets from previous Havok versions.	hkBase	Base
hkGeometryUtilities	Contains utilities to manipulate, transform and convert Havok meshes and hkGeometry structures.	hkBase, hkSceneData	Base
hkInternal	A public interface to Havok's internal common classes and data structures.	hkBase	Internal
hkSceneData	This library contains the data descriptions that are extracted from the animation toolchain.	hkBase	Base
hkVisualize	This module provides the interface and visualization helper classes for the debug viewers and the Visual Debugger.	hkBase	Base
hkaiPathfinding	The central library for AI, including streaming, dynamic cutting, and character control	hkaiInternal	Product
hkaiInternal	Contains nav mesh and nav volume generation functions, as well as utility methods for other AI libraries	hkaiPathfinding	Internal
hkaiSpatialAnalysis	Contains high-level traversal analysis, as well as utility functions to access data produced by traversal analysis	hkaiPathfinding, hkaiInternal	Base
hkaiVisualize	Provides debug visualization for other AI libraries, including AI-specific VDB viewers	hkaiPathfinding	Base
hkaiPhysicsBridge	Provides integration between AI simulation and Physics simulation	hkaiPathfinding	Base
hkaiPhysics2012Bridge	Provides integration between AI simulation and Physics 2012 simulation	hkaiPathfinding, hkpDynamics, hkpCollide	Base
hkaiSpatialAnalysis	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	Product
hkcdInternal	This library contains internal core collision detection functions, used across multiple products.	hkBase	Internal
hkgpConvexDecomposition	This library contains the Havok convex decomposition tools.	hkBase	Internal
hkpDynamics	Core dynamics library. This links the constraint solver library with the collision detection library. This is always required.	hkpCollide, hkpConstraintSolver, hkBase	Product
hkpCollide	This module contains all the collision detection functionality used by Havok.	hkBase	Product
hkpConstraintSolver	The Havok constraint solver. This is always required.	hkBase	Internal
hkpInternal	A public interface to Havok's internal collision and dynamics classes and data structures.	hkpDynamics, hkpCollide, hkBase	Internal
hkpVehicle	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.	hkpDynamics, hkpCollide, hkpConstraintSolver, hkBase	Base
hkpUtilities	Contains a number of handy utilities, including example actions and the Visual Debugger.	hkpDynamics, hkpInternal, hkpCollide, hkpConstraintSolver, hkBase	Base
hkaInternal	Utility methods for data compression (including spline, quantized, predictive) and general quantization methods.	hkBase	Internal
hkaAnimation	The central library for animation, including storage and playback, blending, motion extraction and skinning.	hkaInternal, hkBase	Product
hkaRagdoll	Contains classes to setup and manipulate ragdolls. Havok Physics 2012 and Havok Animation required.	hkBase, hkaAnimation, hkpDynamics	Product

