



# The Wwise Project Adventure

*A Handbook for Creating Interactive Audio Using  
Wwise*

Damian Kastbauer

2019.2.0



# The Wwise Project Adventure

*A Handbook for Creating Interactive Audio Using Wwise*

Damian Kastbauer

Wwise 2019.2.0 Revision 2

Copyright © 2019 Audiokinetic Inc. All rights reserved.

This document (whether in written, graphic or video form) is supplied as a guide for the Wwise® product. This documentation is the property of Audiokinetic Inc. (“Audiokinetic”), and protected by Canadian copyright law and in other jurisdictions by virtue of international copyright treaties.

This documentation may be duplicated, reproduced, stored or transmitted, exclusively for your internal, non-commercial purposes, but you may not alter the content of any portion of the documentation. Any copy of the documentation shall retain all copyright and other proprietary notices contained therein.

The content of the Wwise Project Adventure documentation is furnished for information purposes only, and its content is subject to change without notice. Reasonable care has been taken in preparing the information contained in this document, however, we disclaim all representations, warranties and conditions, whether express, implied or arising out of usage of trade or course of dealing, concerning the Wwise Project Adventure documentation and assume no responsibility or liability for any losses or damages of any kind arising out of the use of this guide or of any error or inaccuracy it may contain, even if we have been advised of the possibility of such loss or damage.

Wwise®, Audiokinetic®, Actor-Mixer®, SoundFrame® and SoundSeed® are registered trademarks, and Master-Mixer™, SoundCaster™ and Randomizer™ are trademarks, of Audiokinetic. Other trademarks, trade names or company names referenced herein may be the property of their respective owners.

---

# Table of Contents

The Adventure Begins .....	ix
1. Setting the Ambient Stage .....	1
Overview .....	2
Building the Foundation .....	3
Importing Audio Files .....	3
Setting Up a Looping Sound .....	6
Adding Detail to a Developing Soundscape .....	11
Randomizing Properties .....	11
Using the Blend Container to Combine Sounds .....	12
Using the Silence Plug-in for Weighted Randomization .....	13
Randomized Positioning of Sounds .....	15
Creating the Combined Daytime Forest Ambient Combination .....	17
Section Summary .....	17
Introduction of a Day and Night Cycle .....	18
Establishing a Game Parameter .....	18
Creating the Ambient System .....	19
Using the Blend Track Editor .....	20
Setting Container Order within Blend Tracks .....	22
Preparing the Event .....	25
Harnessing Event Actions .....	26
Creating Work Units .....	30
Section Summary .....	36
Sound Emitters in the Game World .....	37
Creating Attenuation ShareSets .....	37
Creating and Establishing Generalized Attenuations .....	38
Subscribing Sound Objects to an Attenuation .....	39
Using Spread as Part of an Attenuation .....	40
Using Low Pass Filter as Part of an Attenuation .....	43
Adjusting Cone Attenuation Properties .....	44
Section Summary .....	45
SoundSeed Air -Wind .....	46
SoundSeed Wind - Deflectors .....	48
SoundSeed Wind - Properties .....	49
SoundSeed Wind - RTPC .....	51
Ambient Summary .....	52
Ambient Additional Resources .....	53
2. Establishing Character .....	54
Overview .....	55
Footsteps and Movement - Establishing Needs .....	56
Simple Steps .....	56
Switching System Introduction .....	58

Defining Step Type .....	59
Defining Surface Type .....	62
Defining Character Type .....	64
All Together Now .....	64
Movement .....	66
Defining Armor Type .....	66
Creating the Movement Event .....	67
Character Summary .....	68
Character Additional Resources .....	70
3. Preparing for Combat .....	71
Overview .....	72
Defining Sound Sets for Weapon Types .....	73
SoundSeed Air - Whoosh .....	74
Understanding Impact .....	79
Defining Weapon Type .....	79
Weapon Impact System .....	80
Attenuations for Player vs. NPC .....	82
High Alert .....	82
Listener Considerations .....	82
Combat Summary .....	84
Combat Additional Resources .....	85
4. Making Magic .....	86
Overview .....	87
Summing and Layering with Blend Containers .....	88
Creating Distance-Based Blend Tracks .....	89
Setting up a Game Parameter .....	89
Crossfading Between Containers on a Blend Track .....	93
Section Summary .....	95
Real-Time Parameter Control (RTPC) .....	96
Using Real-Time Effects .....	98
Unleashing Dynamic Synthesis .....	102
Wwise Synth One .....	102
Modulators .....	106
Modulator LFO .....	106
Modulation Envelope .....	112
Magic Summary .....	118
Magic Additional Resources .....	119
5. Dialogue Decisions and Language Lessons .....	120
Overview .....	121
Getting Started with Dialogue and Non-Verbal Vocalizations .....	122
Adding Additional Languages .....	126
Dynamic Dialogue .....	128
Cinematic Dialogue Placement .....	130
Voice Summary .....	132
Voice Additional Resources .....	133

6. Unlocking the User Interface .....	134
Overview .....	135
Creating a Simple Menu Select Sound .....	136
Defining 2D Sound Positioning .....	138
The Complex Negotiation of Pause .....	139
Pause - Defining the Scenario .....	139
Pausing the Game .....	140
Resuming the Game .....	141
Posted Events .....	142
User Interface Summary .....	143
User Interface Additional Resources .....	144
7. Adventures in Music .....	145
Overview .....	146
Starting With the Interactive Music Hierarchy .....	147
Preparing the Content .....	147
Layered: The Vertical Approach .....	148
Creating the Ambient Music Segment .....	148
Grooming the Tracks .....	151
Dynamic Danger .....	152
Adding RTPC to Tracks .....	153
Auditioning RTPC .....	155
Looping a Music Segment .....	156
Section Summary .....	156
The Sequential Approach .....	157
Groups and Behaviors .....	157
Sequencing Groups in the Music Playlist Editor .....	158
Section Summary .....	160
Using States to Switch Between Music Types .....	161
Defining Interactive Music Transitions .....	163
Authoring Transitions .....	164
Defining Transition Behavior .....	164
Transitioning from Ambient to Action Music .....	165
Transitioning from Action to Ambient Music .....	167
Section Summary .....	169
Music Summary .....	170
Music Additional Resources .....	172
8. Adventures in MIDI .....	173
Overview .....	174
Importing MIDI Files .....	175
Section Summary .....	179
Setting up Wwise Synth One .....	180
Adventures in Synthesis .....	181
Section Summary .....	184
Connecting MIDI & Sound .....	185
Importing the Individual MIDI Tracks .....	185

Music Segment MIDI Properties .....	187
Sound Object MIDI Properties .....	188
Section Summary .....	191
MIDI Summary .....	192
9. Mastering the Mix .....	193
Overview .....	194
Routing with Audio Busses .....	195
Routing with Auxiliary Busses .....	197
Using Auxiliary Sends .....	199
User-Defined Auxiliary Sends .....	200
Game-Defined Auxiliary Sends .....	202
States and Mix Snapshots .....	204
Auto-Ducking vs. Side-Chaining .....	206
Auto-Ducking .....	206
Side-Chaining .....	207
Mixing with RTPC .....	207
Using Effects in the Master-Mixer .....	209
Visualizing the Mixing Desk .....	210
Mixing Techniques for Attenuations .....	212
Mix Summary .....	213
Mixing Additional Resources: .....	214
10. HDR Audio Wwizardry .....	215
Overview .....	216
Implementing HDR Audio in Wwise .....	218
Setting up an HDR Audio Mix .....	219
Setting up the HDR Audio Dynamic Range Window .....	219
Enabling HDR Audio in the Master-Mixer Hierarchy .....	221
Setting up HDR Audio Dynamics Properties .....	221
Using HDR Audio in the Actor-Mixer Hierarchy .....	224
Enabling Envelope Tracking .....	225
Editing a Waveform Envelope .....	227
Enabling Source Normalization .....	228
Using Make-up Gain .....	229
Using the Voice Monitor to Understand HDR Audio .....	231
Opening the Voice Monitor View .....	231
Auditioning Sounds in the Soundcaster .....	232
Capturing Data from Wwise .....	233
HDR Audio Summary .....	236
11. Getting Set Up for Adventure .....	237
Overview .....	238
Work Unit Management .....	239
Establishing a Naming Convention Early .....	239
Logical Grouping of Work Units .....	239
Creating Work Units with Sharing in Mind .....	239
Batch Renaming .....	240

Grouping Objects in the Actor-Mixer Hierarchy .....	247
Setting the Audio Channel Configuration .....	249
Speakers vs Headphones Panning Rules .....	249
Creating Simulations with the Soundcaster .....	252
Project Settings .....	254
Project Settings - General Tab .....	254
Workgroup Plug-in Configuration .....	254
Audio File Locations .....	256
Default Conversion Settings .....	259
Defining the Sample Rate Automatic Detection Settings .....	259
Platform Manager .....	260
Obstruction and Occlusion .....	265
Setting Obstruction and Occlusion .....	267
Motion .....	270
Generating a Motion Source from an Existing Audio Signal .....	271
Generating Motion Using a Motion FX Object .....	272
Customizing Layouts .....	274
Layout Docking .....	275
View Docking .....	277
External Sources .....	279
SoundBanks and SoundBank Generation .....	280
Creating a SoundBank .....	280
Populating and Managing the Contents of SoundBanks .....	281
Excluding Elements from a SoundBank .....	282
A New Approach for SoundBank Management .....	283
Conversion Settings .....	284
The SoundBank Definition File .....	284
Using the Integrity Report .....	286
Using the File Packager .....	287
Downloadable Content (DLC) .....	288
Setup Summary .....	289
Setup Additional Resources .....	290
12. Workflow Optimization .....	291
Overview .....	292
Including or Excluding Platforms .....	293
Linking or Unlinking Properties .....	295
Rendering Effects .....	296
Understanding the Different Types of Profiling in Wwise .....	297
Connecting to the Game .....	298
Capturing Data Using the Profiler .....	299
Profiling Sound In-Game .....	300
Instance Limiting, Prioritization, and Virtual Voices .....	304
Playback Limits .....	305
Setting a Playback Limit per Game Object .....	305
Setting a Playback Limit on an Audio Bus .....	305

Global Playback Limit .....	306
Setting Playback Priority .....	307
Understanding Virtual Voice Behavior .....	309
Bridging the Game Engine Integration Gap .....	313
How Does SoundFrame Work? .....	313
Additional Game Engine Integration Techniques .....	314
Optimization Summary .....	315
Optimization Additional Resources .....	316
13. Closing .....	317
The Real Adventure Begins .....	318

---

# The Adventure Begins

While interactive audio continues to scale in complexity, certain implementation and workflow fundamentals remain the same. The Wwise sound engine and authoring application has been built from the ground up to support current and next gen audio integration in a flexible and dynamic productivity pipeline.

Building on the concepts explained in *Wwise Fundamentals*, the *Wwise Project Adventure* shows how to implement interactive and dynamic audio for a fictional game project. This comprehensive overview covers the process from start to finish, and the companion Wwise project allows you to take a hands-on approach to investigating the techniques. Whether you take a hands-on approach with the tutorial, or simply follow along in the accompanying project is up to you.

This book is intended for readers with some experience with digital audio in addition to basic experience with audio for games. Links to additional information to extend your knowledge into different areas of implementation are provided throughout this document. It does not replace the user documentation, but instead provides a way to access the details through specific examples and illustrations of techniques using the Wwise authoring application.

Our case study is a fantasy adventure game set deep in a forest, with danger lurking at every turn. An occasional dungeon gives the hero an opportunity to purge the scourge that lies within its darkened corridors. Armed with a trusty sword and the steadfast courage to persevere, the hero follows the path to riches that lie just ahead on the horizon.

Welcome to the Wwise Project Adventure!

---

# Chapter 1. Setting the Ambient Stage

Overview .....	2
Building the Foundation .....	3
Importing Audio Files .....	3
Setting Up a Looping Sound .....	6
Adding Detail to a Developing Soundscape .....	11
Randomizing Properties .....	11
Using the Blend Container to Combine Sounds .....	12
Using the Silence Plug-in for Weighted Randomization .....	13
Randomized Positioning of Sounds .....	15
Creating the Combined Daytime Forest Ambient Combination .....	17
Section Summary .....	17
Introduction of a Day and Night Cycle .....	18
Establishing a Game Parameter .....	18
Creating the Ambient System .....	19
Using the Blend Track Editor .....	20
Setting Container Order within Blend Tracks .....	22
Preparing the Event .....	25
Harnessing Event Actions .....	26
Creating Work Units .....	30
Section Summary .....	36
Sound Emitters in the Game World .....	37
Creating Attenuation ShareSets .....	37
Creating and Establishing Generalized Attenuations .....	38
Subscribing Sound Objects to an Attenuation .....	39
Using Spread as Part of an Attenuation .....	40
Using Low Pass Filter as Part of an Attenuation .....	43
Adjusting Cone Attenuation Properties .....	44
Section Summary .....	45
SoundSeed Air -Wind .....	46
SoundSeed Wind - Deflectors .....	48
SoundSeed Wind - Properties .....	49
SoundSeed Wind - RTPC .....	51
Ambient Summary .....	52
Ambient Additional Resources .....	53

## Overview

The stage is set before a single piece of geometry has been placed. The goal is simple: create a lush and non-repetitive ambient background to support hours of travel through the deep forest that lies ahead.

This chapter will take you through the process of:

- Importing audio files
- Setting up a looping background ambient audio source
- Randomizing playback of background ambient elements
- Randomizing the 3D position of elements
- Applying real-time parameter control to Blend Containers using Crossfade
- Using sound emitters as part of ambient design
- Creating Attenuation ShareSets
- Subscribing game objects to Attenuations
- Using SoundSeed Air - Wind

## Building the Foundation

### Importing Audio Files

We'll begin by importing a single audio file into a new Wwise project with the intention of using it as a looping ambient background. There are a few things to keep in mind about the process of bringing content into the project.

At a very basic level, Wwise distinguishes between the following two types of assets:

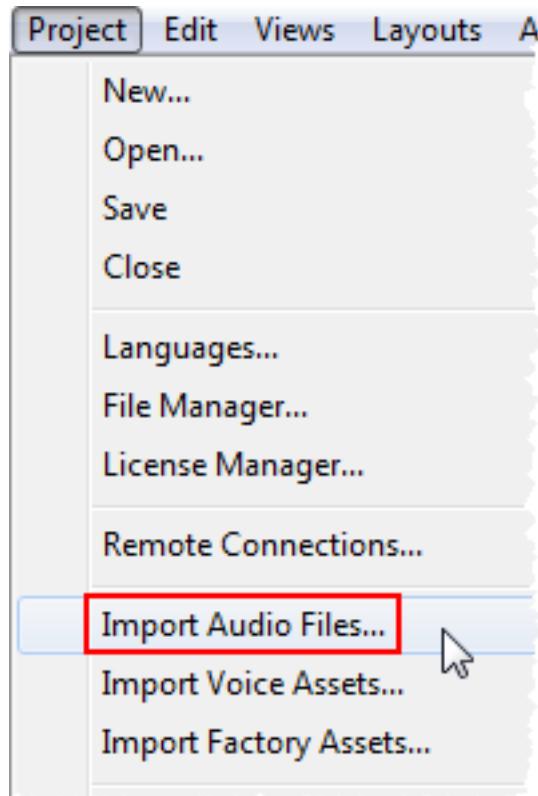
- Original assets imported into Wwise.
- Versions of these assets created for the various game platforms.

Wwise stores these two types of assets in different locations within the project folder, so they can be managed independently. Any imported assets are stored in the Originals project folder by default. Since these assets are usually shared by several people on your team, this folder can be located anywhere on your network and can easily be managed by your source control system.

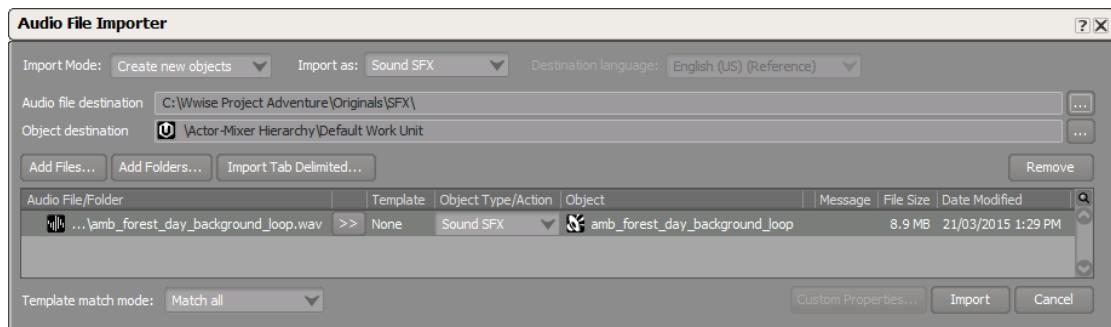
The import process includes the following operations:

- The original media (WAV) files are validated by Wwise before being copied into the project's Originals folder.
- Audio sources are created for the media files.
- Sound or music objects that contain audio sources are created in Wwise and are displayed in their respective hierarchies on the Audio tab of the Project Explorer.

Importing files into the Actor-Mixer Hierarchy using the Audio File Importer allows you to import individual files or groups of files by folder. You can import through the Project menu, or by right-clicking the work unit or container where you want the audio files.



Importing audio files from the Project menu



The Audio File Importer showing a single file to be imported as a Sound SFX

Alternatively, you can drag and drop audio files from folders directly into the Wwise hierarchy.

In the following example we'll import an ambient sound file for use as the basis of our ambient sound system. This file can be found in the project folder: /Originals/SFX/Ambient\_Background/ambient\_forest\_day\_background\_loop.wav.

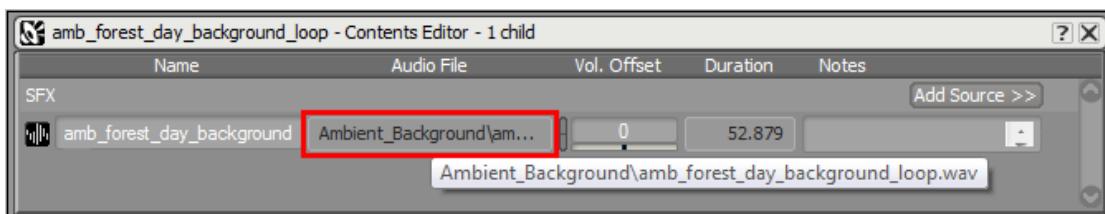


Imported sound file

#### Designer Note

Sound, music, or motion FX objects associated with sources that are not converted for the current platform appear blue in the Audio tab of the Project Explorer. Once a sound source has been converted to the target platform format, the text appears white. If a sound object has been created with no valid source, the text will appear red.

The imported file is added to a sound object, which includes a reference to the imported media file(s). When you load a sound object into the Property Editor, its sources are displayed in the Contents Editor.



Sound SFX object showing the source audio file location

#### Designer Note

A sound object can contain multiple sources including alternate versions, silence, plug-ins and different language versions for localization.

A sound object can be auditioned by using the Transport Control. After double-clicking the ambient\_forest\_day\_background\_loop sound object, click the play icon to play the object loaded in the Transport Control. Alternatively, pressing the spacebar plays and stops the sound object.



Transport Control showing the play icon

## Setting Up a Looping Sound

Now that a background forest day ambience audio file has been added to the project, it needs to be set to loop for it to continue playing. You have the option to loop the sound or motion FX object indefinitely, or to specify the number of times it will be looped.



Looping properties of a Sound SFX

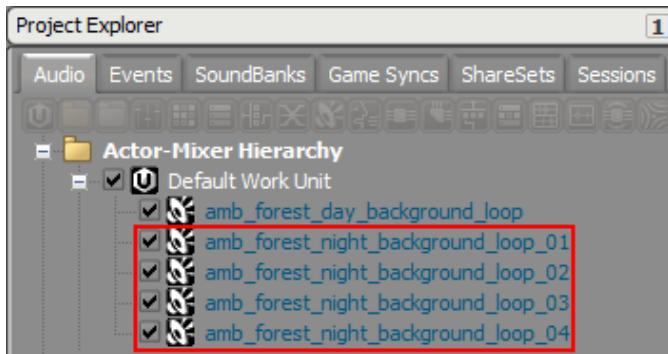
By default, double-clicking an object displays the object's properties. This action changes the focus of the Sound Property Editor to the object that was selected.

### Looping Multiple Sound Files

While playing a single sound file with no audible loop point continues to be a relevant technique, it has become equally common practice to assemble a loop through the randomized combination of individual sound files. For example, instead of playing back a single two minute sound file, you could instead choose to randomize the playback of four 30 second files.

Moving forward with this approach for the night forest ambient background, we'll start with a single ambient sound file that has been edited into four smaller files. These files will then be imported and parented within a random container. By randomizing the order in which these four files are played back, the eventual loop composition will be created differently each time.

Start first by importing the four individual files, either using the audio file importer (as shown above) or by dragging and dropping them into the default work unit from their source folder.



Multiple sound files imported

To accommodate the complex nature of audio within a game, different types of objects can exist within the Wwise project hierarchy. You can use a combination of the following object types to group your assets and build a structure for your project.

- Sound objects
- Motion FX objects
- Actor-Mixers
- Containers

The Actor-Mixer Hierarchy includes four container types:

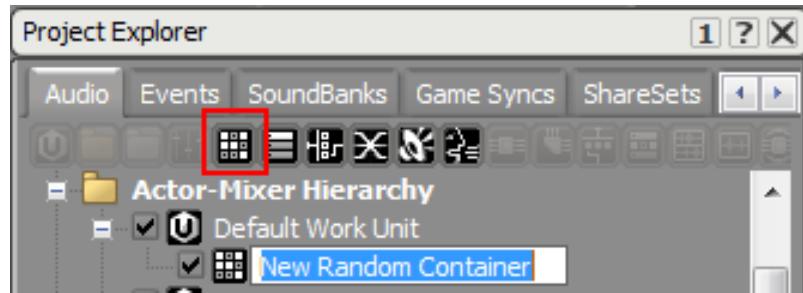
	Random Container: a group of one or more sounds, motion FX objects, and/or containers that are played back in a random order.
	Sequence Container: a group of one or more sounds, motion FX objects, and/or containers that are played according to a specific playlist.
	Switch Container: a group of one or more sounds, motion FX objects, and/or containers that are organized into a series of switches or states that correspond to the different alternatives that exist for a particular element in the game.
	Blend Container: a group of one or more sounds, motion FX objects, and/or containers that are played back simultaneously. The sounds and containers within the blend container can be grouped into blend tracks where sound properties are mapped to game parameter values using RTPCs. Crossfades can also be applied between the sounds within a blend track based on the value of a game parameter.

The next step in the process uses multiple audio files whose order will be randomized when played. For this example we will be creating a random container. A random container or other type of sound object can be created a few different ways:

- Using the Project Explorer toolbar

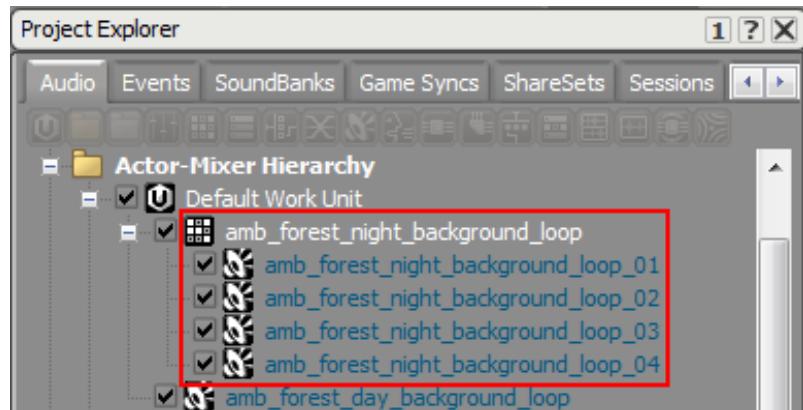
- Through the right-click contextual menu
- Using a shortcut key command (Help menu: Wwise Shortcuts Quick Reference Card)

By selecting the default work unit and then clicking the random container icon in the Project Explorer toolbar, a new random container is created within the selected work unit or sound object.



Creating a random container from the Project Explorer toolbar

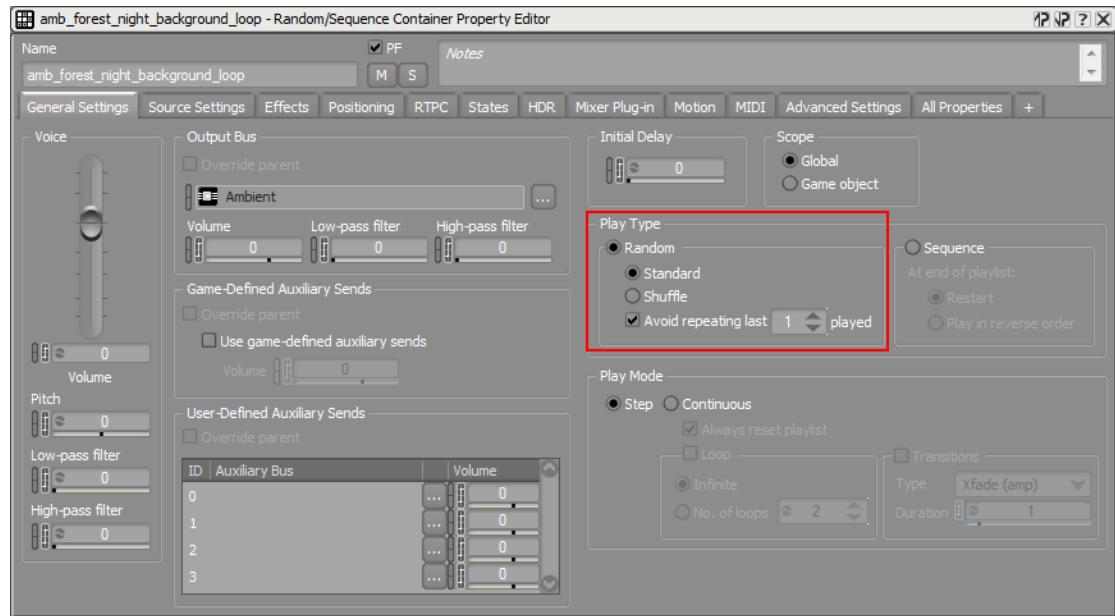
Once the random container has been created and named, sound objects can be dragged and dropped directly into it.



Random container containing multiple ambient sound files

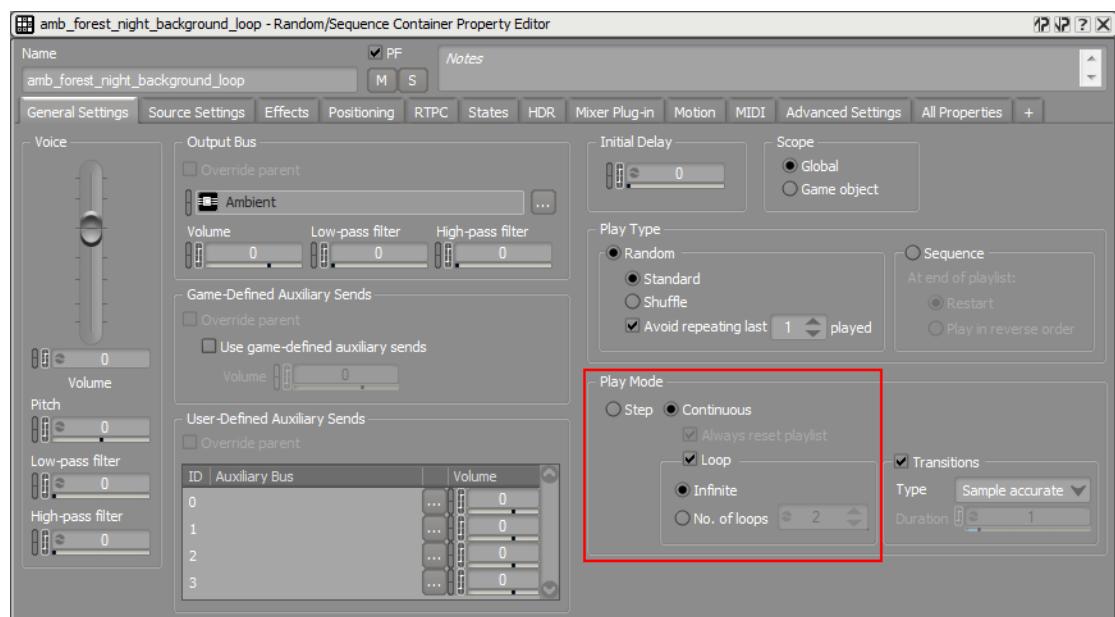
Now that the four smaller pieces of the background forest night ambience loop have been added to the container, their order of playback will be stepped through based on the properties set in the Random Container Property Editor. In Wwise, random can mean either a standard random selection, where each object within the container has an equal chance of being selected for playback, or a shuffle selection, where objects are removed from the selection pool after they have been played. You can double-click your random container to display its random properties.

## Setting the Ambient Stage



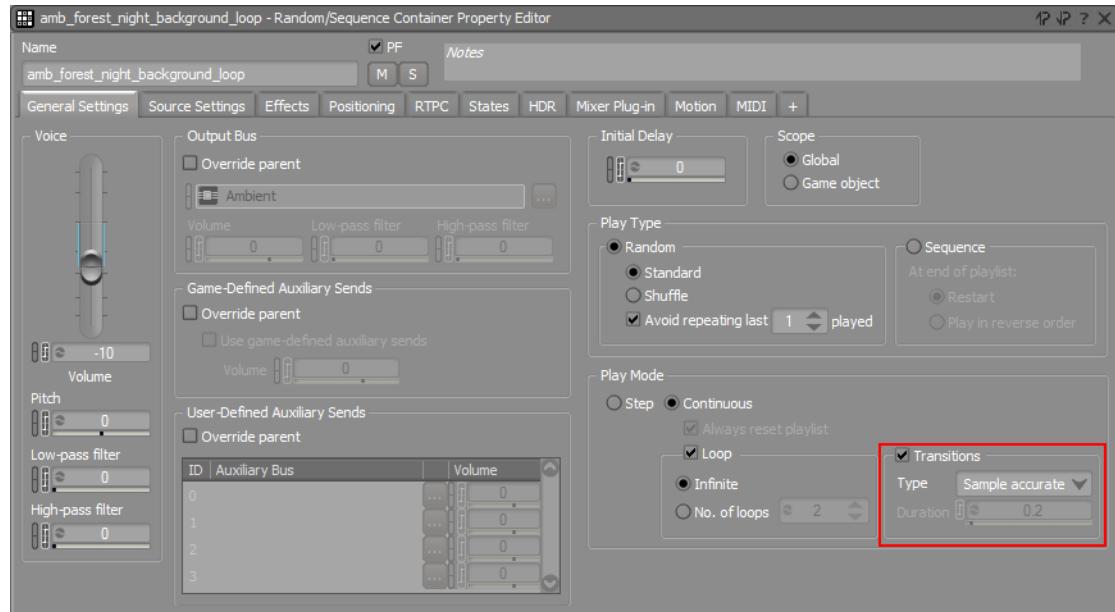
### Play type properties for a random container

The same looping functionality exists within the properties of a random container, where a group of audio files can be randomized within the container and looped in the same way.



### Looping sounds in a random container using continuous play mode

Once the files have been defined as random and are continuously looping, you can also control the way that sounds transition or flow from one sound to the next. Enabling the transition property as part of the looping play mode gives you additional control over the sound.



### Adding a transition type between sound files in a loop

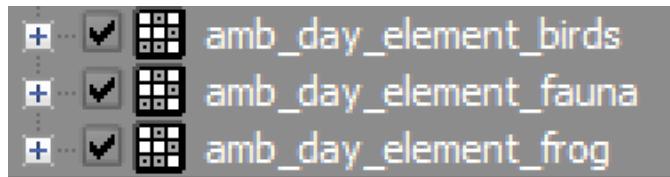
In this example, a sample accurate transition creates a seamless looping ambient file. Other transition options include: delay (silence), Crossfade using constant power, Crossfade using a constant amplitude, and trigger rate.

The power of this approach is in the diversity that can be gained by the recombination of multiple sound files at runtime, as opposed to using a single file of predetermined length. Additionally, by enabling control over the way that sounds transition, the diversity and complexity gained can deliver varied and non-repetitive loops. Whether you're dealing with mono, stereo, quad, or surround ambient backgrounds, each of these looping methodologies are supported.

In this section we've successfully begun the process of creating the ambient background system. With limited resources it's possible to convey a rich sense of environment using creatively designed and implemented loops as the foundation.

## Adding Detail to a Developing Soundscape

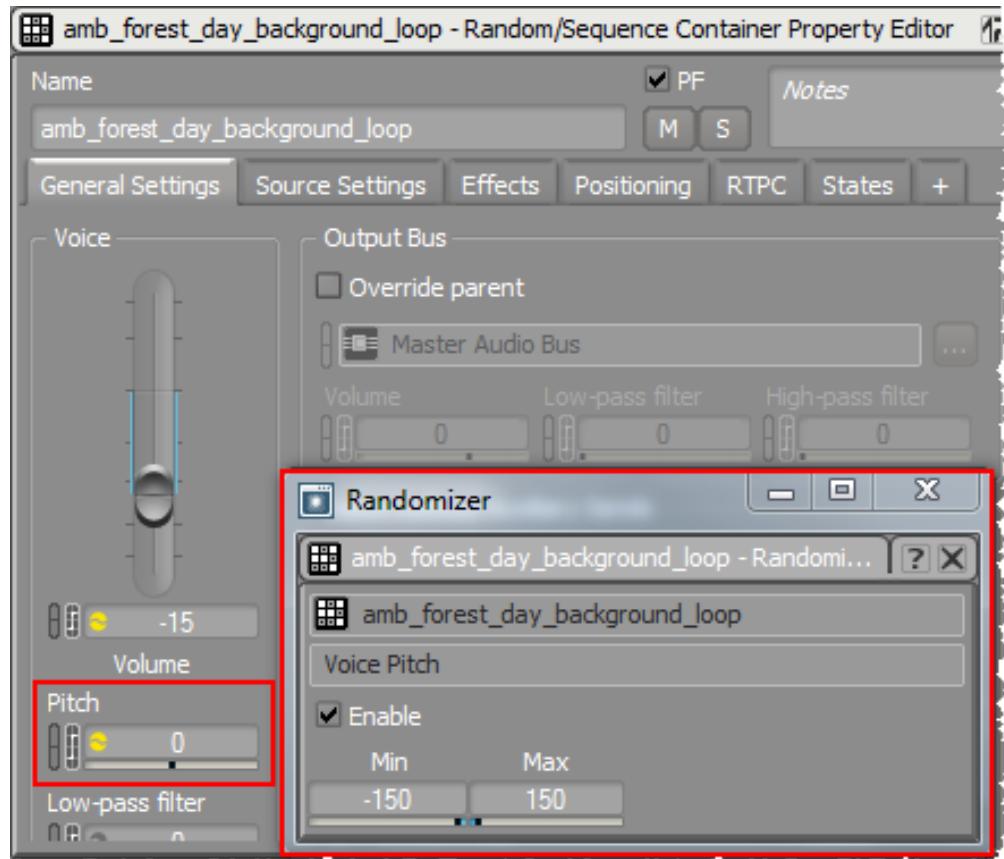
Now that the background ambience has been established, we'll add dynamics and variability using techniques not available with a single looping file or set of files. A forest in the daytime is comprised of many individual sounds, all sounding in unison. We've created several containers of daytime ambient elements such as birds, crickets, owl hoots, or wolf calls that we'll employ to continue building a more complex ambient background.



Random Containers of daytime ambient elements

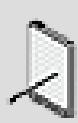
### Randomizing Properties

In addition to setting a static value for volume, pitch, and LPF in the Property Editor, you can also vary these settings using the Randomizer. The Randomizer allows you to specify a range of values that can be randomized for any of the given properties. Double-clicking the Randomizer indicator (seen in yellow, meaning enabled, on the left side of the Pitch and Volume fields of the following screenshot) in the Property Editor allows you to author a minimum and maximum amount of random variation. In this example, the pitch Randomizer has been enabled and a random variation of -150 to 150 cents has been specified.



Showing pitch randomization between -150 cents and +150 cents

When a Randomizer is enabled, it turns from gray to yellow to indicate its activation.



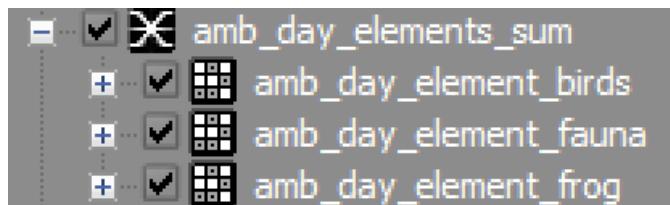
### Designer Note

Maximizing the randomness of each individual set of content will help you get the most out of each type of sound. While you may be able to dramatically vary the pitch of a single cricket chirp without it sounding strange, an animal call may not have the same ability to be pitched in the extreme without sounding abnormal.

### Using the Blend Container to Combine Sounds

Once the individual elements have been added to random containers, and you have spent some time diversifying their pitch, volume, and LPF properties, it's time to combine them into the sound of an animal chorus. One way to play multiple containers of individual elements is by using the blend container. A single blend container with no blend tracks will play all child objects simultaneously. This methodology allows for the summing of multiple sounds and helps keep things organized.

Additionally we'll want each container to loop its contents, so that the sound of each element can repeat over time along with the daytime ambient background.

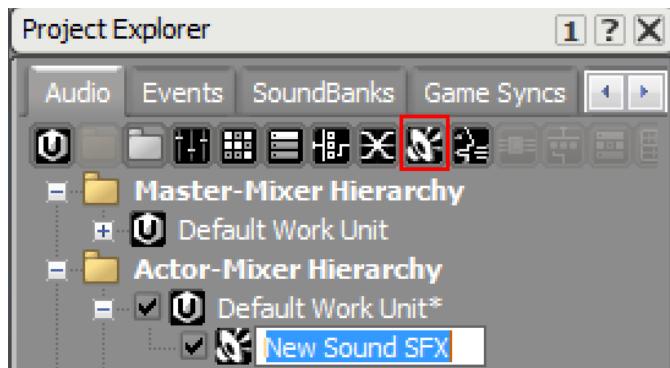


Grouping multiple ambient daytime elements within a blend container

### Using the Silence Plug-in for Weighted Randomization

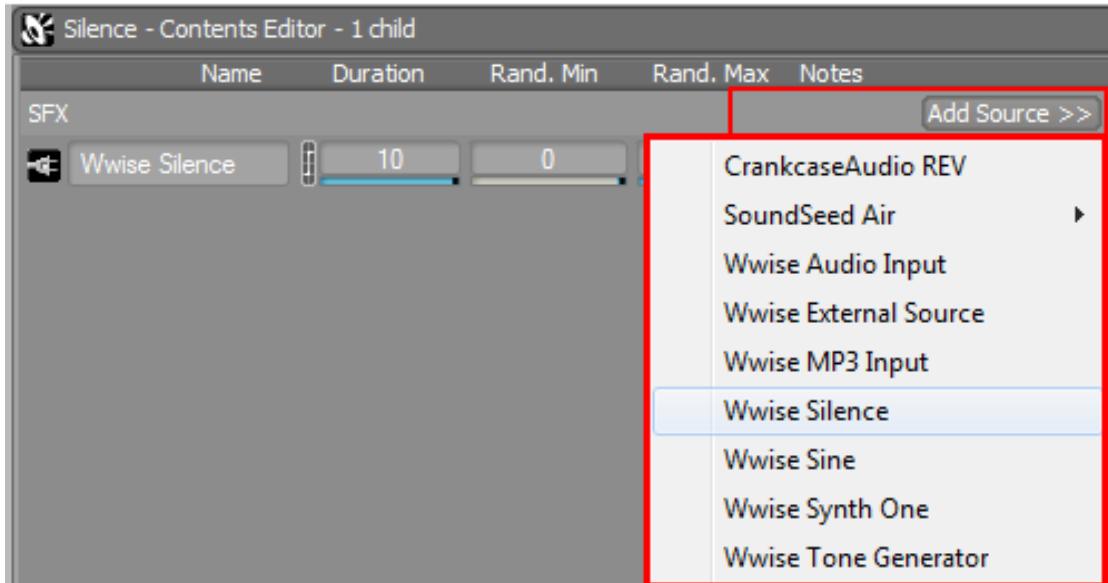
If we were to loop all of these sounds constantly, they would quickly become overwhelming. By introducing silence between each sound file, the sounds play less frequently and the daytime forest soundscape is given balance. The addition of a Wwise silence plug-in to a Sound SFX object allows for silence as part of the randomization between sound files.

Begin by adding a new Sound SFX object to the default work unit by clicking the Sound SFX icon in the Project Explorer toolbar, a new Sound SFX is created. Alternatively, Sound SFX can be created from the contextual menu or by using shortcut keys.



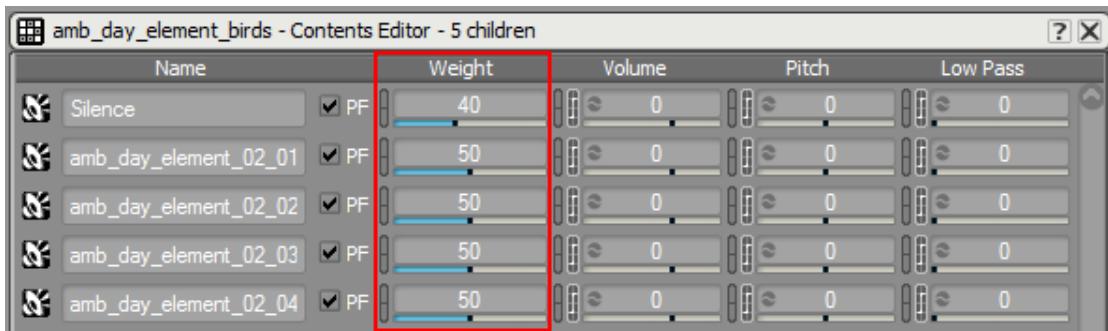
Creating a Sound SFX from the Project Explorer toolbar

After double-clicking to select the new Sound SFX, you can add silence using the Add Source menu in the Contents Editor. You can set a static duration in addition to a random minimum and maximum duration with the Wwise silence plug-in. The random minimum works in conjunction with the static delay by subtracting from the duration value.



### Creating a silence source plug-in for a sound object

By specifying the weight of each Sound SFX, you have control over the probability of playback for each audio file within the random container. By weighting the silence object appropriately, you can predictably add a comfortable sense of repetition during the playback of individual audio files. This ensures that birds don't chirp too often and allows for a randomized duration of silence between bird call variations.



### Controlling sound playback probability using the weight setting

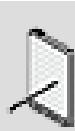


#### Designer Note

You don't have to worry about getting individual weights to add up to 100%; Wwise takes care of the math for this automatically based on the values you input.

### Randomized Positioning of Sounds

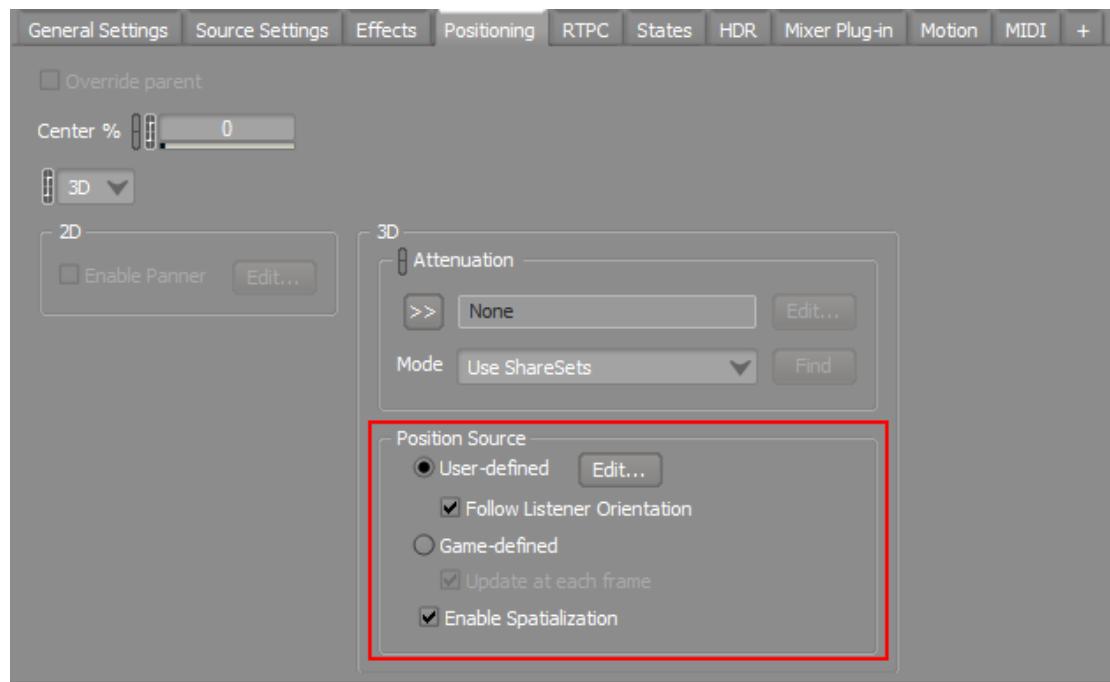
We now have the individual daytime nature sound types randomizing and summed together within a blend container. Next, we'll introduce user-defined 3D positioning in order to give the soundscape some life.



#### Designer Note

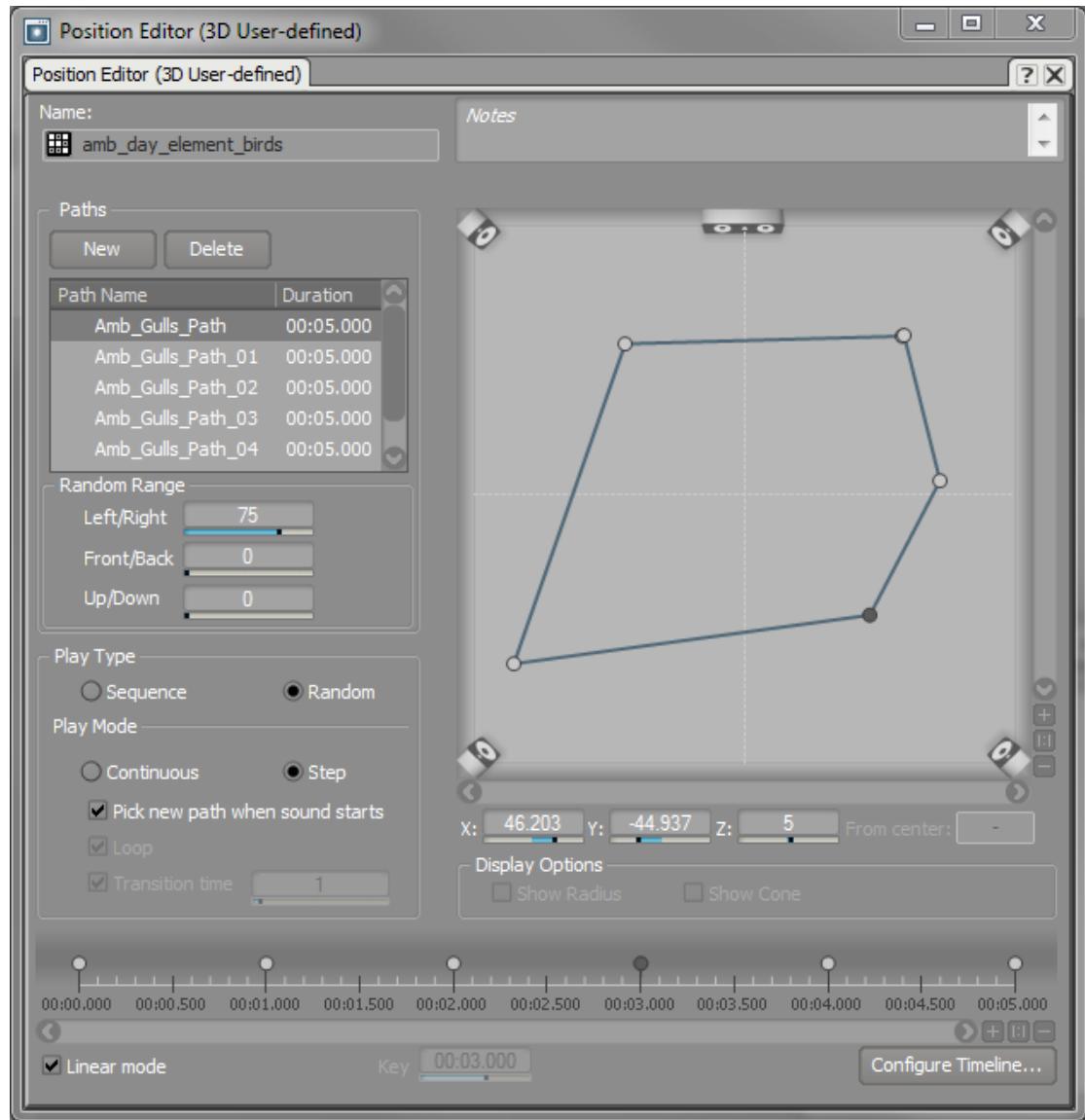
The Position Editor (3D User-defined) allows for the definition of spatial positioning for an object in a surround environment using animation paths. You can create many different versions of a path and then animate the sound or music along these paths. You can also define how the different paths will be played back. The paths you create will override the position and orientation in the game.

After selecting one of the containers of elements, you can make changes in the Positioning tab of the Property Editor. By changing the position source from the default Game-defined to User-defined, you can access the Position Editor where you can author to determine user-defined playback position of sounds. This technique is used to enable positional movement for static sounds and also to add positional randomness to the ambient elements that have been prepared.



Setting up user defined positioning for ambient elements

Multiple paths can be created that can be followed or randomized to simulate the effect of having many individual sounds coming from different directions.



Defining where sounds will play according to speaker placement using the Position Editor



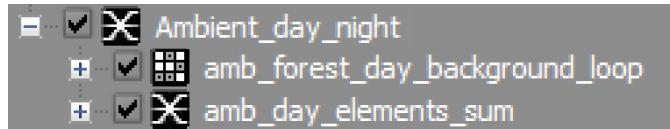
### Designer Note

How to work with the Position Editor is covered extensively in the [Video Tutorial - Positioning and Distance Attenuation](#) and in the documentation: [Wwise Help > Wwise Reference > Positioning > Position Editor \(3D User-defined\)](#).

The ambient day and night elements included in the accompanying project have been created using user-defined 3D positioning and can be referenced for further insight into this technique.

### **Creating the Combined Daytime Forest Ambient Combination**

Now that both the looping ambient background random container and the blend container of individual daytime elements have been created, it's time to combine them within a single blend container.



