Third-Party Integrations

BigWorld Technology 2.0. Released 2010.

Software designed and built in Australia by BigWorld.

Level 2, Wentworth Park Grandstand, Wattle St
Glebe NSW 2037, Australia
www.bigworldtech.com

Table of Contents

Chapter 1. Introduction

BigWorld Technology is designed to be as complete as possible, and as part of our efforts we have worked with various other technology providers to integrate their products. For some of these products there is a demo available in the evaluation version of Fantasy Demo but, depending on licensing, it may not be available in the shipped version of BigWorld Technology. For some of these products you will need to contact the provider to and download patch files in order to achieve a working integration. For all of these products you will need to contact the provider to get more information on licensing. If there is a middleware product you feel we should be supporting please feel free to contact us.

Currently supported integrations:

SpeedTree

Provided by	IDV (www.speedtree.com)
Contact	Kevin Meredith <meredith@idvinc.com>
Supported Version	SpeedTree 4.2 integrated against BigWorld Technology 1.8 and later. Supported by BigWorld's Client and World Editor.
Description	SpeedTree is a powerful toolkit for creating vegetation in games.
Integration Details	SpeedTree is used in Fantasy and Urban Demo, for demonstration purposes. BigWorld maintains the integration. See also *SpeedTree* on page 15 .

FMOD Ex

Provided by	Firelight Technologies Pty Ltd (www.fmod.org)
Contact	<sales@fmod.org>
Supported Version	FMOD Ex 4.24
Description	FMOD is a programming library and toolkit for the creation and playback of interactive audio.
Integration Details	FMOD is integrated in all versions of BigWorld Technology. It is turned on by default in the evaluation but off in the shipped versions. BigWorld maintains the integration in partnership with Firelight Technologies. See also *FMOD* on page 21 .

Scaleform GFx

Provided by	Scaleform Corporation (www.scaleform.com)
Contact	<sales@scaleform.com>
Supported Version	Scaleform GFx 3.0
Description	Scaleform GFx accelerates Flash content. It allows development teams to quickly and easily implement 3D hardware accelerated game content and interfaces, including menu UIs, HUDs, animated textures, mini games, and even full casual games.
Integration Details	BigWorld maintains the integration in partnership with Scaleform. See detailed notes below. This integration is turned off by default in the shipped versions as customers need to buy a Scaleform license. See also *Scaleform* on page 11

Vivox Precision Studio SDK

Provided by	Vivox (www.vivox.com)
Contact	info@vivox.com
Supported Version	N/A
Description	Vivox Precision Studio SDK is a high quality voice chat solution. From basic voice chat, IM, and presence indicators to our more advanced features, such as 3D positional audio, out-of-game connections and voice fonts, Precision Studio enables online game developers to easily bring customised voice communications to their players. Vivox chat users connect to the Vivox servers directly, freeing you from management and scaling issues.
Integration Details	Vivox maintains the integration. Contact them directly for the latest patches.

Dolby Axon

Provided by	Dolby (www.dolby.com)
Contact	Dolby Contact Page
Supported Version	N/A
Description	Dolby Axon is a realistic 3D voice experience that matches the game environment. It delivers a clear signal, free of unwanted noise, echo, and clipping. It expands creative game-play options for developers, and scales seamlessly across servers to support large virtual worlds. Servers are managed by the developer, allowing flexibility. More details atwww.dolby.com
Integration Details	Dolby maintains the integration. Contact them directly for the latest patches.

ForkParticle

Provided by	Fork Particle Inc (www.forkparticle.com)
Contact	Shams Dar <sdar@studiofork.com>
Supported Version	Fork Particle 4.0 integrated against BigWorld Technology 1.9.3.
Description	Fork Particle provides state of the art real time particle effects simulation and an authoring solution for video game and visual simulation developers.
Integration Details	ForkParticle maintains the integration. Contact them directly for the latest patches.

Umbra

Provided by	Umbra (www.umbrasoftware.com)
Contact	<sales@umbrasoftware.com>
Supported Version	Umbra 1.5.4 integrated against BigWorld Technology 1.9.0. Supported by BigWorld's Client and World Editor.
Description	Umbra is rendering optimisation middleware. It determines automatically which triangles are unseen and culls them for better frame rates. Umbra Software's middleware technology is the secret behind some of today's best

	games such as Guild Wars 2 and Age of Conan: Hyborian Adventures. It requires no configuration or developer effort, working transparently in the background.
Integration Details	BigWorld maintains the integration in partnership with Umbra. See detailed notes below. This integration is turned off by default in the shipped versions as customers need to buy an Umbra license. See also *Umbra* on page 9 .