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

1.4.1.1 Havok Library Link Order

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

1.4.2 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 hkbases 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>
```

1.4.3 Rebuilding Havok Library Files

Rebuilding the Havok libraries works in much the same way as building the demos, as mentioned in the ['Building the Demos'](#) section.

1.4.3.1 Building for Windows (32/64-bit)

- Open the Visual Studio project file corresponding to the library that you want to rebuild in Microsoft Visual Studio.
- Select your preferred build configuration, e.g. **Win32** and **Release**.
- Recompile the library.

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.

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.

Please note that you should not link against Havok libraries that use different build configurations. E.g. you should not link your application against the Dev base library and the Release Physics library. Instead only use one build configuration type for all linked libraries.

1.4.4.2 The Havok SDK runtime and the Visual C Runtime (CRT)

As mentioned above Havok's build configurations use the following CRT libraries:

Build Configuration	CRT version used
Release	Multi-threaded DLL (/MD)
Dev	Multi-threaded Debug DLL (/MDd)
Debug	Multi-threaded Debug DLL (/MDd)

However, in some cases it may be preferable to use the static CRT, or to use the debug CRT with Havok's dev libraries, and so on. To facilitate this all CRT operations performed by the Havok runtime have been consolidated inside the `hkBase` library - the other runtime libraries don't use the CRT at all, and are configured to link with any CRT version. So to work with a different CRT all that's needed is a different version of `hkBase` to link against. These versions are provided pre-built as distinct `hkBase` 'variants', as follows:

hkBase variant	CRT version used
<code>hkBase.lib</code> (default)	Varies per build configuration - as above
<code>hkBaseMD.lib</code>	Multi-threaded DLL (/MD)
<code>hkBaseMDd.lib</code>	Multi-threaded Debug DLL (/MDd)
<code>hkBaseMT.lib</code>	Multi-threaded (/MT)
<code>hkBaseMTd.lib</code>	Multi-threaded Debug (/MTd)

For example:

- **To use the debug CRT and the Dev Havok libraries:** Set your library dir to `[Havok]/Lib/win32_vs2015/dev` (for x86 VS 2015) and link against `hkBaseMDd.lib` instead of `hkBase.lib`.

Note that when linking against a "debug" CRT it's often necessary to add the `_DEBUG` preprocessor define, and conversely if you see link errors when linking against a non-debug CRT it may be that `_DEBUG` is defined when it shouldn't be.

- **To use the static CRT:** Link against `hkBaseMT.lib` or `hkBaseMTd.lib`, as appropriate.

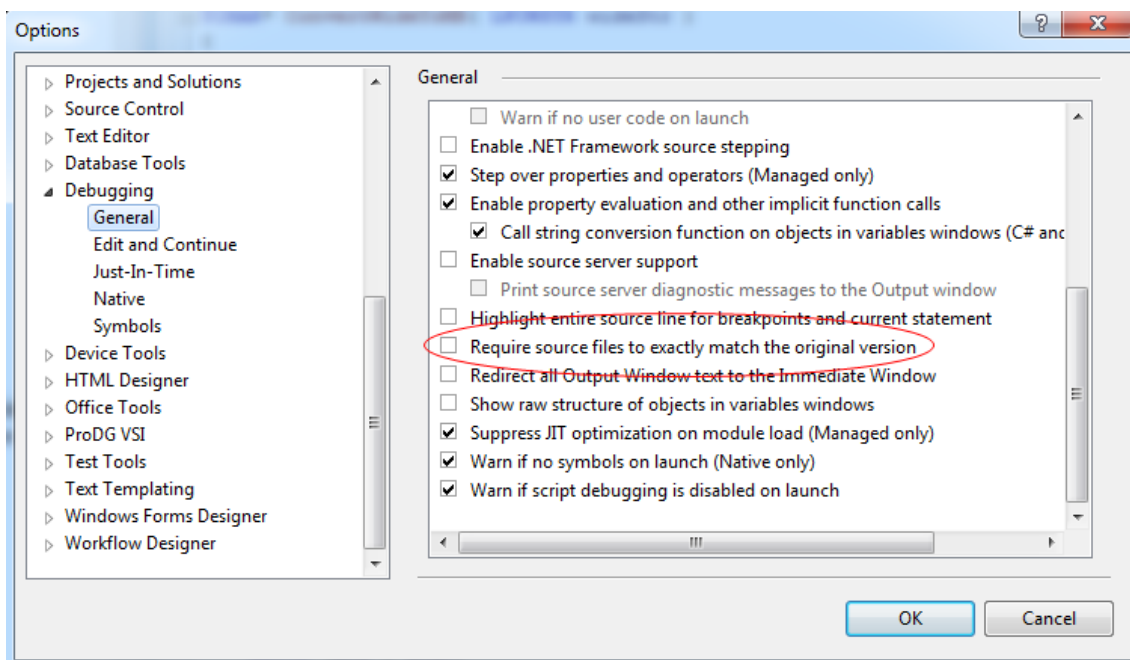
Note that it used to be necessary to download a separate set of Static packages to use the static CRT. This is no longer necessary now that `hkBaseMT.lib` and `hkBaseMTd.lib` are available.

1.4.5 Debugging Havok Libraries

1.4.5.1 Finding Source Code

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"**:



1.4.5.2 Visualizers

Some debuggers support customization of how data is displayed in the debugger. For Visual Studio 2015 and above, the easiest approach is to add the natvis files from Tools/Addons/VisualStudio to your solution. See the readme.txt in that folder.

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 Havok Support

To submit support queries, download the latest builds of Havok, and check for additional help in the form of our online Knowledge Base, please visit <http://support.havok.com>. If you need a login, contact your Account Manager.

1.6 Setting up the Visual Debugger (VDB)

The following are platform-specific details on setting up the VDB.

It's worth noting that as of 2017.1, the VDB has the ability to detect game servers awaiting connection for most current-gen platforms. This Server Discovery is enabled through an advertisement port. The default advertisement port is 15001.

1.6.1 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.

For full instructions please refer to our prior section on [Connecting the Visual Debugger](#).

1.7 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.

1.7.1 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.

1.7.2 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.

1.7.3 Havok Animation

- See all of "[Havok Common](#)" above.
- Use HCT to create skeleton, animation and (optionally) skin. Preview all elements in the 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.

1.7.4 Havok Animation Studio

- See all of "[Havok Animation](#)" above.
- Serialize Havok Animation data from HCT into Havok Animation Tool (HAT).
- Serialize data from HAT into the Havok Behavior Runtime (HBR).

1.7.5 Havok Cloth

- See all of "[Havok Common](#)" above.
- Follow the Integration documentation in the Havok Cloth manual.
- Test a simple sample (e.g. a flag, not character cloth) of your own making in the Cloth/Api/LoadCloth and Cloth/Api/LoadSetupCloth demos.
- Load and test that sample in your runtime using the VDB (and ignoring rendering integration).
- Add rendering of the simple sample.
- Then move on to character cloth.

1.7.6 Havok Destruction 2012

- See all of "[Havok Common](#)" above.
- See all of "[Havok Physics 2012](#)" above.
- Try the Havok Destruction Demo Examples.
- Use the Havok Content Tools (HCT) in a modeler to fracture a sphere.
- Use the HCT in a modeler to create a convex decomposition of a model.
- Load an asset, fractured via the HCT, into the Havok Destruction demo SimpleGraphicsSceneLoader (put the `hkt` file in the `Demo/Demos/Resources/Destruction/Scenes` directory).
- Create a demo using the Havok Demo Framework that implements the breaking of a simple shape (such as a box) in code.
- Integrate Havok Destruction with your graphics pipeline.

1.8 Optimizations

	PC/Linux
Math Library Vector Instructions	SSE intrinsics
Constraint Solver	SIMD
GetSupportingVertex (GSK)	SIMD
CalcAabb (hkpAabbUtil)	SIMD
Broad phase (hkp3AxisSweep)	MMX/SSE hand-optimized

1.8.1 Win32 SIMD

Havok's Win32 libraries (for the x86 architecture) use SIMD optimizations by default, but libraries that don't use any SIMD optimizations are also available for customers who need them. The SIMD-optimized Win32 SDK packages are simply branded `win32_[<visual_studio_version>]`, and the packages containing libraries that don't use any SIMD optimizations are branded `win32_[<visual_studio_version>]_noSimd`. In most situations only one set of Win32 packages, usually the SIMD-optimized set, will be needed. However, in certain situations both sets may be required, e.g. one set for toolchain builds and the other for runtime builds.

A `#define` called `HK_CONFIG_SIMD` controls whether the SIMD optimizations in Havok are enabled or disabled. This `#define` is set on a per-platform basis in `Source/Common/Base/Config/hkConfigSimd.h`. The situation for Win32 is unique in that it's the only platform for which both SIMD-enabled and SIMD-disabled builds are available. When the `ConfigureHavok.exe` application is run it checks to see which, if any, Win32 libraries are available and then does the following:

- If only Win32 libraries that use SIMD optimizations are present then `HK_CONFIG_SIMD` is set to `HK_CONFIG_SIMD_ENABLED` for x86 in `hkConfigSimd.h`.
- If only Win32 libraries that *do not* use SIMD optimizations are present then `HK_CONFIG_SIMD` is set to `HK_CONFIG_SIMD_DISABLED` for x86 in `hkConfigSimd.h`.
- If Win32 libraries that use SIMD optimizations and Win32 libraries that do not use SIMD optimizations are *both* present then the `HK_CONFIG_SIMD` `#define` for x86 is replaced by an `#error` message informing the user that they should define `HK_CONFIG_SIMD` appropriately wherever Havok is included in their own code:

```
HK_CONFIG_SIMD could not be set automatically for win32 as libraries both with and without simd
optimizations appear to be present in your Havok distribution. Please set HK_CONFIG_SIMD manually.
```

This is necessary because `HK_CONFIG_SIMD` needs to be set to a different value depending on whether the SIMD-optimized or no-SIMD libraries are being used.

Note that `HK_CONFIG_SIMD` is set to `HK_CONFIG_SIMD_ENABLED` by default for Win32, so in most situations the `ConfigureHavok.exe` application won't need to modify the `hkConfigSimd.h` header file at all. Please refer to the Havok Setup Guide document for more information on the `ConfigureHavok.exe` application.

1.8.1.1 What happens if the Win32 SIMD setting in `hkConfigSimd.h` somehow ends up set to the incorrect value?

Using SIMD-optimized libraries when `HK_CONFIG_SIMD` is set to `HK_CONFIG_SIMD_DISABLED` typically results in lots of link-time errors that look something like this:

```
error LNK2001: unresolved external symbol "struct hkQuadReal const * const g_vectorConstants"
error LNK2019: unresolved external symbol "public: void __thiscall hkMatrix3::mul(class hkSimdReal)"
```

Using no-SIMD libraries when `HK_CONFIG_SIMD` is set to `HK_CONFIG_SIMD_ENABLED` typically results in lots of link-time errors that look something like this:

```
error LNK2001: unresolved external symbol "union __m128 const * const g_vectorConstants"  
error LNK2019: unresolved external symbol "public: void __thiscall hkMatrix3::mul(class hkSimdReal const &  
)"
```

1.9 PC Performance Tests

This section provides performance statistics for the PC platform.

1.9.1 Animation Timings

This section pertains to Havok Animation. The timings in this section were captured using an Intel Q6600 processor.

1.9.1.1 Sampling

This section describes the runtime performance of decompressing animations.

The timing data in this section is derived from the Havok `SampleOnlyTimingsDemo`. The demo loads 15 animations based on a 38-bone skeleton and compresses them. The animations are duplicated to create 100 animations, each with its own starting time. The total time per to sample the 100 animations is shown in the following table:

Compression Scheme	Single-threaded	3 Worker Threads
Spline	1,420	684
Predictive	1,863	810
Quantized	482	299

For all animation types the cost of decompression is roughly linear in the number of bones, so you can estimate your decompression costs accordingly.

1.9.1.2 Sampling and Blending

This section describes the runtime performance of decompressing and blending animations.

The timing data in this section is derived from the Havok `SampleAndBlendTimingsDemo`. The demo creates 100 `hkaAnimatedSkeletons` having 38 bones, with each one blending 4 animations (from the same set of 15 animations used in the `SampleOnlyTimingsDemo`). Each animation is given a unique local starting time. This produces a total of 400 animations. The total time to sample and blend the 400 animations for 100 characters is shown in the following table:

Compression Scheme	Single-threaded	3 Worker Threads
Spline	5,601	2,193
Predictive	7,439	2,892
Quantized	2,207	932

For all animation types the cost of decompression is roughly linear in the number of bones, so you can estimate your decompression costs accordingly. The cost of blending is roughly linear in the number of animations and the number of bones (with a small amount of overhead per blend).

1.9.2 Behavior Timings

This section describes timing results for Havok Behavior using a dual-core PC.

1.9.2.1 The Behavior Demos

The timings presented below were gathered from some of the demos in Behavior/UseCase/BehaviorPerformance. These demos are run, and their timings are saved to text files, when you run the Bootstrap Stats Demo:

1. Blend of 2 Clips (x100): A behavior containing an `hkbBlenderGenerator` that blends two `hkbClipGenerators`.
2. Blend of 10 Clips (x100): A behavior containing an `hkbBlenderGenerator` that blends 10 `hkbClipGenerators`.
3. Synced Transition Between 2 Synced Clip Pairs (x100): A behavior containing an `hkbStateMachine` that is doing a synchronized transition between two states. Each state contains an `hkbBlenderGenerator` that is doing a synchronized blend between two `hkbClipGenerators`.

All of these demos contain 100 characters all using different instances of the same behavior.

1.9.2.2 Generate

The most computationally expensive operation in Behavior is the call to `hkbBehaviorGraph::generate()`, which generates the current pose of the character. The following table shows timings in milliseconds for performing `hkbBehaviorGraph::generate()` in the demos:

Demo	0 threads	1 thread	2 threads	3 threads	4 threads
Blend of 2 Clips (x100)	5.2	5.0	2.7	3.4	2.7
Blend of 10 Clips (x100)	21.1	21.0	11.3	10.9	10.9
Synced Transition Between 2 Synced Clip Pairs (x100)	11.3	11.1	8.4	5.7	5.8

Notes:

- The column labeled "0 threads" uses the main thread of execution to perform the `generate()` call. The column labeled "1 thread" uses another thread in addition to the main thread to perform the `generate` call, and does not use the main thread. Due to the additional overhead of the extra thread, the cost is slightly higher.
- The first two threads used are executed on the two cores of the PC, so we get substantial speedup. The last two threads are run using hyperthreading and give a modest speedup in some cases, and reduce performance in other cases.
- In the above table, most of the numbers were taken from a single frame of the simulation (frame 0). In the case of the third demo, the timings were taken from frame 11 because that is where the synchronized transition begins.
- All three of the demos in the table have 100 characters, so you can get the per-character cost by dividing by 100.

1.9.2.3 Update

The other important operation on a behavior graph is `hkbBehaviorGraph::update()`, which advances time on the behavior graph. This operation is not currently multithreaded in our demos. The following table shows the timings in milliseconds for performing `hkbBehaviorGraph::update()` in the demos, averaged over the first 100 frames of simulation:

Demo	Update
1. Blend of 2 Clips (x100)	0.7
2. Blend of 10 Clips (x100)	2.4
3. Synced Transition Between 2 Synced Clip Pairs (x100)	1.6