#### **Using a blend container to combine the ambient background loop and daytime elements**

The randomized daytime elements are now playing in conjunction with the looping ambient background to create a soundscape that is different every time. There is always room to add additional variation to sound files themselves, or through property randomization. Finding the right balance between content and implementation is a good way to optimize resources and reduce player listening fatigue. Combining techniques from both of these aspects of integration is oftentimes the best way to create a successful, non-repetitive playback system that fits the scope of the game.

### **Section Summary**

Throughout this section we have created the following sound objects:

- Random container with multiple daytime forest ambient background files looping via transition.
- Blend container containing ambient daytime elements summed with weighted silence and positioned randomly in 3D.

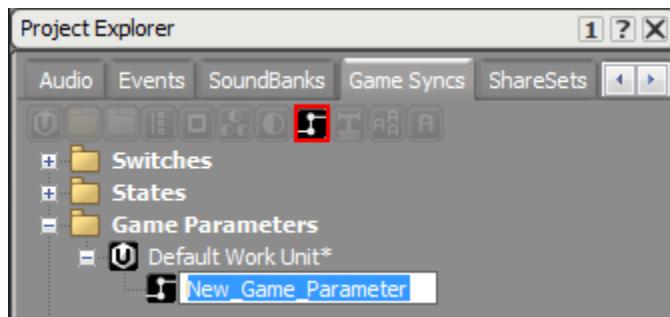
## Introduction of a Day and Night Cycle

After a day of travel through the forest, the sun begins to set and the adventure begins to turn ominous. Growing shadows on the trees transform a once idyllic path into a dark and foreboding passage. In order to effectively convey the passage of time, we expect the sounds to accurately reflect the visual environment and mirror the same sense of despair and unease.

### Establishing a Game Parameter

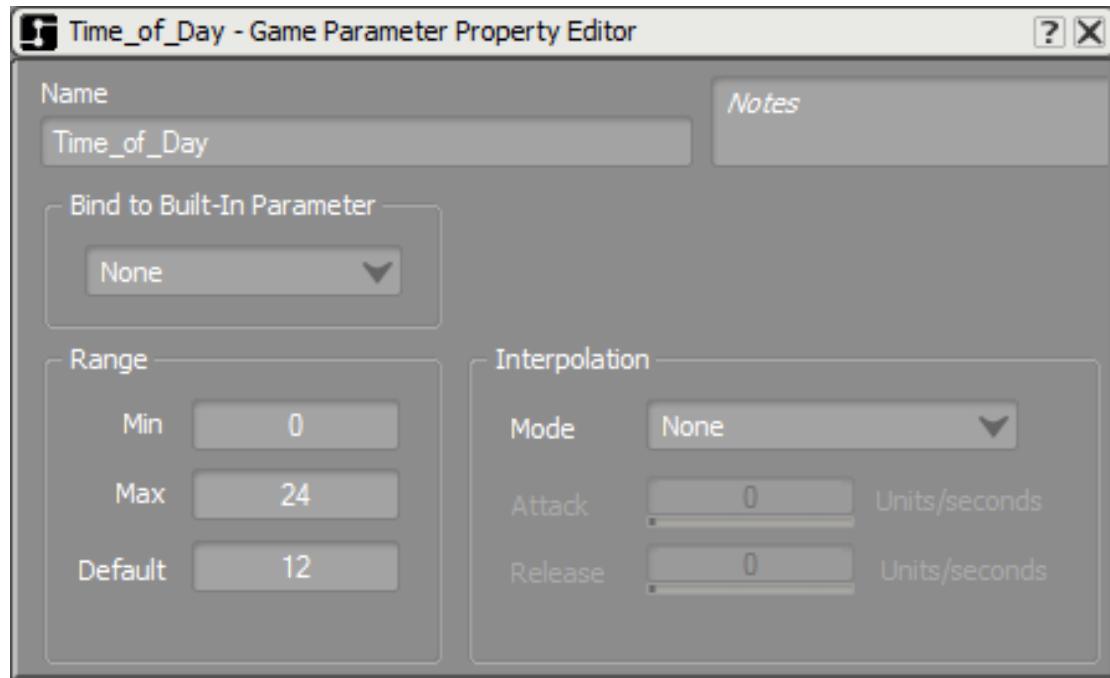
We'll start by having the programmer pass the same information from the game controlling the visual time of day as a game parameter to Wwise. Used in conjunction with a blend container that contains the ambient background and individual elements of the system, we can effectively control the ambient soundscape in time with the rest of the game.

To begin with, we'll create a new game parameter for Time\_of\_Day. Start by selecting the default work unit from the game syncs tab in the Project Explorer. This allows you to add a new game parameter from the Project Explorer toolbar by clicking on the game parameter icon. Alternatively, a game parameter can be created from the contextual menu for a game parameter work unit or by using shortcut keys.



Creating a game parameter from the Project Explorer toolbar

Once the game parameter has been named, the parameter range can be established in the Property Editor. We'll set up the Time\_of\_Day game parameter using values from 0-24, allowing for the parametrization of sounds according to a 24-hour clock with 0 representing 12 AM.

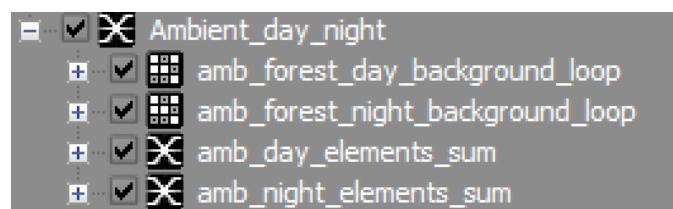


Setting the Time\_of\_Day game parameter values

## Creating the Ambient System

Now that we have a game parameter in place that we can use to drive the time of day simulation, we'll employ a blend container to play back both the looping background and randomized positional ambient sounds simultaneously. Once everything is set up, the forest ambience represents a fully dynamic day and night cycle that can be controlled and auditioned using the game parameter within the authoring application.

Using the techniques outlined in the previous section, we can move forward with the creation of a nighttime ambience that includes the same features implemented for the daytime. However, instead of putting the day and night containers within their own parent blend container, we'll be adding all of them to a single blend container called `Ambient_day_night` that will be used to govern the entire ambient forest system.

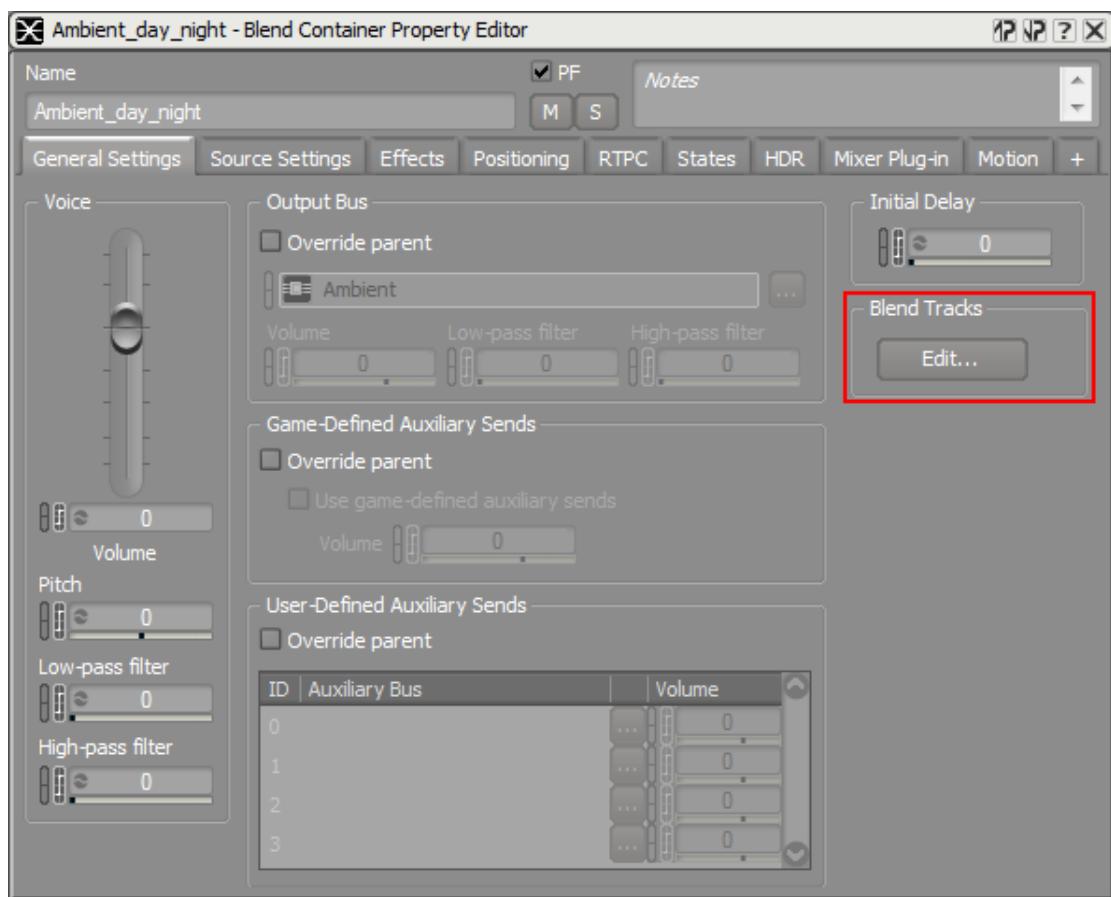


Adding all of the day and night containers to a single ambient blend container

## Using the Blend Track Editor

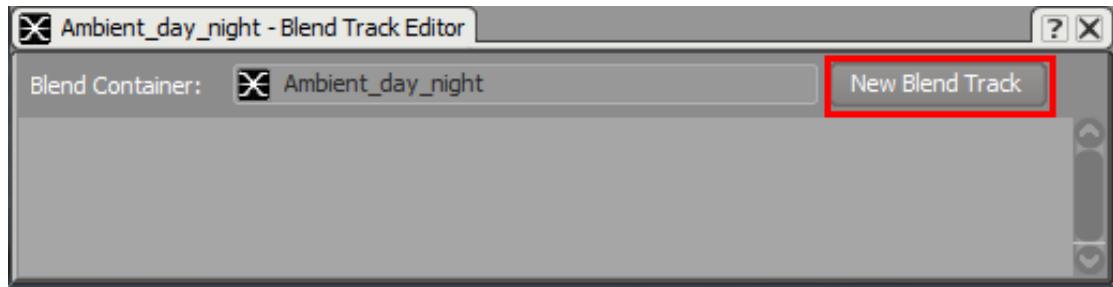
Now that everything is in the blend container and playing simultaneously, we need a way to isolate the day and night elements and integrate the Time\_of\_Day game parameter. A special function of the blend container is the Blend Track Editor, which is used to modify the contents of containers so that their properties can be controlled by RTPC curves and Crossfades.

Blend tracks can be accessed in the General Settings tab of the Blend Container Property Editor.



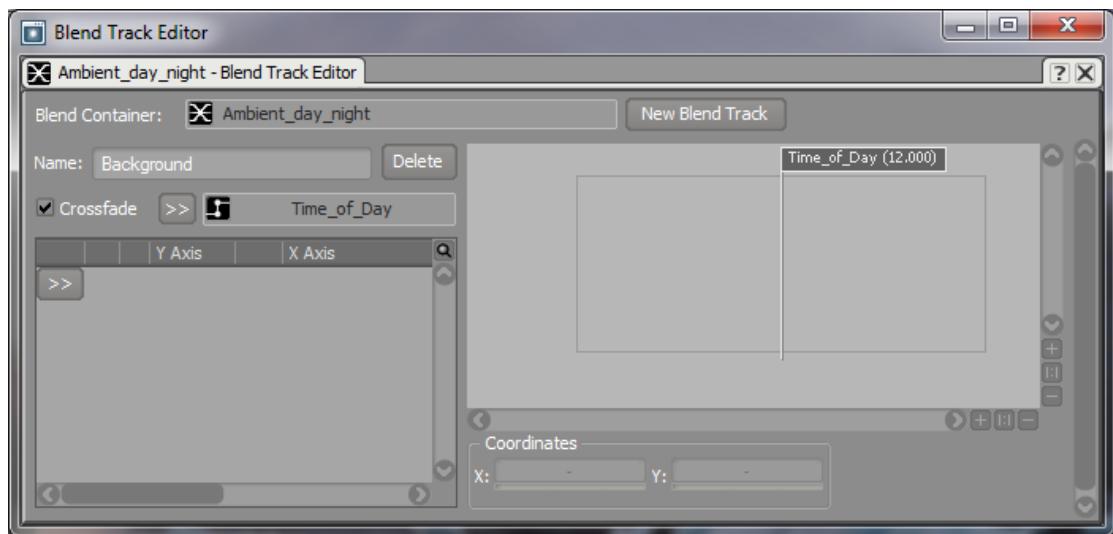
### Opening the Blend Track Editor from the Property Editor

The Blend Track Editor is made up of two main areas: one where blocks can be crossfaded, and another where RTPC curves can be added and modified using the graph view. These areas are enabled by adding a new blend track from the Blend Track Editor.



### Adding a new blend track to an empty blend container using the Blend Track Editor

Once a new blend track has been added, it can be named after the ambient background. Enabling the Crossfade option lets you select a game parameter which can be used to transition between containers or other sound objects on the blend track.



### Enabling the Crossfade on a new blend track

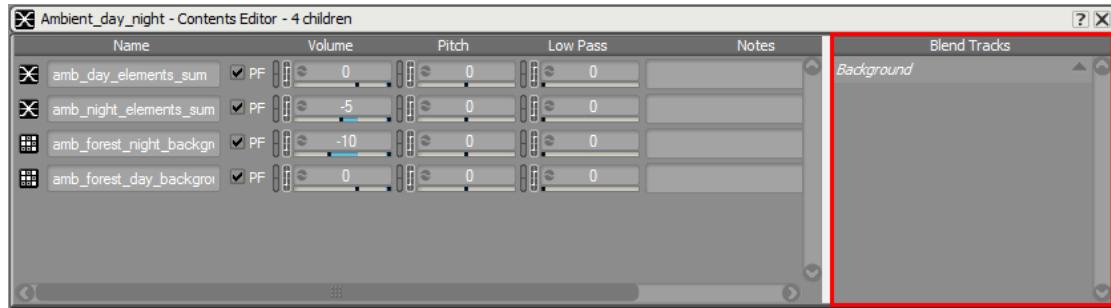
By adding a Crossfade to the Background blend track using the Time\_of\_Day game parameter, you can position containers according to values of the game parameter. The positioning of containers on the blend track determines the amount of overlap or Crossfade.



#### Designer Note

Changing between game parameters with different ranges can affect the range of the blend track's audio blocks. In this case, a decision must be made whether to Stretch the audio blocks or Preserve their position after the change in game parameter.

Now that a blend track has been added and the ability to crossfade has been established, containers can be assigned to blend tracks in the Contents Editor.



Blend tracks section of the Contents Editor

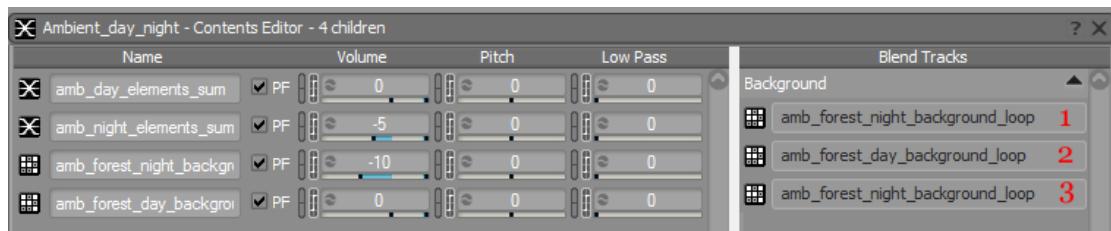
### Setting Container Order within Blend Tracks

Now you can drag and drop containers within the blend container into the blend tracks section. The containers must be dragged and dropped onto the specific name of the blend track. When setting up a blend track in the Contents Editor, it is important to position containers in the order you want them to show up, from left to right in the Blend Track Editor.

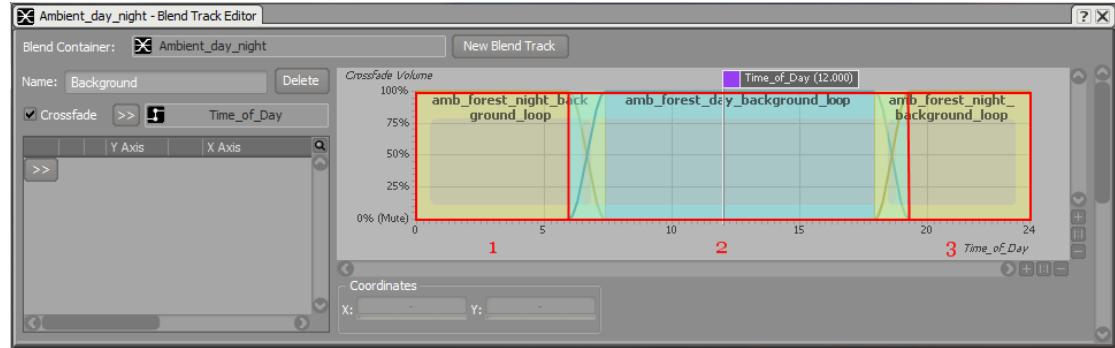
You can see how the order relates to positioning in this example:

#### Blend Track: Background

1. Random container: amb\_forest\_night\_background\_loop
2. Random container: amb\_forest\_day\_background\_loop
3. Random container: amb\_forest\_night\_background\_loop



Random Container order on a Blend Track represented in the Contents Editor

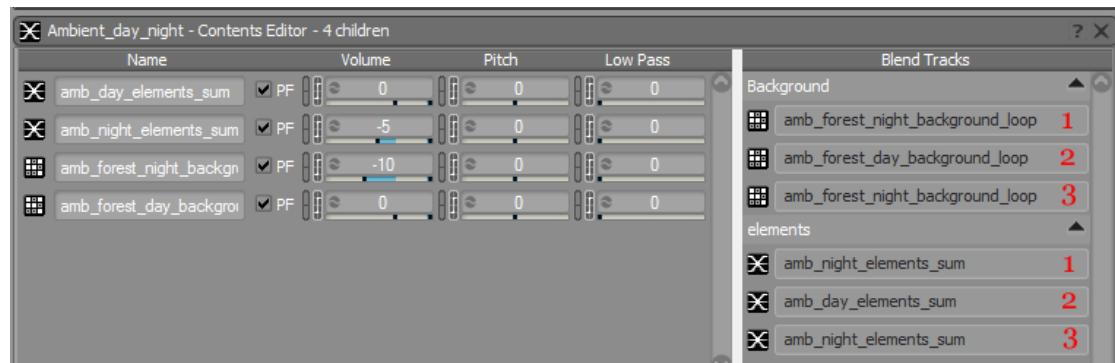


### Adding an additional blend track with ordered containers

To complete the full scope of our ambient system implementation, the ambient day and night elements need to be added to their own blend track.

#### Blend Track: Elements

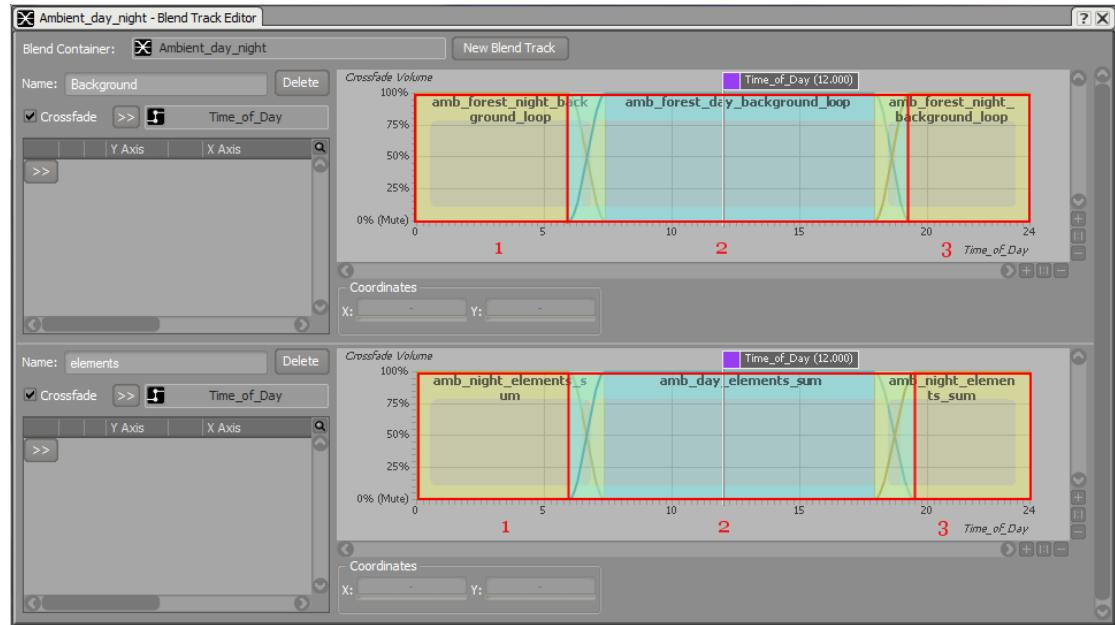
1. Random container: amb\_night\_elements\_sum
2. Random container: amb\_day\_elements\_sum
3. Random container: amb\_night\_elements\_sum



Container order on a Blend Track represented in the Contents Editor

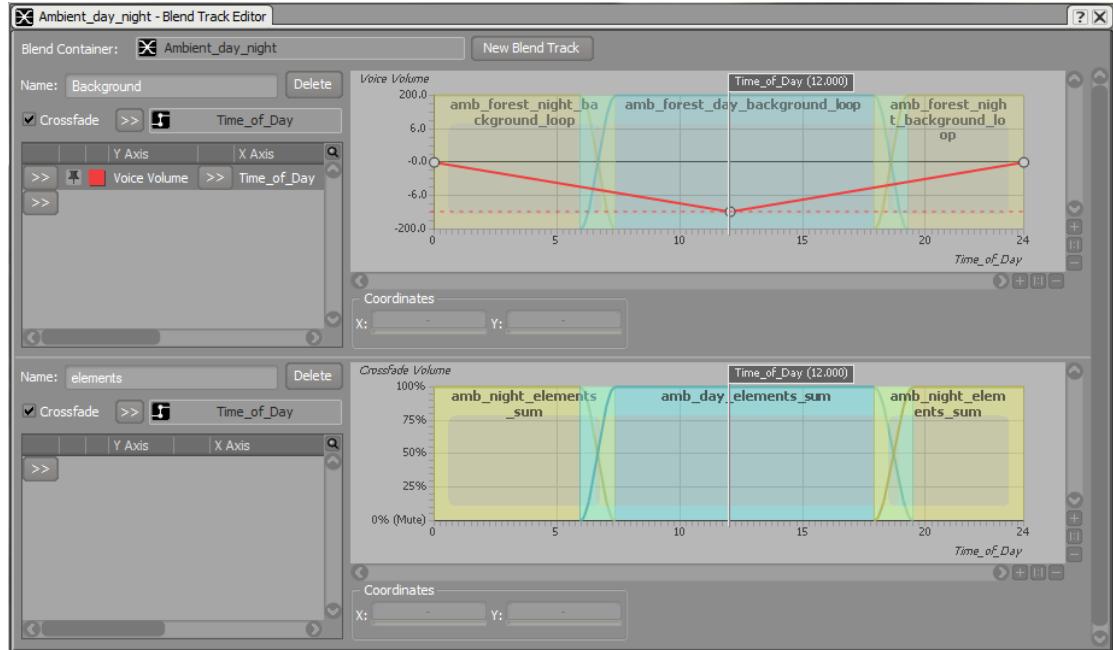
## Setting the Ambient Stage

---



### Adding an additional blend track with ordered containers

The Time\_of\_Day game parameter cursor can now be used to audition the Crossfade between containers for both background and ambient elements. You can also modify the points of these curves and Crossfade types in real time during playback. Furthermore, game parameters can be used in conjunction with blend tracks to affect the volume, pitch, and low pass filter parametrically.



An example of using a game parameter in conjunction with volume to affect a blend track

### Preparing the Event

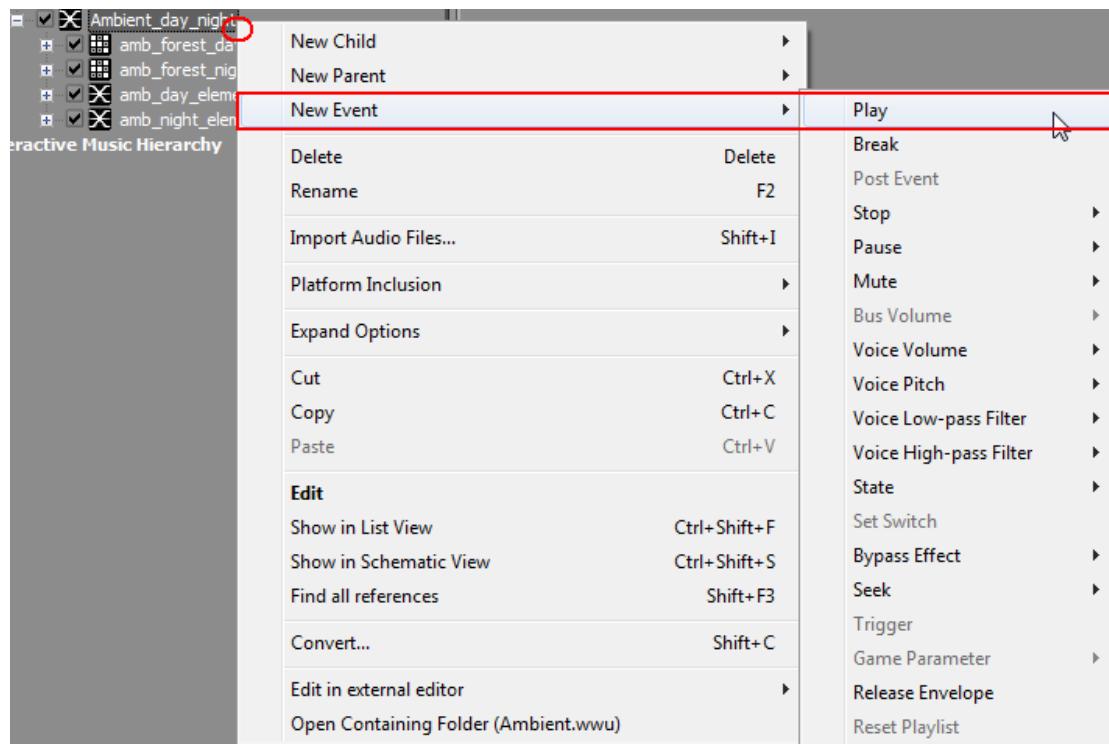
Now that the dynamic ambient system has been created, it's time to prepare the loop to be played and stopped by the game engine. Wwise uses events as sound, music, and dialogue instructions to the game. Events determine which sound, music, motion, or dialogue is played at any particular point in the game and can be used to control other aspects through the use of different actions (Play/Pause/Stop a Sound, change its Volume, Pitch or Low Pass Filter, and so on).

Events can also apply actions to the different structures within your project hierarchy. Each event can contain one action or a series of actions. The actions you select specify whether the Wwise objects will play, pause, stop, and so on.

To prepare our ambient system to be played and stopped by the game engine when the hero enters and exits the forest, two events need to be created:

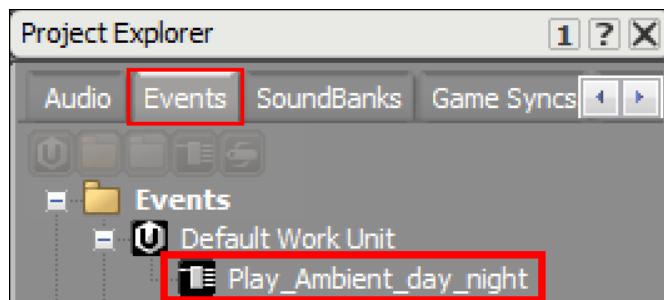
- Play\_Ambient\_Forest
- Stop\_Ambient\_Forest

We'll begin the process of creating a new play event by navigating to the Ambient\_day\_night blend container and selecting New Event > Play in the contextual menu.



Creating a new event using the contextual menu

This creates a new play event named after the container in the events section of the Project Explorer as part of the default work unit.



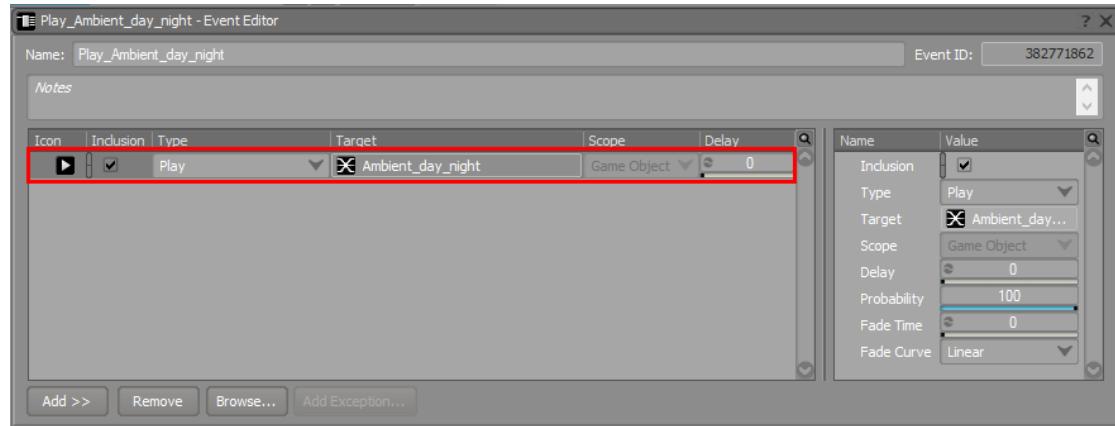
A newly created event in the events tab of the Project Explorer

### Harnessing Event Actions

When the event is created, the Event Editor is brought into focus to reflect the new event. The new event is created with the Ambient\_day\_night blend container added to it with a Play action type.

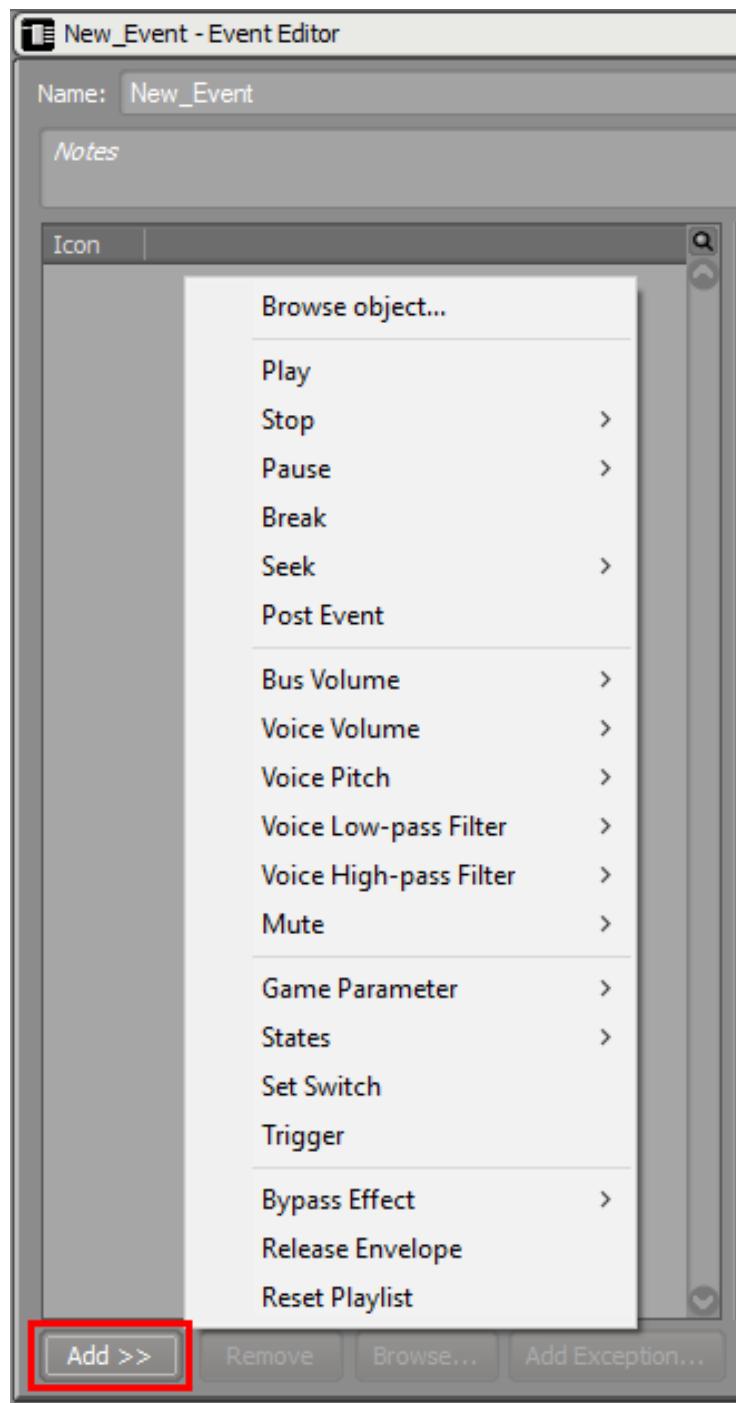
## Setting the Ambient Stage

---



An event created using the Play action in conjunction  
with the Ambient\_day\_night blend container

Event actions can also be removed or added to an event using the Action Selector.



Accessing additional event action options using the Selector

The Event Editor shows any Action Properties available for a given event action.

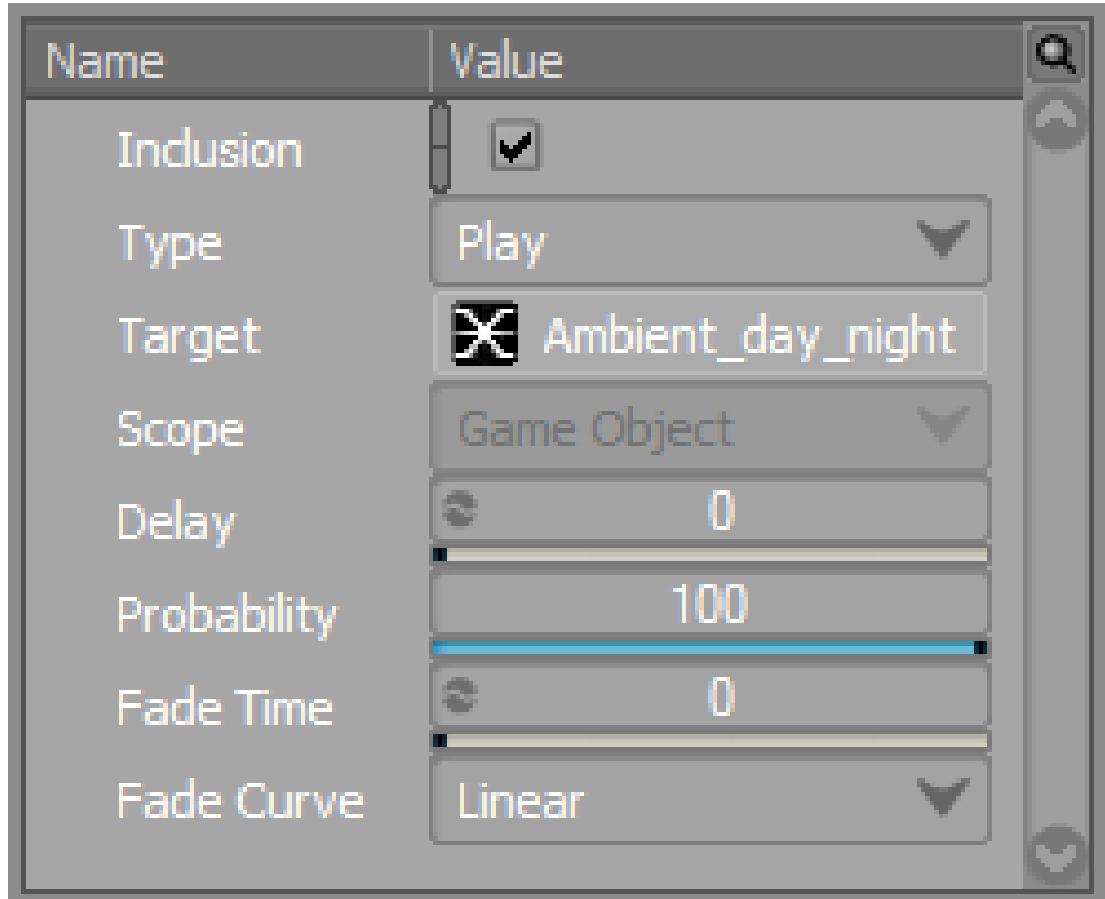
 **Designer Note**

Each event action has a different set of action properties that can be modified to produce the desired effects. See the help documentation for a full listing of event actions and properties:

[Wwise Help > Wwise Reference > Events > Event Editor](#)

[Wwise Fundamentals > Understanding Events > Action Events](#)

[Wwise Help > Interacting with the Game > Managing Events > Overview > Types of Event Actions](#)



Action properties available for the Play event action

The action properties that can be used to modify the Play event action include:

- Delay - The amount of time that goes by before the action is performed.
  - Default value: 0, Default Slider Range: 0 to 10, Input Range: 0 to 600, Units: Seconds.

- **Fade Time** - The amount of time it takes for the object to fade in, after any specified delay.
  - Default value: 0
  - Default Slider Range: 0 to 10
  - Input Range: 0 to 60
  - Units: Seconds
- **Fade Curve** - The curve shape that defines how the object fades in.
- **Probability** - Percentage of probability that an event action will play.
  - Default value: 100
  - Default Slider Range: 0 to 100
  - Input Range: 0 to 100
  - Units: Percentage

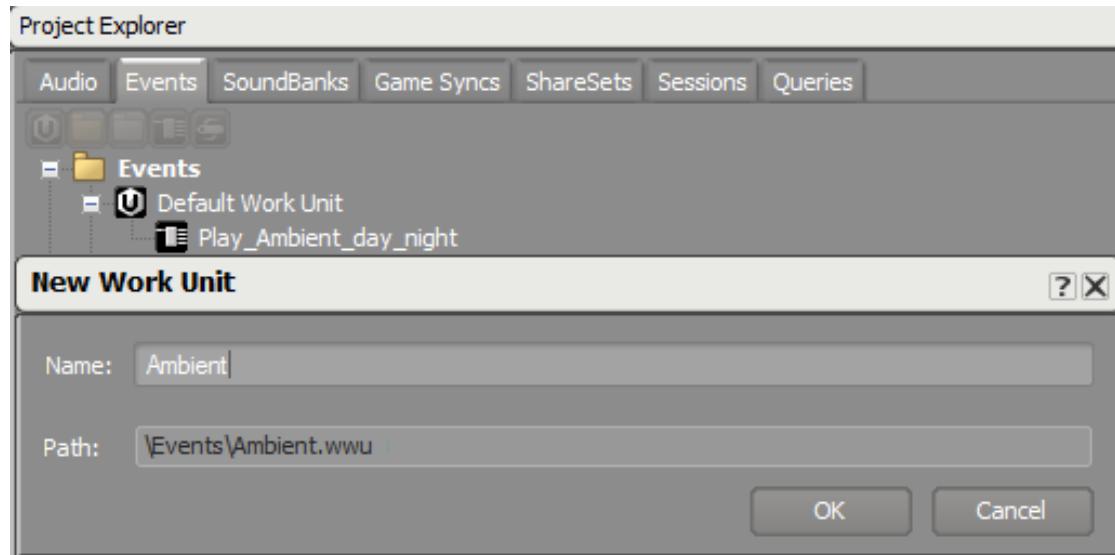
## **Creating Work Units**

We'll approach the creation of a stop event for the Ambient\_day\_night blend container a little bit differently.

First, we'll create a new work unit called Ambient in the events section of the Project Explorer to begin separating out different components of the project.

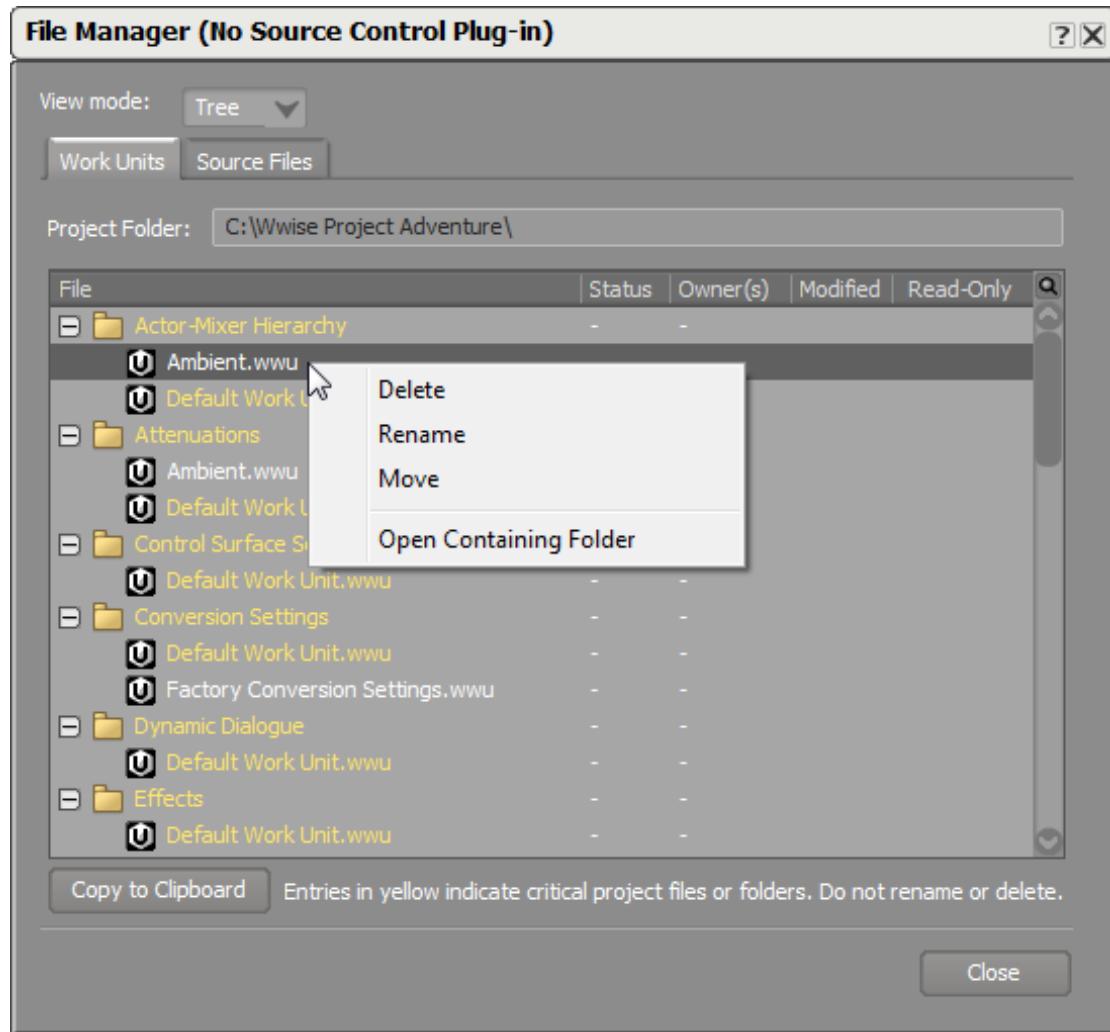
At the foundation of Wwise Workgroups is the work unit. Work units are distinct XML files that contain information related to a particular section or element within your project. These work units can help you organize and manage the different elements within a project. Hierarchies of nested work units can now be created and organized in physical folders and subfolders.

A new work unit can be created through the Project Explorer toolbar by clicking on the work unit icon. Alternatively, a work unit can be created from the contextual menu.



### Creating and naming a new work unit

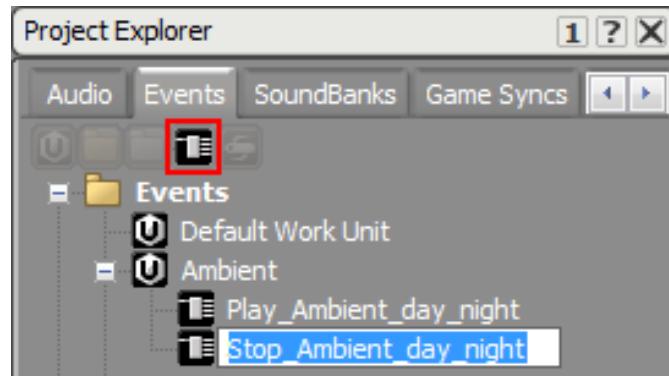
After a work unit is created, you can delete, move and rename it in the Project Explorer. You can also make these changes to work units and source files from the File Manager. The File Manager can be accessed through the Project menu.



### Using the File Manager to delete, rename, or move work units and source files

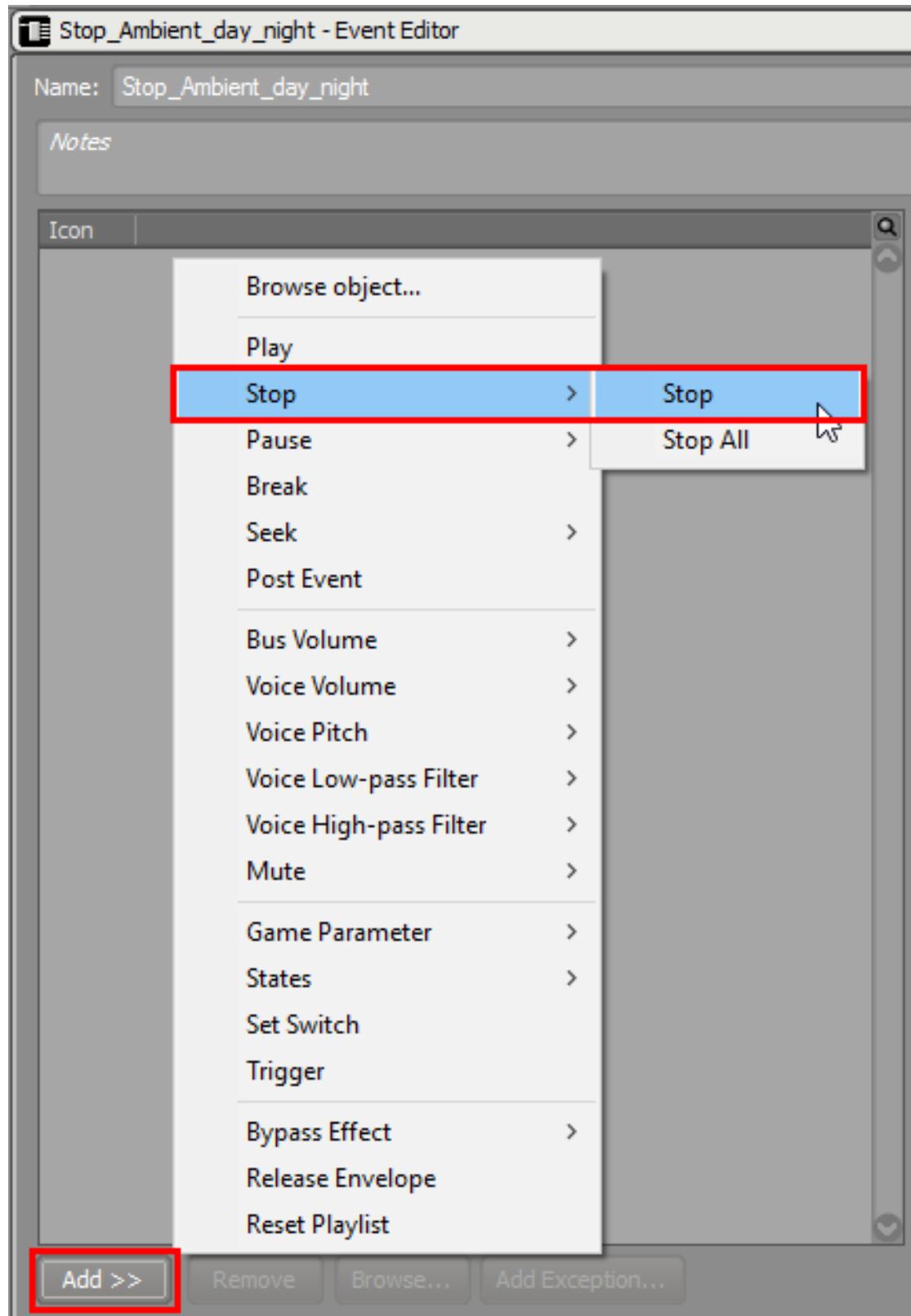
Now that our Ambient event work unit has been created, we can move our previously created Play\_Ambient\_day\_night event by dragging and dropping it from the Default Work Unit.

We can now add a new event called Stop\_Ambient\_day\_night through the Project Explorer toolbar by clicking on the event icon. Alternatively, an event can be created from the contextual menu or by using shortcut keys.



Creating a new event in the Ambient work unit

Double-clicking the newly created Stop\_Ambient\_day\_night event will open it in the Event Editor. The next step is to add a Stop action by using the event action Selector and selecting Stop>Stop from the selection of event actions.