OpenAutomate

Provided by	nVidia (www.nvidia.com)
Contact	N/A
Supported Version	N/A
Description	OpenAutomate is a new standard for application automation. Most applications these days (games included) have their own, unique ways of storing internal parameters and of running benchmarks. OpenAutomate-enabled applications, on the other hand, are automated externally via a common, standardised C-based plug-in. With virtually zero overhead and no complex files to parse or write, the OpenAutomate plug-in can query/set application parameters, run internal benchmarks and measure the resulting performance.
Integration Details	BigWorld maintains the integration. This integration is turned on by default in the shipped version. See also *OpenAutomate* on page 25 .

Chapter 2. Umbra

Umbra is a third party visibility library produced by Umbra Software Ltd (www.umbra.fi). Umbra is not provided as part of the BigWorld Technology license and needs to be licensed separately if required.

Umbra uses graphics hardware to determine what parts of the scene are visible. Any opaque object that writes to the depth buffer can be used as an occluder and any chunk item with a bounding box can potentially be occluded.

2.1. Building BigWorld Client with Umbra enabled

To enable Umbra, the preprocessor definition `UMBRA_ENABLE` needs to be set to `1` in `umbra_config.hpp` in the chunk library (`src/lib/chunk/umbra_config.hpp`).

You will also need to add the Umbra include and library paths to your projects.

To link against the Umbra debug library, change the `#pragma comment(lib, "umbra.lib")` in the file (`src/lib/chunk/chunk_umbra.cpp`).

2.2. Modes of operation

Umbra can run in two different modes: Hardware and Software.

The engine will choose to use hardware mode when hardware occlusion queries are supported.

In hardware mode, Umbra uses the z-buffer for occluding objects, this means that any opaque object that is rendered in the scene can occlude objects that are further away.

In hardware mode only 2 umbra cells are created, one for inside objects and one for outside objects, both cells exist in world space and are connected by umbra portals.

The BigWorld Client will chose to use Umbra in software mode when hardware occlusion queries are not supported.

In software mode only objects specified as occluders can occlude other objects. By default only terrain occludes geometry, additional static models can be set manually as occluders in ModelEditor (for details, see the document Content Tools Reference Guide's section *ModelEditor* → "Panel summary" → "Object Properties panel"). All chunk items with a physical size can be occluded by Umbra. Please refer to the Umbra reference documentation for descriptions of Umbra-related objects.

In software mode chunks correspond fairly well to Umbra cells. Each indoor chunk has a corresponding Umbra cell, which is connected to other Umbra cells by Umbra portals. All outdoor chunks share one common Umbra cell, which is owned by the chunk space. All cells exist in world space.

2.3. Chunk items

When using Umbra to render the scene, every ChunkItem needs to have a corresponding Umbra object. The chunk item is set as the Umbra object's user data, so that when an Umbra object is deemed visible, its chunk item can be retrieved and rendered.

When implementing a specialised chunk item, you will need to add a Umbra object for it. The chunk item contains a UmbraObjectProxy and a UmbraModelProxy, which can store the Umbra object and Umbra model for the specialised chunk item. The object-to-cell matrix is the world transform of the object.

2.4. Umbra proxies

To ease handling of Umbra objects, BigWorld Technology provides proxy objects for Umbra objects (class `UmbraObjectProxy`) and Umbra models (class `UmbraModelProxy`). These objects are a thin wrapper

around the corresponding Umbra objects, allowing BigWorld Technology to use standard smart pointers to reference them.

The proxies are also stored in a managed list, to ensure that all Umbra objects are destructed before Umbra is shut down. For details on these classes, see the C++ API documentation.

Chapter 3. Scaleform

3.1. Introduction

Scaleform GFx is a third party Flash rendering library produced by Scaleform (http://www.scaleform.com/). Scaleform is not provided as part of the BigWorld Technology license and needs to be licensed seperately if required.

BigWorld Technology is integrated with Mozilla using llmozlib which allows showing web pages and Flash movies as part of the game but this solution is less efficient than Scaleform and therefore not suitable for game user interfaces.

Scaleform provides a hardware accelerated renderer for Flash movies suitable for use in games. BigWorld has implemented an integration with Scaleform allowing Flash movies to be displayed, and controlled by Python.

Scaleform IME is an optional add-on for Scaleform customers, allowing display of an in-game Scaleform IME movie to replace the windows IME language bar. BigWorld has integrated against Scaleform IME as well.

While Scaleform has tried its best to optimise rendering of Flash movies, there are still a number of guidelines that artists should follow in order to create Flash movies that behave well and don't bog down performance unnecessarily. It is therefore essential that you read the appropriate Scaleform documentation before launching into full-scale production of Flash movies for use with your BigWorld game.

This document describes how to display and use a Flash movie in BigWorld. It describes the changes required to the client to build in the Scaleform integration. It then describes how to use the Python API to display and manipulate a Flash movie. Finally it describes some BigWorld-specific features and limitations.