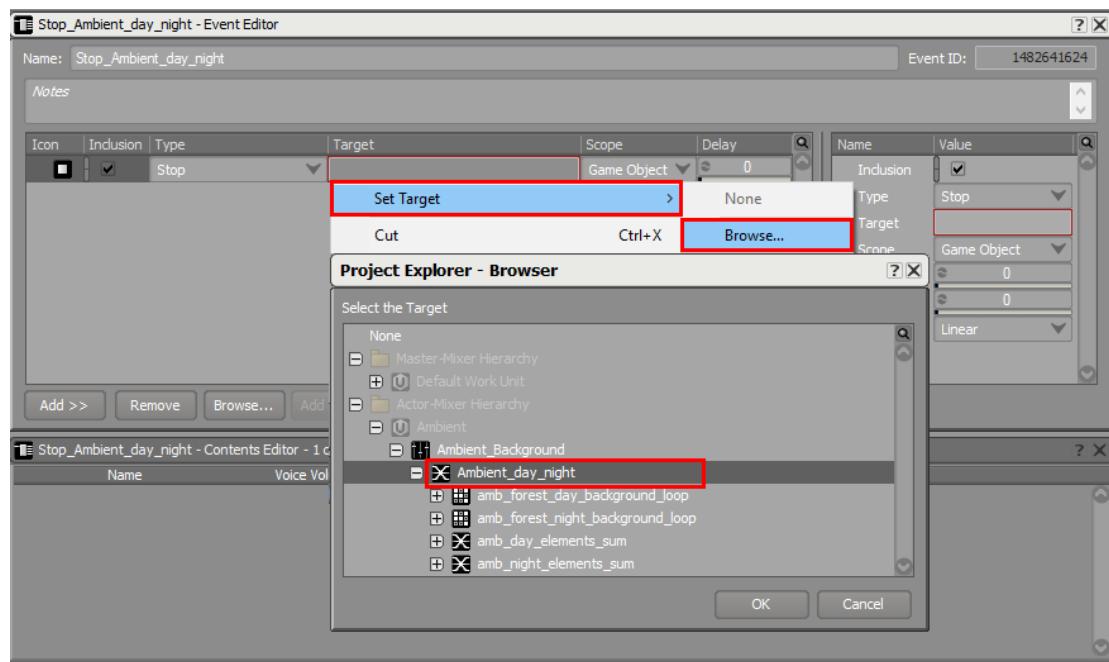


Adding a Stop event action to a new event

Once an event action has been added to the event, it needs to be assigned an object to which the action will be applied or it will show as Missing in the Objects column. When the event is requested by the game, the specified action needs to be applied to one of the four object types:

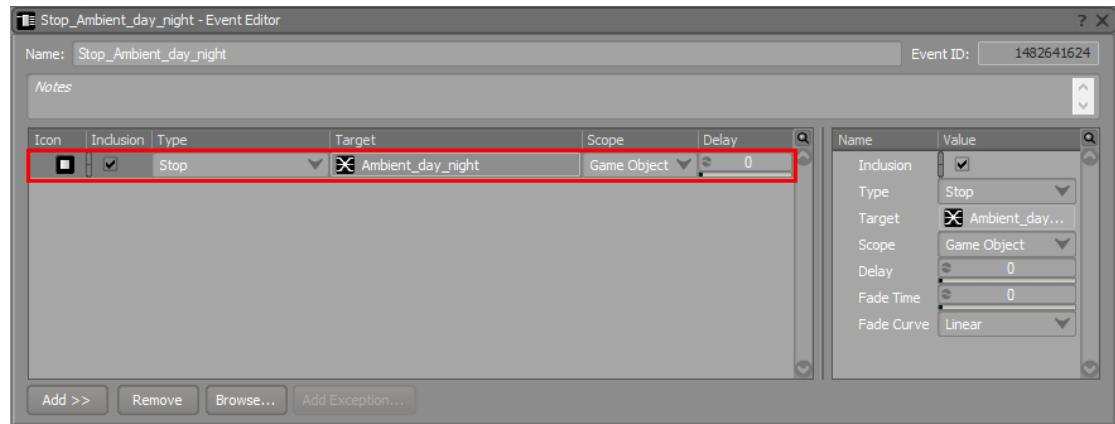
- Sound objects
- Motion FX objects
- Actor-Mixers
- Containers

These objects can be navigated to using the Project Explorer - Browser (prompted by clicking the Browse button) or dragged and dropped from the Project Explorer over text in the object column on the event action in the Event Editor.



Navigating to the `Ambient_day_night` blend container using the Project Explorer - Browser

Once an object has been added, its location in the hierarchy is updated and the action properties are available for adjustment.



An event created using the Stop action in conjunction  
with the Ambient\_day\_night blend container

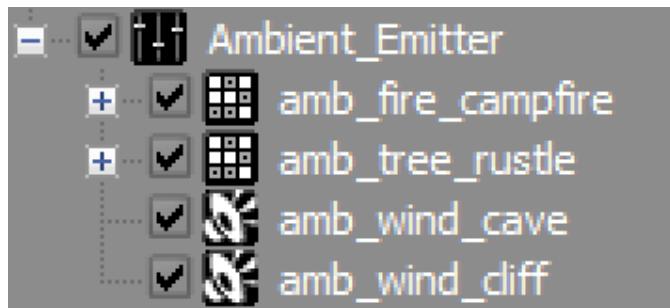
After events have been created, they can be integrated into the game engine so that they are called at the appropriate times in the game. Because of the abstract nature of events, their components can continue to be fine-tuned within the authoring application throughout development without having to re-integrate them into the game engine.

### Section Summary

Adding blend tracks gives you the power to order and Crossfade objects and to dynamically modify them with game parameters within the blend container. The addition of events and event actions determines which sound, music, motion, or dialogue is played and how it will play at any particular point in the game. As the sun sets after a long day of travel, and the sound of birds is replaced with crickets and the occasional wolf howl, the benefits of this dynamic ambience can really be appreciated.

## Sound Emitters in the Game World

With a looping background in place that changes based on the time of day, we can focus on adding positional sounds at specific locations in the forest to help fill in the environment, things like rivers, waterfalls, fire pits, and birds in trees. We'll begin by setting up several random containers that can be used to add further definition to the world.



Prepared sounds for use as 3D Emitters

### Creating Attenuation ShareSets

The fundamental difference between the previous ambient background implementation and the explicit placement of a sound in 3D space is in the way sound changes based on the listener position. A listener is a virtual microphone in the game that helps assign sounds to particular speakers to simulate a 3D environment. Attenuation ShareSets can be authored to control the distance based playback of sounds in relation to the listener.



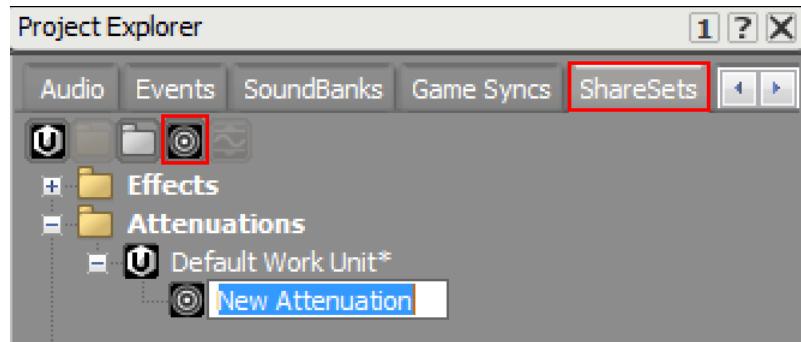
#### Designer Note

The Attenuation Editor allows you to define the distance-based attenuation properties for a particular object. By creating a series of curves to define the relationship between specific Wwise properties, such as volume and low pass filter, and the distance between the emitting source and the listener, you can simulate sophisticated distance-based attenuation for the sounds, music, and motion in-game. You can further refine attenuations using sound cones, which simulate attenuation based on the orientation of the game object in relation to the listener.

Game objects are the central concept in Wwise because every event triggered in the sound engine is associated with a game object. A game object generally represents a particular object or element in your game that can emit a sound, including characters, weapons, ambient objects such as torches, and so on.

## Creating and Establishing Generalized Attenuations

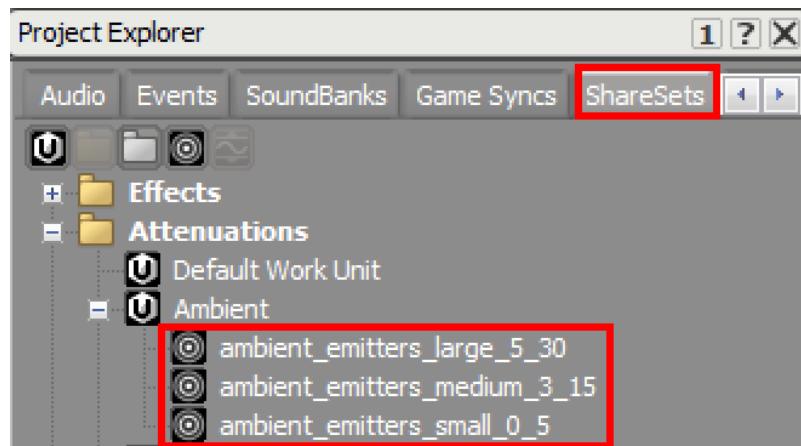
By selecting the default work unit from the ShareSets tab in the Project Explorer, a new attenuation can be added from the Project Explorer toolbar by clicking on the game parameter icon. Alternatively, an attenuation can be created from the contextual menu for an attenuation work unit or by using shortcut keys.



Adding a new attenuation ShareSet

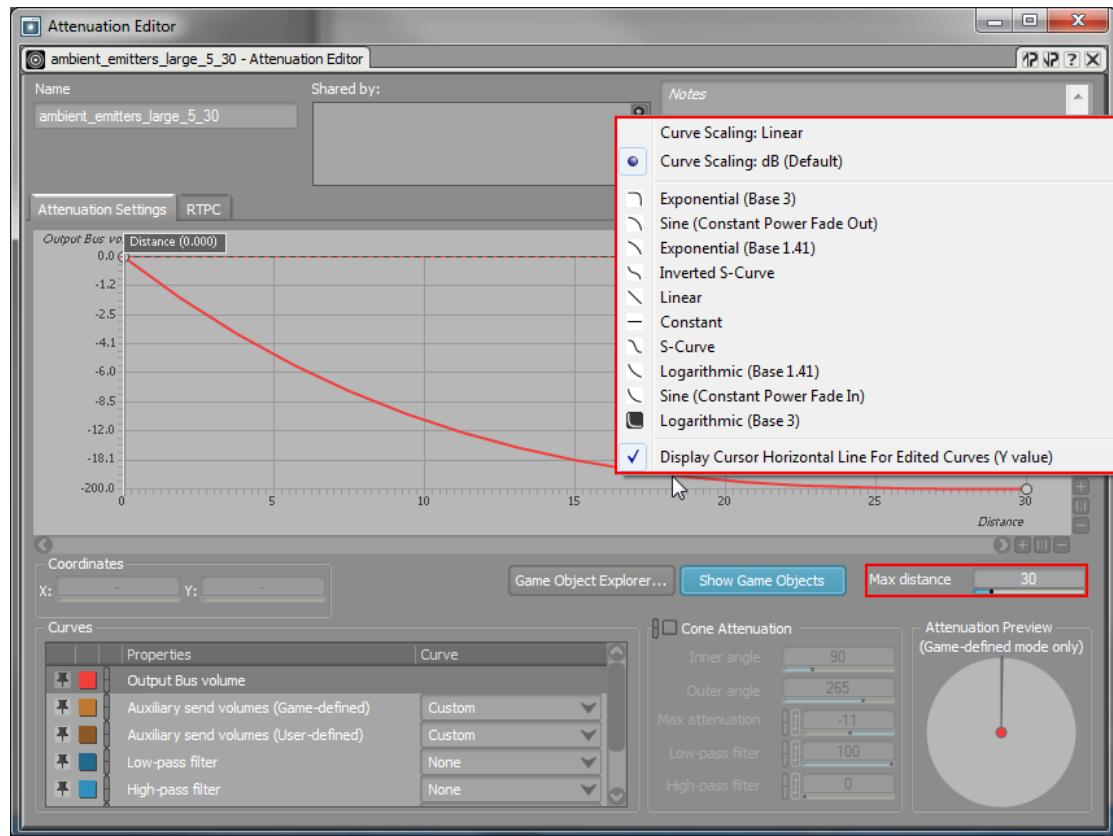
Creating generalized attenuations that can be used in many scenarios is a good place to start. Adding further attenuations that address specific distance-based problems or special effects can be created on an as-needed basis throughout the course of the project.

We'll begin by defining generalized attenuations for Ambient Emitters: Large, Medium, and Small. We'll also imbed some information regarding distance, within the Attenuation name, in order to give some indication of what their intended use is. When working with a naming standard throughout your project, it is important to try and adopt some conventions that will be clear to people who may be working with sound objects, Events, Game Syncs, or ShareSets you have created.



Project Explorer ShareSets showing a set of general attenuations

Once created, you can control the amount of reverb, LPF and spread by adjusting the curve properties. Each curve defines properties based on the maximum distance set for the attenuation. The maximum distance is distance from the emitting source where the sound reaches its lowest level. The attenuation of the object remains constant beyond the max distance.

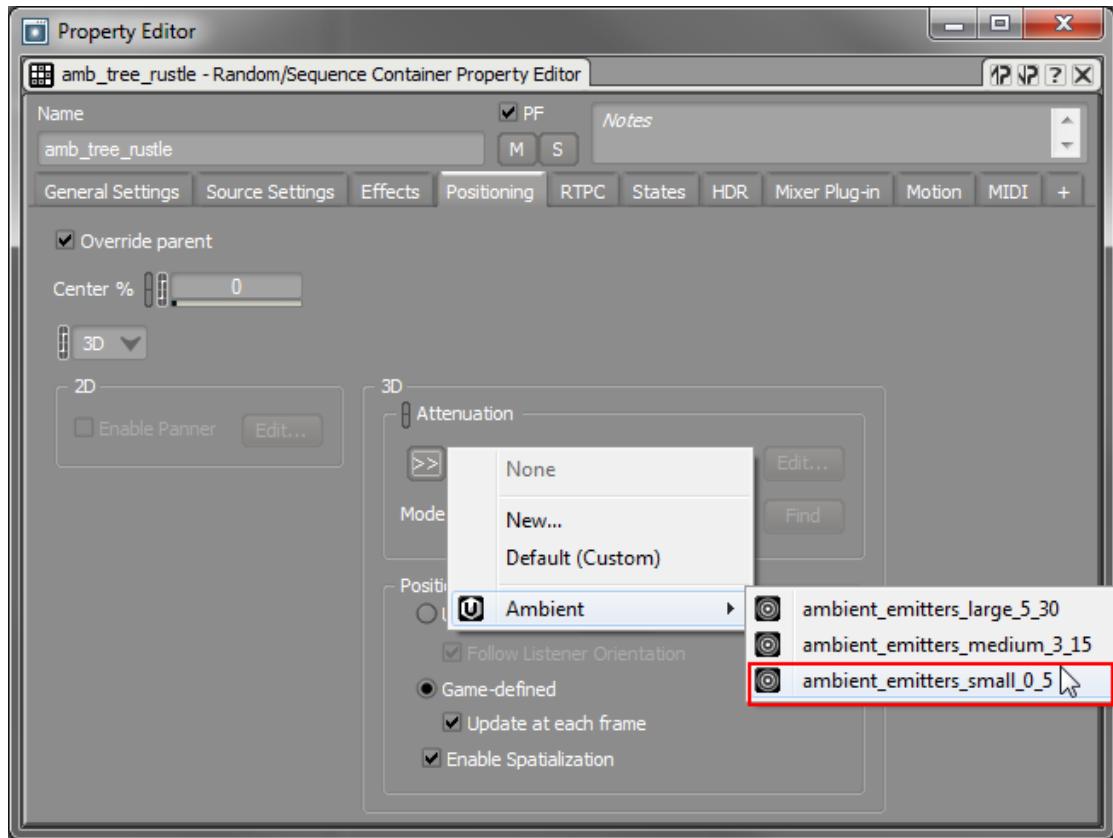


Defining curve properties, max distance, and the attenuation feature set

### Subscribing Sound Objects to an Attenuation

Now that we have established a few generalized attenuations for ambient emitters, any level of the Actor-Mixer Hierarchy can subscribe to them. Remember that any children of the parent sound object will inherit the subscription unless overridden. In the first example, we'll subscribe the `ambient_tree_rustle` sound object to the `ambient_emitters_small_0_5` attenuation so that we only hear trees when the listener is within five game units of the emitter.

We begin by overriding the parent object (if any) and changing the sound to 3D in the positioning tab. Clicking the Selector button ( $>>$ ) allows you to navigate the attenuation ShareSet hierarchy and select the attenuation you want to subscribe to.

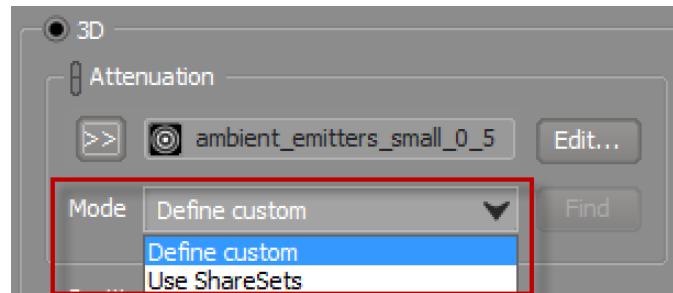


### Subscribing a sound object to an attenuation

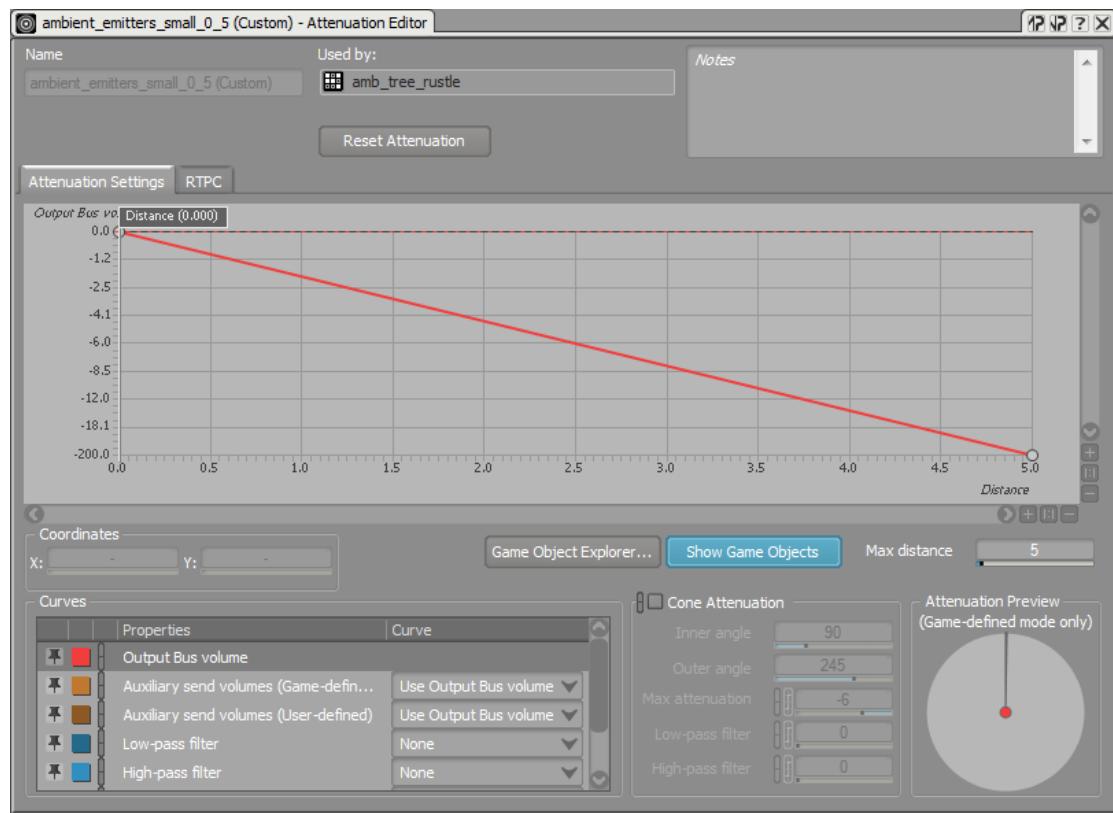
Now, when the ambient\_tree\_rustle sound object is played, it will obey the definition of the ambient\_emitters\_small\_0\_5 attenuation and you will hear the soft rustling of leaves when the listener is close to the emitter. If the properties of the attenuation change, any sound object that is subscribed to it will also change.

### Using Spread as Part of an Attenuation

In the next example, we'll take the same ambient\_emitters\_small\_0\_5 attenuation ShareSet and subscribe the ambient\_fire\_campfire sound object to it. Additionally, we will set the mode to Define Custom in order to use the same ambient\_emitters\_small\_0\_5 attenuation, but also Spread the sound at close distance for that particular object.



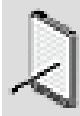
Defining a custom attenuation mode



Attenuation editor displaying custom properties

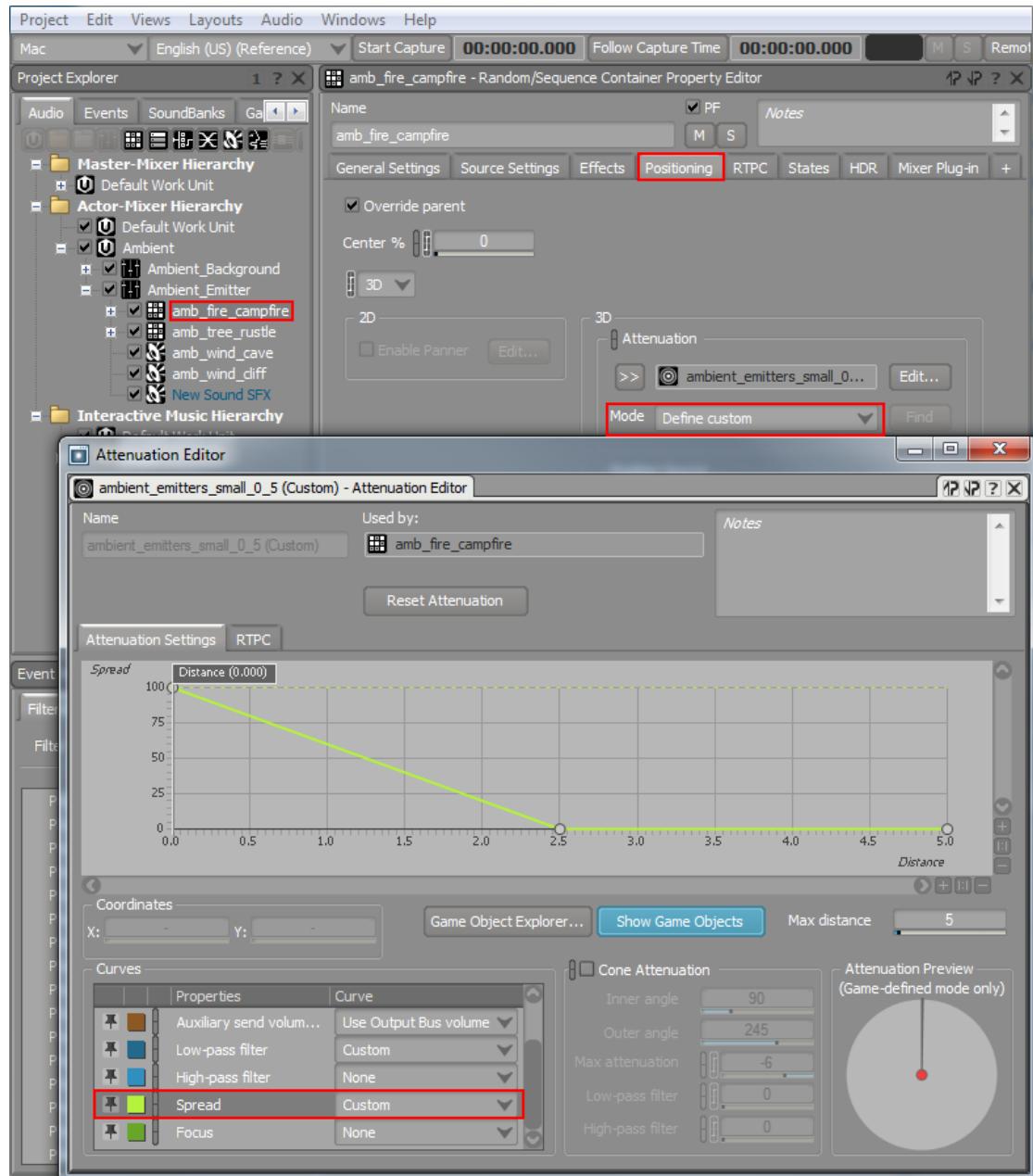
Now that the ambient\_fire\_campfire sound object has been changed to Define Custom as its attenuation mode, it is now using a unique instance of the attenuation that is not shared (or sharable) with any other sound object.

### Designer Note



Spread specifies the amount or percentage of audio that is spread to neighboring speakers, allowing for sounds to change over distance from a point source at low values to a completely diffused propagation at high values. For multichannel sounds, each channel is spread separately.

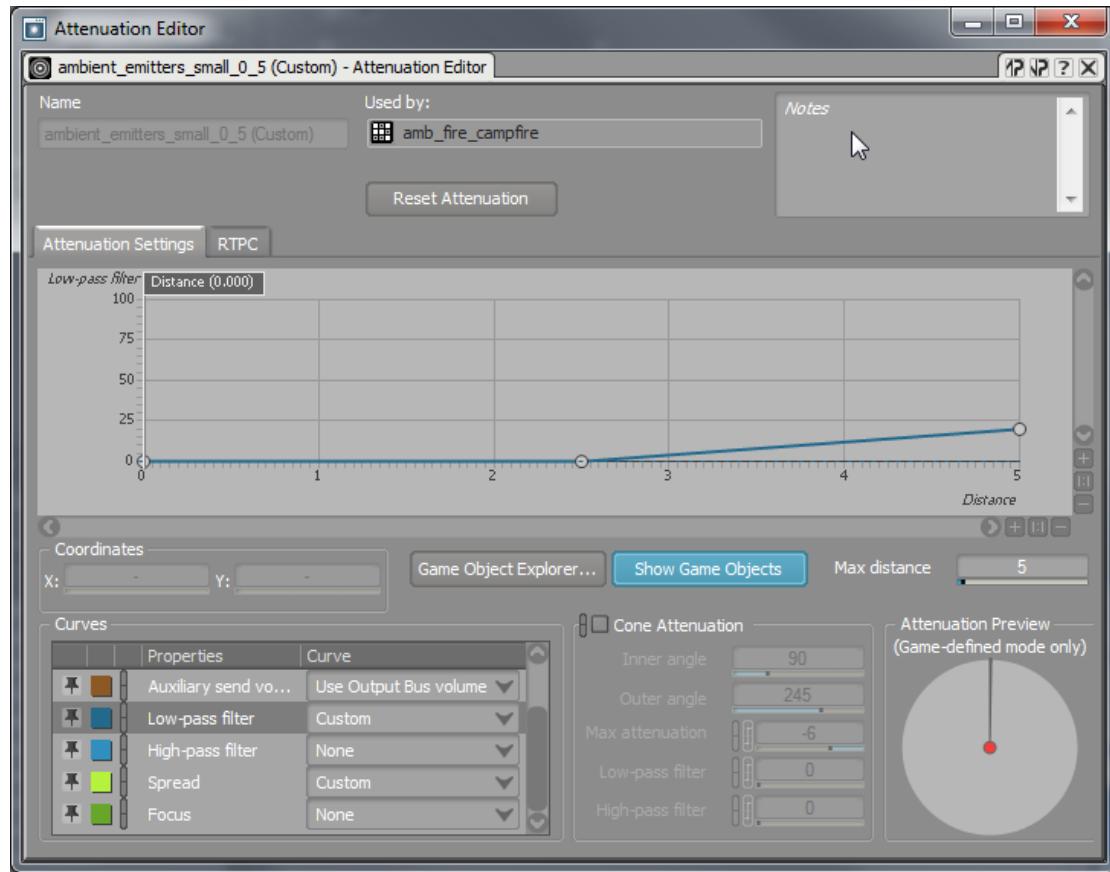
In the following image, the sound of the campfire is set to begin spreading between the available speakers at half its maximum distance (2.5) until it is 100% spread at its minimum (0).



Spread authored in the Attenuation Editor and modified by the distance parameter

### Using Low Pass Filter as Part of an Attenuation

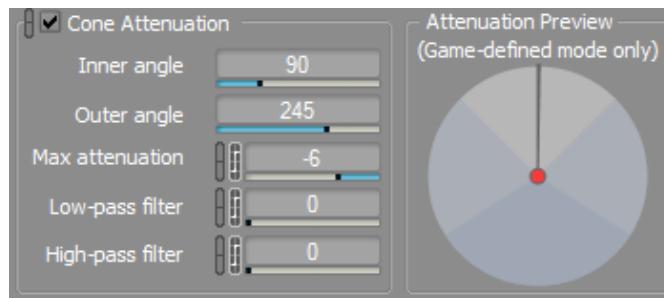
To add further realism to the sound of fire over distance, we can also enable a custom low pass filter that will begin cutting high frequencies as the listener gets further away.



Low pass filter authored in the Attenuation Editor and modified by the distance parameter

### Adjusting Cone Attenuation Properties

Cone Attenuation properties can be adjusted to further influence the directionality of sounds that subscribe to an attenuation ShareSet. Doing so allows you to focus sounds in the world by restricting the angle of sound propagation based on the game objects orientation. This can be especially useful for vehicles, projectiles, and objects that require a more focused positional sound.



Enabling Cone Attenuation properties

## **Section Summary**

Attenuations are specific sets of rules that determine how the sound of objects will change over distance within your game. They're highly customizable and can be used as creatively or realistically as your vision requires. Whether you strive for a heightened sense of reality or a creative impressionistic aesthetic, the flexibility of attenuations allows you to tailor the sonic propagation palette to your needs.

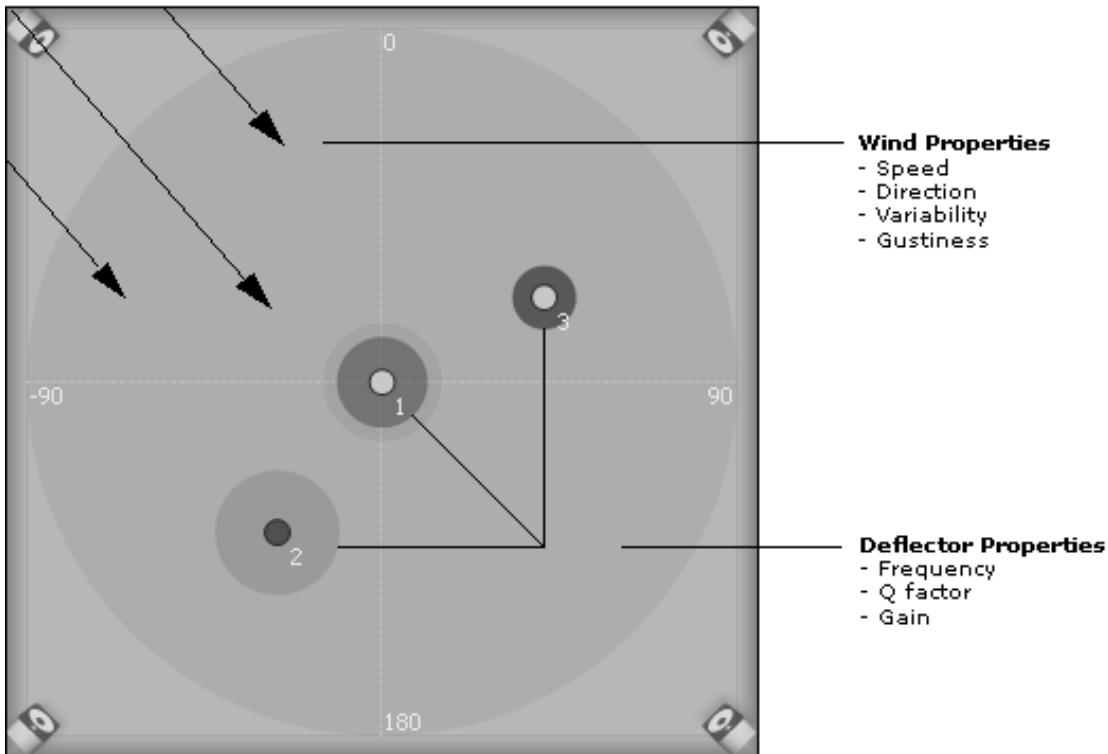
## **SoundSeed Air -Wind**

Whether you find yourself at the edge of a cliff overlooking the path ahead, or standing at the mouth of a dungeon thinking about what might be lurking in the shadows, the sound of wind can help add an epic tone to the situation. The SoundSeed Wind plug-in has been created as a runtime-efficient solution to the challenge of authoring dynamic wind elements.

The SoundSeed Wind plug-in is a Wwise source plug-in that generates wind sounds as they pass through and around objects. These sounds are generated by using time-varying parameter sets to drive a synthesis algorithm. No source audio files are necessary as the wind sounds are completely synthesized. SoundSeed Wind allows you to save memory in-game because there is no longer a need for the long looping wind ambience .wav files that would have traditionally been used.

SoundSeed Wind emulates the flow of wind through a scene; the direction specifies the point of entry of wind flow on to a scene. Thus you should expect pressure waves to hit the deflectors closest to the entry point first. The flow will then propagate through the scene as it is pushed by incoming wind. As it propagates, you will hear deflector objects that are placed farther from the entry point as they encounter the oncoming pressure wave. Note that higher wind speeds at the entry point push the flow at a quicker rate than slower speeds.

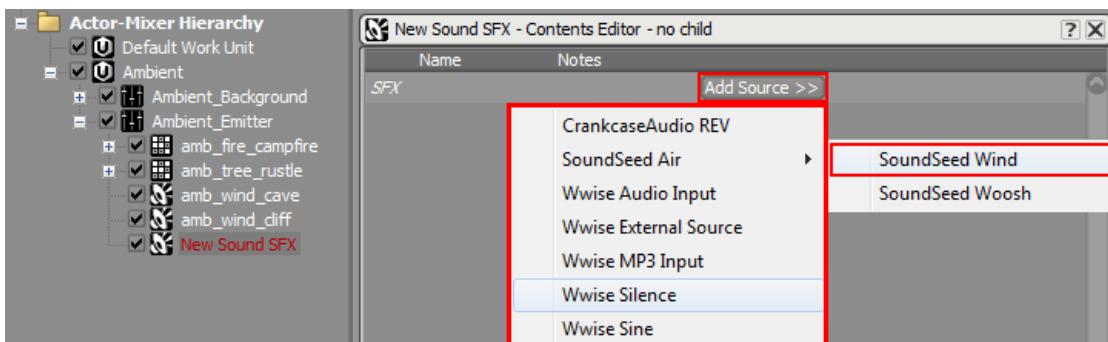
The following illustration demonstrates a typical scene where wind and a series of deflectors are defined using a set of properties. Deflectors will appear smaller or larger in size and lighter or darker in color, depending on the frequency and gain property values assigned to them.



### Deflector and wind properties

The biggest benefits to synthesized wind are the ability to dynamically manipulate the wind model in real time, the randomness of generating sound differently every time, and the memory savings incurred by removing audio content from the equation. What would normally have taken a many seconds long audio file can now be replaced by realistic models authored within Wwise. These authored sounds are further modifiable by game parameters at runtime, as well as from within the authoring application while connected to the game.

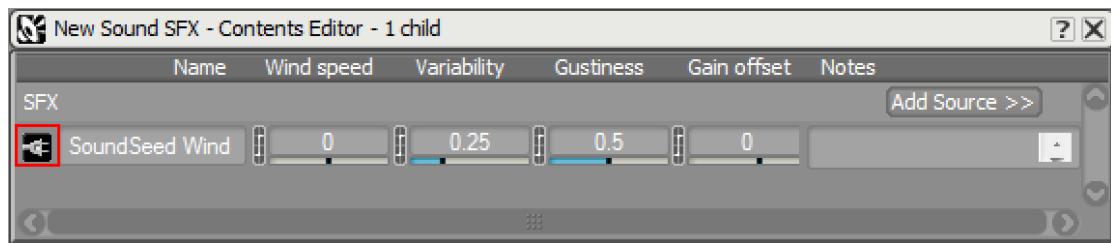
Adding the SoundSeed Wind plug-in to a new Sound SFX is accomplished in the same manner as the silence plug-in covered in the ambient chapter.



Adding SoundSeed Air Wind to an empty Sound SFX

## SoundSeed Wind - Deflectors

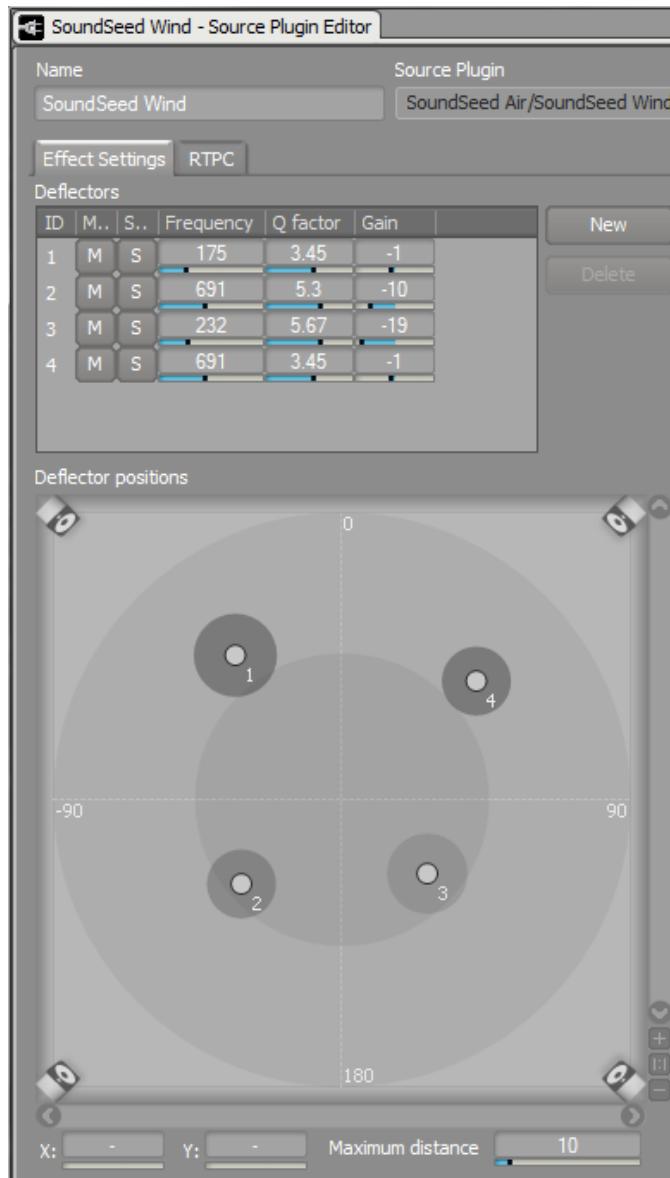
The first step in creating a wind model begins with the placement of a deflector (or deflectors), which is used to generate wind as it passes through and around the objects. The Source Plug-in Editor can be accessed in the Contents Editor by double-clicking the plug-in icon.



Wind Source plug-in icon

The attributes available include:

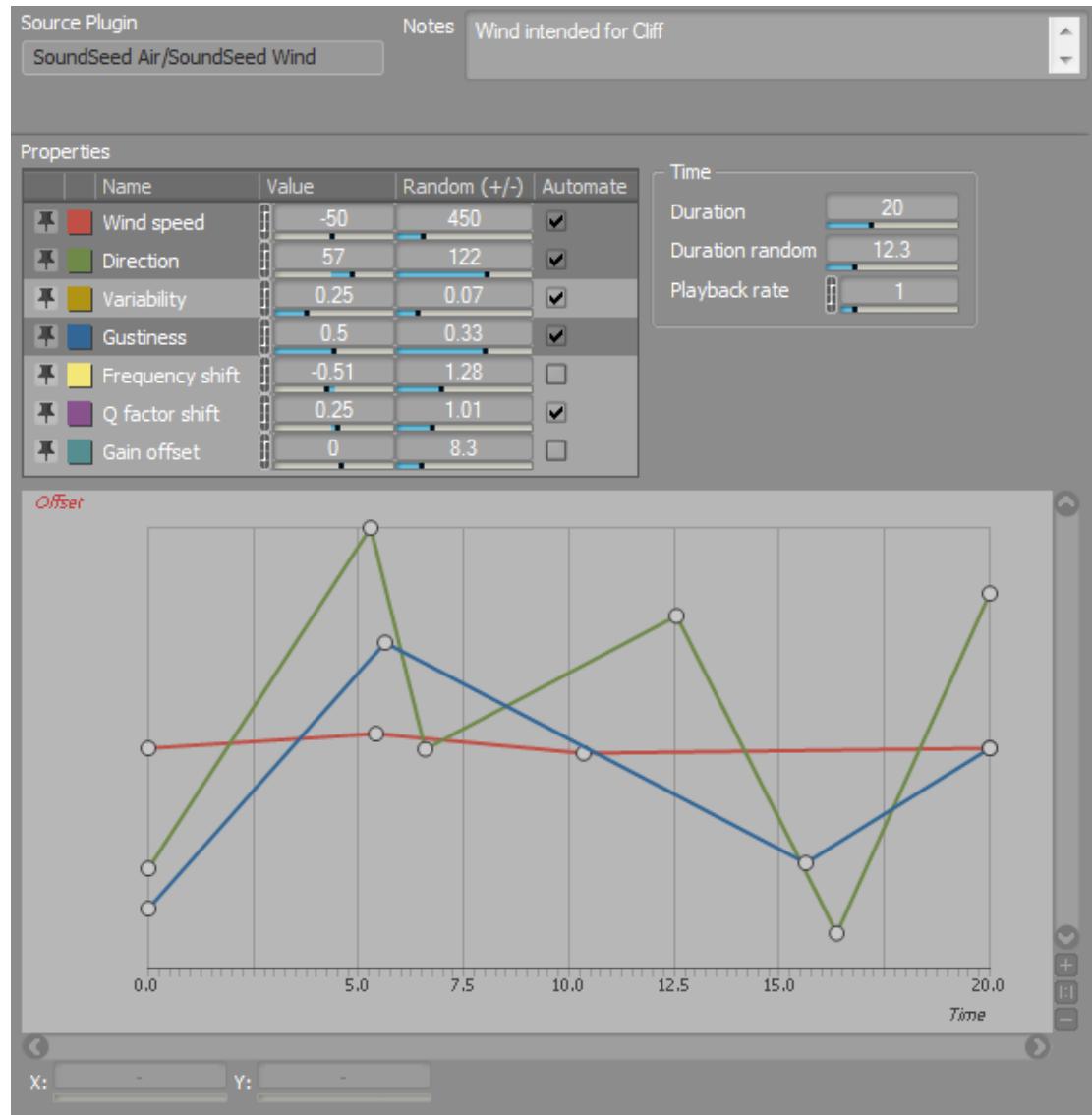
- **Frequency:** Represents the size of the deflector.
- **Q factor:** Is used to simulate deflector surface type.
  - Regular (high Q) /round or irregular (low Q)
- **Gain:** Control of the amplitude of each deflector.



Positioned deflectors

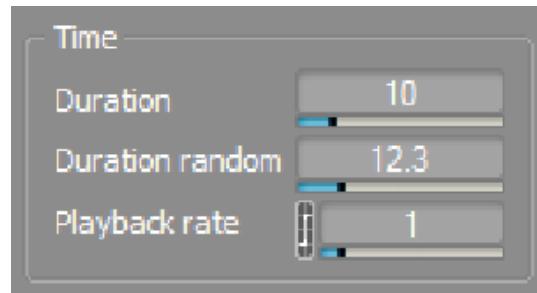
### SoundSeed Wind - Properties

Like any tool, you can experiment to create and discover the sound you want. For both wind and deflector properties, a fixed value can be set along with a positive or negative deviation. All of these properties can be further automated over time, by adding any number of points in the curve editor.



Wind properties and property automation using the curve editor

The Playback rate can be modified using an RTPC in the Time section of Properties, as can any property with an RTPC indicator. Looping needs to be enabled in the General Settings tab, just like any sound object, while fixed parameters for loop duration and additional random duration are set in the Time section.



Properties for time related functionality

### **SoundSeed Wind - RTPC**

With properties mapped to game parameters, you can use information from the game to control various aspect of the wind sound. Imagine using the player velocity to control the wind speed as they plummet through the air after a successful jump, or mapping the gusts to an altitude game parameter that changes as the player ascends along a mountain pass. By bringing parameter control to your wind models, you can dynamically represent and better reflect the intention of the gameplay and scenario.

## Ambient Summary

In this chapter we have established some fundamental processes for building dynamic and varied soundscapes. Many of the techniques covered in this chapter will continue to inform the rest of this document and serve as the building blocks of interactive audio.

**Throughout this chapter we have:**

- Created various types of sound objects
- Used sound objects as building blocks for an ambient soundscape system
- Established looping backgrounds
- Created a system to randomly playback individual ambient elements

**Stepped through the process of:**

- Importing simple sounds to containers
- Using blend containers to sum and playback multiple sound objects simultaneously
- Creating a game parameter that can be used to modify properties
- Using the Time\_of\_Day parameter to Crossfade between containers

**Also touched on:**

- Using attenuation ShareSets to manage distance based falloff
- The use of SoundSeed Air - Wind to author realistic sounding dynamic wind
- Leveraging paths using the Position Editor to randomly position individual ambient elements in 3D space.

**And created the following objects:**

- Looping background random containers.
- Ambient\_day\_night blend container that Crossfades between content using blend tracks based on a time of day game parameter.
- SoundSeed Wind models to be used as positional sound emitters.
- Generalized attenuation ShareSets for ambient emitters.

## Ambient Additional Resources

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Defining Positioning for Sound and Motion > Understanding Positioning in Wwise > Handling multichannel sources with 3D positioning](#)

[Audiokinetic Q&A - How do I simulate a sound that is not a point sound source?](#)

[Video Tutorial - Using the interface](#)

[Video Tutorial - Importing Audio Files](#)

[Video Tutorial - Building Sound Hierarchies](#)

[Video Tutorial - More Learning Resources](#)

[Video Tutorial - SoundSeed Air Wind Overview](#)

[Video Tutorial - Using the Blend Container](#)

[Video Tutorial - Relation between Sounds, Sources and Audio Files](#)

---

## **Chapter 2. Establishing Character**

Overview .....	55
Footsteps and Movement - Establishing Needs .....	56
Simple Steps .....	56
Switching System Introduction .....	58
Defining Step Type .....	59
Defining Surface Type .....	62
Defining Character Type .....	64
All Together Now .....	64
Movement .....	66
Defining Armor Type .....	66
Creating the Movement Event .....	67
Character Summary .....	68
Character Additional Resources .....	70

## Overview

From the hero's first steps through the forest, to the upgrading of armor after a dungeon crawl, movement sounds help to anchor the player firmly in the game world. While the content demands of character sounds can vary wildly across different game types, the fundamentals remain consistent across any genre.

This chapter will take you through the process of:

- Creating player and non-player character (NPC) movement sets
- Understanding switch groups, switches, and how to use them
- Establishing multi-level switches to manage:
  - Step type
  - Surface type
  - Player type
- Creating movement events

Character sounds persist throughout every area and scenario of the game world. Special care and handling should be used when defining the aesthetic and implementation that result in sounds heard frequently throughout gameplay. Thankfully there are several different ways to approach movement sounds for any situation.

## Footsteps and Movement - Establishing Needs

To establish audio content needs, it's important to understand a few things about how to best support gameplay when it comes to movement sounds. You can easily create a simplified step system, but a varied system produces a better experience. The importance of varied footstep sounds in a game can be a delicate balance of finding the right places to put resources without ending up with a system that uses up all of the available memory.

Think about the following questions to help you determine how extensive the footstep system should be:

- How many step types will characters have? (Walk, run, scuff, turn...)
- How many surface types need to be represented with sound? (Dirt, stone, sand...)
- How will footstep sounds be triggered? (Animation, programmatically...)
- How often will you hear footsteps for an extended period of time? (2-5 minutes, 10-20 minutes...)

### Simple Steps

A simplified footstep system might consist of a randomized container of individual footstep audio files played each time a character's step reaches a certain frame of animation. These animation-based footsteps are usually marked (or tagged) with a Wwise event for each frame of animation, where the footstep is meant to play back a sound. Different types of steps such as walk, run, turn, scuff, jump, land, etc. would then be explicitly tagged to a frame of animation, corresponding to an event comprised of the correct sounding steps.



Simplified footsteps implementation

While this technique may still have its place in games where there are few footsteps, the simplified system quickly shows its limitations because of its recognizable repetition and the inability for sound designers to manage variables from within the authoring application.

## Switching System Introduction

Switches represent the different alternatives that exist for a particular game object within the game. Sound, music, and motion objects are organized and assigned to switches so that the appropriate sound or motion object will play when a change is made from one alternative to another in game. The Wwise objects that are assigned to a switch are grouped into a switch container. When an event signals a change, the switch container verifies the switch and the correct sound, music, or motion object is played.

To take this a step further, we are going to build a cascading set of switch groups that work in concert to play the correct footstep sound based on:

- Step Type
- Surface Type
- Character Type

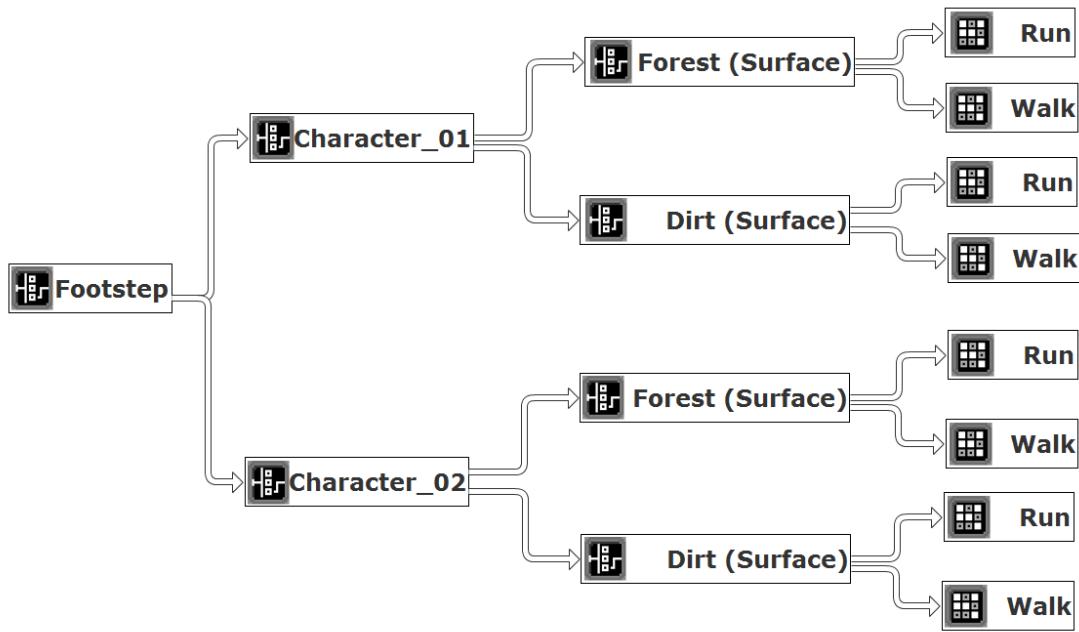
To unleash the full power of switches a programmer will need to define these in the game engine to drive the system.

It's common for every asset in a game to include metadata about what the asset is, be it a rock texture or character model, along with the properties that define it. By using these already existing definitions, or creating this data to control the switching system, we move closer to a data-driven pipeline making things easier to manage and scale throughout development.



### Designer Note

Using information from the game to drive the switching system in Wwise is something that should be discussed with an audio programmer as soon as possible during the development cycle. There are great opportunities to automate much of the challenge that comes with this aspect of implementation.

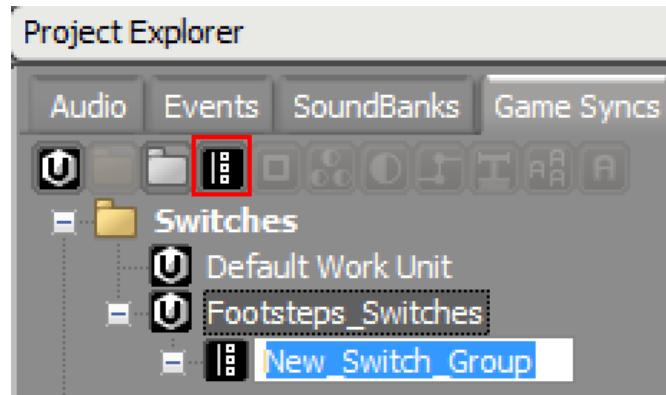


Multi-level switch system showing switch between:  
character type, surface type, and step type

While seemingly complex compared to the simple method of footstep implementation, the flexibility of using information from the game allows for greater control of the resulting sound and puts creative control in the hands of the sound designer.