3.2. Building the Client with Scaleform

In order to build the client with support for Scaleform, you must download and install the SDK (version 3). The installer will create an environment variable called `GFXSDK` which is used by the BigWorld client solution to locate the SDK installation. You can check that the environment variable is correctly set by opening a command prompt and entering the command `set GFXSDK`. It should print out the root installation path of the Scaleform SDK.

3.2.1. Setting the BigWorld Build Configuration Macros

The Scaleform integration library is included with your BigWorld Technology package in the `bigworld/src/lib/scaleform` directory. Contained in this directory is a file called `config.hpp`, which should be changed to look like this:

```
#define SCALEFORM_SUPPORT 1
#define SCALEFORM_IME 1              //only necessary if also using Scaleform
 IME
```

Then rebuild the client. That's all you should have to do.

3.3. Displaying Flash Movies

This section aims to show you how to load up a Flash movie and display it in your game's user interface. There are two approaches to achieving this. The first approach involves displaying a global movie which bypasses the BigWorld GUI system completely. The second approach involves showing a Scaleform movie within a BigWorld GUI tree. It is recommended to use the second approach, but for simplicity and to get up and running quickly you can display a flash movie as part of a global flash movie list.

3.3.1. Getting up and running quickly

The simplest way to display a flash movie is to load a Movie Definition File (representing a .swf file), create a view of that (Movie View) and display the movie. This type of movie will always handle key and mouse inputs, and will always display at the full size of the client window. Note that just like any other resources loading in BigWorld, Scaleform resources must exist within your resource paths. Any attempts to load a flash file from outside of your specified resource paths will throw a Python IOError Exception.

```
import Scaleform
movieDef = Scaleform.MovieDef( filename )
movieView = movieDef.createInstance()
movieView.showAsGlobalMovie()
```

3.3.2. Python controls the lifetime of Scaleform Objects

In the above example, two objects were created, a Movie Definition and a Movie View. Both of these are Python objects that internally maintain a reference to one or more Scaleform C++ objects. As a game programmer, all you need to know is that as long as you keep your Python objects alive, the underlying Scaleform objects are kept alive too. Here's a code snippet that demonstrates the lifetime of these two Python objects :

```
#movieDef, movieView attributes created in the above example

# First we will free the movie definition.
del movieDef

# The Movie View will still play, however in future if you require a new view
 of that movie, you'll have to reload the entire movie.

# Now free the movie view.
del movieView

# Notice that deleting the movieView object means it is no longer displaying
```

If you hold onto Scaleform objects after the Personality Script is fini'd(), then you will run into problems, as the Scaleform subsytem is shut down before Python. When Python is shut down, it frees all its outstanding objects, and the Python-controlled Scaleform objects will be released too late, causing an assert. It is up to you to make sure you release these objects before shutdown.

3.3.3. Better Control over the Display of your Movie

In the first example, the movieView created was displayed as part of a simple, global list of movies. In practice, you may want to mix BigWorld GUI objects and Scaleform movies, for example, to embed a Scaleform movie within a moveable window, or to pop up a Scaleform movie like an Inventory display in the HUD.

In order to do this, you simply wrap the MovieView object in a Flash GUI Component. Once you do this, the GUI object takes control over the lifetime, display, size, position and input handling for your movie.

```
# movieDef, movieView attributes created in the above example

import GUI
g = GUI.Flash( movieView )
GUI.addRoot(g)

# movie is now displayed via the GUI tree, instead of displayed via a global
 Scaleform movie list.
```

```
# The movie can now be positioned just like other GUI components.
g.position = (1.0,-1.0,0.6)
g.verticalAnchor = "BOTTOM"
g.horizontalAnchor = "RIGHT"
g.widthMode = "PIXEL"
g.heightMode = "PIXEL"
g.size = (256, 256)

# The movie now responds to key / mouse events just like other GUI components
g.focus = True             # respond to key events
g.mouseButtonFocus = True  # respond to mouse button events
g.moveFocus = True         # respond to mouse events
```

Note

If you attach a script object to a FlashGUIComponent, you have to pass input events through to the MovieView yourself. This allows the script to filter events before they get passed through. For example, the following snippet passes through all input events except the escape key which is handled as a special case:

```
class MyFlashScript( object ):
    def __init__( self, component ):
        self.component = component
        component.script = self

    def handleKeyEvent( self, event ):
        if event.key == Keys.ESCAPE:
            self.visible = False
            return False

        return self.component.movie.handleKeyEvent( event )


    def handleMouseButtonEvent( self, comp, event ):
        return self.component.movie.handleMouseButtonEvent(
 event )

    def handleMouseEvent( self, comp, event ):
        return self.component.movie.handleMouseEvent( event )
```

3.4. Interacting with Flash Movies

No doubt you will want to build complex UI's in Flash that interact with your BigWorld game, sending and receiving events and swapping data back and forth. All method and data interactions are handled via the PyMovieView Python object. This chapter details a number of different ways to interact with your Flash movie and its action script.

3.4.1. Using fscommand in Action Script to call into Python

In earlier versions of Action Script, the `fscommand()` call was the way to call out from Action Script and into its container. `fscommand()` is supported by Scaleform and BigWorld. In order to handle `fscommand()` calls, you simply register a Python callback function to handle any such calls :

```
def myFSCommandHandler( cmd, arg ):
```

```
    # cmd, arg are both strings.  Dispatch to wherever is appropriate.

movieView.setFSCommandCallback( myFSCommandHandler )
```

3.4.2. Using External Interface calls from Action Script to Python

In later versions of Action Script, external interface is used instead to call out of Action Script and into the containing application. External Interface calls have the advantage of allowing different types and number of arguments to be passed around (instead of fscommand supporting only a single string argument). Additionally, the container can set return values for the external interface call. Usage is similar to handling fscommand, however the callback function has a different interface :

```
def myExternalInterfaceHandler( cmd, *args ):
    # cmd is still a string, but now we have a list of arguments, with
 arbitrary types.  Dispatch to wherever is appropriate

movieView.setExternalInterfaceCallback( myExternalInterfaceHandler )
```

3.4.3. Setting and Getting Action Script variables directly from Python

Action Script variables tend to be buried in a hierarchy, often looking like : `"_root.levelN.someVariableName"`. Unfortunately this syntax is misinterpreted by Python and currently there is no elegant solution to work around this. However it is still easy to exchange data with Action Script. All that is required is to use Python's `setattr` and `getattr` syntax. For example:

```
setattr( movieView, "_root.level0.maxHealth", 10 )
currentHealth = getattr( movieView, "_root.level0.health" )
```

Using this syntax you can get and set any of the following types of data : boolean, int, floats, string and unicode string.

3.4.4. Calling Action Script methods from Python

The final interaction you will want to have with your Flash movie is to call Action Script methods directly from Python. In order to do this, simply use the `invoke()` method:

```
result = movieView.invoke( "_root.level0.flashHealthBar", 10.0 )
```

Chapter 4. SpeedTree

SpeedTree is a third party library used for creating and rendering vegetation in games and is produced by Interactive Data Visualization, Inc (http://www.speedtree.com/). SpeedTree is not provided as part of the BigWorld Technology license and needs to be licensed separately if required.

4.1. Building with SpeedTree support

BigWorld's source code ships with support for SpeedTree turned off by default. In order to turn SpeedTree support on in BigWorld's game client and tools, in `bigworld/src/lib/speedtree/speedtree_config.hpp`, set the `SPEEDTREE_SUPPORT` macro to `1`. You will also need to have SpeedTreeRT SDK 4.2 properly installed and configured.

4.2. Resource files

For a SpeedTree to be available in WorldEditor, or to be loaded from a chunk file into a space within the game client, the following resource files need to exist in the resources tree:

- The SpeedTree file (`.spt`).
- The branch texture map file.
- The composite texture map file.

For each tree, these files should reside in the same folder. More than one tree can coexist in the same folder, as long as no filename clash occurs.

For performance reasons, we assume that all trees use a composite map for the frond, leaf, and billboard textures. For details on how to have a tree to use composite maps, see the SpeeTreeCAD's User Manual.

To take advantage of SpeedTreeCAD's alpha noise cross-fading feature (a fundamental feature which use we highly recommend), you will also need an accompanying `.texformat`[1] filefor the composite texture map, or an appropriate wildcard entry in `<texture_detail_levels>.xml`. For details, see “Level of detail” on page 16 .

Optionally, you can have a `SpeedWind.ini` file in this same folder if you require finer control over how this tree wind animates. For details, see “Wind animation” on page 16 .

4.3. SpeedTrees in WorldEditor

Once the resource files are in place, trees are ready to be used within WorldEditor. To insert a tree, either drag a `.spt` file from the **Asset Browser** panel[2] into the viewport, or select the desired `.spt` file and press `Enter`.

Once a tree has been added, it can be selected, moved, rotated, and scaled like any other model in WorldEditor. You can do this interactively (see “Mouse controls”) or by editing the trees' properties in the **Properties** panel (see “Properties panel”). In the **Properties** panel, you can also edit the name of the `.spt` file and the seed number used to generate the tree geometry. The results of editing a tree's properties can be immediately previewed in the viewport.

If a tree cannot be loaded during chunk loading then a red box model will be rendered in its place. The box can still be manipulated and most of its properties edited with the exception of its seed. Any changes will be saved and restored if that tree is loaded in the future.

---

[1] For details on this file's grammar, see the document File Grammar Guide's section `.texformat`.

[2] For details on the Asset Panel see “Asset Browser panel”.

4.4. Wind animation

BigWorld uses SpeedTree's new SpeedWind library to animate trees swaying to the wind. SpeedWind's parameters can be tuned and previewed from within SpeedTreeCAD, and then saved as a .ini file. This file can then be loaded into BigWorld's 3D engine to control how trees animate.