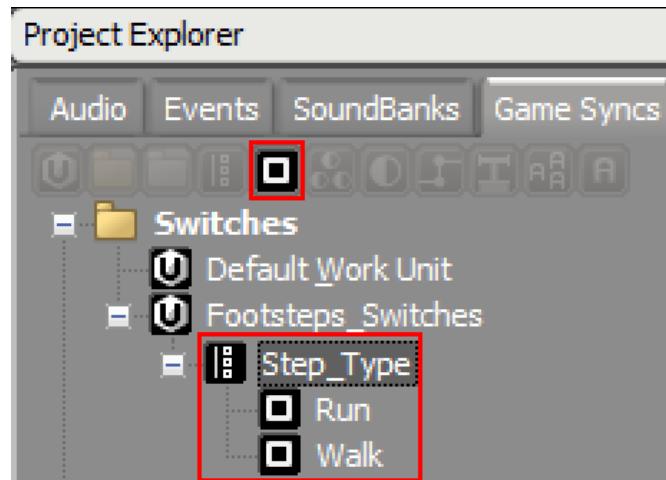
### Defining Step Type

We'll start by creating a new work unit for our footstep switch and add a Switch Group for Step\_Type to change between walking and running switches. By selecting the new Footstep\_Switches work unit and then clicking the switch group icon in the Project Explorer toolbar, a new switch group is created.



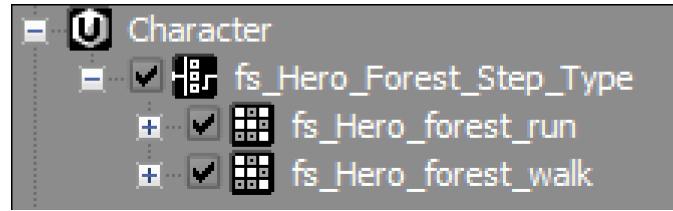
Creating a switch group in the “Footstep” switch work unit

Once the new switch Step\_Type switch group has been created, switches can be added by selecting the new switch group and then clicking the switch icon in the Project Explorer toolbar. New switches are created: one for each step type: Walk and Run. Alternatively, switches and switch groups can be created from the contextual menu or by using shortcut keys.



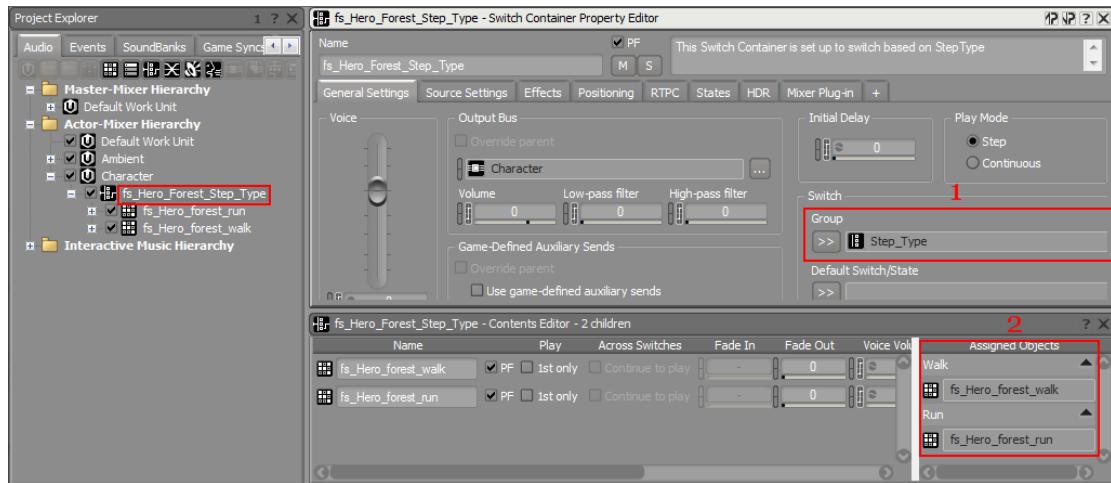
Creation of the Step\_Type switch group and corresponding Run and Walk Switches

A new switch container which contains both run and walk random containers can now be created.



A switch container containing both run and walk random containers

The switch type can then be defined by using the switch type Selector button (>>) and selecting the new Step\_Type Switch Group (1). Next, the walk and run random containers can be assigned by dragging and dropping them to the appropriate Switch in the Assigned Objects area of the Content Editor (2).



Adding the Step\_Type switch group and assigning objects

Auditioning the different step types can be done through the Transport Control. After selecting the fs\_Hero\_Forest\_Step\_Type switch container by double clicking it, clicking the play icon or pressing the spacebar will play the object. Setting the switch can be done from the game syncs section of the Transport Control. When selected, any associated game syncs available for the object will be displayed.



Transport Control showing the selected switches game sync and available switches for the selected game object

## Defining Surface Type

One of the greatest potentials for diversification, and something that immediately communicates a sense of place to the player, is the inclusion of different footstep surface types. While these may be represented in the game visually, it may take extra effort to communicate information about the surface type a character is traversing as a switch. Ideally the game engine has a way to specify the surface type of a traversal area, and this specification can be posted to Wwise as a switch.



### Programmer Note

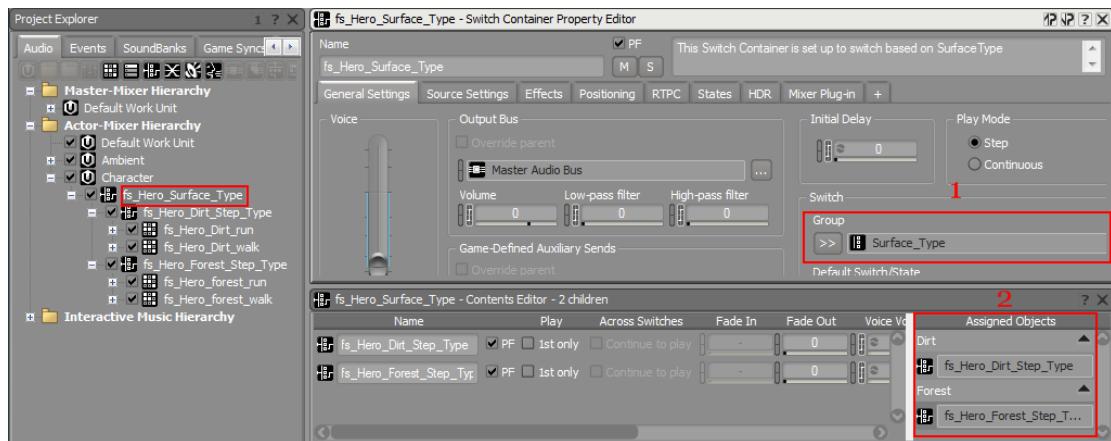
There is a helpful footstep integration example that comes as part of the Wwise Programmer SDK.



### Designer Note

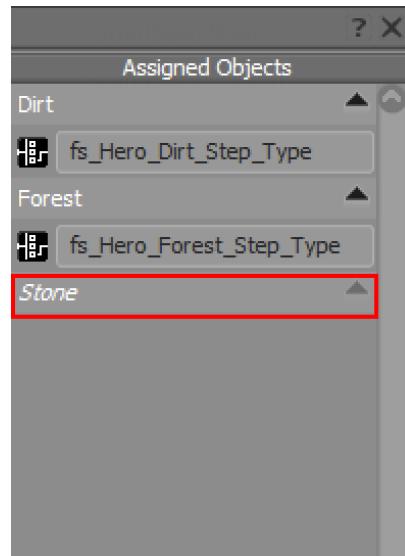
Once surface type is communicated to Wwise, there are other applications and uses for this information. If, for instance, you were building a dynamic rain system, you could use surface type to change rain drop impact playing back positionally from different surfaces in the game world. Also, most physics impact sounds benefit from knowing the surface material being impacted. As an example, the sound of a wood plank falling on dirt will have a much duller sound than on concrete.

As in the previous section, we'll create a new switch container and subscribe it a new Surface\_Type switch group (1). We can then add the random containers to the appropriate switches for either dirt or forest in the assigned objects area of the Content Editor (2).



Adding the Surface\_Type switch group, and assigning objects

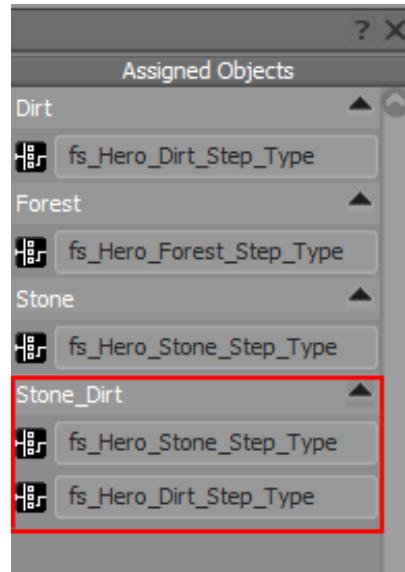
Using a multi-level switching system contributes to an organized and consolidated working methodology which is supported within the Wwise hierarchy. It also helps to keep the implementation clear and concise. Furthermore, switches can be added to switch groups as new types become available in the game, which makes scaling a straightforward endeavor. As an example, if we want to add an additional switch for the Stone surface type, we can add it to the switch group, and it will show up in the Assigned Objects section of the Contents Editor for any sound object subscribed to the Surface\_Type group.



### Adding an additional switch to an existing switch group

Let's say the sound content for the Stone footsteps hasn't been delivered yet. We can temporarily use any available audio content until it is ready to be replaced. Temporarily assigning content allows for immediate feedback and helps validate a working system. This can be further verified by looking at the capture log returned from the game as part of the Wwise Profiler. (See Optimization 10.4 [Understanding the Different Types of Profiling in Wwise](#).)

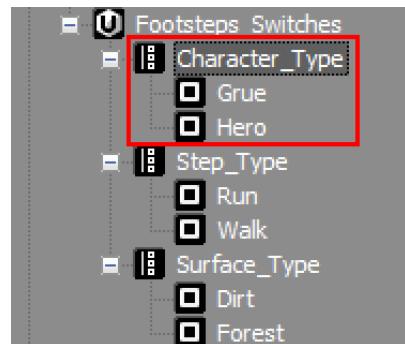
This level of abstraction also allows you to share content between Switches and, when combined elementally, can produce additional variations without the need for additional content. For instance, combining Dirt and Stone sounds could produce a dirty/stone surface type variation using content that already exists.



Adding multiple sound objects to a single switch

## Defining Character Type

Taking this a step further, we can also add a Switch Group for the Character\_Type:



Character\_Type switch group and switches

## All Together Now

We now have switching information communicated from the game to Wwise for three different aspects of character sound:

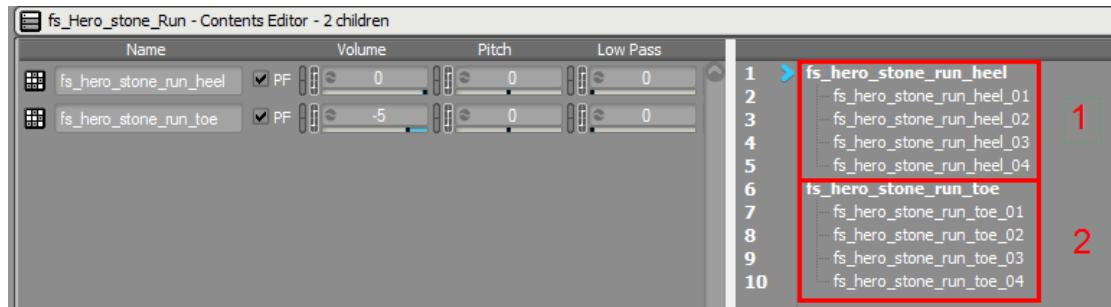
- Step Type
- Surface Type
- Character Type

The combination of these switches when the Footstep event is triggered by the game will cause a sound from the correct character, step and surface type sound object to be played. Now instead of having multiple events to keep track of, we can simply pass a single event for a 'footstep' from the game and let switches handle the correct playback of sound as defined by the game.



### Designer Note

Another technique used sometimes at the content level is the separation of heel and toe variations. This adds further diversity through randomized combination and, if the content is designed appropriately, can exponentially increase the variations and further reduce the chance for repetition. By setting up a sequence container with a randomized Heel container (1) followed by a randomized Toe container (2), each step can be composed on-the-fly when a footstep is requested. Further memory savings may be possible if you're able to share heel or toe components between different surfaces. By working elementally with the techniques available within the Wwise authoring application, you can increase variation and reduce your memory footprint, which is a winning prospect for any development.



#### Sequencing heel and toe to reproduce a single footstep

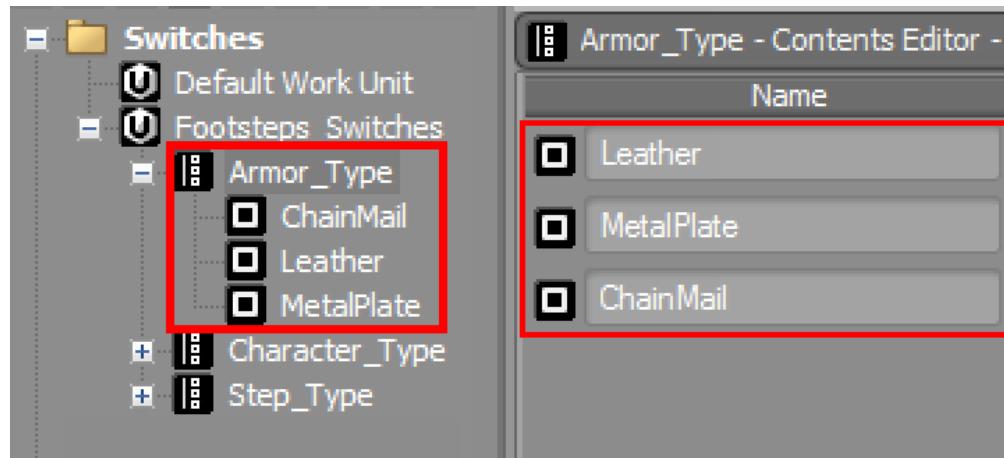
Regardless of the level of detail your game and implementation requires, Switches provide an efficient way to deal with groups of content whose playback is dependent on information from the game engine.

## Movement

Another key aspect to movement sound is the sound of clothing material as a character travels through the world. Whether it's age-old leather, chain-mail, or plate style armor, the distinguishing characteristics add to the footstep sounds and help bring an additional level of believability. The sound of movement is especially important if different types of wearables can be changed or added during the course of gameplay. Changing out the sound whenever possible helps to reinforce a choice made by the player and adds to the level of realism throughout their experience.

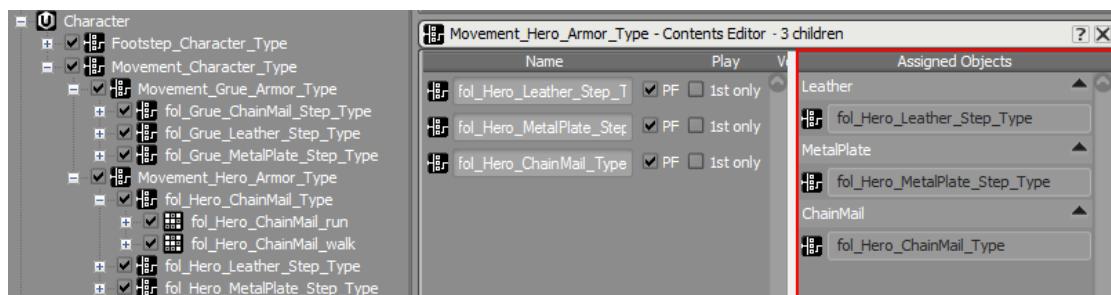
### Defining Armor Type

Similar to the switching system implemented for footsteps, we'll use the Character\_Type and Step\_Type switches in addition to a new switch for Armor\_Type.



Armor\_Type switch group and switches

With each of these switches being used to drive the switching system, the hierarchy can be organized to switch between appropriate sound content.



Movement switches with assigned step types

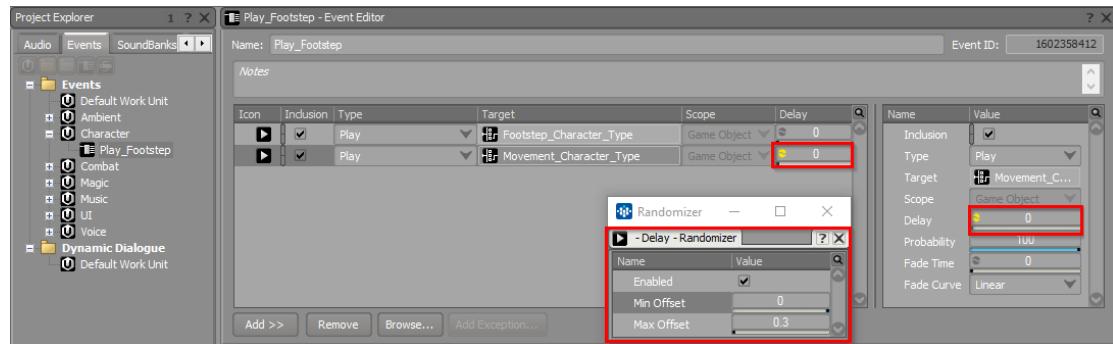
## Creating the Movement Event

Once we've got everything switching appropriately and all the content in place, it's time to add the parent movement switch container to the footstep event using the Play action.



### Design Note

Depending on the sound content for movement, playing the armor material sound simultaneously with the sound of the footstep may not be desirable. In situations where it is preferable to give the footstep some room to be heard, the delay property can be used as part of the movement play action. Further adding a fixed delay offset to the movement sound, in addition to delay randomization, increases the randomness and lends itself to a more realistic representation.



### Movement switch and delay offset properties

Using this technique we can better represent the sound of movement between footsteps, it also adds a feeling of variation and uniqueness to each footstep event.

Other switch specific functions, such as the ability to drive switches based on game parameters, continuous switch mode, and fade behaviors are additional techniques for creative implementation. Furthermore, switches can be a handy organizational tool and a convenient way to harness game information as part of a streamlined workflow.

## Character Summary

Throughout this chapter we've come to a greater understanding of switch groups, switches, and their use as part of a system that works in conjunction with game information. The strength of building a system based on information from the game is in the ability to change, control, and manipulate content within the tool, without changing code. It also puts control over behaviors independent of the game in the hands of the sound designer.

**Throughout this chapter we have:**

- Discussed establishing a simplified model of footsteps.
- Developed an understanding of switch groups, switches, and their use.
- Discussed the randomization of properties within a sound object.
- Arranged containers within the assigned objects section of the Contents Editor.

**Stepped through the process of:**

- Creating sets of player and Non-Player Character (NPC) footstep random containers based on:
  - Character type
  - Surface material type
  - Footstep type
- Creating sets of player and Non-Player Character (NPC) movement random containers based on:
  - Character Type
  - Footstep Type
  - Armor Type
- Establishing switches to manage:
  - Surface Type
  - Player Type
  - Step Type
- Creating a multi-level switching system for footsteps and movement.

**Also touched on:**

- Using delay offset in the Event to introduce natural randomness in movement sounds.

**Throughout this section we have created the following objects:**

- A multi-level switching system that determines the character, surface, and step type when the footstep event is played.
- A multi-level switching system for movement sounds with randomized delay offset in the footstep event.

- A footstep event comprised of play actions for both footstep and movement switch systems.

## Character Additional Resources

[Wwise Fundamentals > What are Game Syncs? > Understanding Switches](#)

[Wwise Help > Interacting with the Game > Working with Switches](#)

[Wwise SDK > Sound Engine Integration Walkthrough > Integrate Wwise Elements into Your Game > Integrating Switches](#)

[Video Tutorial - Creating Footsteps using Random and Switch](#)

---

## **Chapter 3. Preparing for Combat**

Overview .....	72
Defining Sound Sets for Weapon Types .....	73
SoundSeed Air - Whoosh .....	74
Understanding Impact .....	79
Defining Weapon Type .....	79
Weapon Impact System .....	80
Attenuations for Player vs. NPC .....	82
High Alert .....	82
Listener Considerations .....	82
Combat Summary .....	84
Combat Additional Resources .....	85

## Overview

The sounds of combat serve as punctuation marks along the journey, as the hero battles across the ambient soundscape we've created thus far. With each new encounter, each enemy variation, each weapon upgrade, sound has the ability to reward the player with dynamic audio that equals the unfolding drama on-screen.

**This chapter will take you through the process of:**

- Defining sound sets for different weapon types
- Generating weapon swings using SoundSeed Whoosh
- Understanding different implementation techniques for impacts
- Attenuations for player vs. NPC.

Starting with a list of available weapon types is one way to approach the sometimes sizable task of combat sound. Once the list has been defined, it's important to understand the different ways that weapons will be triggered, and how to organize them within the project.

**Some questions to ask about combat:**

- How will combat sound be triggered by the game engine?
- Do most of the combat actions rely on the animation system?
- Is hit detection for impacts handled separately from animations?

When locked in mortal combat with a foe of ill repute, one of the roles that sound plays is in communicating success or failure for each attempted strike. Once you have determined how combat sounds will be played back, you can make sure the actions are represented with the appropriate sound.

## Defining Sound Sets for Weapon Types

At the basic level, you can usually count on the playback of swings and impact to share the same implementation technique across different weapons. Additional accents and special moves may be unique; however, there is usually a logic to what attacks can be shared by multiple weapons and which may require special handling. Whether designing a single sound for each action, or working from a pool of basic sounds that can be used as building blocks, how you approach the design often begins with knowing how it will be implemented in the game.

In this example, we are going to use unique weapon swing sounds created using the SoundSeed Air - Whoosh plug-in. The Whoosh plug-in generates sounds made as a simulated object passes through the air. This makes it perfectly suited for close combat weapon swooshes, bullet fly-bys, and other motion sounds.

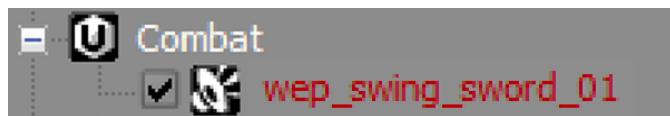
We'll also take a different approach for weapon impacts and combine different material type sounds, based on the surface type (or types) being impacted. This methodology allows us to get a maximum of diversity out of a core set of audio files, by utilizing the surface type switches already created for footsteps.

## SoundSeed Air - Whoosh

The SoundSeed line of plug-ins has been created to further extend the functionality of the Wwise authoring application and provide a unique solution to a common problem. When faced with the challenge of keeping repetitive sounds fresh, dynamic, and varied, SoundSeed provides resource conscious creative tools that bring the design of sound one step closer to the game.

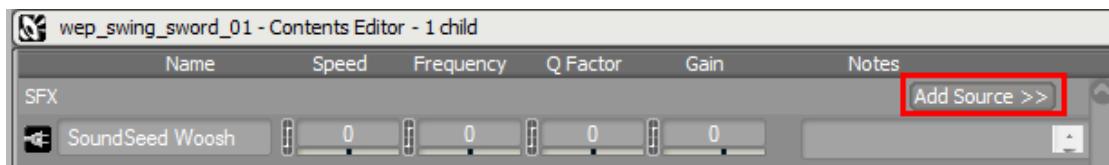
The SoundSeed Whoosh plug-in is a source plug-in that generates sounds as an object passes through the air. To create these types of sounds, the characteristics of the deflector object are defined along with the trajectory movement and speed of the object. No source audio files are necessary as the whoosh sounds are completely synthesized. SoundSeed Whoosh allows you to save memory in-game because .wav file variations are no longer necessary. Additionally, the Whoosh source can be used to create variations using the multiple randomization features.

We'll begin by creating the first of many sword swings that will be available to the hero. It begins with an empty Sound SFX in the Combat work unit nested into the Swing Actor-Mixer.



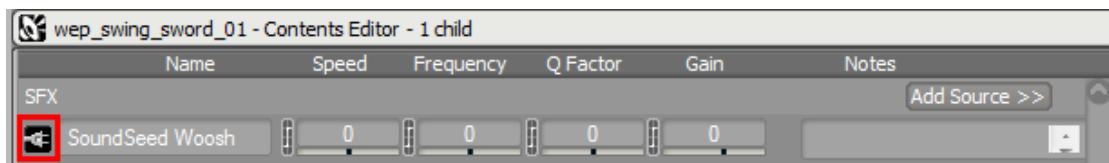
An empty Sound SFX in preparation for adding a whoosh source

We can now add a SoundSeed Air - SoundSeed Whoosh plug-in by navigating to the Add Source Selector button (>>) in the upper-right corner of the Contents Editor.

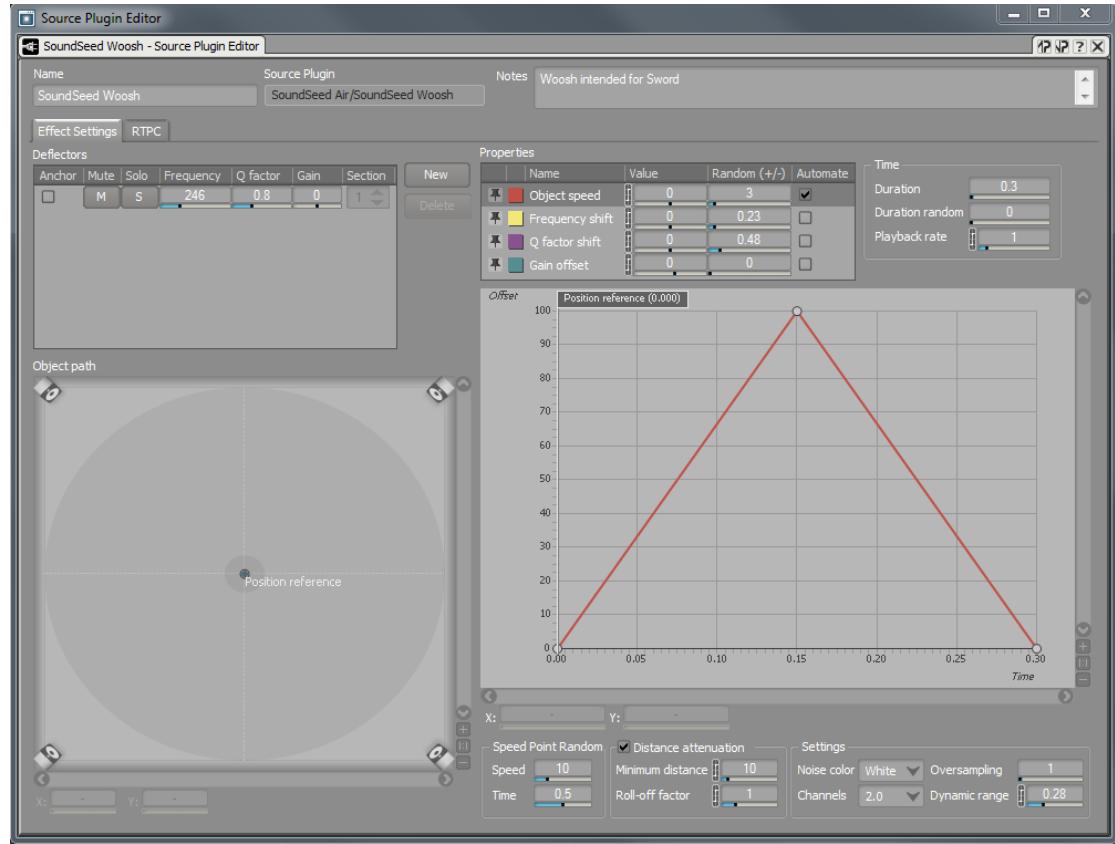


Whoosh Source plug-in Add Source Selector button (>>)

The Source Plug-in Editor can be accessed in the Contents Editor by double-clicking the plug-in icon.



Whoosh Source plug-in icon

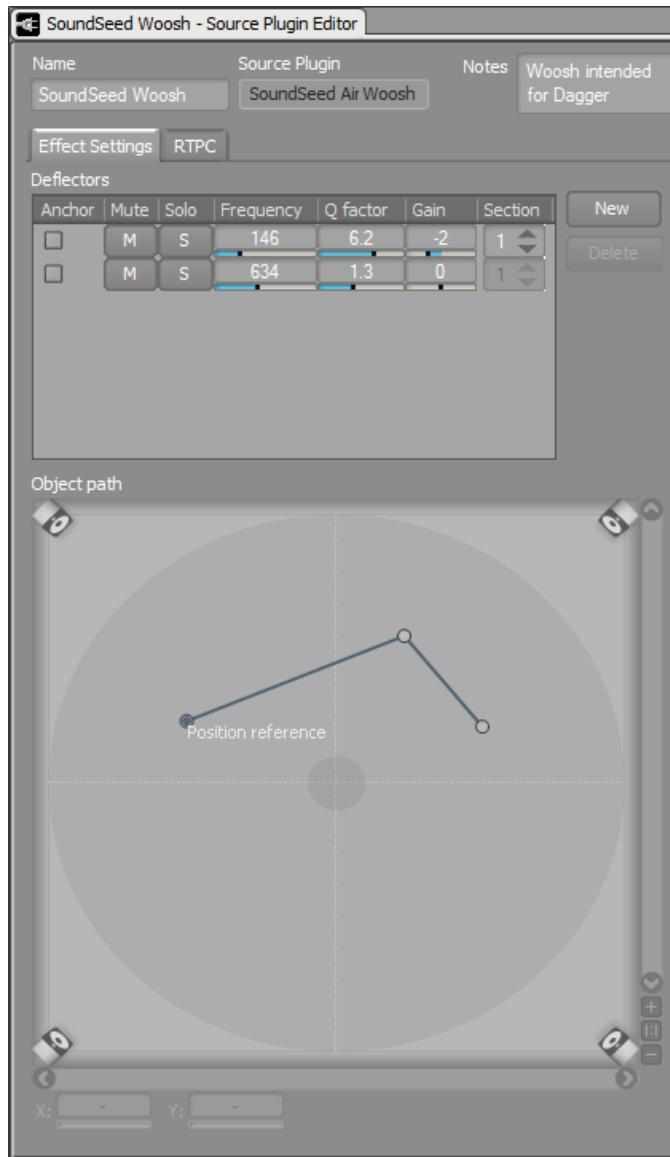


### Authoring Whoosh sounds in the source plug-in editor

Whooshes are a combination of one or many deflectors representing an object and its traveling path. Deflector settings are used to determine the shape of the object, while anchor defines whether the object is spinning or not.

- Frequency: Represents the size of the object.
- Q: Is used to simulate deflector surface type. (regular [high Q]/round or irregular [low Q])
- Gain: Control of the amplitude.

The object path is comprised of points in a path and defines the movement direction of the object through space.



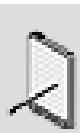
Deflector properties and object path

Any number of points can be connected to produce the desired travel path, which appropriately pans based on speaker setup and the number of channels specified in the Settings.

For the weapon swings, we'll be using the two channel setting in conjunction with a path that traces an arc in front of the player from left to right. For the sake of the example, we'll create Whoosh models for sword, axe, and dagger weapon types.

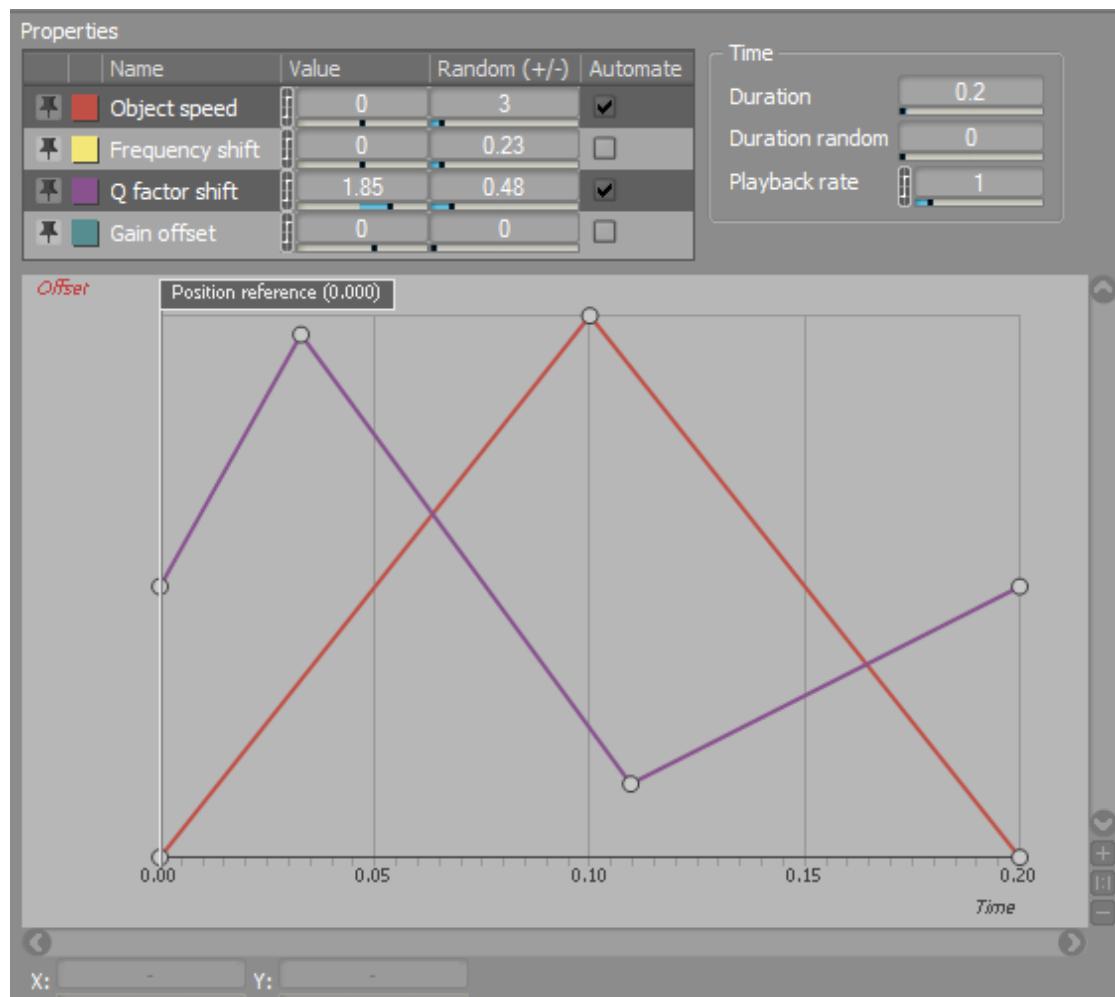


Prepared Whoosh sounds for use by weapon swings

 **Designer Note**

Not just reserved for short duration swings, coupling a circular path with a long time duration can add additional movement sound as a special effect. Due to the generative nature, the simulation and resulting sound will be different every time. Adding a randomized multichannel Whoosh element as part of a spell casting effect could be used to further wrap the player in a swirling maelstrom of sound, without having an impact on the memory budget.

Similar to SoundSeed Wind, properties that control the object's movement can be fixed, randomized, or parametrized via RTPC. The added ability to create automated curves for properties allows you to add further movement and controlled randomness to help sculpt whoosh sounds.



### Whoosh properties, automation curves, and time settings

Ignoring all of the memory saving benefits of generative sound design, there is an incredible amount of creativity encapsulated in the SoundSeed suite of plug-ins. By taking a common sound design tool and making it available as part of a game audio authoring application, generative synthesis at runtime can help bridge the gap between design and implementation. Add to this the ability to affect properties in real time using game parameters and you have a winning combination of dynamic flexibility and memory efficiency.

By establishing the weapon swing sounds within a single swing Sound SFX , the need for content variation has been removed by leveraging the features in Whoosh.

## Understanding Impact

With the beginnings of a weapon swinging in place, we'll move on to the finer details involved with impact sounds and the bashing of bad guy skulls. Object impacts are often handled with either a composite sound (meant to convey the emotional intent of the impact) or by creating a multi-layered system (aimed at recreating the sound of reality). The first decision about which direction to take lies in the type of game, and what will best support the style and expectation of the player.

In a system that relies on a minimum of variation, it may be enough to simply play a group of randomized audio files for an impact - for any weapon, on any surface. Taken a step further, the weapon type could be used to determine the impact sound and play, regardless of surface type. Still further, each weapon type could have a different impact sound on each surface type in the game, making a sword impact on stone sound different than dirt or an axe impact on wood.

Regardless of whether you choose to represent object interactions abstractly or by embracing a model that skews closer to reality, the Wwise authoring application can accommodate your decision.

### Defining Weapon Type

In the continued pursuit of randomness and variation, we are going to build a system that utilizes switches in a similar fashion to the previous footstep and movement examples. Instead of focusing on character and step type, we'll use weapon type in conjunction with an expanded list of surface types to allow for an increased level of detail.



Weapon\_Type switch group and switches

This switch group will change based on the weapon type being wielded by either the player or NPC and will be assigned a unique sound object for each type. Now that there is an established switch group for weapon type, it can be used to implement the swings created in the previous section by parenting them within a switch container, setting the switch group, and assigning the objects.



Weapon type switch group with assigned swing sounds

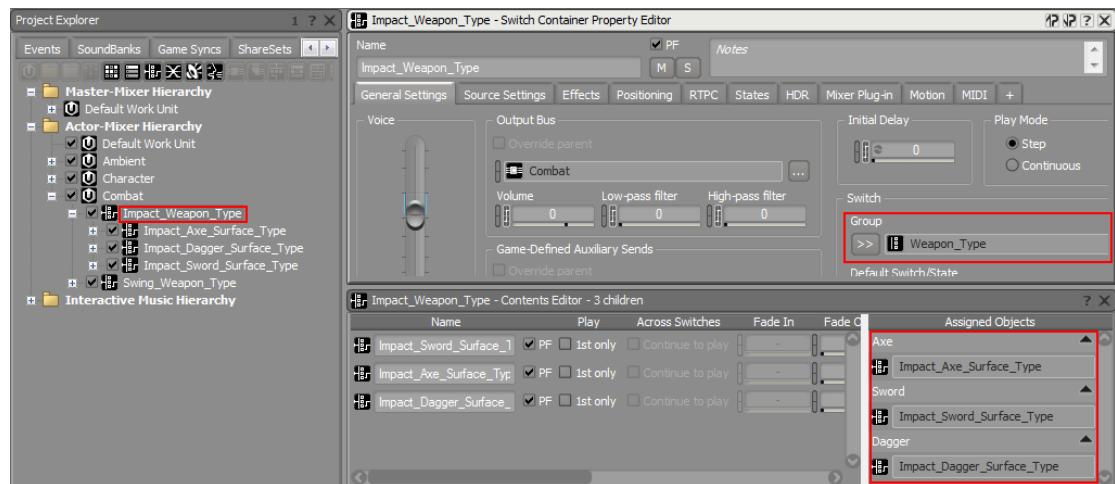
## Weapon Impact System

It should be assumed that throughout countless battles our hero will inevitably taste the sting of a blade. In order to add further detail to the impact system, we'll add a Switch for flesh to the surface material Switch Group.



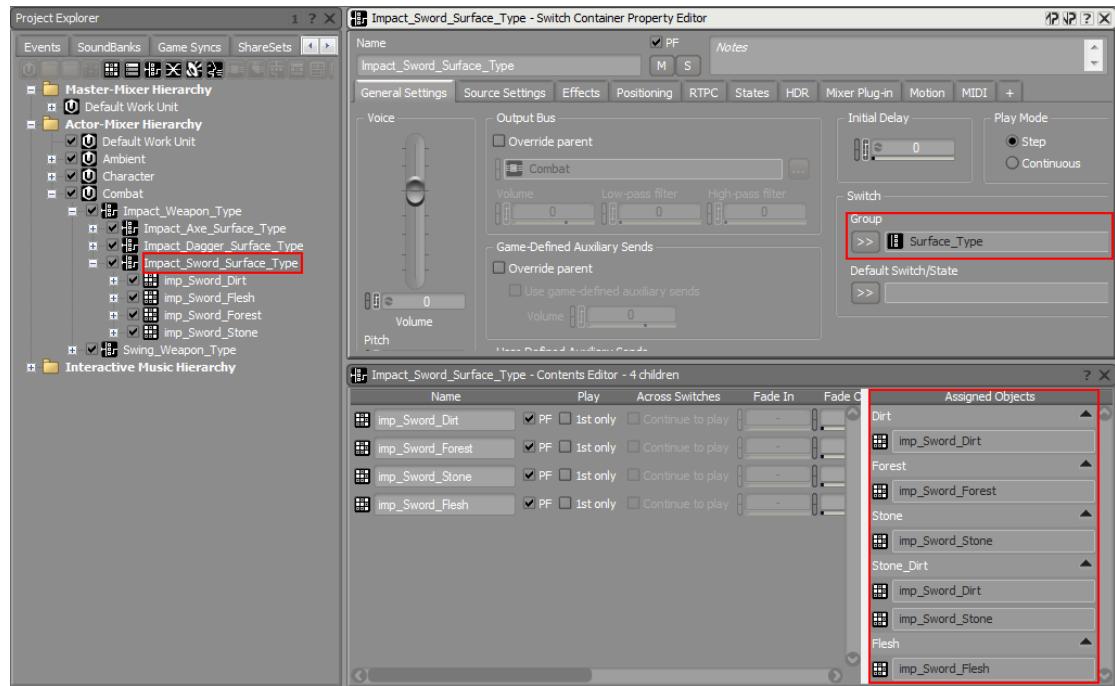
Switch for new surface type

Once switches have been defined in the game and material settings have been set, the properties of the game object being requested will drive the switches and play the appropriately assigned sound for the combination of weapon and material type.



Assigned weapon type objects

## Preparing for Combat



Assigned surface type objects

With the escalating number of surface materials in today's games, you could continue adding detail to the sound for as long as the memory budget allows. By utilizing the previously mentioned technique [Character 2.2] for sharing different surface material content between switches, you can help extend the content while increasing detail. With thoughtful sound design and well designed systems, you can overcome many limitations through the creative use of implementation.

## Attenuations for Player vs. NPC

Now that we have a fully switching impact system that can be used for any defined weapon in the game, it's important to begin establishing and subscribing attenuations to ensure that the sound appropriately falls-off over distance. While we've already discussed simple attenuations in the Ambient chapter, they'll be put to further use in helping to define the attenuations of combat sounds.

### High Alert

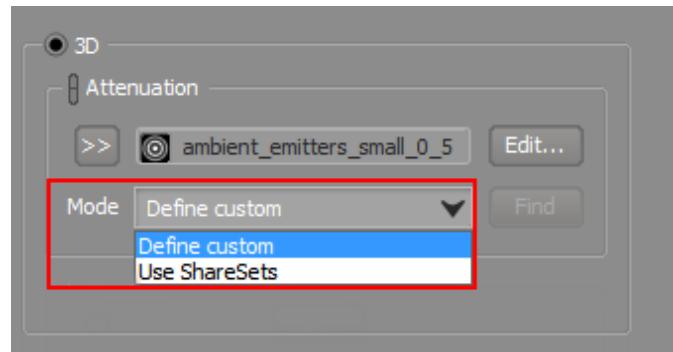
Before the battle has even begun, there is an opportunity to alert the player of an enemy's presence by allowing their idle sound and footsteps to be heard outside of the range where they would attack. Using sound to inform the player of approaching danger can help heighten awareness and prepare for the pending struggle. It starts with knowing at what distance an enemy will attack, and continues with an attenuation that allows the player to hear movement before combat begins. Let's assume that an enemy will engage the player within 15 game units. By extending the movement sounds' max distance for the attenuation to 20 game units, we can make sure the player has time to equip appropriately for the oncoming altercation.

### Listener Considerations

In most cases the listener is positioned at either the player or camera, or at some (possibly parametrized) interpolation between the two. The sound changes based on the proximity of the listener to the NPC, and the attenuation provides the falloff properties. This scenario is widely used in 3D games and well understood in game audio. However, choosing where to implement the listener in the game may require special considerations for player sounds.

While some situations may benefit from being played back in 2D on the player (such as surround magic effects), there are potential pitfalls that could cause undesirable results, specifically, in the common scenario where the camera moves through the world to highlight action taking place elsewhere in the environment. If a sound playing from the position of the player is set to 2D when this happens, the sound will persist even though the player is no longer part of the visual scene. In this way it's usually safest to continue treating the player attenuations in the same way we've begun with NPCs, as 3D sounds.

While you may start out sharing several of the same attenuations, a unique attenuation ShareSet can always be applied at any level of the hierarchy. Additionally, you can create a custom attenuation that will apply only to a single object. Changing the mode of a sound objects attenuation from Use ShareSets to Define Custom applies a unique instance of the attenuation that is not shared (or sharable) with any other sound object.



### Taking an attenuation ShareSet and defining custom properties for a sound object

While defining custom attenuations can be a good solution as an exception to the rule, there is more power and transparency to creating ShareSets which are all managed in one place.

Once you're gotten the feel for the way sound propagates in the game, you'll find yourself instinctively using attenuation ShareSets you've created for new sounds. By establishing well defined and flexible attenuations, a distance falloff model can be created that suits the game.

## Combat Summary

In the Combat chapter we've been working on establishing swings and impact based on weapon type. Along the way we looked into the functionality of SoundSeed Whoosh to replace the traditional audio file variation model with a powerful synthesis toolset. We also continued investigating the use of attenuations in conjunction with listener positioning and how they can work together to positively affect gameplay and help define propagation models.

**Throughout this chapter we have:**

- Discussed the aesthetic choices involved in designing the sound of different weapon types and implementations.
- Discussed the use of generative weapon swings using SoundSeed Whoosh to replace content based variations.
- Exposed different implementation techniques for impact sounds.

**Stepped through the process of:**

- Creating swing sounds for different weapon types using SoundSeed Whoosh.
- Creating a switching system for weapon swings based on weapon type.
- Creating a switching system for weapon impacts based on weapon and surface type.

**We've also touched on:**

- Scaling complexities of material based impact detection.
- Using different attenuations for the player vs. NPC.
- Considerations when choosing the position for the listener.

**Throughout this section we have created the following objects:**

- Swing Sound SFX objects for sword, axe, and dagger authored with SoundSeed Whoosh.
- A weapon swing switch system based on weapon type.
- A weapon impact switch system used to determine impacts based on weapon and surface type.

## **Combat Additional Resources**

[Video Tutorial - SoundSeed Introduction](#)

[Video Tutorial - SoundSeed Air Whoosh Overview](#)

---

## Chapter 4. Making Magic

Overview .....	87
Summing and Layering with Blend Containers .....	88
Creating Distance-Based Blend Tracks .....	89
Setting up a Game Parameter .....	89
Crossfading Between Containers on a Blend Track .....	93
Section Summary .....	95
Real-Time Parameter Control (RTPC) .....	96
Using Real-Time Effects .....	98
Unleashing Dynamic Synthesis .....	102
Wwise Synth One .....	102
Modulators .....	106
Modulator LFO .....	106
Modulation Envelope .....	112
Magic Summary .....	118
Magic Additional Resources .....	119

## Overview

Whether casting spells to fend off a grue attack, getting blasted by dark wizards, or summoning succubus from the nether realms, the rules of magic have been handed down from generation to generation by sorcerers older than time itself. You can experiment with sound in the Wwise authoring application to deploy your sonic alchemy and unleash a powerful torrent of audio mayhem.

This chapter will take you through the process of:

- Summing and layering with the Blend Container
- Leveraging blend tracks as part of a distance based perspective model
- Using real-time effects
- Using Real-time Parameter Control (RTPC)
- Creating a synthetic element using Wwise Synth One
- Using Modulator Envelopes and LFOs to diversify properties of a sound

The approach for creating unique magic effects begins with inspiration. From there, audio files and synthesis define the sound palette for the eventual effect. You can seriously up the creativity quotient in your effects by taking individual sound element variations, or layers of sound, and recombining them using blend containers. Then draw on parameters from the game to add dynamic variability.

## Summing and Layering with Blend Containers

As discussed in [Chapter 1, Setting the Ambient Stage](#), a blend container simultaneously plays the sum of its contents. By taking individual sound element variations, or layers of sound design, and recombining them using the blend container, startling new effects and composites can be achieved that help extend the content. This is especially true when coupled with randomized properties or additional effects available at any level of the Actor-Mixer Hierarchy.



The Magic blend container is comprised of multiple random containers

## Creating Distance-Based Blend Tracks

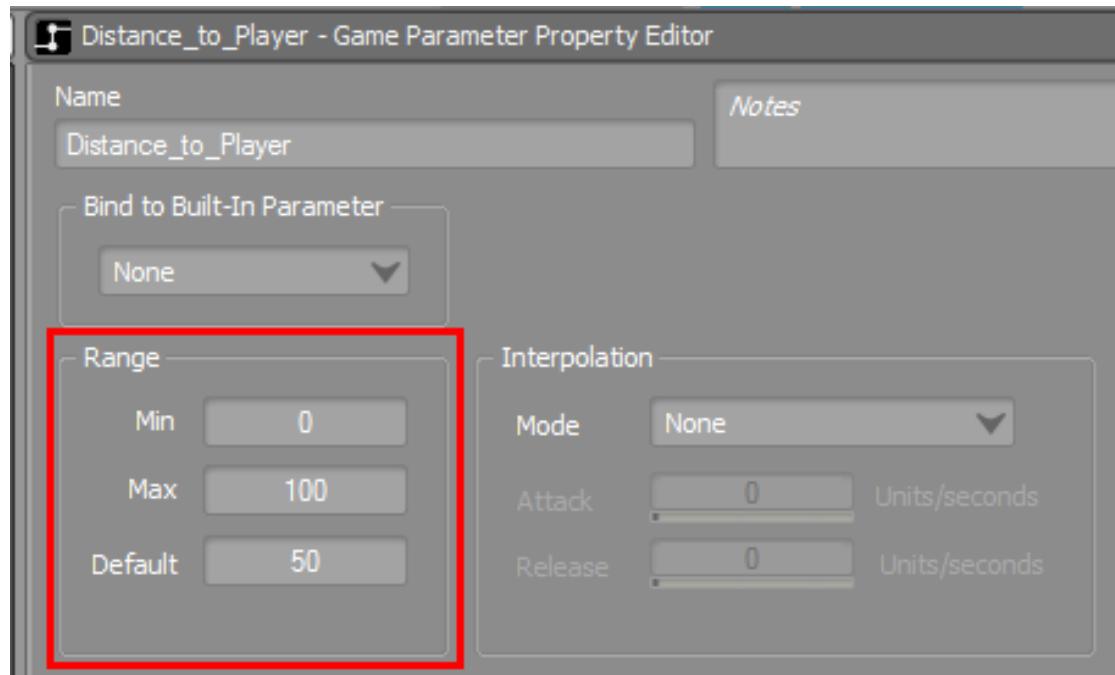
Another way to use blend tracks is to create a complex distance-based blend to diversify perspectives or, simply put, to change the sound content based on distance. This technique can be quite effective when implementing ranged weapons, moving objects, or in this case, magic effects.