A global .ini file is used by default for all trees. Optionally, a unique SpeedWind.ini file can be tied to a specific tree, by placing it in the same folder as the .spt file (the name of the file is required to be `SpeedWind.ini`). If neither file is present, then SpeedWind's engine will be setup with its default parameters.

Although SpeedWind's parameters define how a tree should animate given a wind of certain direction and strength, the actual wind's direction and strength fed into SpeedWind's engine is provided by BigWorld's dynamic weather system, conveying a convincing and immersive experience for the player.

4.5. Level of detail

BigWorld's 3D engine fully supports SpeedTree's LOD model. Branches and fronds switch detail levels discretely, while leaves and billboards LODs cross-fade according to the distance to the camera.

By default, the actual LOD level of a tree — a value that ranges from 0 (minimum detail) to 1 (maximum detail) — is a linear function based on the distance from the tree's origin to the camera's eye position. The lodFar and lodNear parameters define the distance interval within which the trees' LOD levels will vary. In the default LOD mode, the LOD function is the same for all trees.

Optionally, you can rely on SpeedTree's engine to compute the LOD level for each tree based on parameters contained in the .spt file (which can be edited from SpeedTreeCAD). Roughly speaking, they work in the same way as explained above, but with specific near and far values for each tree. This allows a more fine-grained control of LOD transitions, and a better match between what is previewed on SpeedTreeCAD and what is rendered on the tools and the client. The only drawback is that relying on SpeedTreeRT for the LOD computation is a bit slower than letting BigWorld do it.

You can switch between this two LOD modes by editing the `lodMode` parameter in SpeedTree's XML file (for details, see “XML file” on page 18) or interactively, via watchers (for details, see “Watchers” on page 18). The parameters `lodNear` and `lodFar` are also accessible this way.

The `lodOverlap` parameter defines how much overlapping should happen during the cross-fading of LOD levels. The valid range goes from 0 to 2. The default value is 1. This parameter is also accessible via XML file and watchers.

Leaves and billboards, when phasing in or out their LODs, cross-fade their alpha test values in order to smooth level transitions. This technique requires composite texture maps with homogeneous noise baked into the alpha channel and pre-generated mipmaps. Also, BigWorld's auto mipmapping generation should be disabled for this texture map.

To disable the automatic generation of mipmaps in BigWorld, you will need to setup a `.texformat`[3] file for each tree's composite texture map, which is illustrated below:

```
<root>
  <mipCount>       -1    </mipCount>
  <horizontalMips> true  </horizontalMips>
  <mipSize>        0     </mipSize>
</root>
```

Example `.texformat` file

The example above configures the following:

[3]For details on this file's grammar, see the document File Grammar Guide's section *`.texformat`*.

- **`mipCount`**

  -1 ensures that all mipmap levels baked into the texture will be used.

- **`horizontalMips`**

  Should always be true for SpeedTreeCAD's generated mipmaps.

- **`mipSize`**

  Width of the top-most level mipmap. If set to zero, then will use half the width of the original texture file, which is right for composite maps exported from SpeedTreeCAD.

A more general way of achieving this is by adding a wildcard entry to `<texture_detail_levels>.xml`. The entry should match all composite maps used by SpeedTree, and define the same attributes shown above. An example of a wildcard entry is illustrated below:

```
<detailLevel >
  <prefix>         speedtree </prefix>
  <contains>       composite </contains>
  <mipCount>       -1        </mipCount>
  <horizontalMips> true      </horizontalMips>
  <mipSize>        0         </mipSize>
</detailLevel >
```

Example `<texture_detail_levels>.xml` entry

For details on how to create texture maps with alpha noise and pre-generated mipmaps, see SpeedTreeCAD' **User Manual**.

4.6. Collision geometry

The first time a tree is loaded, BigWorld uses the collision geometry specified in the .spt file to generate the BSP trees that will populate the collision scene in a space. In the game client, you can preview these BSP trees by enabling the `SpeedTree/BSP` watcher (for details, see “Watchers” on page 18). In WorldEditor, enable the **General Options** panel's **Show** list box's **Scenery → BSP** item (for details, see “General Options panel”).

If the engine has write access to the resource path, then a cache of the BSP tree will be saved, to speed upload time in the future. This cache file will be re-created whenever the `.spt` file is modified.

Once a tree's BSP tree has been inserted into the collision scene, it will be used in all collision queries in the engine, including line-of-sight, navigation paths generation, static lighting, and more. When loaded from within WorldEditor, if a tree has no collision geometry defined, then a temporary BSP tree in the shape of a box will be created, to allow the world builder to select and interact with the tree. This temporary BSP tree is not cached, and will not be created when the tree is loaded into the game client.