We'll be creating a blend container that changes the sound of magic impact blasts near the player, and from a far off distance perspective. When the magic blast event is triggered, the perspective (or combination of perspectives) is played based on the blast game object's distance to the player. So, the content will have more high frequency detail and sense of danger at a close distance from the player and a slightly low pass filtered and reverberant sound when far away.

Magic effects are often associated with particle or visual effects within a game environment. While many particle effects are comprised of different techniques that contribute to the resulting visual effect, there is usually a point of origin, referred to as the game object that travels as the center of the effect in 3D space and can be used to attach sounds. Using the distance to player as part of a dynamic effect can help give a sense of movement to the sound and increase the dramatic effect by lending a positionally significant modifier.

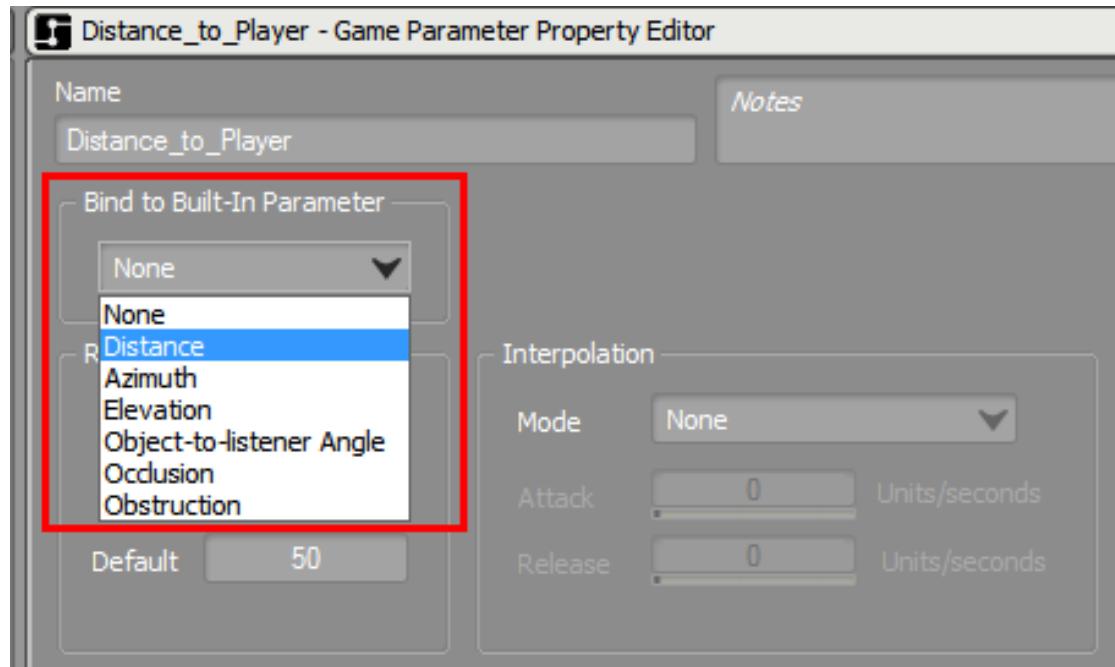
### Setting up a Game Parameter

We can get information about the game object's proximity to the player from the game engine and use it through a new game parameter called `Distance_to_Player` throughout the project.



Setting up the Distance\_to\_Player game parameter

It's easy to get distance info and other values directly from the Wwise audio engine. You can subscribe to a built-in value for the distance between the listener and the game object from a list of built-in values calculated inside of the audio engine using the "Bind to Built-In-Parameter" drop-down menu.



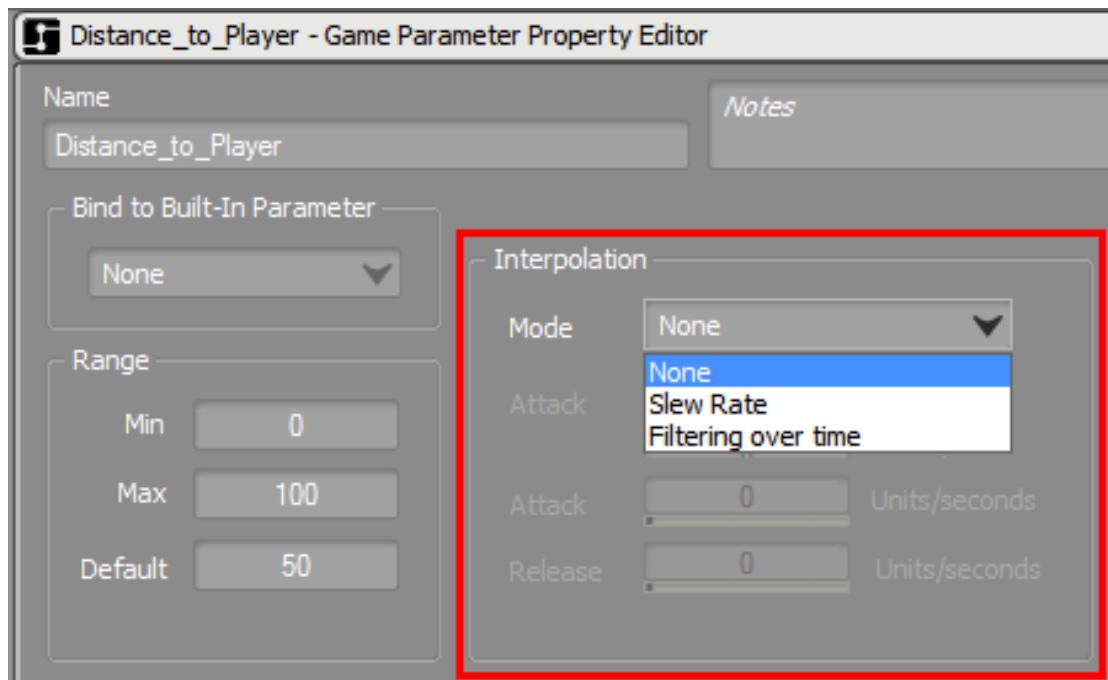
Binding the Built-In Distance value to the Distance\_to\_Player Game Parameter

The following built-in parameters are calculated by the audio engine and are available to be bound to game parameters:

- Distance - Distance to game object.
- Azimuth - Horizontal angle.
- Elevation - Vertical angle.
- Emitter Cone - Angle between object's orientation and the listener.
- Occlusion - As set by the game on the game object.
- Obstruction - As set by the game on the game object.
- Listener Cone - Angle between listener's orientation and the emitter.

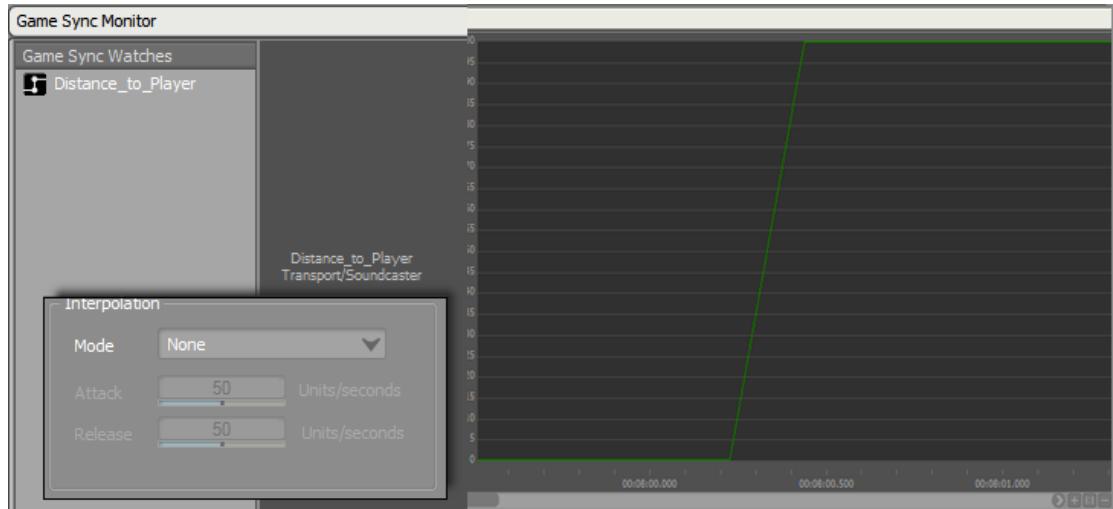
Interpolation can be used to apply a kind of "smoothing" to the values sent by the audio engine. The modes (other than None) let you modify the rate (in Units) or time (in seconds) when target values go upward (Attack) or downward (Release). Adding an Interpolation Mode helps keep values from the game engine from "jumping" or "stepping" too quickly which may cause, for example, a voice volume to abruptly increase in volume. The Interpolation properties allow for controlled transitions between Game Parameter values sent from the game or from within the Wwise audio engine.

In this example, the behavior of the Distance\_to\_Player game parameter is modified using the Interpolation properties available in the Game Parameter Property Editor.

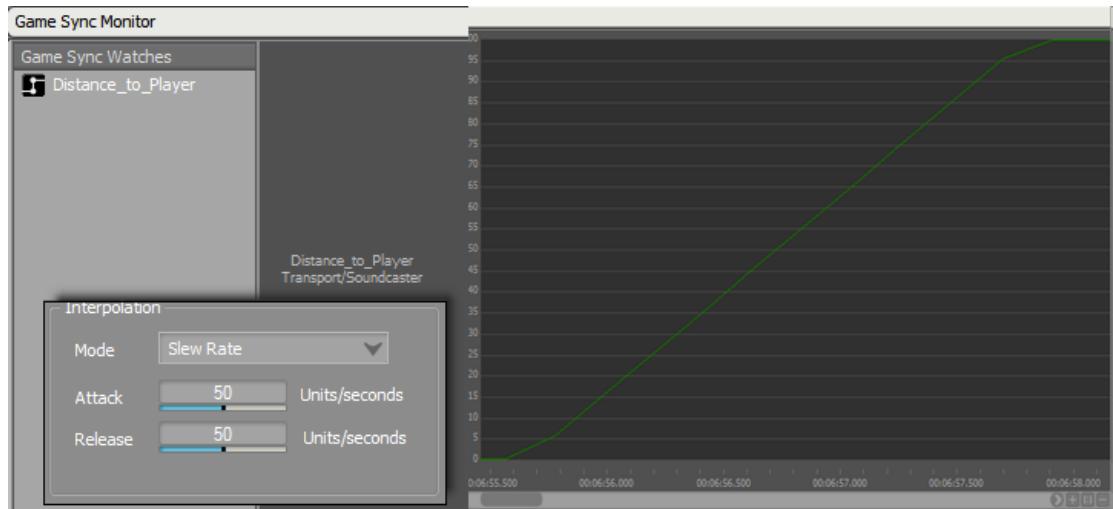


Setting the Game Parameter Interpolation Mode

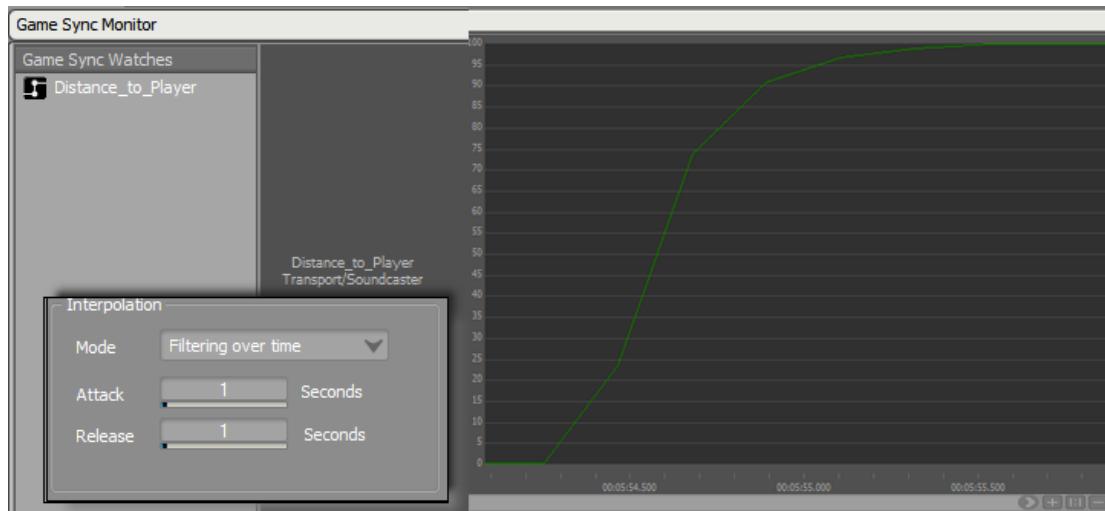
Interpolation Modes can be set using the Mode drop-down list and include:



- **Interpolation Mode - None:** Jump straight to the target value.



- **Interpolation Mode - Slew Rate:** Limit the game parameter variation rate to the specified Attack and Release rate. (Units)



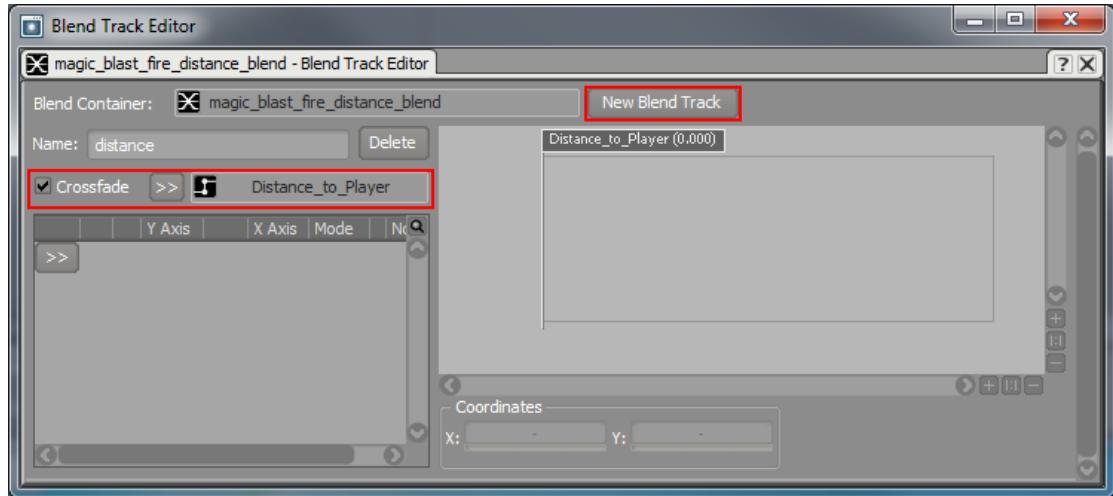
- **Interpolation Mode - Filtering Over time:** Filter the game parameter current value to target 99.5% of the target value within the specific Attack/Release time. (Seconds)

The Interpolation Mode for the Distance\_to\_Player Game Parameter will be set to "Slew Rate" with a Attack (in Units) of 50 and Release (in Units) of 50 to produce the desired effect.

### Crossfading Between Containers on a Blend Track

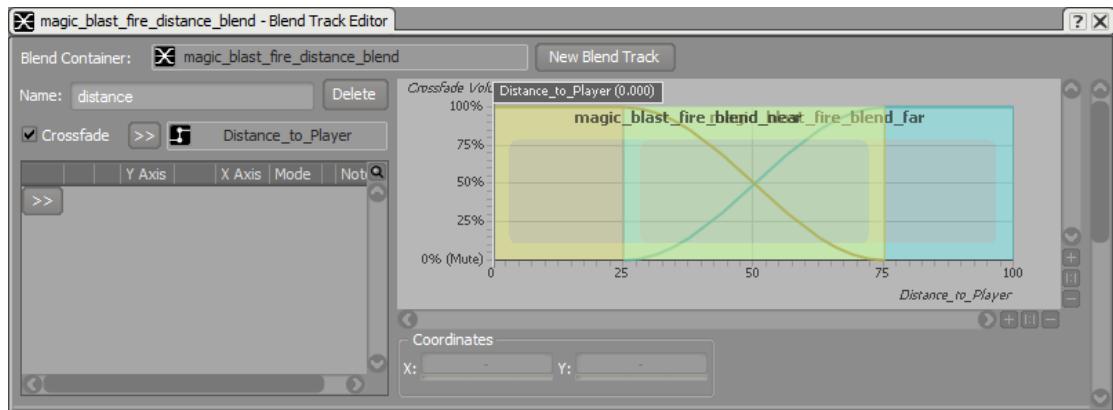
Now that there is a way to track how far an object is from the player, we can use this information to drive the Crossfade between various containers on a blend track inside of a blend container we have named magic\_blast\_fire\_distance\_blend.

As we did for the blend containers created for the ambient system, a new blend track will be created with a Crossfade enabled based on the new game parameter for Distance\_to\_Player.



Creating a new blend track with Crossfade enabled based on the Distance\_to\_Player game parameter

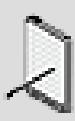
Randomized containers can then be added to the assigned objects area in the distance blend track. As previously mentioned, the order of these containers in the assigned objects section is reflected in their positioning on the blend track.



Adding containers to the blend track in the correct order

After the information is passed successfully from the game, the Distance\_to\_Player game parameter controls the Crossfade between containers, playing only the container (or containers) beneath the game parameter cursor.

- When the game object plays the magic blast event within 0-25 units from the player, the magic\_blast\_fire\_blend\_near is heard alone.
- When the game object plays the magic blast event within 25-75 units from the player, a blend of both perspectives is heard.
- When the game object plays the magic blast event within 75-100 units from the player the magic\_blast\_fire\_blend\_far is heard alone.



### Designer Note

While similar distance-based blending can be done using multiple attenuations, there are workflow benefits to managing these within blend tracks.

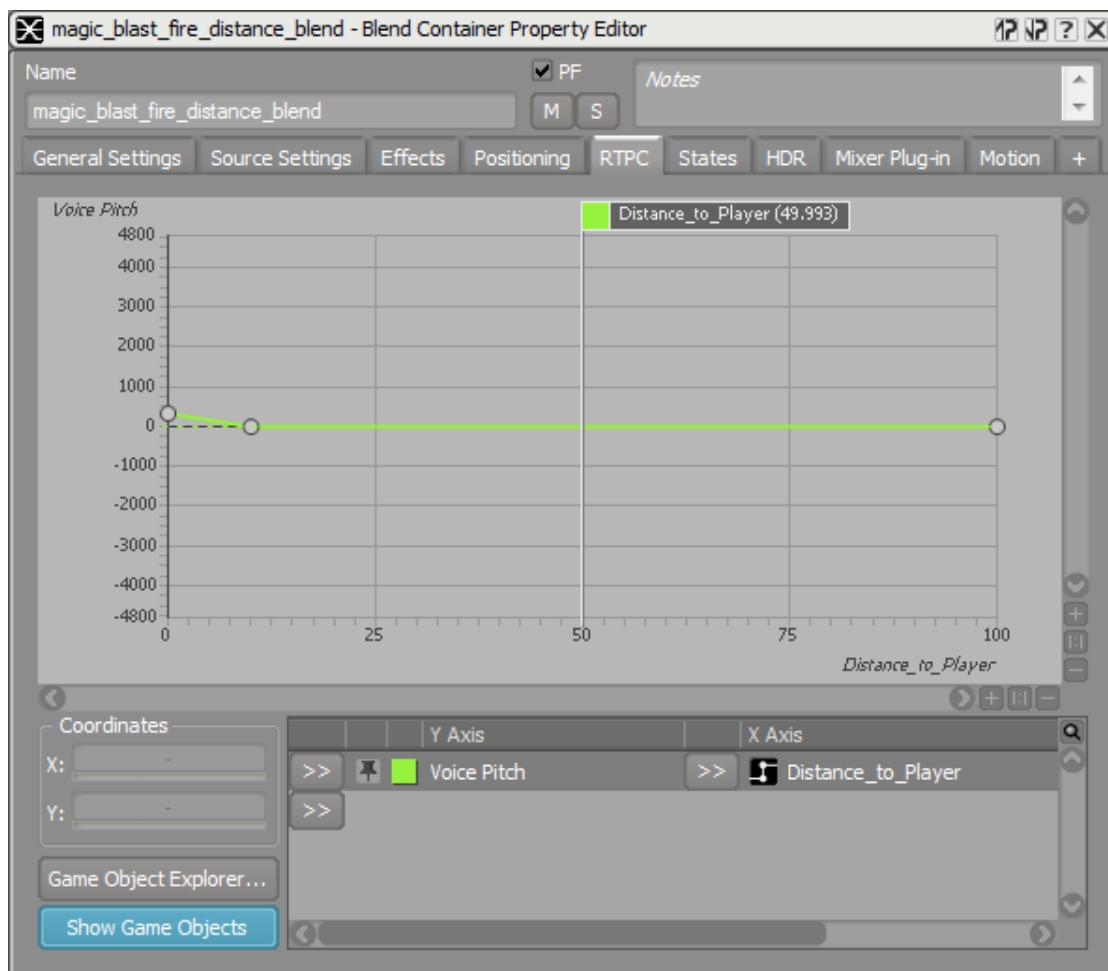
### Section Summary

We have successfully implemented magic blasts that change between audio content dependent on the location and distance of the game object from the player. This technique is best used when there is a need to represent different perspectives or levels of detail at the content level. While simple filtering can also be applied to help change the tone of the resulting sound, you can add richer levels of detail by designing the content appropriately.

## Real-Time Parameter Control (RTPC)

To push magic effects further into dynamic territory, you can employ Real Time Parameter Control (RTPC) for any sound object in the Actor-Mixer Hierarchy or any audio bus in the Master-Mixer Hierarchy. RTPC can be accessed in the Property Editor for any sound object, and new modifiers can be added using the Selector button.

Using the `magic_blast_fire_distance_blend`, a pitch curve can be added using the `Distance_to_Player` game parameter as part of the object's RTPC. The distance curve value can be used to dynamically affect the sound of the magic blast by modifying pitch based on the magic blast's proximity to the listener. Similar results can be obtained using RTPC within the blend track.



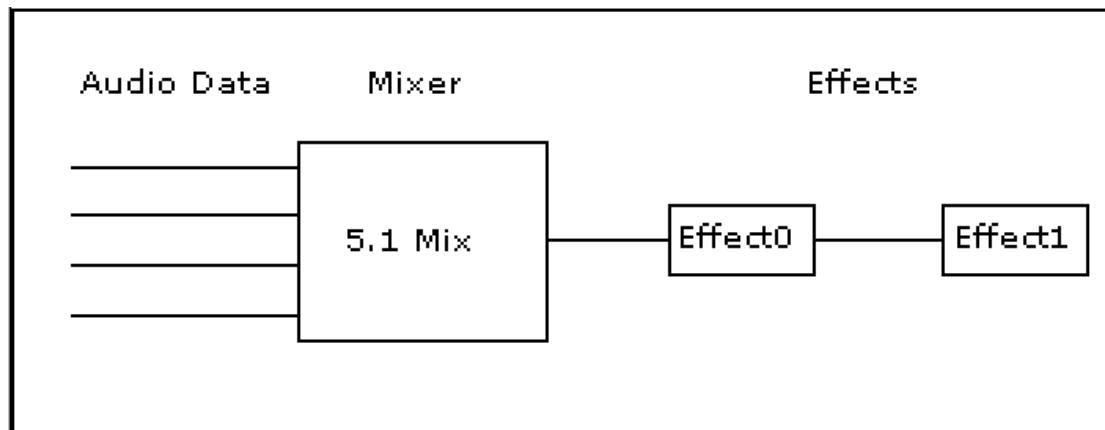
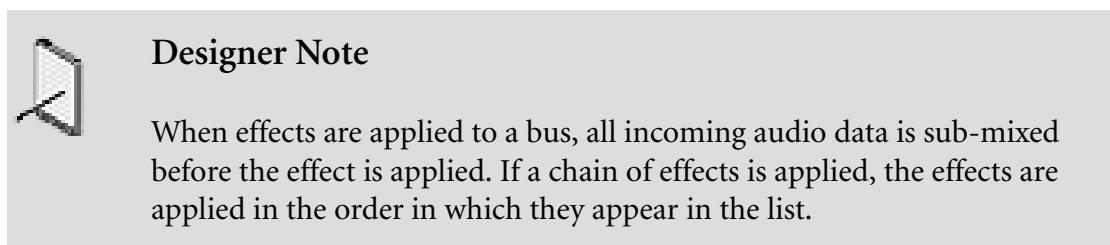
RTPC using the `Distance_to_Player` game parameter to control the amount of pitch

RTPC further modifies properties of a sound object by opening dynamic possibilities for making your content change based on information from the game. In this example, we have taken and modified the magic blast, pitching it up as it approaches the player. This gives the approach of the magic blast an immediacy that is reflected by its pitch.

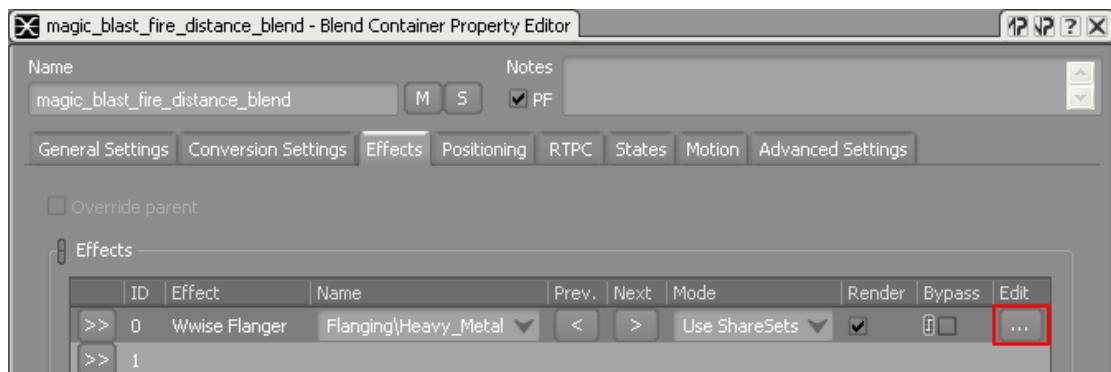
## Using Real-Time Effects

In addition to the unique features within the blend container, Wwise ships with a suite of DSP Effects that can be used to modify content after it's been imported, enabling additional sound design possibilities from within the authoring application.

Available in the Property Editor of every audio object, and additionally within the Master-Mixer hierarchy, these effects can be added to modify sounds using fixed settings or dynamically at runtime using game parameters.

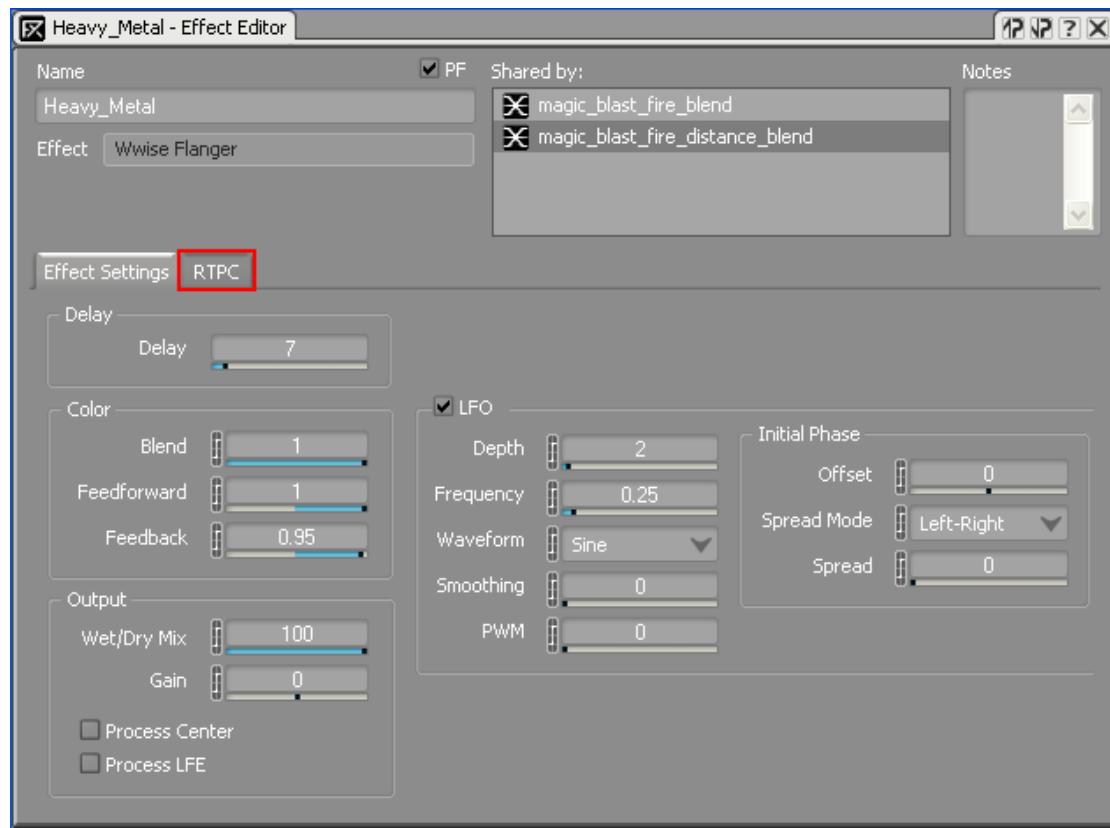


RTPC for effects can be found in the Effect Editor after clicking the Edit button for an enabled effect.



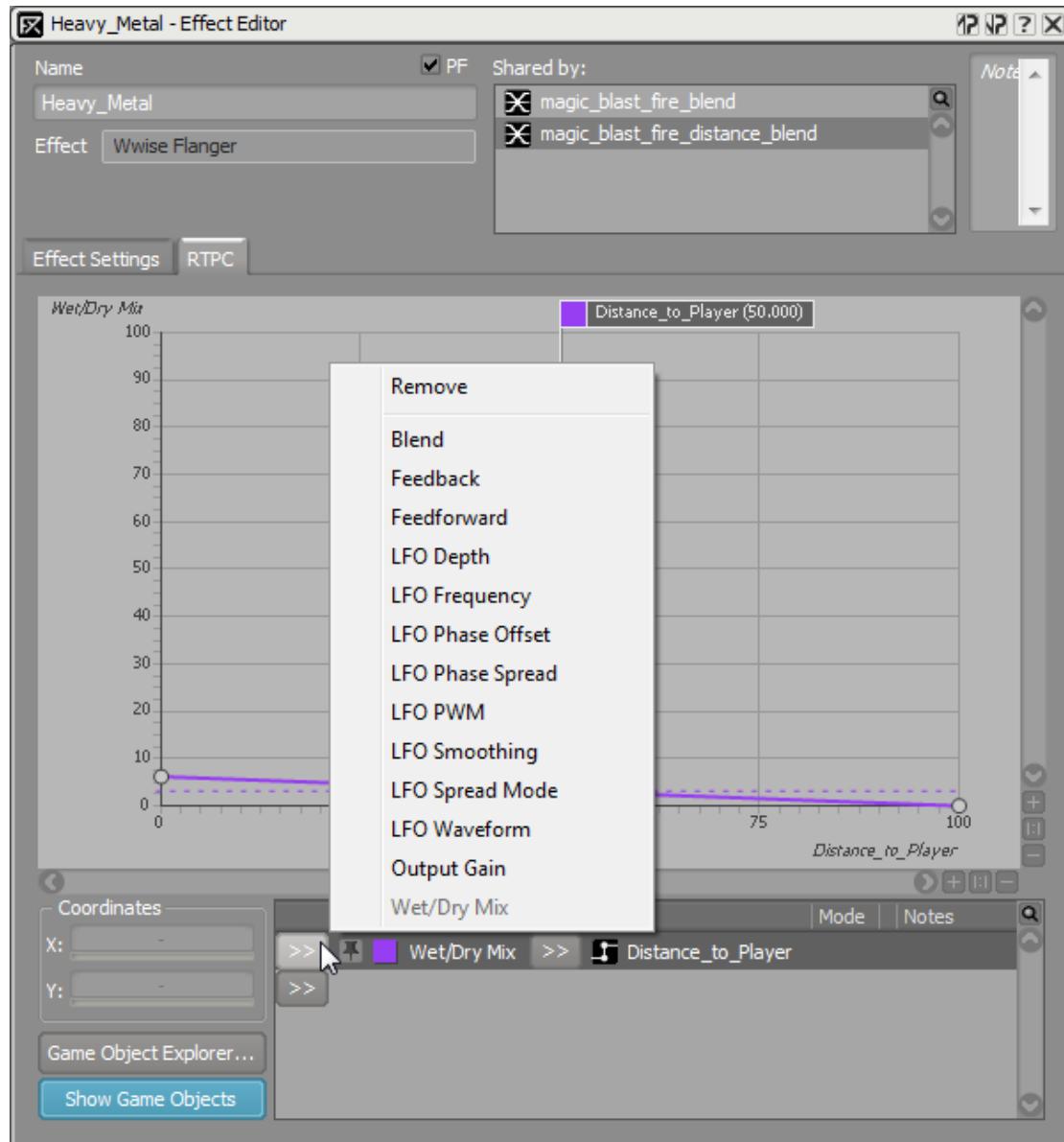
The Edit button is used to access the effect editor

The Effects Editor displays all of the settings of an enabled effect, including access to any properties that can use RTPC.



The effects settings panel and access to the RTPC tab

From the RTPC tab, you can create relationships between effects settings and game parameters to produce unique effects driven by information coming from the game.



RTPC settings added to the Distance\_to\_Player game parameter

Using game parameters in conjunction with DSP effect settings opens up a whole new world of dynamic content manipulation. You can use the game parameters in conjunction with effects in the following ways:

- Add a slight flange to a rushing river sound to make it sound eerie, only from a distance.
- Add stereo delay to all vocalizations when the player is about to die in combat, based on health.
- Add tremolo to an air conditioner fan loop that changes, based on the rotation speed of the fan.

Whether you're designing iconic spellcasting sounds that need to be instantly recognizable or gaining diversity through recombination, the power of real-time parameter control and dynamic DSP effects can create special effects that are only possible within the environment of an interactive game world.

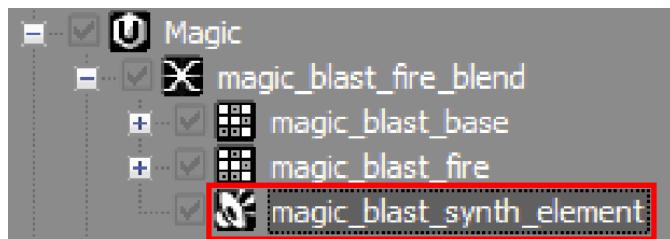
## Unleashing Dynamic Synthesis

Another way to replace, augment, or supplement audio files is with the Wwise Synth One plug-in. Synthesis provides a flexible and dynamic audio platform that doesn't depend on system memory (RAM); it uses CPU to generate sounds at runtime instead. Chances are good that you're already using synthesis as part of your sound design process within a DAW. With the power of Wwise Synth One, synthesis can become a part of your pipeline within the Wwise authoring environment.

Game Parameters, MIDI Messages, Modulator LFO, and Modulator Envelope can be added as part of any Sound Object, Audio Bus, or Aux Bus RTPC and then used to modify properties of Wwise Synth One, SoundSeed Whoosh, SoundSeed Air, or any DSP plug-in. Being able to manipulate sound using synthesis at runtime gives you greater flexibility to modify properties using parameters from the game or from within the Wwise authoring environment.

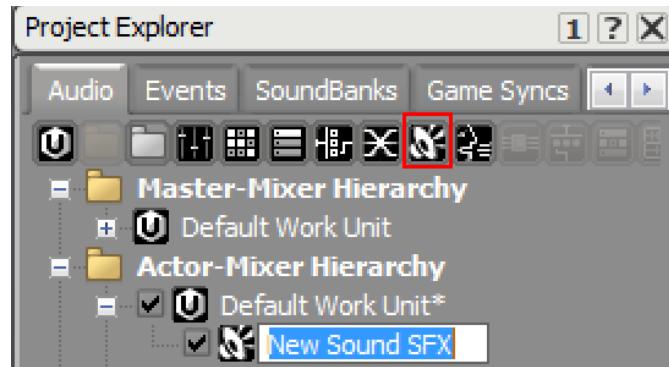
### Wwise Synth One

We'll begin by creating a `magic_blast_synth_element` that contains the Wwise Synth One plug-in as part of the `magic_blast_fire_blend` created in Section 3.



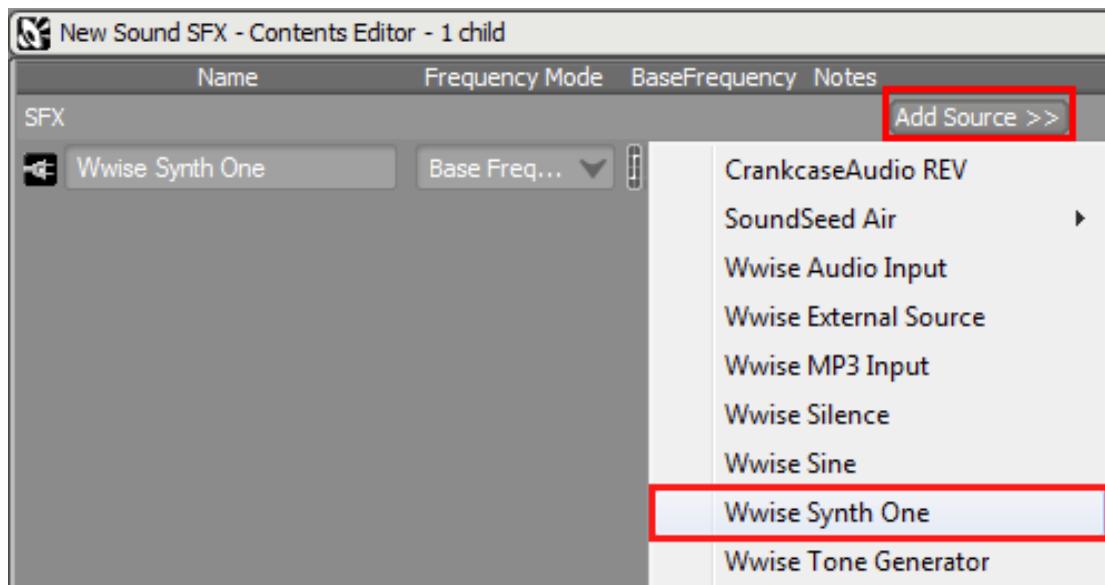
Sound SFX containing the Wwise Synth One plug-in inside the `magic_blast_fire_blend`

The Wwise Synth One plug-in can be added as an input source to a Sound SFX (or Sound Voice) in the same way that the Silence plug-in was added in Chapter 1's [Using the Silence Plug-in for Weighted Randomization](#). To begin, add a Sound SFX object to the default work unit by clicking the Sound SFX icon in the Project Explorer toolbar. A new Sound SFX is created. Alternatively, Sound SFX can be created from the contextual menu or by using shortcut keys.



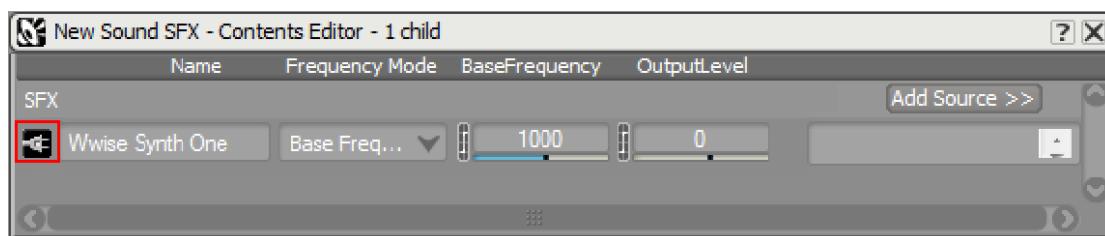
Creating a Sound SFX from the Project Explorer toolbar

After double-clicking to select the new Sound SFX, you can add the Wwise Synth One plug-in using the Add Source menu in the Contents Editor.

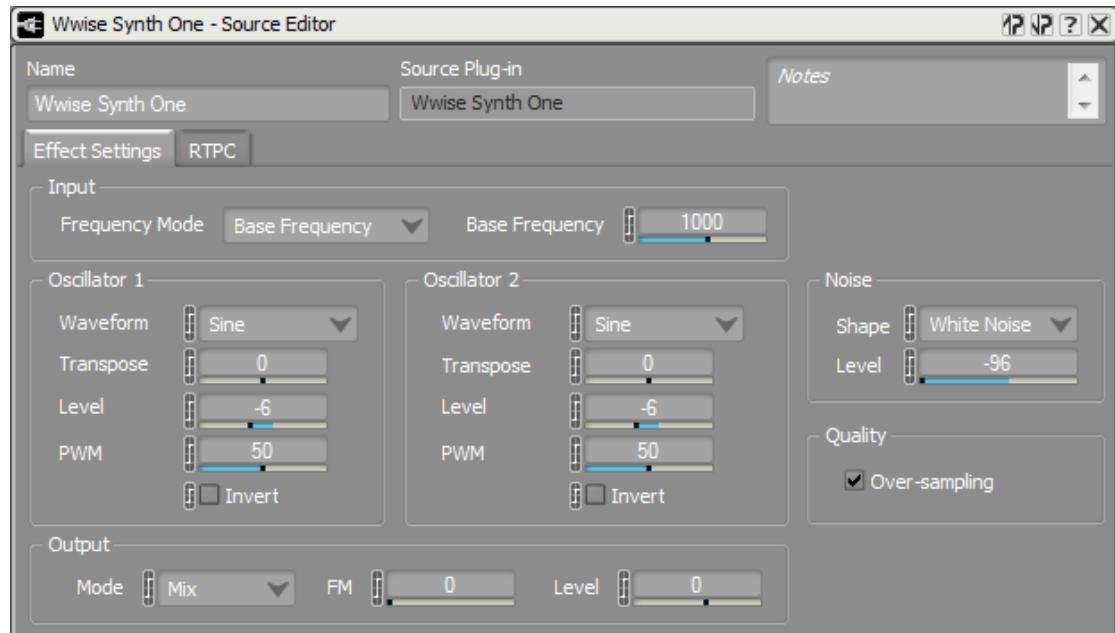


Creating a Wwise Synth One source plug-in for a sound object

The Source Editor can be accessed in the Contents Editor by double-clicking the plug-in icon.



Wwise Synth One plug-in icon



Wwise Synth One - Source Editor



### Designer Note

Features of the Wwise Synth One plug-in are described in Chapter 8's [Setting up Wwise Synth One](#) section.

We'll begin by creating an additional synthetic element for the `magic_blast_fire_blend` Blend Container that will play in addition to the audio files already established earlier in this chapter. This element adds to the synthetic characteristics of the sound and is modified using a Modulator Envelope on the Output Level property of the Wwise Synth One plug-in. Additionally, Modulator LFOs and the Distance\_to\_Player Game Parameter will be used to modify various properties of the Wwise Synth One plug-in in conjunction with a Modulator Envelope.

Now add the `magic_blast_synth_element` that contains the Wwise Synth One plug-in inside the `magic_blast_fire_blend` created in Section 3.

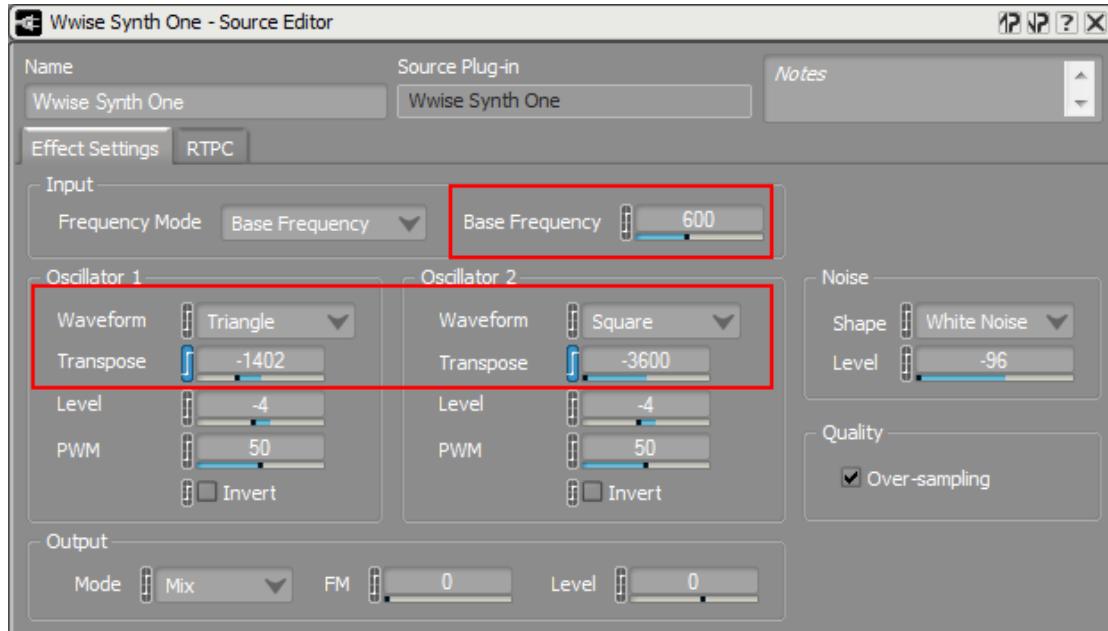


Sound SFX containing the Wwise Synth One plug-in inside the magic\_blast\_fire\_blend

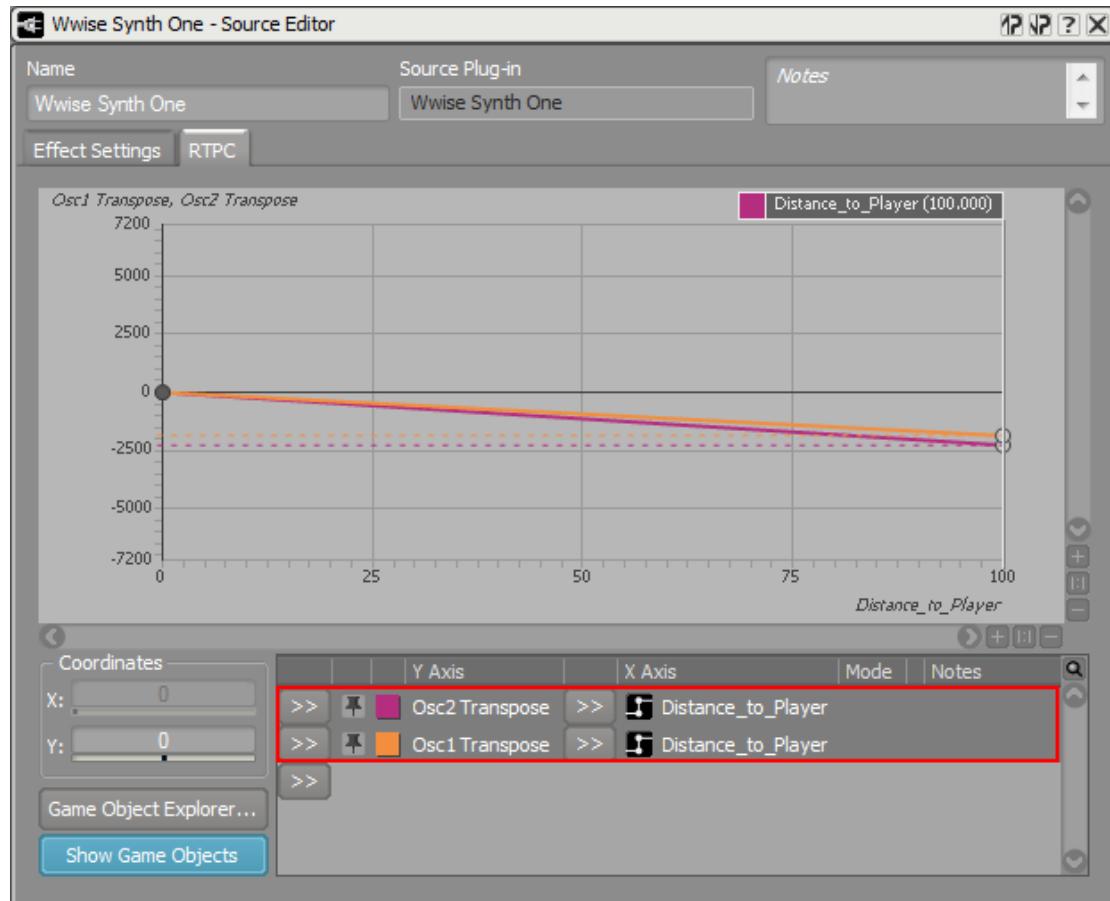
The magic\_blast\_synth\_element will start out with a low base frequency and a combination of Triangle and Square waveforms. Each Oscillator is then further transposed and modified using the Distance\_to\_Player Game Parameter to increase the transposition as the magical blast gets closer to the player.

 **Designer Note**

You can find more information on Game Parameters in Chapter 1's [Introduction of a Day and Night Cycle](#).



Wwise Synth One Base Frequency and Waveform properties



Wwise Synth One Oscillator 1 & Oscillator 2 Transpose property RTPC

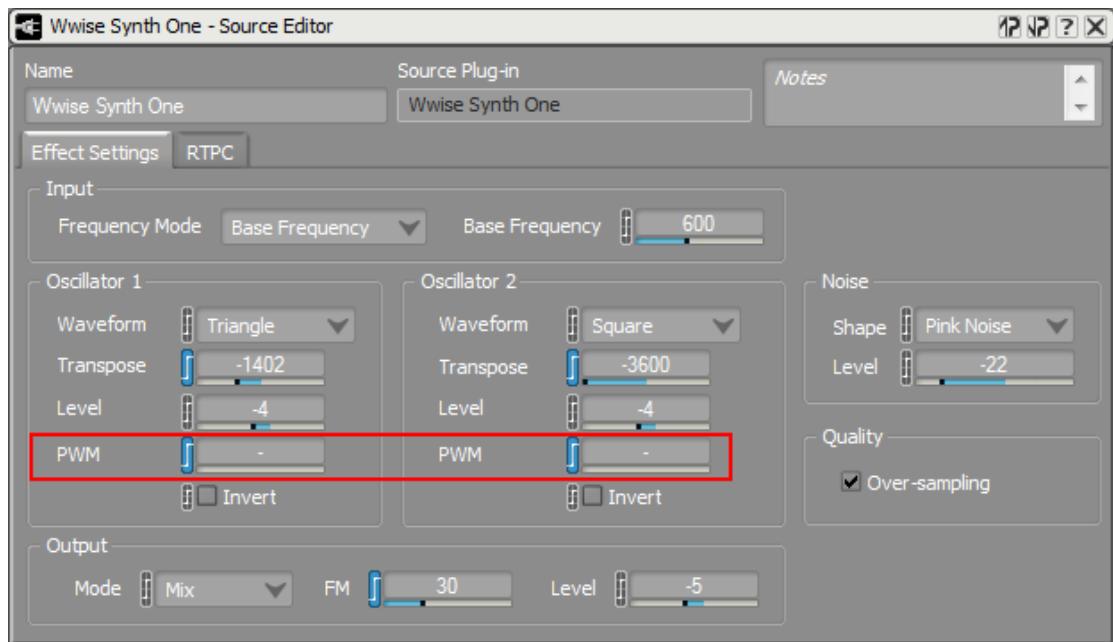
## Modulators

Parameterizing properties throughout a Wwise project can become a powerful tool for dynamic creation. Imagine modifying the pitch of the amb\_tree\_rustle ambient emitter loop by randomizing the properties of a Low Frequency Oscillator (LFO) or varying the length of an audio file by applying an Envelope with a randomized decay time to the voice volume. In addition to Game Parameters, there are a suite of Modulators that can be used to modify an object's properties. These include: LFOs, Envelopes, and MIDI Messages.

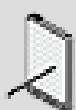
### Modulator LFO

The Modulator LFOs (Low Frequency Oscillator) can be used to create a modulation of property values over time. When added to an RTPC, a Modulator LFO modifies the values of a property between a range of values. Additionally, the values of the LFO properties can be further parameterized or randomized to achieve a high-level of variability.

To diversify the synth for the purpose of the magic\_blast\_fire\_blend element, a Modulator LFO added as an RTPC to each Oscillators PWM (Pulse Wave Modulation) property gives the sound extra "movement" and vibration.

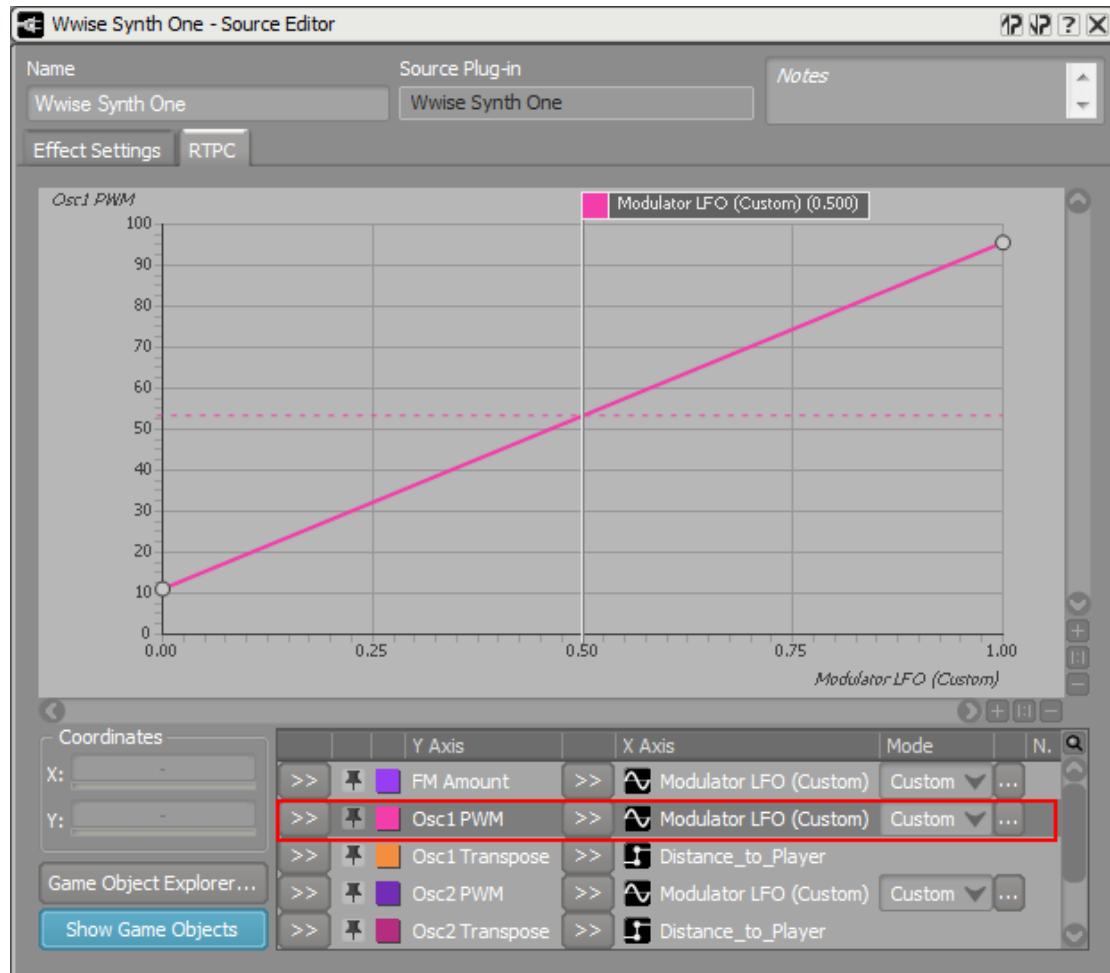


Wwise Synth One PWM (Pulse Width Modulation) properties attached to LFO (settings can be edited in the RTPC tab)



### Designer Note

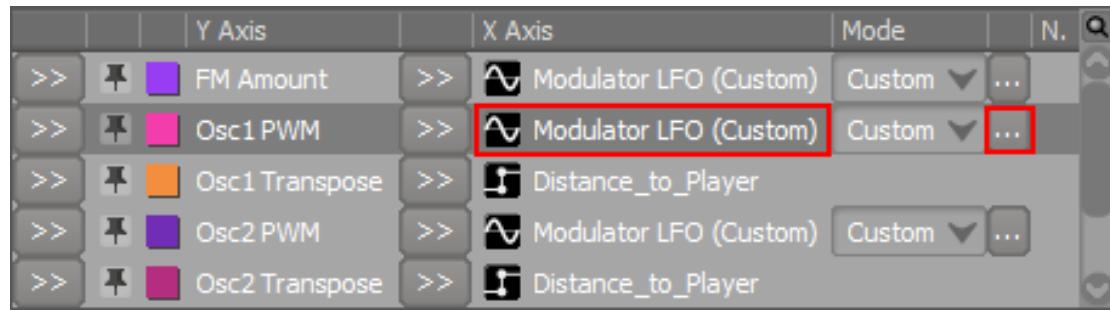
In Wwise, some properties are additive (Voice Volume, Voice Pitch, and so on...), and some are exclusive. When adding an LFO on the additive properties, the LFO modulation is added to the current value of the property. When adding an LFO on the exclusive properties, the LFO modulation replaces the current value of the property. Properties modified by an LFO are represented by a dashed line ("--") in the Property field.



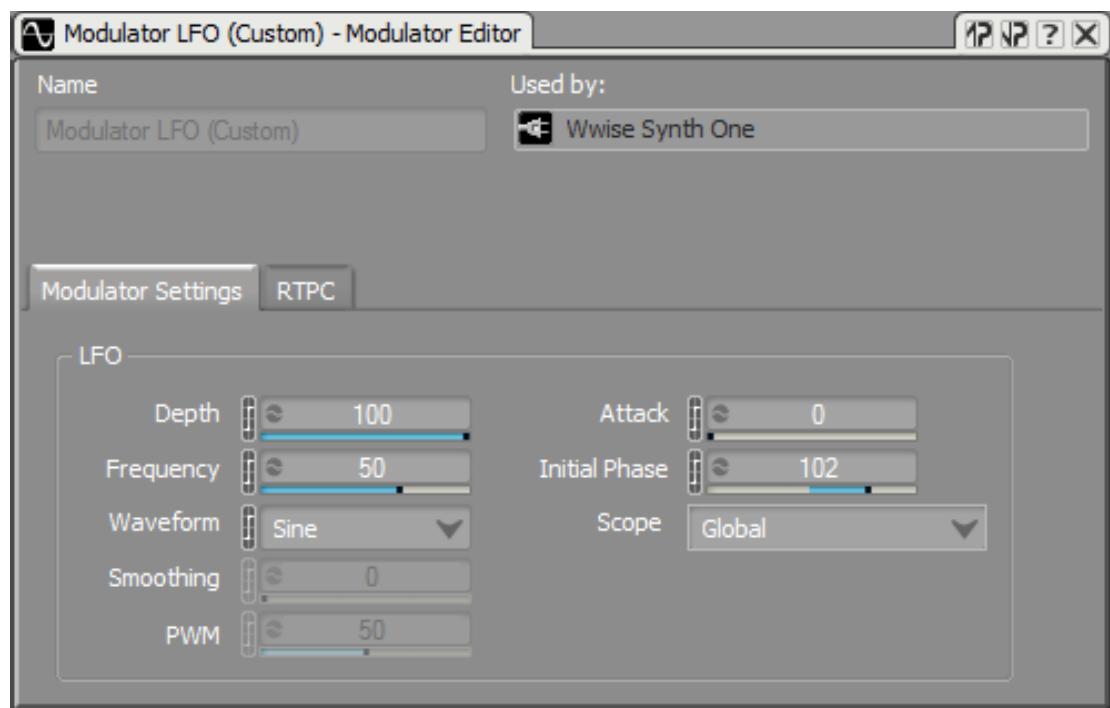
Wwise Synth One Oscillator 1 showing the effective range of the PWM property modulated by the LFO.

LFOs (Low Frequency Oscillator) are used to create modulation of property values over time. The minimum and maximum RTPC values of the LFO Envelope in the Sound Property Editor affect the base and top of the LFO (nil and peak values). Swapping the min and max values effectively phase reverse the LFO.

Access the Modulator Editor by double-clicking the Modulator LFO in the RTPC dialog or by clicking the [...] button.



Accessing the Modulator Editor by double-clicking the Modulator LFO in the RTPC dialog or by clicking the [...] button.

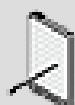


**Modulator LFO Modulator Editor**

The properties of the Modulator LFO are:

- **Depth**
  - The amplitude variation of the oscillator (in percentage). Maximum amplitude is 1.0.
  - Default value: 100
- **Frequency**
  - The number of cycles per second (in Hz).
  - Default value: 1
- **Waveform**
  - The shape of the modulator (sine, triangle, square, sawtooth).

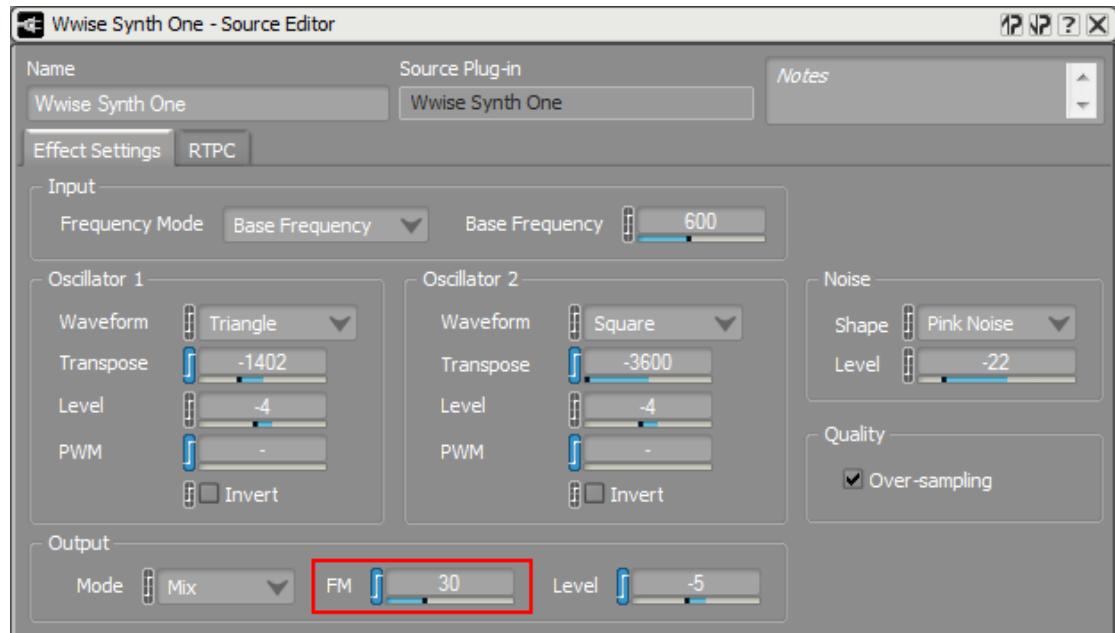
- Default value: Sine
- **Smoothing**
  - Low-pass filter over the waveform to smooth hard edges (in percentage).
  - Default value: 0
- **PWM (Pulse Width Modulation)**
  - The width of the pulse wave, only applies to the Square waveform (in percentage).
  - Default value: 50
- **Attack**
  - The time it takes for the oscillator to reach full amplitude (in seconds).
  - Default value: 0
- **Initial Phase**
  - The initial phase of the oscillator waveform (in degrees).
  - Default value: 0
- **Scope**
  - Controls how LFOs are created:
    - **Voice:** An instance of an LFO is created for every voice of the synth when used in MIDI context.
    - **Note/ Event:** An instance of an LFO is created for every playing instance of the synth when used in MIDI context.
    - **Game Object:** An instance of LFO is created for each game object instance in relation to the game engine.
    - **Global:** A single LFO is created for the whole project and used globally by all instances.
  - Default value: Voice



### Designer Note

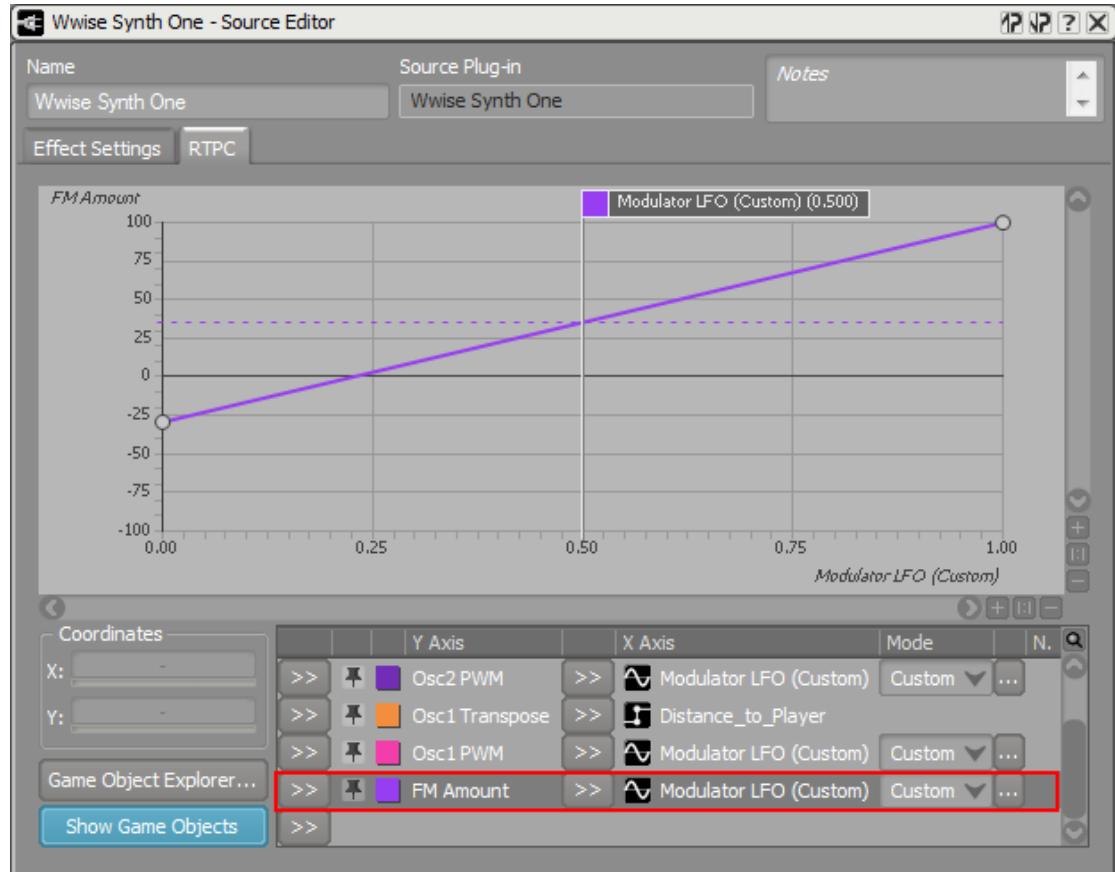
Additional information for working with LFOs can be found in the Wwise Help Document: [Wwise Help > Interacting with the Game > Working with RTPCs > Working with LFOs](#)

Additionally, Frequency Modulation (FM) between the two Oscillators is used to "soften" the sound and helps it blend better with the sample based components of the magic\_blast\_fire\_blend.

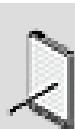


Wwise Synth One FM (Frequency Modulation) property

The "FM Amount" property is further modified by an LFO Modulator.



Wwise Synth One FM Amount property RTPC



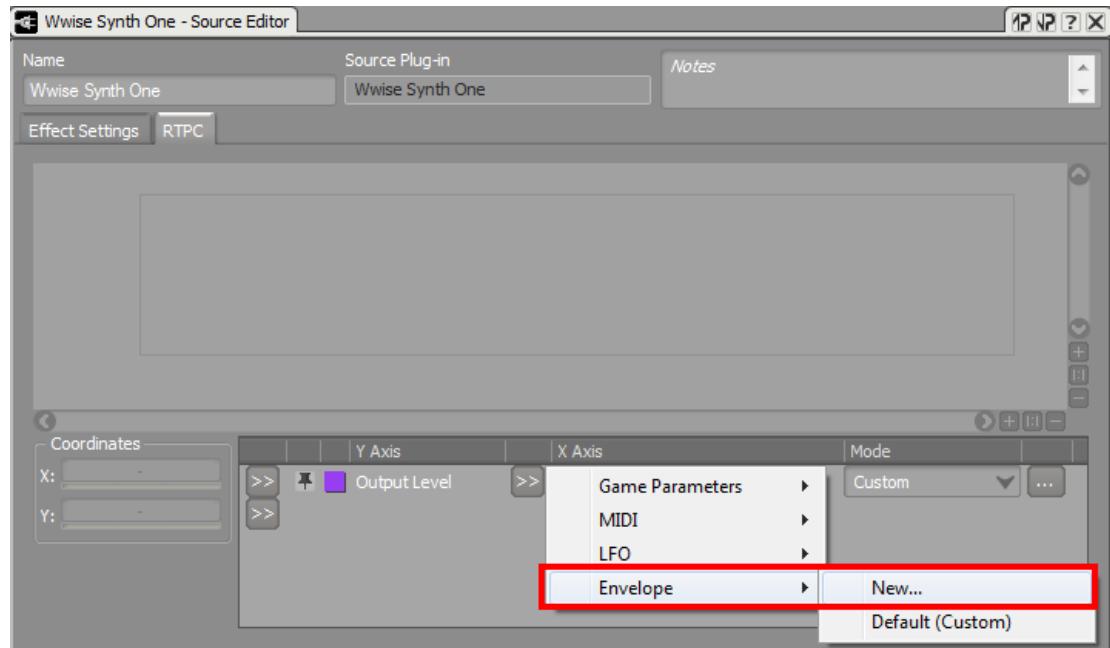
### Designer Note

LFO objects can be created as Custom or ShareSet. Custom objects are stored in-place, directly inside the object that has it. ShareSets are stored in a separate work-unit and can be re-used across objects.

### Modulation Envelope

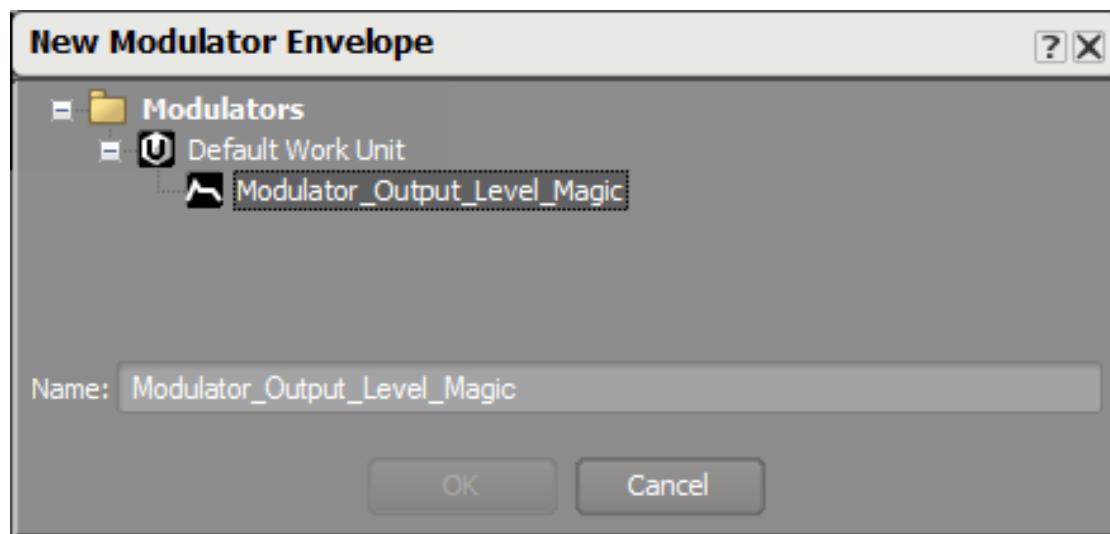
You can control the amplitude envelope of the Wwise Synth One by adding an Output Level RTPC modified by a Modulator Envelope. The Modulator Envelope provides properties to modify the Attack, Decay, Sustain, Release (ADSR) in addition to Attack Curve, Sustain Time. You can also use the Modulator Envelope to stop playback with the envelope. Many of the properties of the Modulator can be randomized and modified by RTPC.

From the RTPC Tab for the `magic_blast_synth_element`, use the Selector to add the Output Level parameter which will be modified by a new Modulator Envelope ShareSet.



Adding a Modulator Envelope to the Output Level RTPC

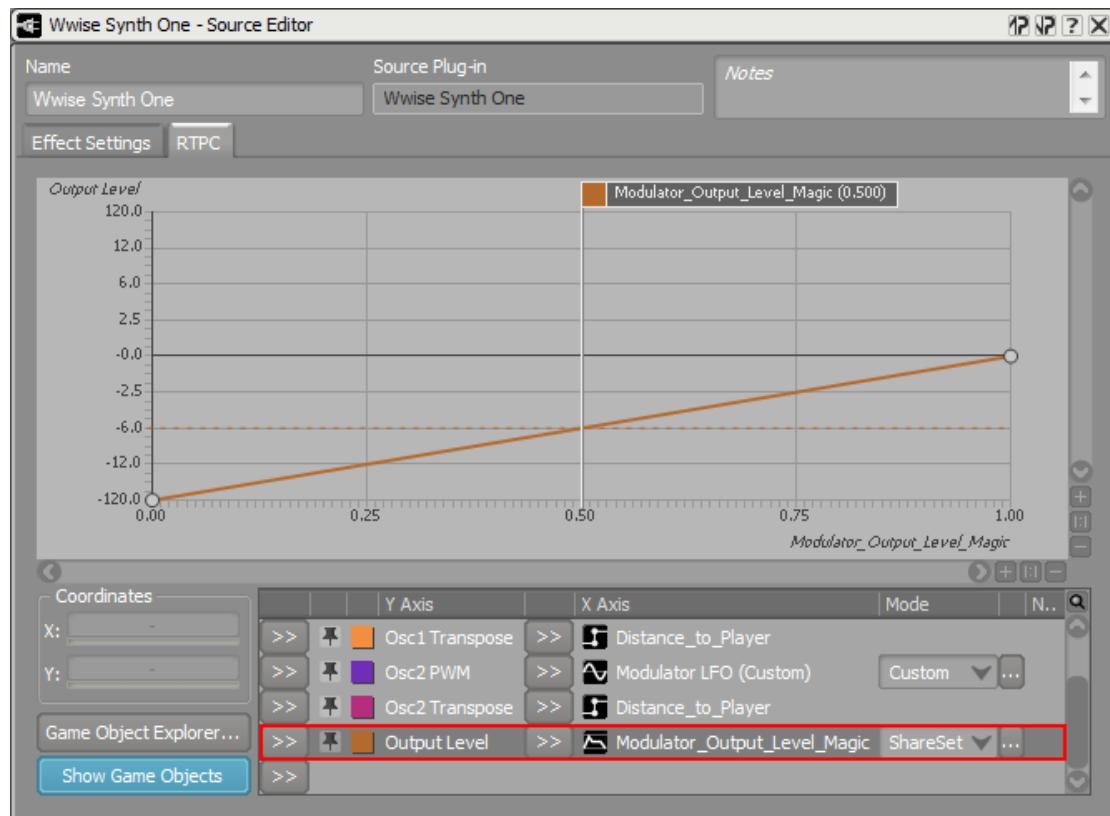
By default, the mode for the newly created Modulator Envelope is set to *ShareSet*.



Creating a new Modulator Envelope ShareSet  
named Modulator\_Output\_Level\_Magic

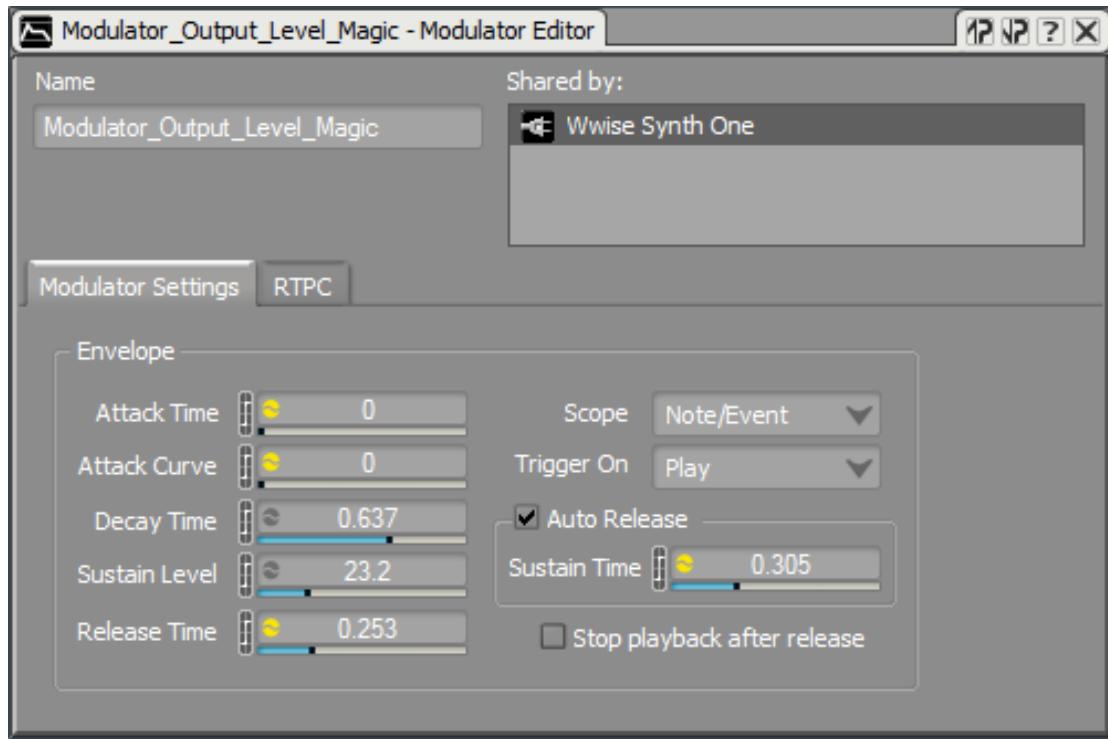
## Designer Note

Envelope objects can be created as Custom or ShareSet. Custom objects are stored in-place, directly inside the object that has it. ShareSets are stored in a separate work-unit and can be re-used across objects.



Modulator Envelope Mode set to ShareSet

The minimum and maximum RTPC values of Modulator Envelope in the Sound Property Editor affect the base and top of the ADSR envelope (nil and peak values). Swapping the min and max values reverses the ADSR.



Editing the Envelope Modulator

The properties available to the Modulator Envelope are:

- **Attack Time**
  - Defines the time taken for initial run-up of level from nil to peak, beginning when the key is first pressed (in seconds).
  - Default value: 0.2
- **Attack Curve**
  - Adjusts the Attack Curve from its linear default slope (50%) to either:
    - an exponential-style envelope (0%) where the rate of change starts slow and then increases
    - a logarithmic envelope (100%) where the rate of change starts fast, then decreases
  - Default value: 50
- **Decay Time**
  - Defines the time taken for the subsequent run down from the attack level to the designated sustain level (in seconds).
  - Default value: 0.2
- **Sustain Level**
  - Defines the level during the main sequence of the sound's duration, until the key is released (in percentage of the range).
  - Default value: 100
- **Release Time**

- Defines the time taken for the level to decay from the sustain level to zero after the key is released (in seconds).
  - Default value: 0.5
- **Scope**
  - Controls how Envelopes are created:
    - **Voice:** An instance of an Envelopes is created for every voice of the synth when used in MIDI context.
    - **Note/ Event:** An instance of an Envelopes is created for every playing instance of the synth when used in MIDI context.
  - Default value: Note/Event
- **Trigger On**
  - The actions/MIDI events that may trigger the envelope (that is, enter the attack phase):
    - **Play:** either a play action or a MIDI note event
    - **Note-Off:** only a MIDI note-off event
  - Default value: Play
- **Auto Release**
  - Determines if the envelope requires an action/MIDI event to exit the sustain phase and enter the release phase. If set, the envelope exits the sustain phase after Sustain Time. If not set, the envelope exits the sustain phase following a certain condition:
    - The envelope may be released by the game via a **Release Envelope** event.
    - An envelope may also enter the release phase via a MIDI note-off event, provided the envelope was triggered by a MIDI note-on event.
  - Default value: False
- **Sustain Time**
  - Defines the time which the envelope will remain in Sustain before the Release is applied (in seconds).
  - Default value: 0
- **Stop Playback After Release**
  - If set, the playback of the associated sound is terminated after the release phase is complete.
  - Default value: false

Modulator Envelope Properties can be accessed using the Modulator Editor:



### Designer Note

Additional information for working with Envelopes can be found in the Wwise Help Document: [Wwise Help > Interacting with the Game > Working with RTPCs > Working with Envelopes](#)

The combined power of synthesis, modulation, and parameterization is at its best when used in conjunction with the dynamics of gameplay. With so many options to modify and affect the properties of sound within the project, it is important to understand the impact these can have on your platform's memory and CPU performance.

A modulator's processing time depends on its RTPC usage. For most properties, a modulator is evaluated once per audio sample. However, for the property voice volume, the associated modulator is evaluated at every frame. Use modulators selectively as they can consume a significant amount of a platform's memory and CPU. Always work within the constraints of your target platform.

## Magic Summary

The creative sound design process for magic and spell casting effects begins deep within your own imagination. From there it extends from sound, to the microphone, eventually ending up in the sound library where it can be assembled in a digital audio workstation as the basis for the creation of fantastical effects. A further step in this process is how the content is prepared for the eventual implementation. Creatively combining, recombining, and dynamically manipulating sounds within the authoring application using game parameters is a working methodology that leverages the inherent interactivity of games as a medium. Coupled with the power of synthesis, LFO, and Envelopes, the potential for the dynamic interaction between gameplay and game sound can make a sound designer's dreams come true.

**Throughout this chapter we have:**

- Learned about summing and layering with the blend container.
- Changed perspectives based on distance using RTPC.
- Discussed the application and modification of effects.
- Created a Wwise Synth One element.

**Stepped through the process of:**

- Recombining sounds using the blend container.
- Using effects.
- Employing Real Time Parameter Control (RTPC).
- Leveraging blend tracks as part of a distance based perspective model.
- Creating a Modulator Envelope ShareSet.

**Also touched on:**

- Effects chain ordering.
- Audio bus sub-mixing of effects.
- Modulator LFOs.

**Throughout this section we have created the following objects:**

- A blend container called `magic_blast_fire_distance_blend` that uses a distance game parameter to Crossfade between different perspectives of magic blast sounds.
- A Wwise Synth One element that augments the audio content in `magic_blast_fire_blend`.

## **Magic Additional Resources**

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Defining Object Playback Behaviors > Defining the Contents and Behavior of the Blend Container](#)

[Wwise Help > Wwise Reference > Actor-Mixer Objects > Blend Containers](#)

[Video Tutorial - Creating Dynamic Sounds Using RTPCs](#)

[Wwise Help > Interacting with the Game > Working with RTPCs > Working with LFOs](#)

[Wwise Help > Interacting with the Game > Working with RTPCs > Working with Envelopes](#)

[Wwise Help > Wwise Source/Effect Plug-ins > Wwise Synth One > Wwise Synth One Plug-in](#)

---

## **Chapter 5. Dialogue Decisions and Language Lessons**

Overview .....	121
Getting Started with Dialogue and Non-Verbal Vocalizations .....	122
Adding Additional Languages .....	126
Dynamic Dialogue .....	128
Cinematic Dialogue Placement .....	130
Voice Summary .....	132
Voice Additional Resources .....	133

## Overview

A single shriek of terror resonates through the forest, shaking dead leaves from the branches of trees and setting birds to flight. What follows is an exchange of inhuman grunts and growls followed by a plea for help.

From simple conversational interactions, to a fully branching multi-language epic storytelling device, the art of dialogue production has become an exercise in asset management and system integration of the highest order. With the escalating line counts of modern console titles, voice over can quickly become a high quality headache without a comprehensive localization strategy.

This chapter will take you through the process of:

- Getting started with voice
- Preparation for multi-language support
- Dynamic Dialogues
- Positioning and panning

## Getting Started with Dialogue and Non-Verbal Vocalizations

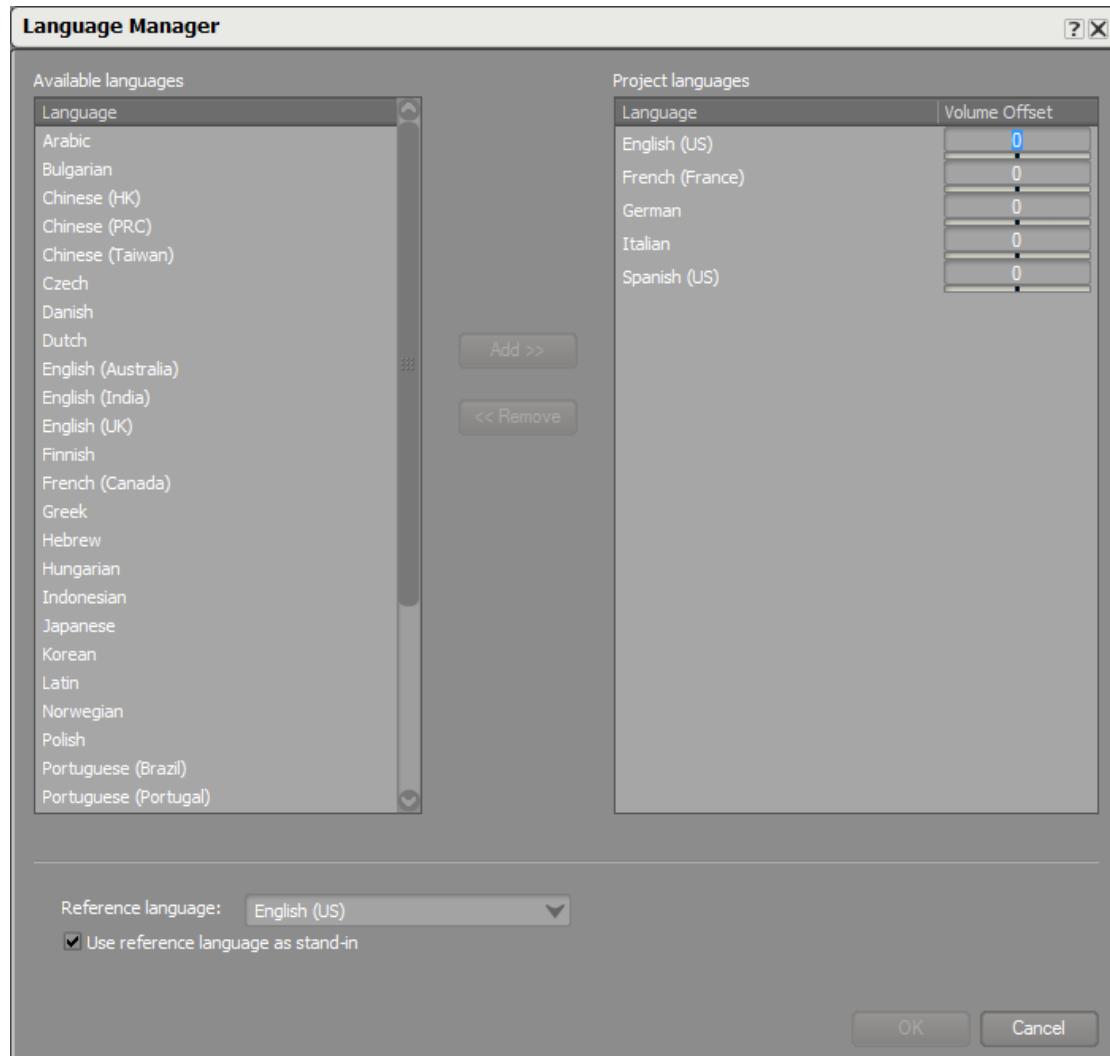
When working with dialogue you need to distinguish between spoken and non-verbal dialogue. Spoken dialogue usually needs to be localized in different languages whereas non-verbal dialogue represents the palette of emotional vocalizations not usually translated. In the case of non-localized audio files, variations can be added directly to a sound object, organized within the Actor-Mixer Hierarchy, and added to events just like any other sound effect. To handle these dialogue considerations, the process for importing Voice assets into the project differs from music and sound effects. The fundamental difference being the opportunity to determine project languages in preparation for localization.

First you need to define the game project languages in the Language Manager located in Wwise>Project>Languages menu. You also need to define the reference language used throughout the project.

The reference language for the project is used in various situations:

- **Importing language files** - when you are importing an audio file, the import conversion settings of the reference file are used.
- **Converting language files** - when you are converting a language file, the conversion settings of the reference language can be selected.
- **Before language files are available** - when certain language sources are not ready, the reference language can be used in their place.

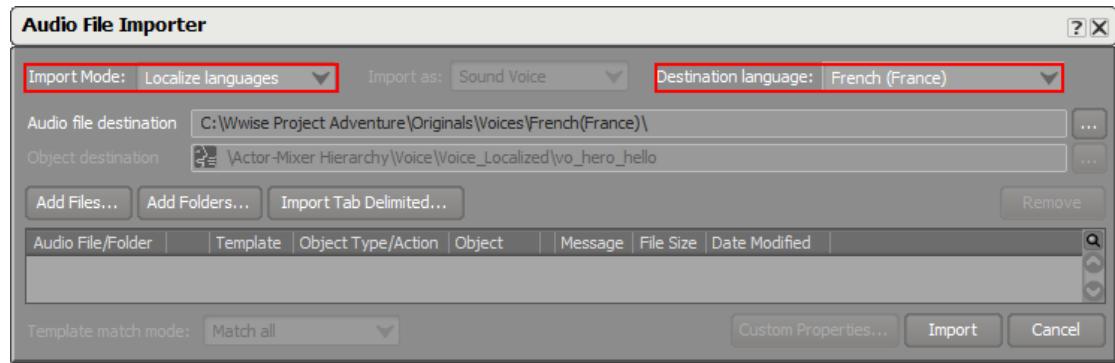
## Dialogue Decisions and Language Lessons



### Defining project languages in the Language Manager

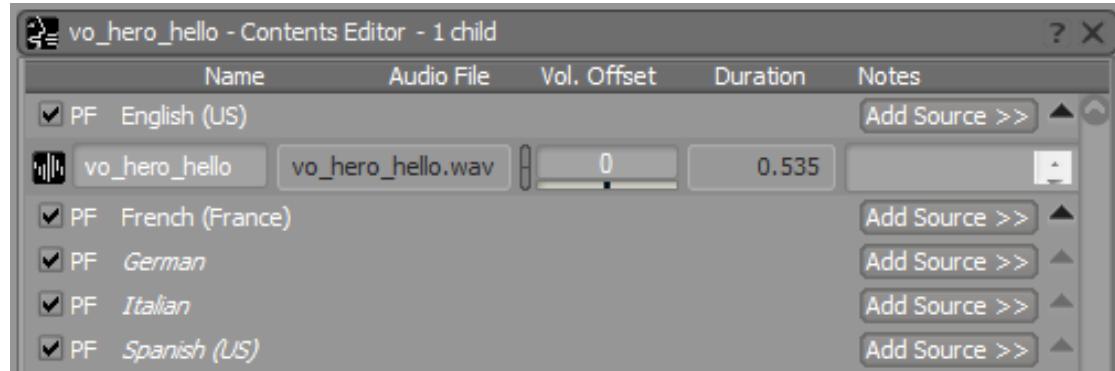
After you have selected the languages, localized voice files can be imported either through the Project menu, or by simply dragging and dropping files into the Project window. Selecting Import as Sound Voice prepares the object for localization based on project languages established in the Language Manager.

## Dialogue Decisions and Language Lessons



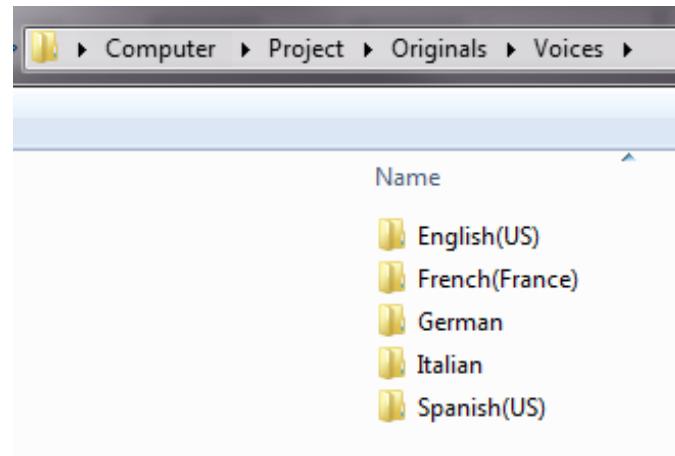
### Importing dialogue as a Sound Voice in the Audio File Importer

After a Sound Voice has been imported in the reference language, additional localized files can be added.



### Adding an audio source for English language

A copy of the source audio files are then added to the project based on language. By default these files are copied to the Originals/Voice location within the project directory.

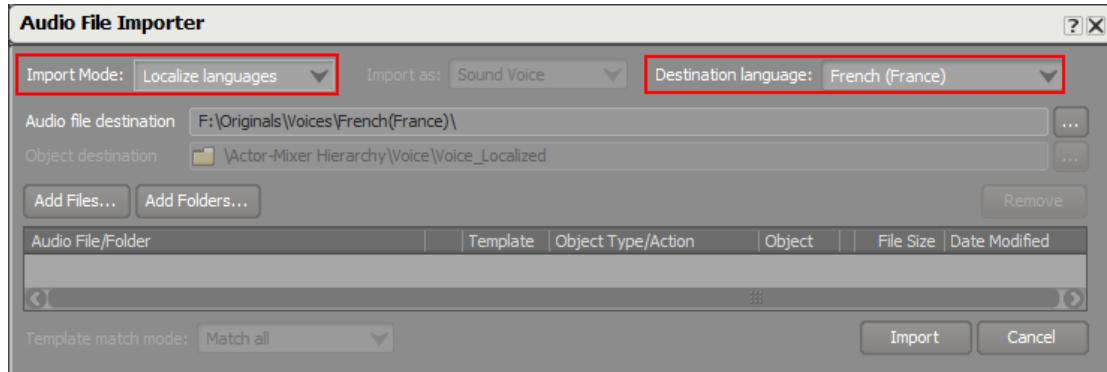


The folder hierarchy for copied Originals in the project Voices file folder

## Adding Additional Languages

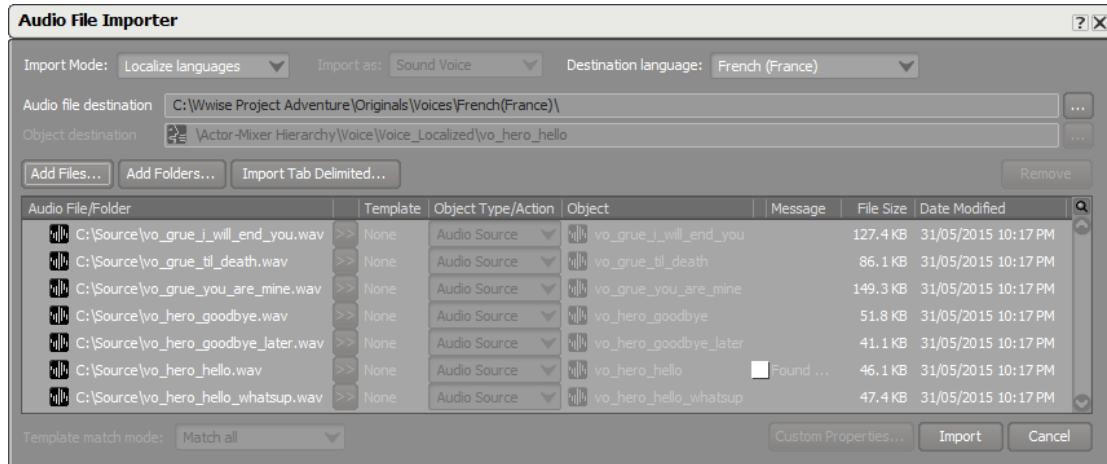
When localized language files are ready, they can be imported directly into the project using the Audio File Importer.

To begin the process, first change the import mode to Localize Languages and set the appropriate Destination language for localization.



Preparing to import localized voice files using the audio file importer

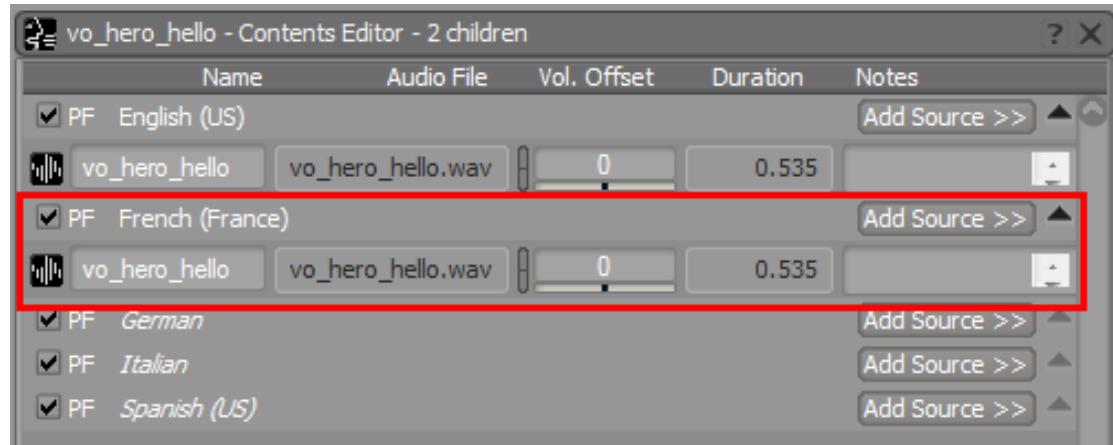
You can then select files or folders to be imported in the Destination language using either the Add Files or Add Folders button.



Adding localized files to already existing voice objects

Once imported, the localized files appear in the Contents Editor for the associated Sound Voice object.

## Dialogue Decisions and Language Lessons



Localized dialogue added to an existing Sound Voice

## Dynamic Dialogue

There comes a point in every quest when the path splits, and a decision must be made about how to proceed. The art of branching dialogue relies on the player choosing from a list of possible interactions in order to move the conversation forward. In addition to the potential outcome, you may want to carry forward the history of past responses to characters during previous conversations.

At the center of Wwise's Dynamic Dialogue system is the dialogue event, which is a set of rules or conditions that determines which piece of dialogue to play. The dialogue event allows you to re-create a variety of different scenarios, conditions, or outcomes that exist in your game. To ensure that you cover every situation, Wwise also allows you to create default or fallback conditions.

All these conditions are defined using a series of Switch Groups and Switches. These Switch Groups and Switches are combined to create Switch Group paths, which define the particular conditions or outcomes in the game. Each path is then associated with a specific sound object in Wwise. As the game is played and dialogue events are called, the game verifies the existing conditions against the Switch Groups defined in the dialogue event. The Switch Group paths that match the current situation in game along with the mode, probability, and weighting of each path determine which piece of dialogue is played, if any.

For example, the following dialogue event contains the arguments related to the names of each player in a sports game. The Switches of each Switch Group are combined to create the different paths or conditions that may exist. In this particular example, the commentator can use either the player's last name or full name.

Dialogue Event: Name		
Switch Groups	Player Name	Name Length
Switches	Tony Cross	Full
	John Patrick	Last
Switch Group Paths		Assigned Object
Cross - Full	Cross_Full	
Cross - Last	Cross_Last	
Patrick - Full	Pat_Full	
Patrick - Last	Pat_Last	
Player Name - Name Length	He	

To deal with situations where there are no Switches that match the current situation in game, you can create a path with a default or fallback Switch. These fallback paths contain one or more Switch Groups instead of Switches and are usually associated with a more general sound object. In the previous example, the fallback Switch Group path is associated with the sound object “he” instead of one of the player’s names.

After you have re-created all the conditions in the dialogue event, it can be integrated into the game engine. When the dialogue events are called by the game, the sound engine resolves the dialogue event by returning the audio object that corresponds to the matching argument path. The sound engine can then decide whether to insert the audio object into a dynamic sequence for playback. The relationship between returning an audio object and inserting it into a dynamic sequence does not have to be 1:1. This means that for each resolved dialogue event, a returned audio object can be added to the dynamic sequence as many times as necessary.

Since the game engine uses the dialogue event name, you can create the events, integrate them into the game and then build and fine-tune the contents of each event without ever having to re-integrate them into the game again. This gives you a great deal of flexibility to add or remove argument values, and to experiment with different sounds, all without additional programming.