For details on how to define a tree's collision geometry, see SpeedTreeCAD's User Manual.

Note

Although higher numbers of volumes defining a tree's geometry result in higher load on the physics system, no BSP traversal is done if the collision query falls outside the bounding box of the tree.

Thus, using more volumes is not as expensive as it may at first seem (although it definitely more expensive than using little or no volumes at all).

Note

Due to the nature of BSP trees, spheres and cylinders can be inefficient in terms of memory and runtime performance when compiled into a BSP.

To compensate that, BigWorld uses low-resolution tessellation when generating them.

For this same reason, whenever possible you should try to use boxes instead of cylinders and spheres to approximate the collision geometry of a tree.

Normally, the server is not built to use SpeeTreeRT, and requires the BSP files to be present in the resource path in order to load SpeedTrees into the collision scene. The recommended approach is to check all SpeedTree BSPs into the repository so that they will be accessible to the server. When running the server, if the associated BSP file is not available when loading a SpeedTree, then it will fail to load (the chunk will be loaded successfully, though).

4.7. XML file

All game-specific settings related to rendering SpeedTrees are defined in a XML file, which actual name is defined in `resources.xml` (for details, see the document Client Programming Guide's section *Overview*→ "Configuration files" → "File `resources.xml`") , under the `system/speedTreeXML` tag, and defaults to `speedtree/speedtree.xml`.

This file is read upon initialisation of BigWorld's 3D engine. For details on these tags, please see inline comments in `fantasydemo/res/speedtree/speedtree.xml`.

4.8. Watchers

In both the game client and WorldEditor, several watchers (for details, see the document Client Programming Guide's section *Debugging* → "Watchers") are available to instantly customise and debug the behaviour of SpeedTrees.

The list below describes them (all watchers listed are prefixed by `SpeedTree/`):

- **Batched rendering**

  Toggles batched rendering. Default is on.

- **BB Optimiser/ Active buffers**

  Number of buffers in pool that are currently active — these are not available for recycling. Read-only.

- **BB Optimiser/ Frames to recycle**

  Number of frames during which a billboard optimiser buffer should stay unchanged before it is set for recycling.

- **BB Optimiser/ Frames to delete**

  Number of frames during which a billboard optimiser buffer should stay unchanged before it is set for deleting.

- **BB Optimiser/ Pool size**

  Number of buffers in pool. Read-only.

  This number minus the number of active buffers (BB Optimiser/Active buffers) is the total number of buffers currently available for recycling.

- **BB Optimiser/ Visible buffers**

  Number of buffers in pool that are currently visible. Read-only.

- **Bounding boxes**

  Toggles the rendering of trees' bounding boxes. Default is off.

- **BSP Trees**

  Toggles the rendering of trees' BSP trees. Default is off.

- **Counters/Instances**

  Total number of trees instantiated. Read-only.

- **Counters/Species**

  Number of speed tree files currently loaded. Read-only.

- **Counters/Unique**

  Number of unique tree models currently loaded. Read-only.

- **Counters/Visible**

  Number of trees currently visible in the scene. Read-only.

- **Draw billboards**

  Toggles the rendering of billboards. Default is on.

- **Draw branches**

  Toggles the rendering of branches. Default is on.

- **Draw fronds**

  Toggles the rendering of fronds. Default is on.

- **Draw leaves**

  Toggles the rendering of leaves. Default is on.

- **Draw trees**

  Toggle the rendering of speed trees. Default is on.

- **LOD far**

  Distance for minimum LOD.

- **LOD Mode**

  Mode of handling of LODs. Value range is:

  - **-1** — Dynamic — near and far LOD defined per tree.
  - **-2** — Dynamic — uses global near and far LOD values.
  - **0.0 to 1.0** — Forces static value.

- **LOD near**

Distance for maximum LOD.

- **LOD overlap**

  LOD overlap value. Value range is 0.0 to 2.0.

- **Optimise billboards**

  Toggles billboard compounding. Default is on.

- **Play animation**

  Toggles wind animation. Default is on.

- **Texturing**

  Toggles texturing. Default is on.

The watchers **LOD Mode**, **LOD Near**, **LOD Far**, and **LOD Overlap** are read from SpeedTree's XML file when the engine is initialised. Modifying them via WebConsole's **ClusterControl** module (for details, see the document Server Operations Guide's section *Cluster Administration Tools* → "WebConsole") does not modify the contents written into the file.

4.9. Troubleshooting

The list below describes some errors that you might come across when implementing SpeedTree:

- **Speed trees are not rendered.**

  SpeedTreeRT SDK is not installed.

  —or—

  SpeedTreeRT SDK license has expired.

- **Leaves are textured with whole composite map, instead of just the leaf part.**

  Trees have not been correctly exported for real-time use.

  The best approach is to export the tree to a folder different from the original tree. This will guarantee that all files in the exported folder are for real-time use, and also preserve the original tree files.