For more information on dynamic dialogue:

[Wwise Help > Interacting with the Game > Managing Dynamic Dialogue > Working with Dialogue Events](#)

[Wwise Help > Interacting with the Game > Managing Dynamic Dialogue > Understanding the Dynamic Dialogue System](#)

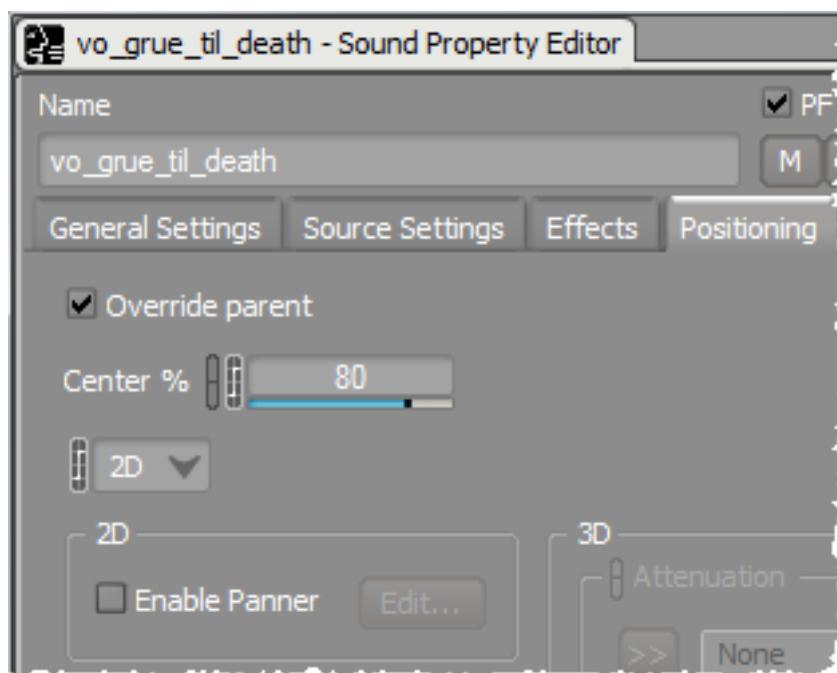
[Wwise Help > Interacting with the Game > Managing Dynamic Dialogue > Dialogue Events Tips and Best Practices](#)

[Wwise Help > Interacting with the Game > Managing Dynamic Dialogue > Creating Dialogue Events](#)

## Cinematic Dialogue Placement

Several examples are helpful to have a clear picture of how dialogue should be handled positionally in games. Linear media, such as film and television, can help guide the way we present dialogue for the screen, but doesn't accommodate the entire scope of concerns when it comes to handling interactions that are presented dynamically.

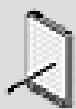
With cinematic conversations usually anchored in the front speakers during sections of the game where control is taken from the player, setting the position of dialogue during these sequences to 2D will appropriately manage their placement. By default, 2D objects are only played through the front left/right speakers for sound objects without any propagation behaviors. Adding a percentage of the center channel allows for an additional level of control.



Cinematic dialogue set to 2D with 80% center channel volume

In situations where the player is present during the delivery of gameplay dialogue and able to move throughout the world, it may be necessary to play the dialogue positionally in 3D. Simply subscribing an attenuation to a Sound Voice with the game object at the location of the character speaking the line will apply the specified volume reduction based on the defined attenuation. This case allows for situations where you are free to move about while dialogue is spoken by characters positioned in the game world.

Adding realism to positioned in-game dialogue doesn't have to end with volume attenuation. As described in the campfire example in the Ambient chapter, most voices could benefit from the addition of a spread across adjacent speakers at close distance. Taking the implementation a step further, the addition of cone attenuation could be used to focus the direction of the sound controlled by the orientation of the game object, or in this case the mouth of the character speaking. Applying a cone attenuation allows for the attenuation and filtering of the sound as the listener moves outside of, or behind, the oriented game object. This simulates the obstruction of the character's head on the spoken line of dialogue.



### Programmer Note

The OrientationFront vector defines the direction that the listener's head is facing. It should be orthogonal to the OrientationTop vector, which defines the incline of the listener's head. For a human listener, one could think of the OrientationFront vector as the listener's nose (going away from the face), while the OrientationTop vector would be orthogonal to it, going between the listener's eyes and away from the chin.

Introducing additional techniques for simulating realism with dialogue are worth investigating, but knowing whether the dialogue being delivered is critical to the game play can quickly determine how much realism can be used. If, for instance, it's important that the player hear the secret to subduing the dragon, then realism should not get in the way of delivering the dialogue by obstructing it with filters or volume attenuation. Ideally every conversation in the game needs to be built along the fine line between realism and storytelling to deliver, coupled with game play, an immersive conversational experience to the player.

## Voice Summary

As games continue to grow into their role as a storytelling medium, there are few devices available greater than the human voice in its ability to convey emotion and establish character. Whether you're dealing with non-verbal communications or building complex trees of conversational outcomes, there exists a suite of tools designed for the express purpose of integrating dialogue within the flow of gameplay in Wwise.

**Throughout this chapter we have:**

- Discussed getting started with voice.
- Differentiated between verbal and non-verbal vocalizations.
- Established a methodology for localizing dialogue.
- Discussed the role of dialogue positioning in different scenarios.

**Stepped through the process of:**

- Importing sound voice objects for English language dialogue files.
- Importing localized language dialogue files using the Audio File Importer.
- Adjusting the percentage of center channel contribution for 2D positioned sounds.

**We've also touched on:**

- Using arguments to creating branching dialogue.

**Throughout this section we have created the following objects:**

- Spoken dialogue sound voices in the English language and localized in French.

## Voice Additional Resources

[Wwise Help > Interacting with the Game > Managing Dynamic Dialogue](#)

[Wwise SDK > Sound Engine Integration Walkthrough > Integrate Wwise Elements into Your Game > Integrating Markers](#)

[Audiokinetic Q&A > Can we import work-units from other projects?](#)

[Video Tutorial - Managing Voices and Language Localizations](#)

---

## **Chapter 6. Unlocking the User Interface**

Overview .....	135
Creating a Simple Menu Select Sound .....	136
Defining 2D Sound Positioning .....	138
The Complex Negotiation of Pause .....	139
Pause - Defining the Scenario .....	139
Pausing the Game .....	140
Resuming the Game .....	141
Posted Events .....	142
User Interface Summary .....	143
User Interface Additional Resources .....	144

## Overview

The User Interface is often the hub around which the game world is created. Creating a solid experience with sound begins with the ability to understand the interface flow and how sound can be used to reinforce the aesthetic, without breaking immersion.

This chapter will take you through the process of:

- Creating simple menu events
- Defining 2D positioning and panning
- Creating complex events
- Posting events

From the simple playback of exactly the right sounds, to the complex negotiation that happens when pressing pause, the User Interface is a case study in the solutions and techniques available within the Wwise authoring application.

## Creating a Simple Menu Select Sound

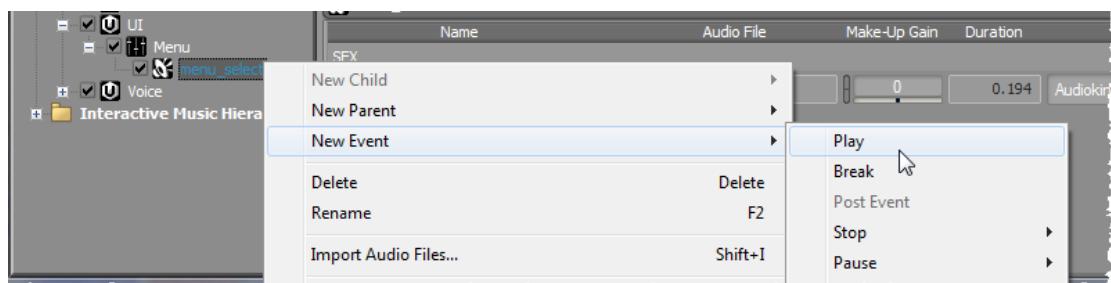
Let's create a Sound SFX for a menu\_select sound which will be played whenever a menu selection is confirmed. Start first by importing the menu\_select file, either using the audio file importer or by dragging and dropping from the source folder into the default work unit.



menu\_select sound object

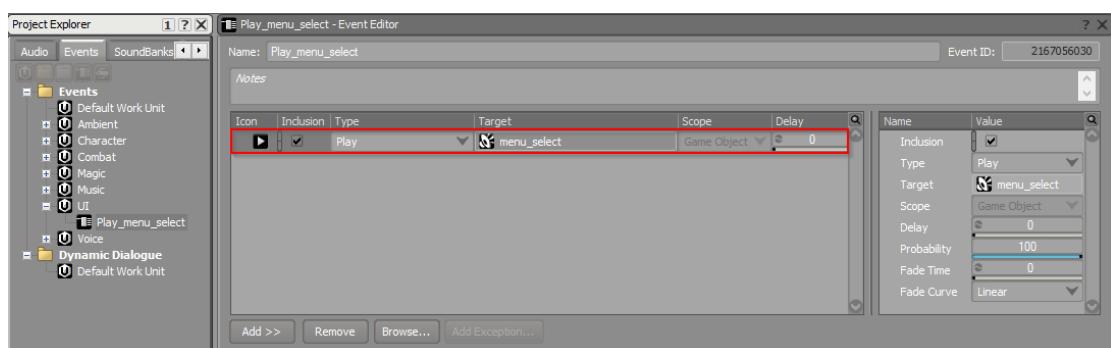
## Creating a Simple Menu Select Event

By selecting the Sound SFX and entering the contextual menu we can immediately create a play event for this sound:



Creating a play event from the contextual menu of a sound object

This results in the following event:



Play\_menu\_select event

The event name Play\_menu\_select can now be passed to the programmer who will post the event for each time an item in the menu is selected.

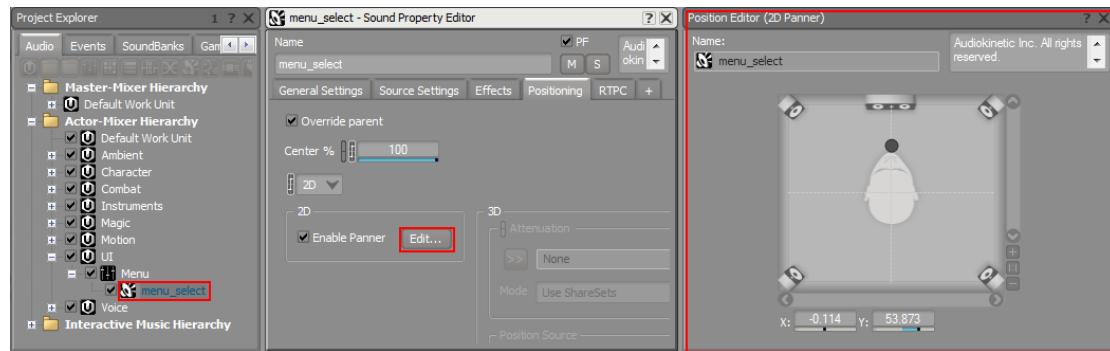


### Programmer Note

Events are triggered by calling AK::SoundEngine::PostEvent()

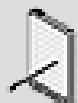
## Defining 2D Sound Positioning

With most visual aspects of the menu displayed on a two-dimensional plane, it's common to also ignore 3D positioning for a sound object and instead manually control the positioning in 2D. Wwise enables direct control over 2D positioning of a sound object from the positioning tab in the Project Explorer.



2D Position editor

By switching the positioning to 2D you gain control over the sound's position through the use of the 2D panner interface and also via a percentage parameter for the center channel that sends a contribution of the sound to the center speaker.



### Programmer Note

Usually, UI events are triggered from a global game object (a game object without positioning info)

Of primary concern when scoring for the user experience is the feel and aesthetic of the sound itself. While there is always room to use advanced techniques to achieve certain effects, the role of sounds that help to provide feedback to the user are usually much more difficult to design than they are to implement.

## The Complex Negotiation of Pause

While the playback of discrete sounds may end up being quite simple, there is often a complex series of interactions that need to take place during the transition from gameplay to the menu. It is expected that at any point during gameplay that the pause button can be pressed, allowing for the swift transition from in-game to in-menu actions.

For this to appear seamless from the sound perspective, several things should happen:

- Music playing in-game should be muted, paused, or allowed to play through the menu.
- In-game sound effects should be paused, stopped, or muted.
- In-game ambient sound should be muted, paused, or allowed to play through the menu.
- Non-critical dialogue should be paused or stopped.
- Critical dialogue should be either paused or restarted.
- Menu music or ambient sound could be played.

At first glance, this list of actions may seem outside the scope of what would normally be available to the sound designer; however, the event system allows for these decisions to be controlled and manipulated from within the authoring application. Wwise events can apply actions to the different structures within your project hierarchy. Each event can contain one action or a series of actions. The actions you select will specify whether the Wwise objects will play, pause, stop, and so on.

A complete list of event actions can be found in the help documentation:

[Wwise Help > Wwise Reference > Events > Event Editor](#)

[Wwise Fundamentals > Understanding Events > Action Events](#)

[Wwise Help > Interacting with the Game > Managing Events > Overview > Types of Event Actions](#)

### **Pause - Defining the Scenario**

Let's take a scenario where we specifically want to:

- Pause all music, ambient, sound, and voice.
- Play a transitional sound effect.
- Loop a piece of menu specific music.

### Designer Note

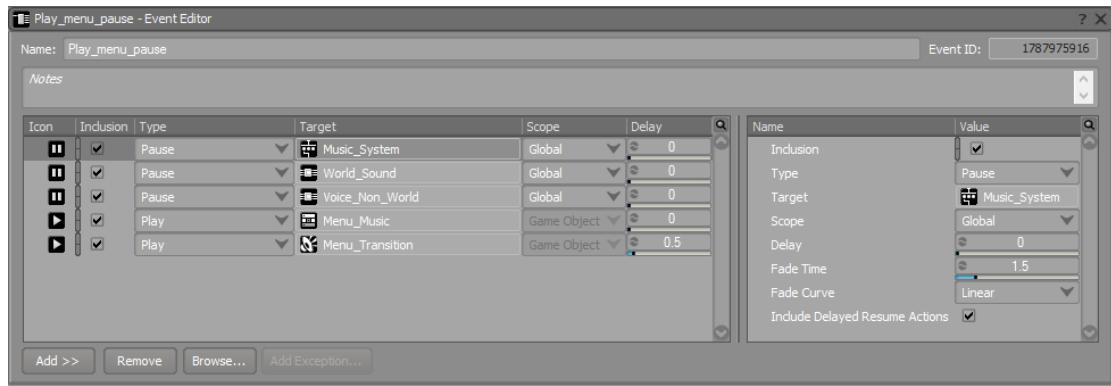


In the case of multi-player or persistent worlds that carry on while one user is paused, there is also the ability to mute the audio.

Ideally, from a simplified perspective and likely that of the programmer, only two events need to be communicated: Pause Audio and Resume Audio. The sound behavior that happens when each of these is posted by the programmer can be specifically determined by the sound designer. To make things easy in the example, we are going to pause based on busses contained in the Master-Mixer Hierarchy.

### Pausing the Game

The following event shows each level of the mix hierarchy using the pause action with each object receiving a 1.5 second fade out:



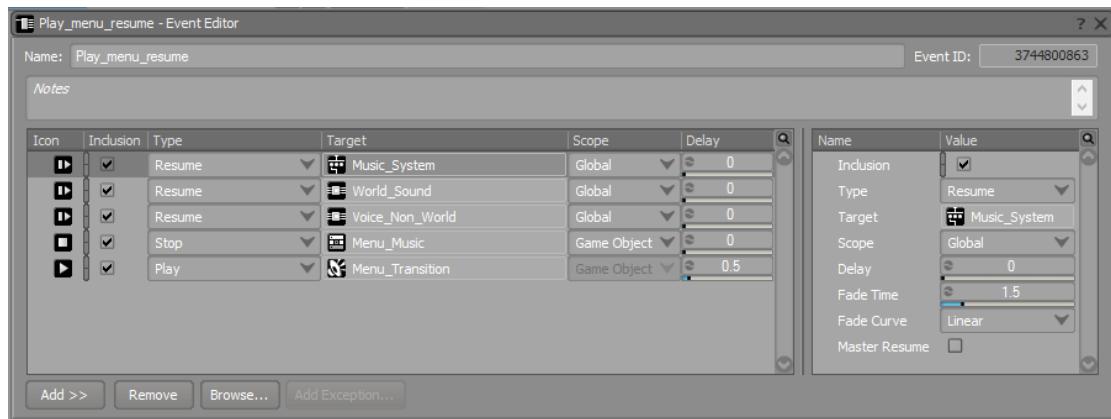
Pause event showing event actions

Event Action	Object	Location
Pause	Music_System	(Interactive Music)
Pause	World_Sound	(Master Audio Bus)
Pause	Voice_Non_World	(Master Audio Bus)
Play	Menu_Music	(Interactive Music)
Play	Menu_Transition	(Actor-Mixer)

Additionally, the menu\_music and menu\_transition use the play action simultaneously.

## Resuming the Game

When we resume the game, we only need to use the resume action on the paused busses of the Master-Mixer hierarchy in order to continue playing everything back from the point at which they were paused. Simultaneously we will use the stop action on menu\_music with a 1.5 second fade out, and play the menu\_transition sound again:



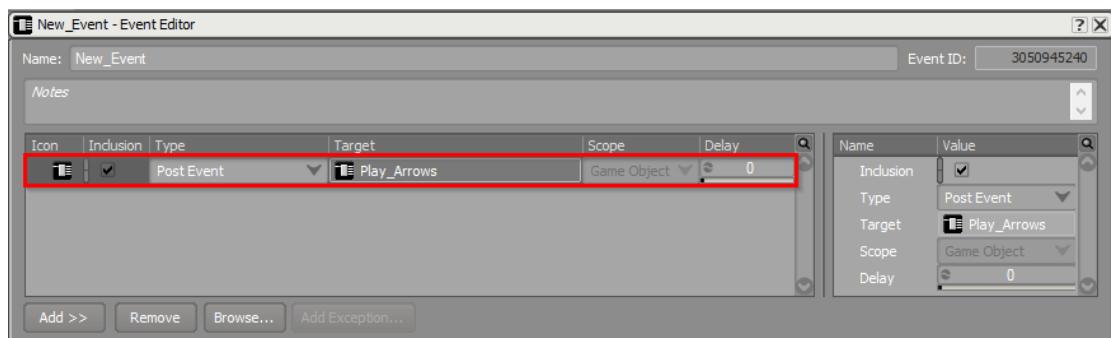
Resume event showing event actions

Event Action	Object	Location
Resume	Music_System	(Interactive Music)
Resume	World_Sound	(Master Audio Bus)
Resume	Voice_Non_World	(Master Audio Bus)
Stop	Menu_Music	(Interactive Music)
Play	Menu_Transition	(Actor-Mixer)

The important take away from this example is that with just two Wwise events you can control all of the sound for the game. Using event actions, the audio experience for transitioning from gameplay to pause and back again can be authored according to the desires of the sound designer.

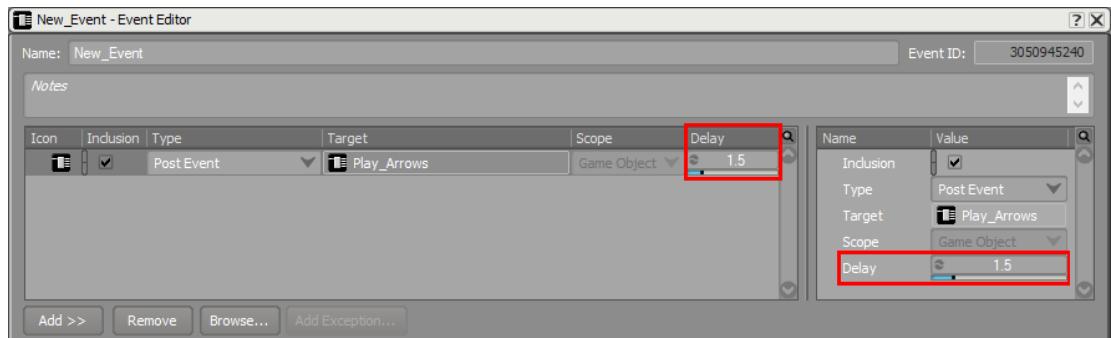
## Posted Events

While Event actions directly affect the playback and manipulations of sounds within the project, the ability to post an Event from another Event allows for the further extension and flexibility of an Event. Posting an Event from within another Event opens up a world of uses that includes things like sequential playback and templating of functionality that can be reused across multiple Events.



Post Event action added to New\_Event

One example of the flexibility of nested events is the potential to simply delay an existing event that contains a complex set of established event actions.



Setting a delay offset on a Post Event action

Other considerations are the pause and resume Events created in the previous section. If there were other moments in the game when sound and music needed to be manipulated in the same way, these two Events could be triggered from within other Events. Creating a template of functionality allows for the leveraging of reusable implementation intelligence throughout a project without the need to duplicate efforts. It can, therefore, cut down on the number of Event actions used in a single Event.

## User Interface Summary

Implementing the sound in your user interface can be as simple or complex as necessary in order to support the feel and aesthetic. Simply put, the sound should serve as an extension of the experience; whether navigating through or transitioning between gameplay and the menu system, everything should support the artistic scope. Wwise puts the decisions into the hands of the sound designer by allowing for the finer details of interaction to be managed through the event system.

**Throughout this chapter we have:**

- Reviewed importing sound files.
- Discussed the role of User Interface sounds to support game aesthetic.
- Outlined some of the complexities inherent with pausing and resuming.

**Stepped through the process of:**

- Creating a simple Event for menu selection.
- Defining 2D sounds using the Position Editor.
- Creating pause and resume Events.

**Also touched on:**

- Different types of Event actions.
- Triggering Post Event actions.

**Throughout this section we have created the following objects:**

- A simple menu select Sound SFX and event.
- Pause and resume events.

## User Interface Additional Resources

[Wwise Help > Interacting with the Game > Managing Events > Overview > Types of Event Actions](#)

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Defining Positioning for Sound and Motion](#)

[Video Tutorial - Creating Events](#)

---

## **Chapter 7. Adventures in Music**

Overview .....	146
Starting With the Interactive Music Hierarchy .....	147
Preparing the Content .....	147
Layered: The Vertical Approach .....	148
Creating the Ambient Music Segment .....	148
Grooming the Tracks .....	151
Dynamic Danger .....	152
Adding RTPC to Tracks .....	153
Auditioning RTPC .....	155
Looping a Music Segment .....	156
Section Summary .....	156
The Sequential Approach .....	157
Groups and Behaviors .....	157
Sequencing Groups in the Music Playlist Editor .....	158
Section Summary .....	160
Using States to Switch Between Music Types .....	161
Defining Interactive Music Transitions .....	163
Authoring Transitions .....	164
Defining Transition Behavior .....	164
Transitioning from Ambient to Action Music .....	165
Transitioning from Action to Ambient Music .....	167
Section Summary .....	169
Music Summary .....	170
Music Additional Resources .....	172

## Overview

Heroic journeys are usually accompanied by the thundering of trumpets and fanfare, but several hours of steadily looping brass wouldn't scale across a single piece of music. Interactive and dynamic music in games continues to evolve as an art form in its own right. Complex techniques such as authoring sync points, defining time signatures, and preserving the musical entry and exit points can now be specified from within the authoring application to create a complex fabric of composition created specifically for the medium of games. With such potential comes the ability to tailor the music to support gameplay and the player experience.

The first consideration when choosing an implementation strategy for a music system begins with matching the emotional scope and intent of the game with appropriate music. By nature, systems are adaptable and varied, but the basic thematic intent should create a seamless blend between the game and its implementation. Once the mood has been established, it's time to begin thinking about how to support gameplay and then how to shape compositional elements into building blocks that will become the emotional river running throughout the game.

This chapter will take you through the process of:

- Starting with the Interactive Music Hierarchy
- Preparing the content
- Layering vertically
- Adding a sequential approach
- Switching between music types
- Defining interactive music transitions

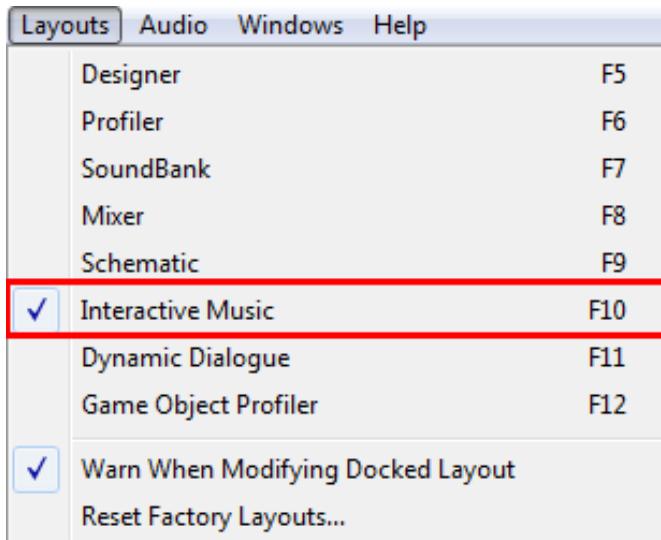
## Starting With the Interactive Music Hierarchy

### What is the Music Engine?

The music engine is the high-level engine that handles complex, high-level scheduling of music segments of the interactive music hierarchy. The fundamental difference between the Interactive Music and the Actor-Mixer Hierarchies is the use of tempo and time signature to rule the behaviors of the different music containers.

Audio files added to the Interactive Music hierarchy are imported as Music Segments and can also be added as tracks within a music segment. Segments can then be added to Music Playlists Container or Music Switch Container in order to add additional functionality.

To view the music specific functionality of containers in the Interactive Music hierarchy, the Layout must be changed to Interactive Music.



Changing to the Interactive Music layout

### Preparing the Content

Before beginning the integration of music content into the project, the content must first be prepared for the type of system that's going to be used. Throughout this section we'll walk through two ways of presenting dynamic music that will end up working together to provide an emotional backdrop to the hero's quest. By leveraging both a layered approach (where the volume of time-synced music layers reacts to gameplay) and a sequential approach (which adds diversity to a looping section of music through randomization) we'll create a simple music system that changes based on player interaction with the game.

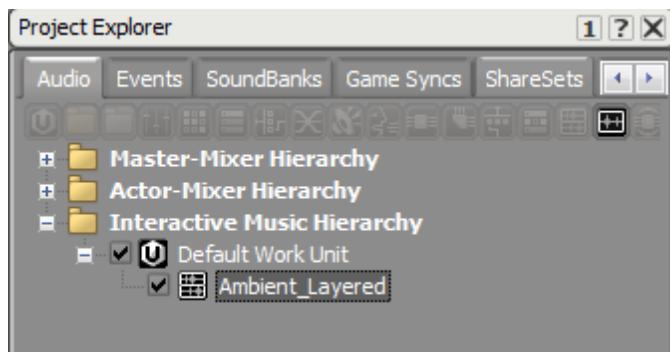
## Layered: The Vertical Approach

During a walk through the forest, the music helps to set a serious and foreboding tone as we quest to adventures unknown and dangers hidden just beyond sight. In an attempt to keep the music fresh, varied, and dynamic, the introduction of a vertical layering approach modified based on the players proximity to danger will be instituted. This parameter is passed from the game engine based on the player distance to an enemy or location and is used to increase the musical intensity during gameplay.

The musical content for the layered implementation has been prepared as ten tracks of layered audio that is intended to play back in sync within a single music segment.

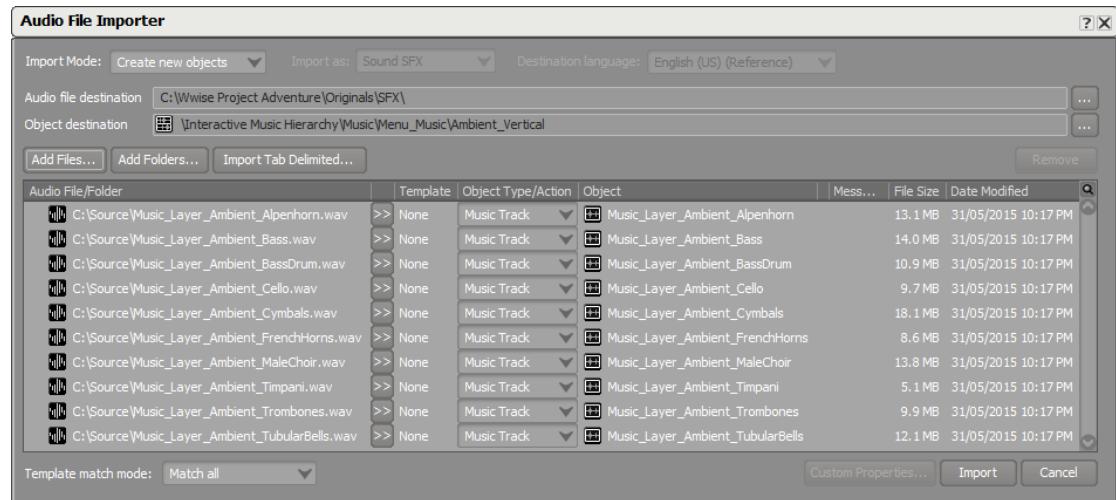
### Creating the Ambient Music Segment

Adding a child Music Segment to the Default Work Unit in the Interactive Music Hierarchy called Ambient\_Layered will begin the process of establishing the ambient music system.



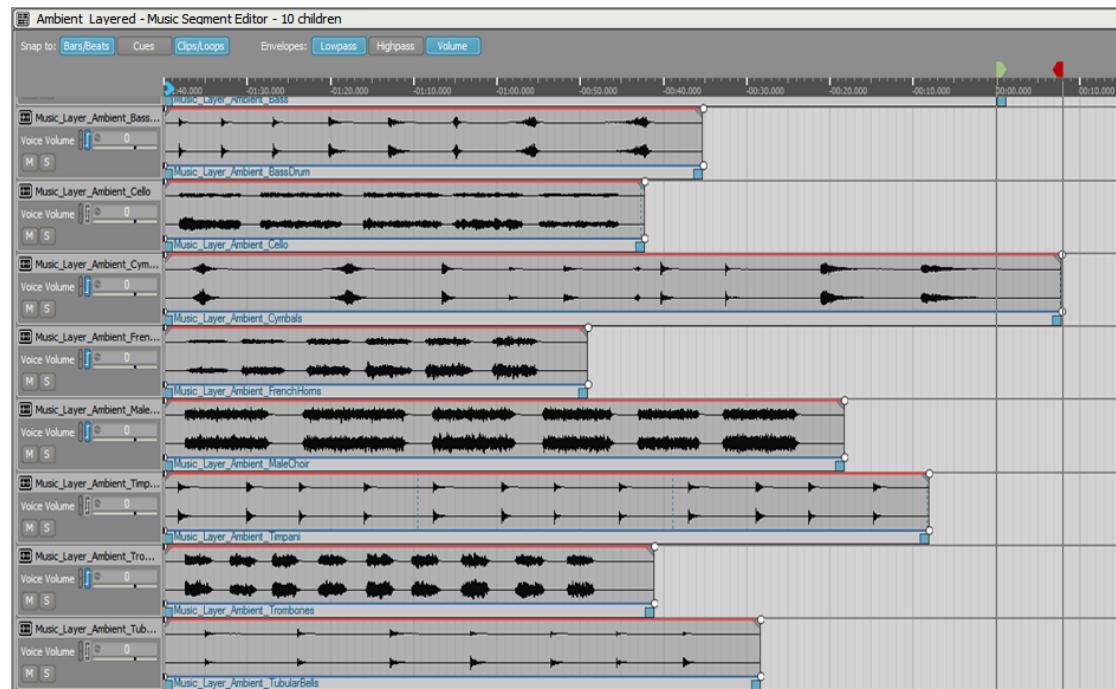
Creating the Ambient\_Layered segment for the vertical music system

The music files can now be imported as tracks in the segment by either using the Audio File Importer for the segment or dragging and dropping it over the Music Segment.



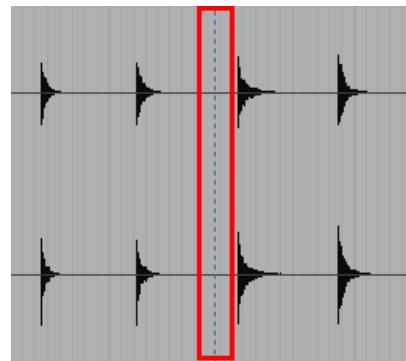
### Importing Music Tracks to a segment using the Audio File Importer

Once files have been added as tracks to the segment, you can access them through the Music Segment Editor, where tracks can be arranged and edited within the Music Segment.



### Editing tracks in the Music Segment Editor

The basic component of a track is called a clip, a rectangular area representing a single WAV file. You can adjust when a clip plays by dragging it left or right along its track. You can move clips from one track to another by dragging them up or down. You can also overlap clips on a single track. You can make a clip shorter by dragging one of its handles inwards, or extend it by dragging the handle outwards. When you extend a clip, it repeats itself. Each repeat is called a loop. Loop points can be identified by a vertical dashed line in the clip.



Loop points represented by a vertical dashed line in the clip

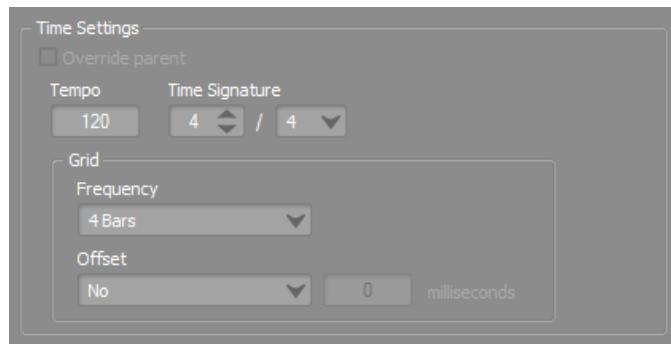
Cues and cursors are also part of the Music Segment Editor. Cues are markers appended to segments to indicate key points, such as its entry and exit points.



Adjusting Entry and Exit cues in the Music Segment Editor

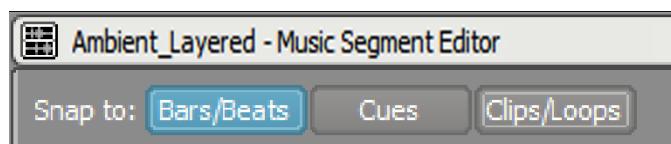
You can also create custom cues to indicate when property changes or transitions should occur, or when stingers should be played. The play cursor moves along as you play a segment, but you can also move it to control where playback begins. The end cursor marks the end of the segment.

The time settings for a segment, which is inherited by any tracks held within a segment, can be specified under the general settings tab. Segments can inherit the time settings of their parent containers, as well as override them for further timing control.



Settings for Tempo, Time Signature, or Grid based measurement

The tempo for this multi-layered music is 120 BPM with a 4/4 time signature. Once the time settings have been defined for a segment and its tracks, the ability to use the Snap to functionality in the Music Segment Editor gives a more precise level of control when adjusting clips, cursors, or cue markers.

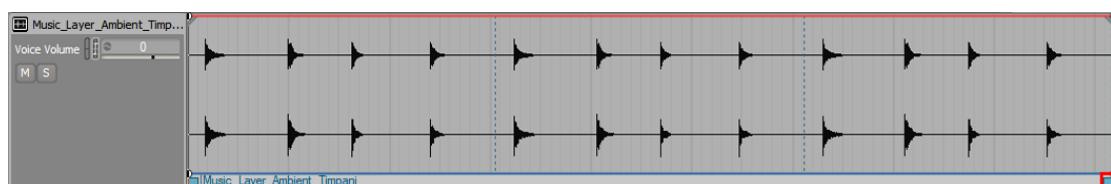


Setting the Snap to functionality in the Music Segment Editor

## Grooming the Tracks

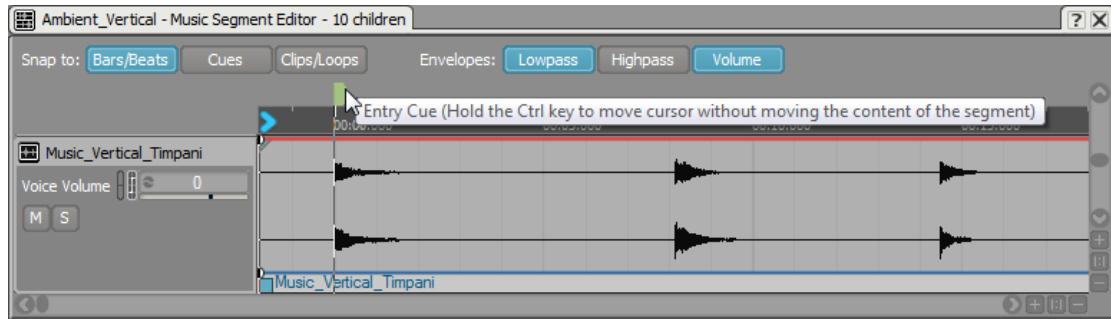
We can now add and adjust any cues or clip handles in order to prepare this ambient segment for use throughout the rest of the music system.

First, there are several percussion tracks (timpani, bass drum, and tubular bells) which can be extended (looped) to match the duration of the cymbals track. This can be accomplished by clicking and dragging on the blue clip handle at the bottom right of each track and matching the approximate length of the target track duration.



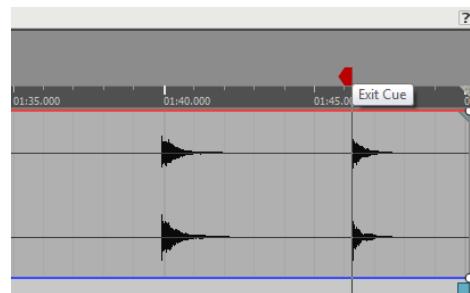
Using the clip handle to loop and extend the duration of a track

Next, adjust the entry cue to the first beat of the timpani by holding the Ctrl key and moving the cursor without moving the segment. This defines the entry cue of the segment and will ensure that any music transitioning to the ambient music segment will start on beat. The area to the left of the entry cue is the pre-entry area of a segment. The pre-entry area may or may not be played in game, depending on transition settings.



Entry cue moved to the first beat of the timpani

Next, adjust the exit cue by dragging the exit cue cursor to the last beat of the timpani. This defines the exit cue of the segment. The area to the right of the exit cue is the post-exit area of a segment. The post-exit area may or may not be played in game, depending on transition settings.

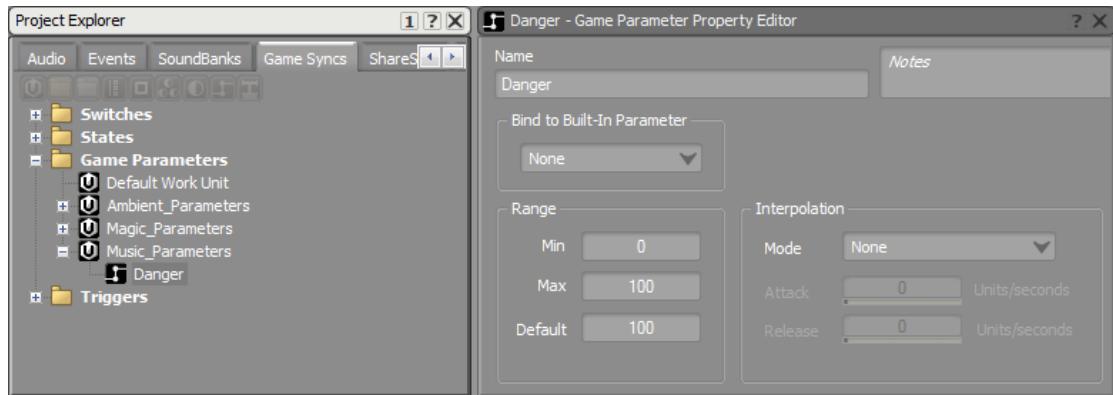


Positioning the exit cue

## Dynamic Danger

Now that entry and exit points have been established and a final duration for the ambient music has been set, it's time to set up volume based dynamics driven by potential danger. While the implementation of danger on the game side might include a value attached to each deadly forest creature which would then be passed to the audio engine, this feature can be auditioned directly within the authoring application.

The process begins with the creation of a game parameter called Danger in the game syncs tab of the Project Explorer.



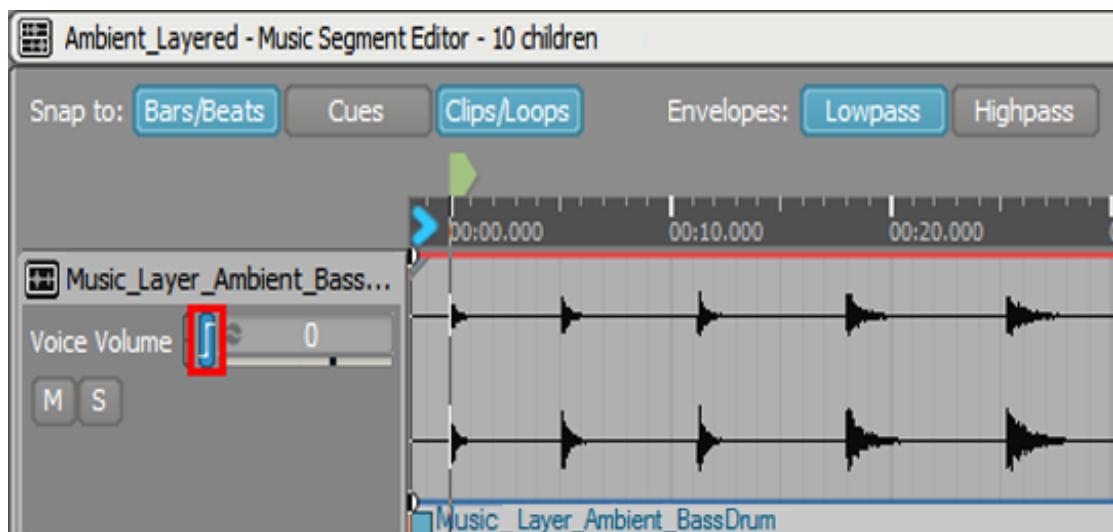
### Creating a Danger parameter to be used by the ambient music system

The Danger parameter will be used in conjunction with a volume RTPC for several tracks in the ambient system. Additional track properties are also configurable using RTPC such as: States, Effects, and Positioning as well as setting different busses, priorities, and limiting.

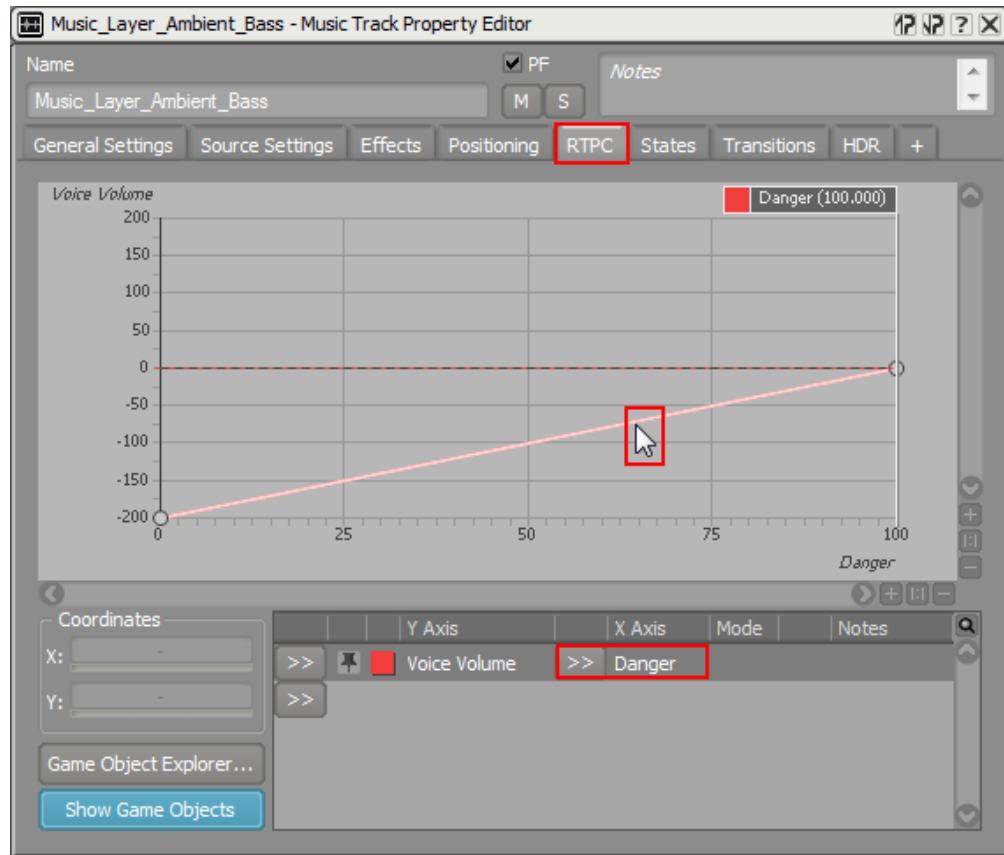
### Adding RTPC to Tracks

The intention in the multi-layered ambient system is to drive intensity by using volume in conjunction with several key tracks within the segment, specifically: Bass Drum, Cymbals, French Horn, Male Choir, and Trombones.

Double clicking the RTPC icon for the Bass Drum track opens up the RTPC tab for the track, where volume can then be added and associated with the Danger game parameter.



Accessing the property editor via the RTPC icon



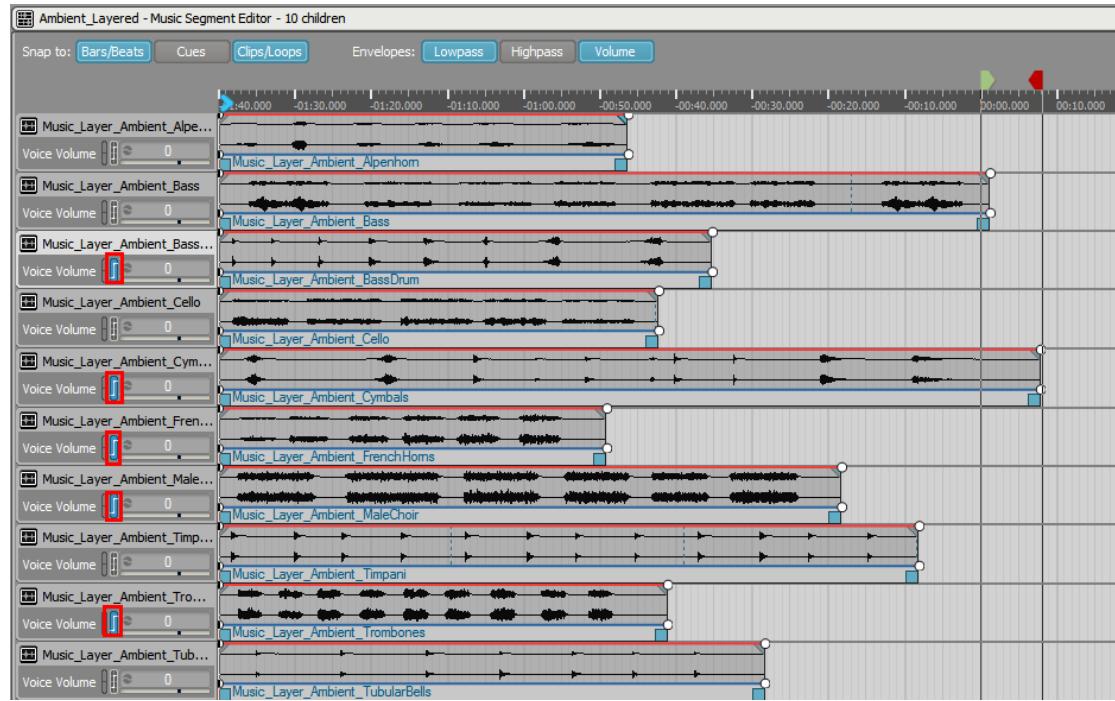
Adding volume adjustment in conjunction with the Danger game parameter for the Bass Drum track



### Tip

Click the line to select a different curve type. For example, choose Exponential so that the volume increases at a more progressive rate with increasing danger levels.

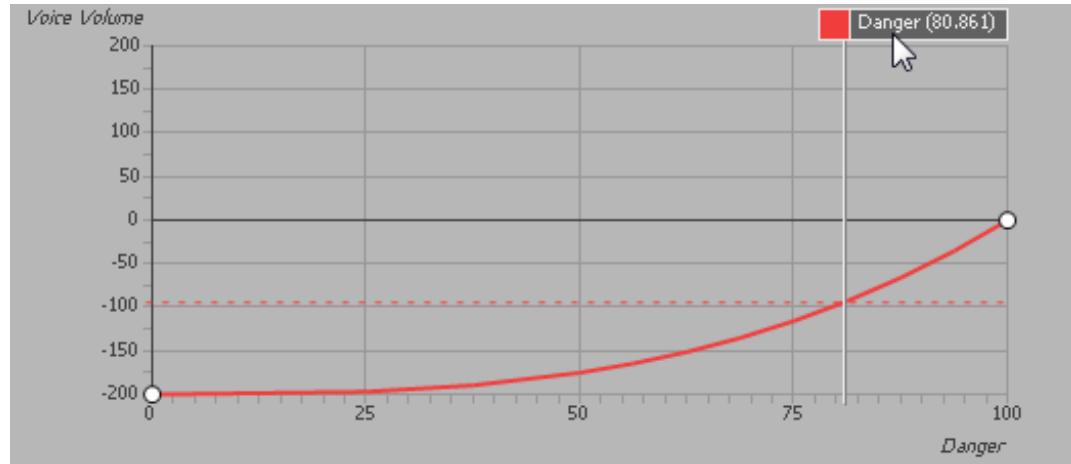
These same changes can be applied to additional tracks so that as the Danger parameter increases, additional instruments fade up and reveal layers of musical intensity.



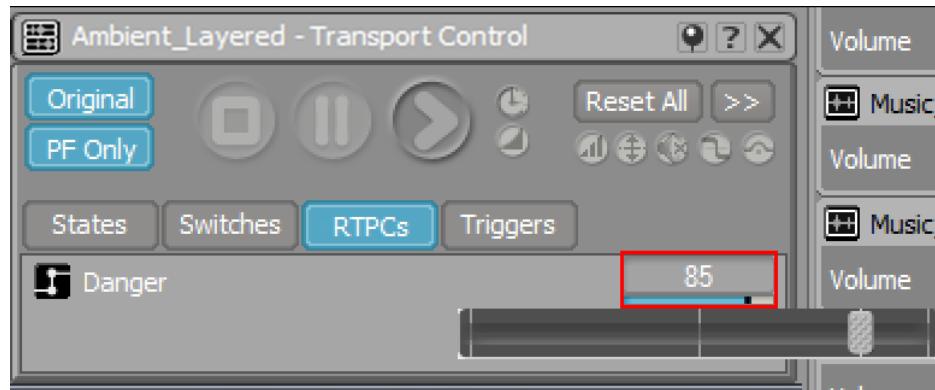
Completed ambient music system showing tracks with RTPC enabled

## Auditioning RTPC

Auditioning changes can be done either using the game parameter cursor in the RTPC view or through the RTPC tab in the Transport Control.



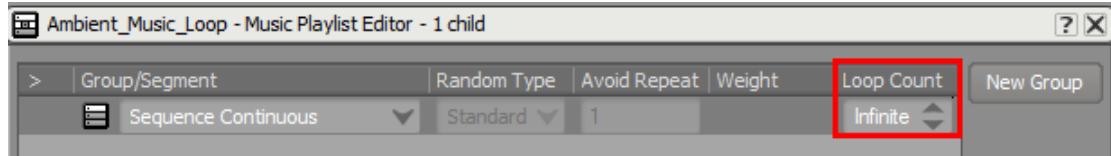
Auditioning RTPC with the game parameter cursor



Auditioning RTPC from the Transport Control

## Looping a Music Segment

Looping a Music Segment within the Interactive-Music hierarchy can be done by parenting the segment in a playlist container and managing its looping properties within its group. Once it's been added, the properties for looping can be found in the default group loop count. Clicking the down arrow changes the loop count to infinite, ensuring that the group will continue to play the segment until it is stopped by the audio system.