- **Cross-fading of leaves and billboards does not look correct (leaves on current LOD disappear before next LOD leaves are displayed).**

  Composite texture map was not generated with alpha noise.

  —or—

  BigWorld's automatic generation of mipmap is enabled. For the cross-fading with alpha noise to work correctly, you must generate the composite texture map with pre-generated mipmaps, and instruct BigWorld to use them, instead of automatically generating the mipmaps on load. For more details, see "Level of detail" on page 16 .

- **Terrain vertex shadow is not correct**

  Trees do not have collision geometry, and BigWorld uses the collision scene to calculate terrain shadows.

  Note that, because of a known limitation in BigWorld's outdoor static lighting model, shadows of tree-like objects may not look correct under some sun angles.

Chapter 5. FMOD

FMOD is a third party library for music and sound effects by Firelight Technologies (www.fmod.org). BigWorld provides sound support via the FMOD sound library. A license to use FMOD is not included with BigWorld. Thus in order to release a title using FMOD, a separate licensing agreement with Firelight Technologies needs to be entered into.

Whilst FMOD is enabled in BigWorld Fantasy Demo, it is disabled in the shipped source by default. To enable FMOD, edit `bigworld/src/lib/fmodsound/fmod_config.hpp` and set `FMOD_SUPPORT` to `1` and rebuild the client. If FMOD support has been disabled, the FMOD Python module will still be available, however none of the function calls will succeed.

This documentation provides an overview of how to use the FMOD sound features exposed by BigWorld Technology, which is essentially a thin interface to the FMOD Designer API. For documentation related to general FMOD design principles and sound issues, please see the FMOD documentation itself.

The features exposed by the engine are:

- Loading of one more more sound banks (preloaded or streamed).
- Playback of 2D and 3D sound events.
- Remotely connecting to the BigWorld client using the FMOD Designer in order to tweak event parameters in real time and view profiling metrics.
- Spatial reverb using reverb presets as defined in the FMOD Designer.
- Access to the FMOD dynamic music system.

5.1. Overview

5.1.1. Sound Engine Configuration

High level settings can be configured by editing the soundMgr section in the engine_config.xml file. Fantasy Demo provides an engine configuration file which can be used as an example.

The following is a list of available settings.

5.1.1.1. enabled

This controls whether sound support is enabled. When this is set to false, the SoundManager will not be initialised on startup.

5.1.1.2. errorLevel

This can be set to either 'silent', 'warning', or 'exception' to control what happens when FMOD calls (such as `Entity.playSound` or `FMOD.playSound`) hit an error.

5.1.1.3. channels

Defines the number of virtual channels to be made available to FMOD. Please see the FMOD documentation for `EventSystem::init` for a detailed description on channels.

5.1.1.4. mediaPath

The directory inside the res tree where your sound banks reside. Note that FMOD requires that all your sound banks be in the same directory, and so only the first directory in your res path that matches the `<mediaPath>` will be used for loading sounds.

5.1.1.5. soundBanks

This defines the sound banks to be made available, and which will be preloaded when the engine starts up. This section is made up of one or more `project` tags. The schema for each `project` tag is:

```
<project>
    <name> project_name </name>
    <preload> true|false </preload>
</project>
```

The project name is the name of the `.fdp` file located in the `mediaPath` excluding the extension.

If `preload` is set to true, then the sound bank will be registered on startup. If it is set to false, then it will only be registered when requested by the `BigWorld.loadSoundBankIntoMemory` Python API.

5.1.1.6. networkUpdates

Enables/disables whether or not the FMOD Designer tool can connect to the client and make live updates to properties on existing sound events. This is always disabled on consumer release builds.

5.1.1.7. enableProfiler

Enables/disables external access (using FMOD Designer or FMOD Profiler) to view, in real-time, information such as memory usage, CPU usage and the DSP network graph in. This is always disabled on consumer release builds.

5.1.1.8. maxSpeed

This controls the maximum speed for 3D sounds. If the speed exceeds this value, then the sound system will assume the sound source has been teleported. This prevents audio glitches from excessive changes in object speeds, you should set this value to some realistic maximum for your application. If any 3D sound source exceeds this speed the Doppler effect will be zero.

5.1.2. Python API

The FMOD API can be accessed from Python by importing the FMOD module. See the client Python API documentation for a description of what is available in this module.

5.1.2.1. Backwards compatability

For backwards compatability with previous versions of BigWorld, the FMOD module will alias legacy sound functions into the BigWorld module (e.g. FMOD.playSound is the same as calling BigWorld.playSound). To make sure this function aliasing occurs as early as possible, be sure to import the FMOD module at the top of your personality script.

Since these older API functions will be deprecated, we encourage you to use the new FMOD module.

5.2. Loading Sound Banks

To implement sounds in your game, an FMOD sound bank needs to be created using the FMOD Designer tool. A very small example sound bank project is provided, and is located at `fantasydemo/audio/fd_sound.fdp`. Please see the documentation for FMOD Designer for details on how to create and edit your own sound banks.

Sound banks must be specified in the engine configuration XML file. Multiple sound bank projects can be specified, and can be configured to load on client startup or only by request from the scripts. See “Sound Engine Configuration” on page 21 for more detail on this.

Note

If an FMOD sound bank has been configured to stream from disk, it cannot be packed within a zip file system and must reside within an operating system directory.

5.3. Sound Playback

BigWorld provides easy to use functions for playing back sound events defined by the FMOD Designer tool.

See the Client Python API documentation for more details on these functions and the `FMOD.Sound` object.

5.3.1. FMOD.Sound Object

5.3.1.1. Just Fail If Quietest

The FMOD Event system provides a variety of max playback behaviour modes. These determine the behvaiour when the maximum number of instances of an event have been generated and then another is requested. One of the these is "just fail if quietest". This means that once the maximum number of concurrent instances of a sound event has been reached, the next attempt to playback that event will fail if it's the quietest. If it is not the quietest, it will succeed by stealing the handle from the quietest instance. The SoundManager will update all these events each frame so that only the quietest ones will be virtual.

In other words, JFIQ ensures that the loudest N instances of the event are audible at any point in time, where N is the max playbacks property of that event. Loudness is determined by the distance from the listener as well as any sound cone/listener angle effects and occlusion geometry. When an event is set to have the behaviour 'just fail if quietest' there is no limit to the number of Sounds which can be placed in the world using that event. Calling the following methods on Sounds which are virtual will return errors: volume, pitch, paused, muted, duration. There is an `isVirtual` method provided to check if a Sound is virtual or not.

An example of when to use JFIQ would be if you have 100 torches crackling in a room you would only need to hear a few of the nearest (loudest) torches to get the desired effect.

5.3.2. Sound Playback Functions

- For 2D sound events (e.g. GUI sounds), use FMOD.playSound and pass in the full path to the desired sound event (e.g. "ui/boop"). It returns an FMOD.Sound object representing the sound event being played.
- For 3D sounds, use:
  - PyModel.playSound, which will play a sound at the location of the model (it will track the model's current position). It will return an `FMOD.Sound` object representing the sound event being played.
  - FMOD.getSound to retrieve an `FMOD.Sound` handle, set its position attribute, and then call its `play` method directly. This technique has the disadvantage that it will not take advantage of the FMOD channel virtualisation features, as a single `FMOD.Sound` object represents a single virtual sound instance.

Chapter 6. OpenAutomate

OpenAutomate is a testing and benchmarking automation library produced by NVIDIA. OpenAutomate is shipped as part of the BigWorld Technology license. See http://developer.nvidia.com/object/openautomate.html for more details.

BigWorld is using and extending the Open Automate API to test the BigWorld client in house. The `res\scripts\client\OpenAutomate.py` script contains the code which runs the tests based on the Open Automate instructions.

There are several ways to use the BigWorld Technology Client with the OpenAutomate framework to test a game:

6.1. Using oaman to execute tests.

NVIDIA provides a tool called oaman which allows setting specific game settings and running tests. The oaman tool is available in `src\lib\third_party\openautomate\oaman\oaman.exe`.

Using oaman can be done as follows:

- Start `oaman.exe`
- Choose **Application→Add OA Application**
- Choose the BigWorld client executable being used to run your game.

6.2. Scheduling command line tests.

Running nightly testing to detect performance degregations and stability issues is a recommended practice for game development. The BigWorld Technology Client can read and execute open automate commands (and additional commands) based on xml files provided as part of a command line. Following is the syntax for running a test using an xml instructions file:

```
<BigWorld Technology Client> -openautomate INTERNAL_TEST <text xml file>
```

For example:

```
bigworld.exe -openautomate INTERNAL_TEST \
 <fd res>\scripts\testing\openautomate\simpleSetAutoDetect.xml
```

The above example illustrates how to auto detect the client settings and use the highest video mode. Please refer to the corresponding

```
currentCommand == "CMD_AUTO_DETECT"
```

in `OpenAutomate.py` to understand the implementation. This command should be used before running other tests to make sure that we are using the same graphics and video settings when running tests.

Running the command:

```
bigworld.exe -openautomate INTERNAL_TEST \
 <fd res>\scripts\testing\openautomate\testAdvancedRunBenchmarks.xml
```

will run multiple benchmarks and quit.

Customers wishing to add more tests can add logic to the `OpenAutomate.py` file. The function `testOAGetNextCommand()` available at `src\lib\open_automate\open_automate_tester.cpp` contains the implementation which reads the xml files and sends commands to the `OpenAutomate.py` file. Changing this function might be required for advanced customers wishing to add additional testing capabilities to their xml files.