Setting the loop count to infinite playback

**Designer Note**

Do not confuse looping segments with looping audio clips inside segments.

## Section Summary

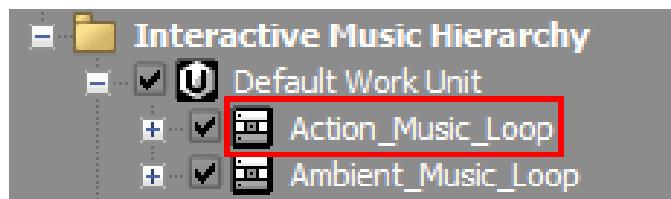
Throughout this section we have worked to create a dynamic loop of music that responds to the level of danger present in the game. Through the use of layering and real time parameter control, in conjunction with volume adjustment, this looping section of exploration music now reflects the hero's journey.

## The Sequential Approach

With a dynamic layered system in place to handle music during the exploratory section of gameplay, there must also come a time when the battle begins. In the case of the triumphant hero, the sequential approach will increase intensity level through the use of an action specific piece of music. In order to introduce a bit of randomness into the loop, the action music will be broken into two sections with variations, and a bridging segment within the playlist. We'll build these within the Music Playlist Container using groups.

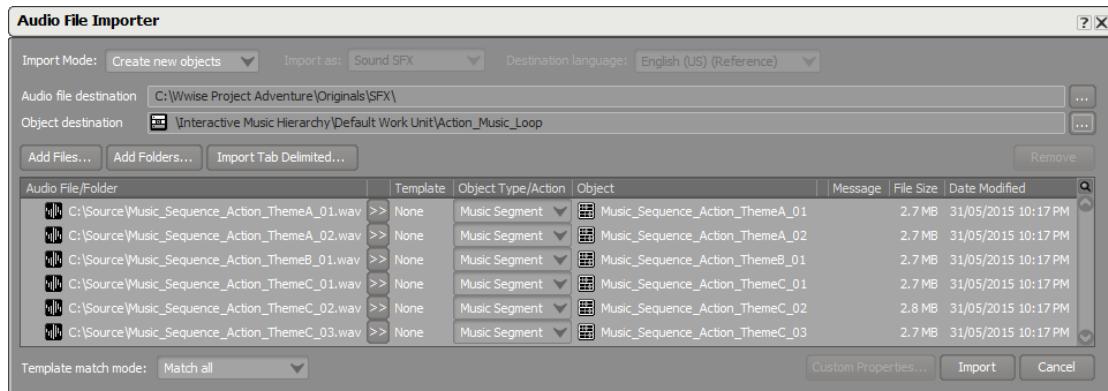
### Groups and Behaviors

The first step in the process begins with the creation of a Music Playlist Container that will represent the looping action music in the system.



Creating the action music loop playlist

Next, import the WAV files that will be used. By default these will be added to the Music Playlist Container as segments that can be further arranged within the Music Playlist Editor.



Adding the action music WAV files to the Music Playlist Container

Now that action music segments have been added to the project, they need to be dragged and dropped into the Music Playlist Container. Once added, they can be assembled into groups for further arranging. The playlist determines which objects will be played in what order when the playlist container is called by the game engine.

The four types of groups and segments available are:

- **Sequence continuous:** Plays all music objects in the group in sequential order each time the group is played.
- **Sequence step:** Plays only one music object in the group each time the group is played. The next time the group is played, the next music object in the group is played.
- **Random continuous:** Plays all music objects in the group one after the other in random order each time the group is played.
- **Random step:** Plays only one music object in the group selected randomly each time the group is played.

Additional randomization types, weighting, and repetition variables can be accessed in the Music Playlist Editor.



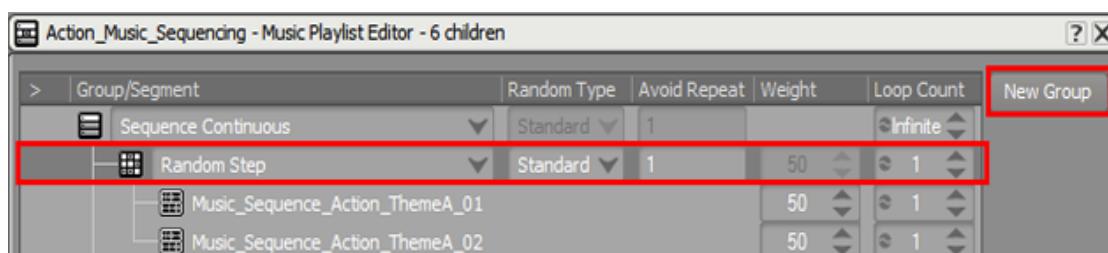
Variables for grouping segments and playback properties in the Music Playlist Editor

## Sequencing Groups in the Music Playlist Editor

The action music files are broken into 3 different themes: A, B, C.

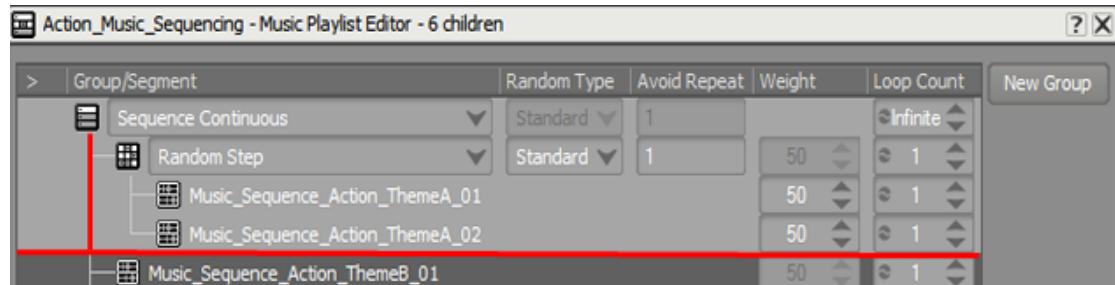
Themes can be arranged together by creating groups in the Music Playlist Container. In this case, taking the two Action Theme A variations, the only Action Theme B variation, and the three Action Theme C variations and sequencing them to create a varied action loop.

Start by creating a new group that will play as a random step and drag in the two Action Theme A variations.



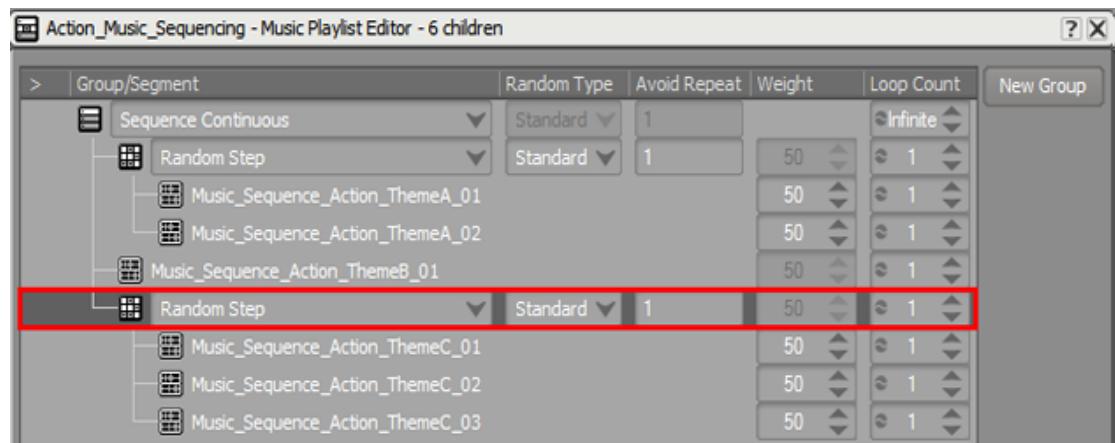
Creating a new continuous random step group for Action Theme A

Next, use the Action Theme B segment as a bridge between the two randomized groups created with the Action Theme A and C segments. Action Theme B can be dragged in and positioned within the parent sequence continuous group using the red indicator as a guide to its positioning.



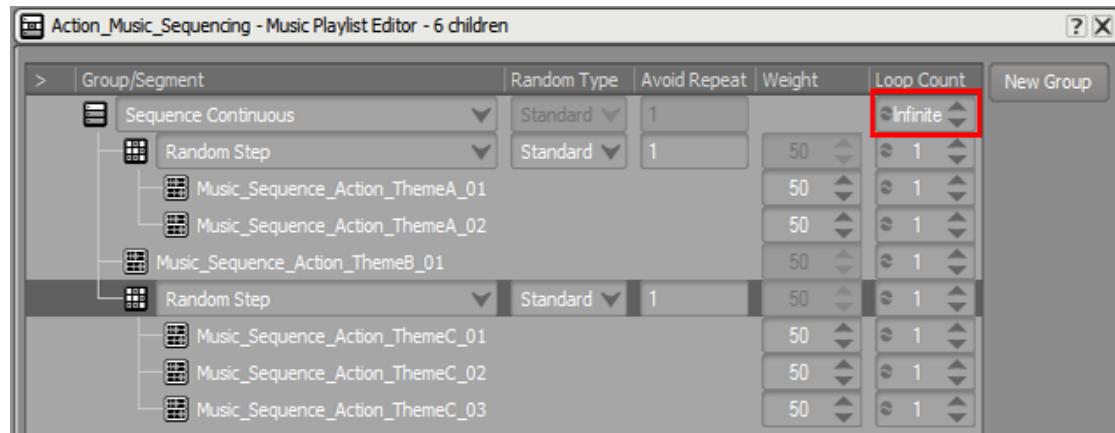
#### Arranging Action Theme B in sequence with the Action Theme A random group

Finally, Action Theme C will play as a random step for one of the three variations within its own group.



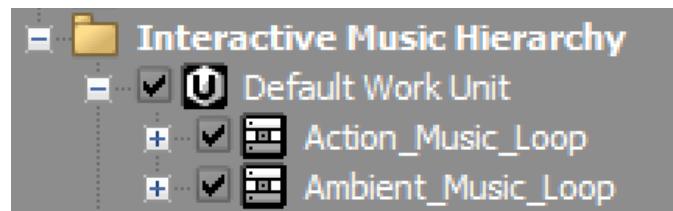
#### Creating a new random step group for Action Theme C and arranging it in sequence

Setting the parent sequence container to loop count infinite will cause the sequence to repeat itself until a change is called by the audio engine.



Setting the playlist container loop count to Infinite

There should now be two playlists containers: one for ambient music and another for action music.



Playlist containers for ambient and action music

## Section Summary

Here we have tackled the creation of looping action music that is randomly assembled using playlists of segments. The added sequencing of different segment types helps to give definition, progression, and drama to the unfolding music.

## Using States to Switch Between Music Types

Now that different types of music have been established for different scenarios, the next step is to integrate them so they can react to changes in gameplay. This can be done by introducing States (or game states) that can be used to switch between music containers, in addition to affecting other aspects of sound or music.

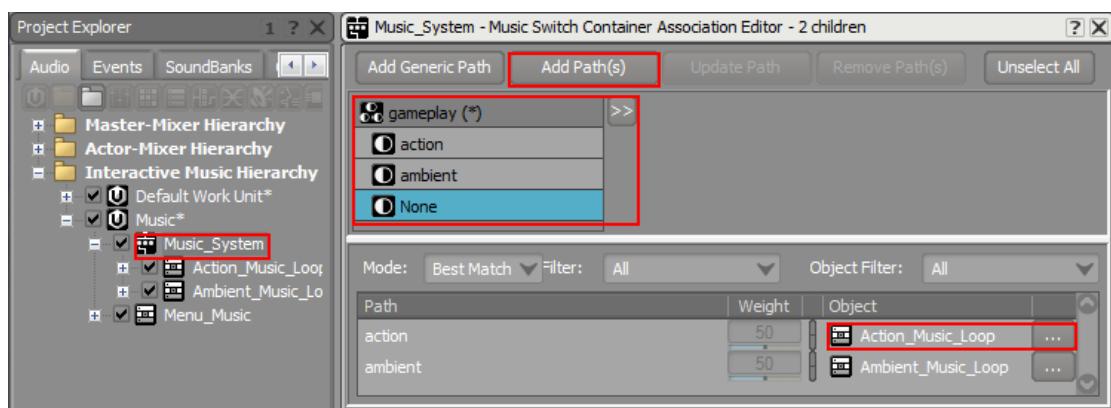
The process of creating a music switching system begins by defining states that will be communicated from the game and used to switch between different types of music, in this case: action and ambient music. States represent changes in the game that affect the properties of existing sounds, music, or motion on a global scale. The music switch container operates on the same fundamental principal as the switch container in the Actor-Mixer Hierarchy. The addition of time and tempo functionality allows for musical transitions when switching between containers.



Creating gameplay states to switch between music types

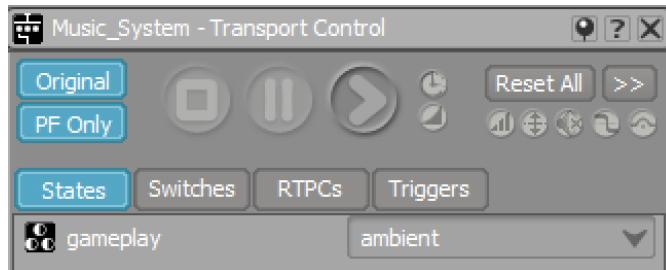
Parenting the already existing ambient and action music loop playlists in a music switch container allows for the application of the newly created gameplay state group.

Music Playlist Containers can now be assigned to each of the states in the Music Switch Container Association Editor either by dragging and dropping them from the hierarchy or using the Selection button.



Creating a music switch container using the gameplay state group and assigning a Music Playlist Container to its states

Now that the simple assignment of music playlist containers to states has been established, the default transition between music types can be auditioned via states in the transport control.



Auditioning music changes based on states

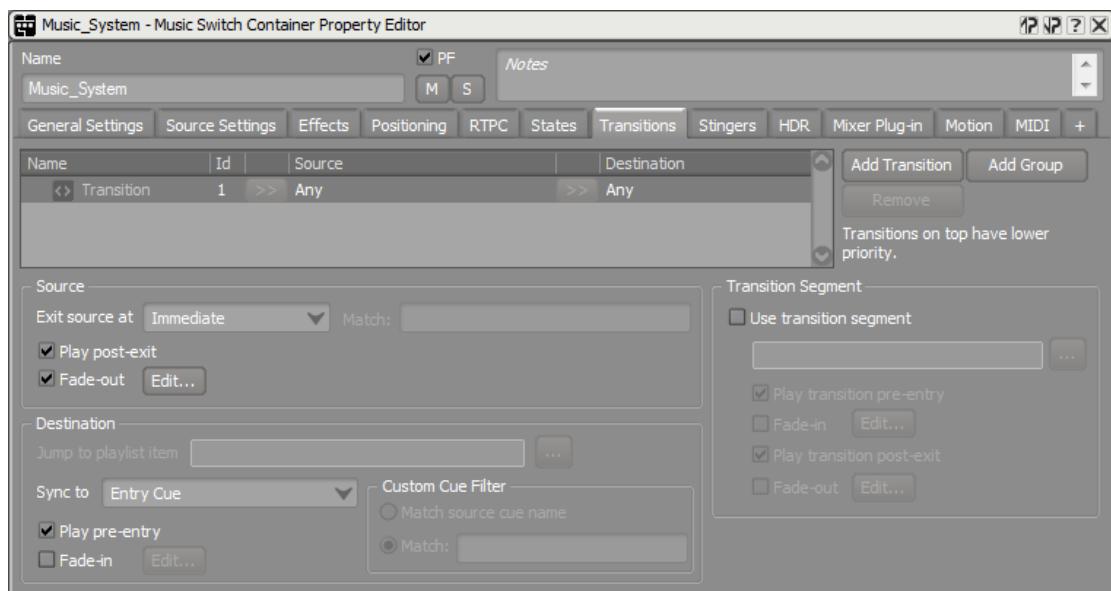
## Defining Interactive Music Transitions

In order to gain more control over the way music segments and music playlist containers transition, the music switch container provides the ability to author specific transition behavior.

The Transitions tab is where you define the transitions between music objects within either music playlist or music switch containers. A transition is a musical behavior you can define to be used when switching from the music object currently playing to another. Each transition has a source and a destination. An additional segment, called a transition segment, can be used as a musical bridge between the source and destination.

The Transitions tab contains the Transition Matrix, which allows you to create a set of rules that define how each object transitions to every other object within the container. You can create explicit rules for each object, or more general rules that apply to more than one object. The default Any to Any transition rule applies to all remaining undefined transitions.

The list of transitions is displayed in the Transitions tab in descending order. When a transition is needed, Wwise begins searching at the bottom of the list until it finds one that applies to the situation at hand. If no matching transition is found, Wwise uses the default Any to Any transition.

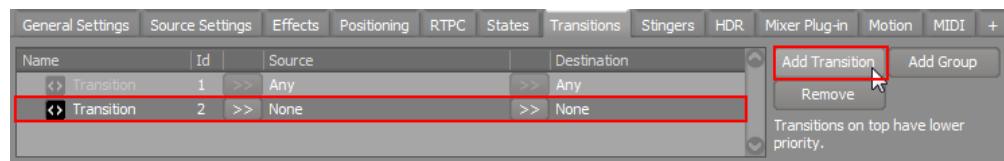


The Transitions tab and Transition Matrix

## Authoring Transitions

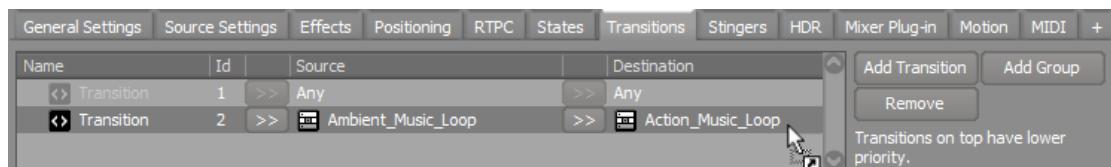
We'll begin by adding explicit transition segments in order to move musically from the ambient loop, to the action loop, and back again. This can be accomplished by adding a new transition to the transition matrix and defining its behavior, starting with the transition from the Ambient Music Loop playlist to the Action Music Loop playlist.

First add a new transition using the Add button.



Adding a new transition to the transition matrix

Next drag and drop the source and destination music objects, in this case the Ambient Music Loop playlist (Source) and the Action Music Loop playlist (Destination).



Dragging and Dropping music objects to the transition matrix

## Defining Transition Behavior

Transition behavior between the source and destination can now be authored, specifically: exiting the source, entering the destination, and the specifying of any transitional segment.

By default, a transition is a simple change from one music object to another. The real power of transitions appears when you customize the source and destination to make a unique musical passage. By setting source and destination properties, you can make a transition between objects sound both smooth and musical.

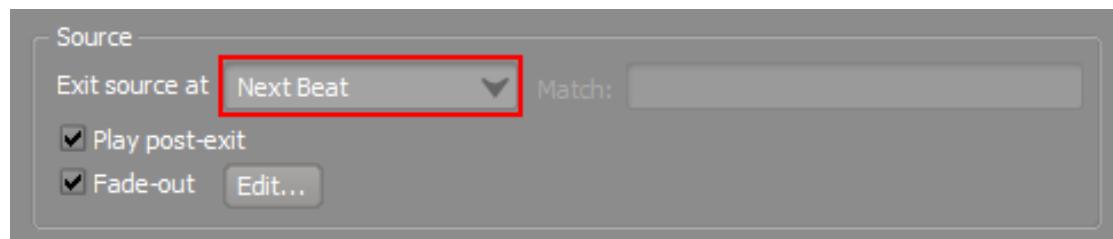
To give you additional control and flexibility over the transitions between objects, you can choose from a number of different exit and entry points for the source and destination respectively.

There are many potential options associated with exiting the source:

- **Immediate:** The source stops playing immediately.
- **Next Grid:** The source stops playing at the next grid interval. The grid is an arbitrary method by which music objects can be virtually partitioned.
- **Next Bar:** The source stops playing at the next bar.
- **Next Beat:** The source stops playing at the next beat.
- **Next Cue:** The source stops playing at the next cue, whether it be a custom cue or the exit cue.
- **Next Custom Cue:** The source stops playing at the next custom cue. If the current music segment doesn't contain a custom cue, Wwise continues to the next segment until it finds a custom cue.
- **Exit Cue:** The source stops playing at the exit cue.

## Transitioning from Ambient to Action Music

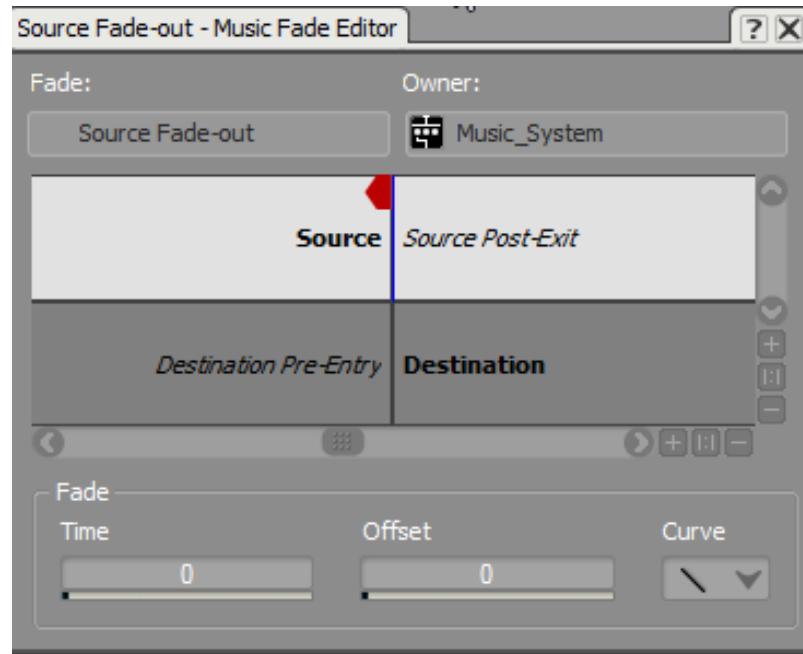
Transitioning at the next beat will suit the purpose of moving quickly from ambient to action music and so specifying Next Beat in the Exit Source window will accomplish this.



Defining the source exit source

By default the source transition is set to play the post-exit of the source during the transition, however, the post-exit of a source will only play if that source exits at its exit cue, or fades out at or beyond its exit cue. Otherwise, the post-exit will never play during a transition.

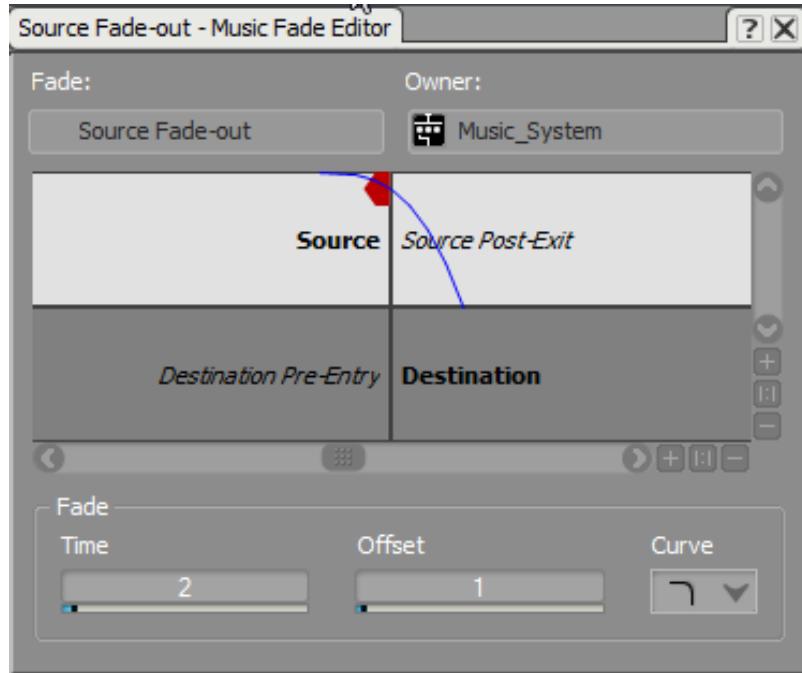
In this transition we also want to end playing with a fade-out. This can be accomplished by selecting Fade-out and pressing edit to enter the Music Fade Editor.



Adding a source fade-out in the music fade editor

The Music Fade Editor is where you can define the properties of each individual fade used when transitioning from one music object to another. This includes fade-ins for destination objects, fade-outs for source objects, and fade-ins and fade-outs for transition segments. You can define the length and offset of each fade, as well as the curve shape to further customize the transition.

For the example, it has been determined that a two second fade with a one second offset using a steep convex curve will allow the ambient music to fade quickly out and sound natural as it transitions to the action music playlist.



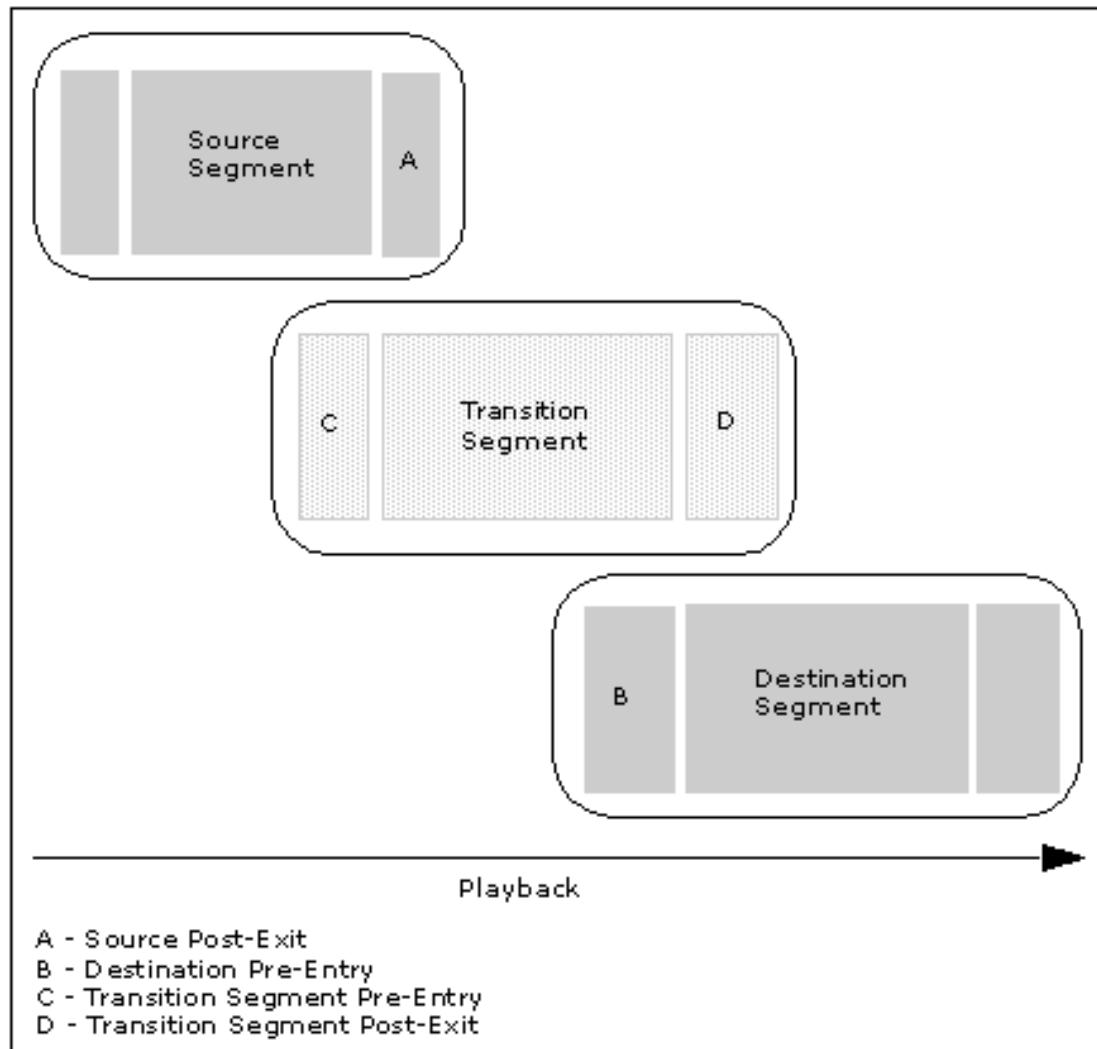
Specifying the fade settings for the ambient to action music transition

Auditioning the transitions can be done in the same way as outlined previously.

### **Transitioning from Action to Ambient Music**

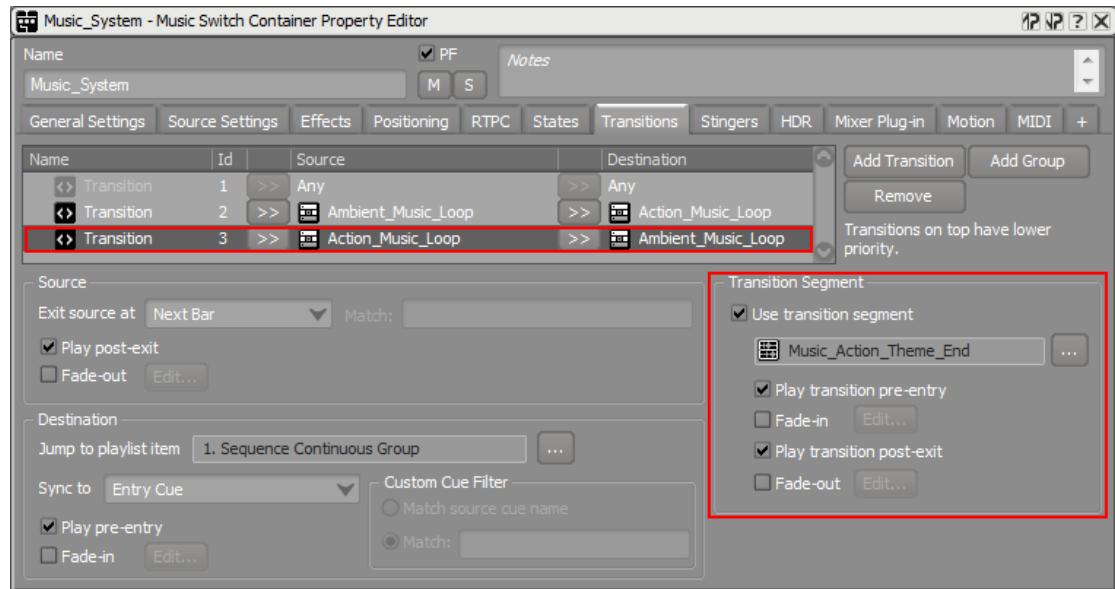
Handling the transition from the action music to ambient requires special handling because of the looping nature of the action playlist, and also in order to capitalize on the dramatic intent signaled by the end of the action music.

At times, a transition sounds better if another piece of music plays over the end of the source and the beginning of the destination. This bridging piece of music is called a transition segment, and you can use one for any transition in Wwise. The following illustration shows how a transition segment is played between the source and destination of a music transition.



You can also use any combination of the pre-entry and post exit areas of the source, destination, and transition segments to create even more seamless transitions.

In order to bridge the transition from the Action Music Loop playlist (Source) back to the Ambient Music Loop playlist (Destination) the addition of the Action Theme End must be used as a transition segment as part of a new transition in the transition matrix.



Specifying a transition segment between the Action Music Loop playlist (Source) and the Ambient Music Loop playlist (Destination)

## Section Summary

Handling the delicate relationship and transitioning between music playlist containers and segments is one of the cornerstones of providing a seamless musical experience to the player. With the introduction of switching based on game state and the inclusion of a transition segment serving as a bridge between action and ambient loops, the music is allowed to naturally progress and respond to the game.

## Music Summary

This chapter illustrates a very simple state based music system that operates both in layers, based on intensity, and sequentially, based on game state. The use of tempo based transitions helps keep the music in-sync across state changes, while the amount of danger dynamically controls the volume of different ambient music tracks. The combination of these techniques results in a musical soundtrack that is customized to the player's experience.

**Throughout this chapter we have:**

- Discussed the role of interactive and dynamic music in games.
- Discussed the use of different types of implementation strategies.
- Talked about preparing music content for integration.
- Compared the Interactive Music Hierarchy with the Actor-Mixer Hierarchy.
- Established the use of Music Segments, Music Playlist Containers, and Music Switch Containers.

**Stepped through the process of:**

- Creating a music implementation based on the Sequential Approach using RTPCs to control the dynamics of the ambient music:
  - Creating Music Segments.
  - Adjusting track lengths.
  - Looping Music Tracks.
  - Adding Music Segments to Music Playlist Containers.
  - Looping a playlist.
  - Creating game parameters.
  - Modifying Music Track volume based on RTPCs.
- Creating a music implementation based on the Sequential Approach using a playlist to randomly recombine action music segments.
- Switching between music based on state:
  - Defining the transition matrix between source and destination.
  - Defining transition behavior.
  - Setting variables for exiting a source.
  - Using the music fade editor.
  - Defining a transition segment.

**We've also touched on:**

- Working in the Interactive Music Layout.

**Throughout this section we have created the following objects:**

- A music system Switch Container that switches between ambient and action music based on game state.

- A looping ambient Music Playlist Container which includes an ambient Music Segment and ten tracks of music with volume RTPC based on the danger game parameter.
- A game parameter used to simulate danger.
- A playlist container which includes Music Segments that are randomized to create a seamless loop of action music.
- An action music theme ending used as a transition segment when switching from the action to ambient state.

## **Music Additional Resources**

[Video Tutorial - Creating Interactive Music Structures](#)

[Video Tutorial - Defining Interactive Music Transitions](#)

[Video Tutorial - Creating Stingers](#)

[Game Sound Design - Making Music Interactive: Elaboration of the Feature Set in Wwise](#)

[Game Sound Design - Dynamic Music Creation Using Wwise](#)

---

## **Chapter 8. Adventures in MIDI**

Overview .....	174
Importing MIDI Files .....	175
Section Summary .....	179
Setting up Wwise Synth One .....	180
Adventures in Synthesis .....	181
Section Summary .....	184
Connecting MIDI & Sound .....	185
Importing the Individual MIDI Tracks .....	185
Music Segment MIDI Properties .....	187
Sound Object MIDI Properties .....	188
Section Summary .....	191
MIDI Summary .....	192

## Overview

The hero lies in wait for the winds of change to blow their favorable tidings across the wind-swept field of battle. He seeks knowledge from the oracle, the all-knowing eye of history, brimming with stories of the old days - tales of long-forgotten alchemy and strategies that could help reclaim his position as the rightful heir to the throne. Equipped with an understanding of these ancient ways, there can be no failure- he has the tools at hand and the power of the past to create the ultimate good.

A comprehensive pipeline to utilize MIDI (Musical Instrument Digital Interface) in conjunction with synthesis and sound samples is a powerful combination. The standardized MIDI format is at the heart of every modern DAW (Digital Audio Workstation) and serves as an interoperable format that communicates note, velocity, and other control information for playback. This data can be mapped to sound samples or synthesizers within Wwise to create efficient workflows for music or sound design. The power and flexibility of MIDI is well documented as part of a DAW workflow and the benefit of this format is available within Wwise.

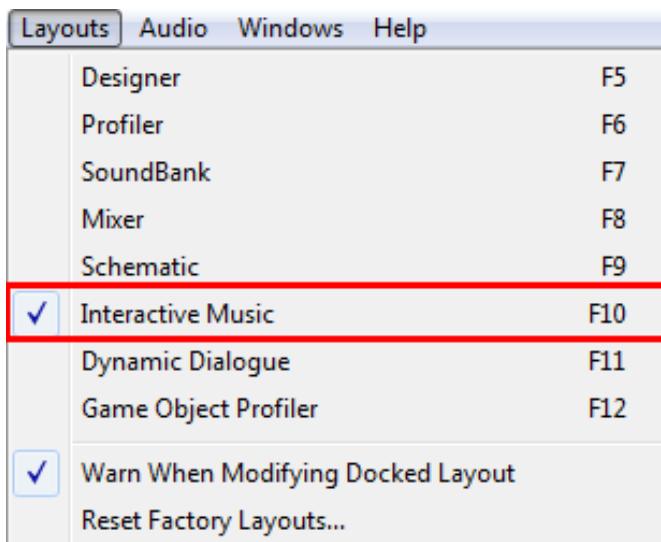
This chapter will take you through the process of:

- Importing MIDI files
- Understanding the features and functionality of Wwise Synth One
- Adding MIDI files to a music track
- Modifying the MIDI target
- Adjusting the MIDI clip tempo

## Importing MIDI Files

Importing MIDI files can be achieved using the same processes outlined in Chapter 1's [Building the Foundation](#). MIDI files added to the Interactive Music Hierarchy are imported as Music Segments and can also be added as tracks within any existing music segment. The MIDI file includes all of the messages that specify notation, pitch, velocity, and other control information. After the MIDI file has been established in the Wwise Project, it can be replaced or edited like any audio file.

To view the music specific functionality of containers in the Interactive Music Hierarchy, you need to switch to the Interactive Music layout.

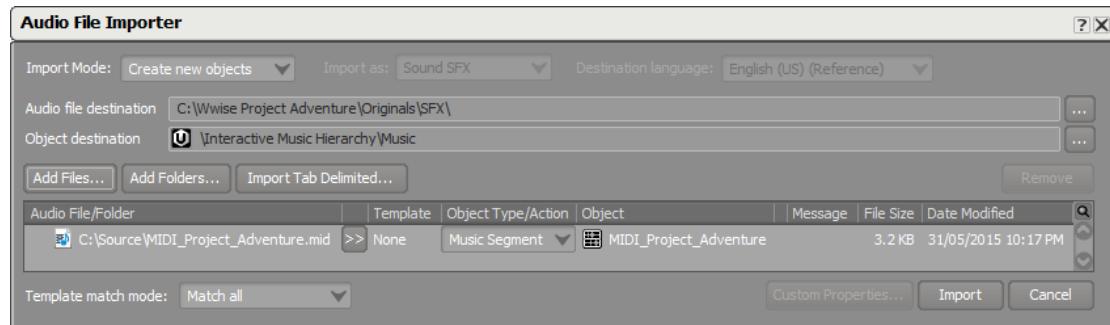


Switching to the Interactive Music Layout

MIDI Files can be imported in any of the following configurations:

- One MIDI file containing all tracks, with each track assigned to a unique MIDI Channel (1-16)
- One MIDI file for each track, with each track assigned to a unique MIDI Channel (1-16)
- One MIDI file for each track, with each track assigned to the same MIDI Channel (1)

MIDI files can be imported as Music Segments or as tracks within an existing Music Segment by either using the Audio File Importer or by dragging and dropping them into the Interactive Music Hierarchy. After files have been added as tracks to the segment, they can be accessed through the Music Segment Editor like any other music track.



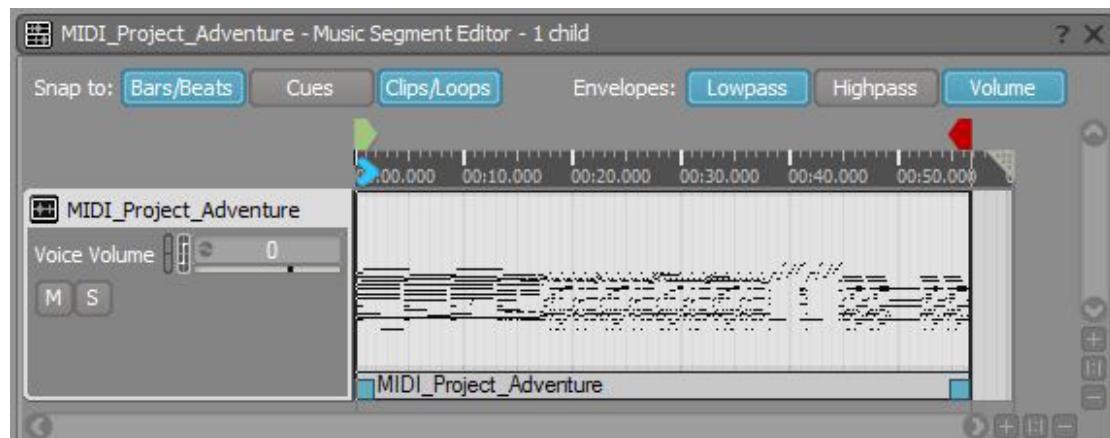
The Audio File Importer showing a single  
MIDI file to be imported as a Music Segment

Alternatively, you can drag and drop MIDI files from folders directly into the Wwise Interactive Music Hierarchy.



Imported MIDI\_Project\_Adventure .mid file

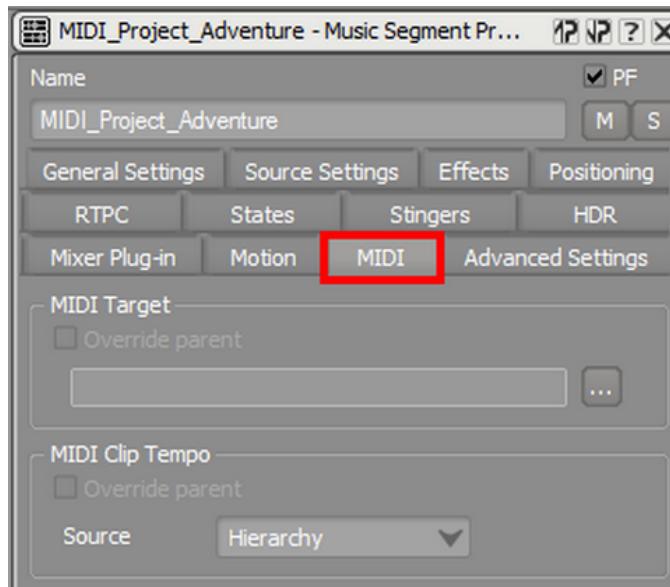
After MIDI files have been added as tracks to the segment, you can access them through the Music Segment Editor, where any combination of audio and MIDI tracks can be arranged and edited within the Music Segment.



Editing tracks in the Music Segment Editor

MIDI Tracks within a Music Segment exhibit the same behavior and functionality as an audio file, with one exception: audio properties are not modifiable in the Music Segment. Since MIDI is a data representation only, the audio properties are a part of the MIDI Target or "Instrument".

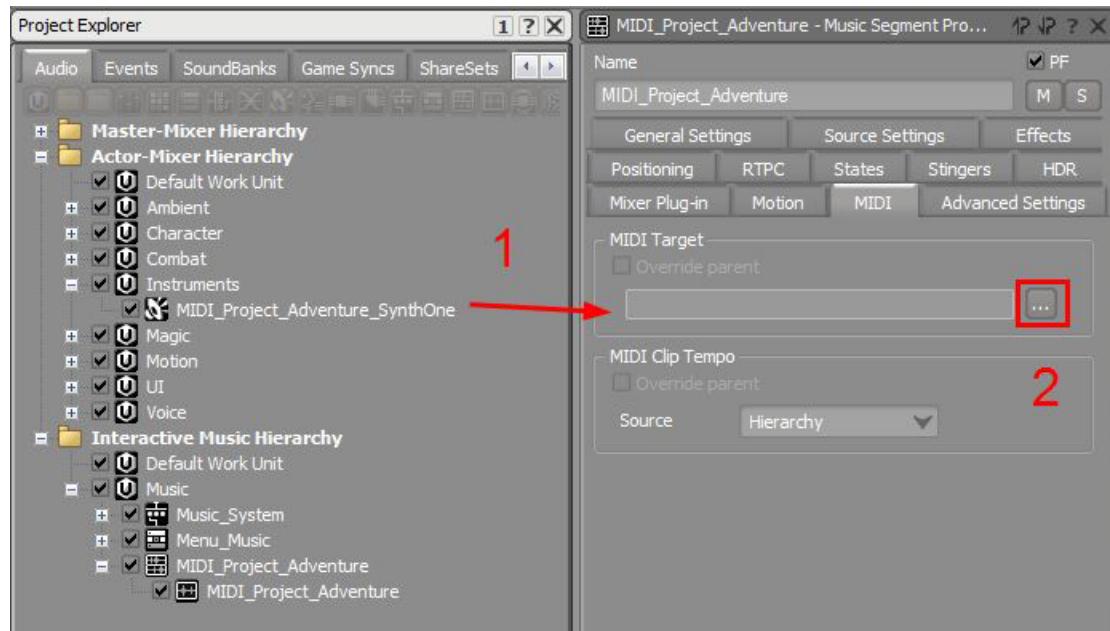
The MIDI Target and MIDI Clip Tempo can be defined for either the Music Segment (Parent) or Music Track (Child) in the MIDI tab in the Music Segment Property Editor or Music Track Editor.



The MIDI tab in the Music Segment Property Editor

The MIDI Target is specified as the Sound Object through which all MIDI events are routed. This property can target a specific Sound Object or a Blend Container including multiple Sound Objects, each with their own individual MIDI properties.

The MIDI Target can be assigned to a Music Segment or Music Track by navigating to a Sound Object through the Project Explorer - Browser or by dragging and dropping a Sound Object from the Actor-Mixer Hierarchy onto the MIDI Target field.



Assigning the MIDI Target using drag and drop  
(1) or using the Project Explorer - Browser (2)

A MIDI Clip can use either the Tempo specified in the MIDI file to control its tempo or it can use the tempo set for the Music Segment.

The MIDI Clip Tempo can either use tempo information from the General Settings tempo specifications for the Music Track hierarchy or tempo from the (MIDI) file.



Assigning the MIDI Clip Tempo Source

There are some caveats to using MIDI Clip tempo information within the Wwise Music System:

- Transition scheduling between segments is always done using the tempo set for the segment and is never influenced by tempo events indicated in the MIDI clips/files.
- Triggers are scheduled to play on the beat, bar, grid of the music segment and not based on the tempo of the MIDI clips/files.
- MIDI files/clips can be exported with one or more tempo changes and, if the option 'file' is selected, the clip will listen to these tempo changes to play back the MIDI notes and CC on time.

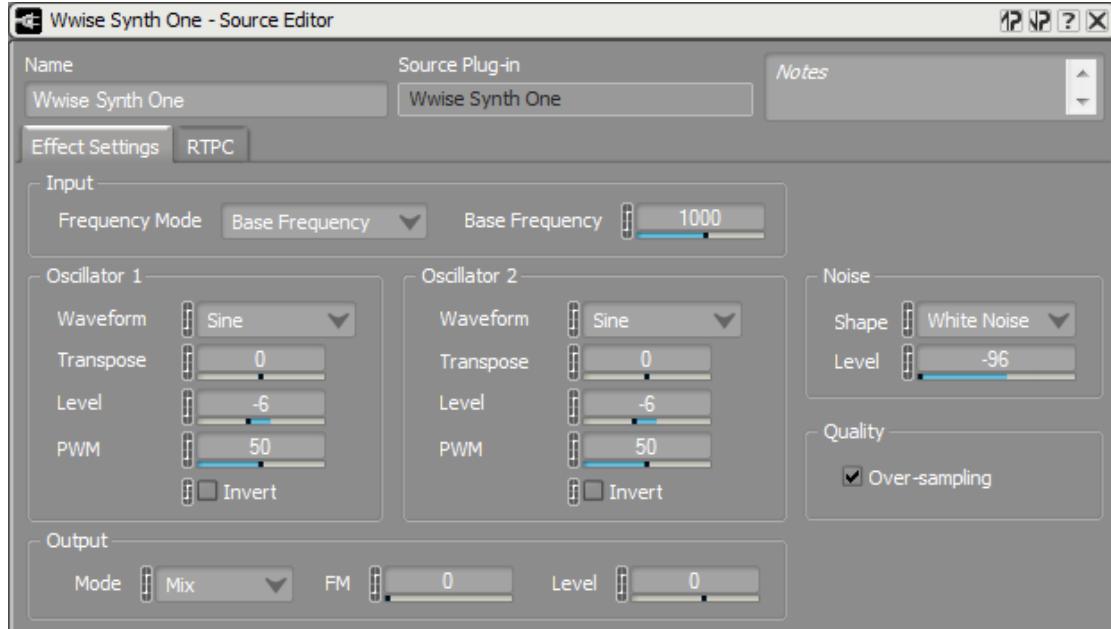
- These MIDI file/clip-based tempo changes do not influence the tempo of the Segment; thus MIDI clip and segment transitions can easily become out of sync.

### **Section Summary**

In this section we have imported a MIDI file and described the track-specific MIDI functionality. The MIDI file is the currency of communication between a composition created within a DAW and its eventual presentation in the game.

## Setting up Wwise Synth One

Before we begin working with MIDI in the Interactive Music Hierarchy, let's take a look at the synthesizer functionality of the Wwise Synth One source plug-in.



Source Editor - Wwise Synth One

- **Input**
  - **Frequency Mode** - The source of the input frequency used by the oscillators.
    - Base Frequency - Frequency is obtained from the Base Frequency property.
    - Midi Note - Frequency is obtained from received MIDI note events.
  - **Base Frequency**
    - 20-20000 Hz - Input frequency for oscillators in Hz.
- **2 Oscillators**
  - **4 Waveform Types** - Available for each Oscillator.
    - Sine
    - Triangle
    - Square
    - Sawtooth
  - **Transpose:** The Pitch of the Oscillator
    - -3600 to +3600 cents - Transposition of input frequency, in cents.
  - **Level** - Output level of Oscillator, in dB, applied before the Oscillators are combined.
  - **PWM (Pulse Width Modulation)** - A technique that conforms the width of the pulse based on the modulator signal.

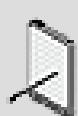
- **Invert** - Inverts the Output of the Oscillator.
- **Output** - How the Oscillator outputs are combined
  - **Mode**
    - Mix: the samples are added
    - Ring: the samples are multiplied
  - **FM (Frequency Modulation)** - This value determines how much of Oscillator 2's output is used to generate oscillator 1's output.
    - 20-20000 Hz
  - **Level (Volume)** - Level applied to the final signal (combined Oscillator outputs mixed with noise generator output).
- **Noise**
  - **Noise Shape** - The type of noise generated.
    - White Noise
    - Pink Noise
  - **Noise Level (Volume)**: Level applied to the output of the noise module, in dB. The level is applied before the output of the noise module is mixed with the combined Oscillator outputs.

## Adventures in Synthesis

The MIDI Project Adventure uses four instruments that were recreated using the functionality of the Wwise Synth One plug-in. These instruments include: two lead square wave synths, a bass synth comprised of a combination of sine and sawtooth waveforms, and a percussion synth utilizing white noise. Experimentation is the key when working with synthesis, and often the best sounding synths are the result of the creative exploration within the limitations of a feature set.

Here are some things to keep in mind when designing your synths:

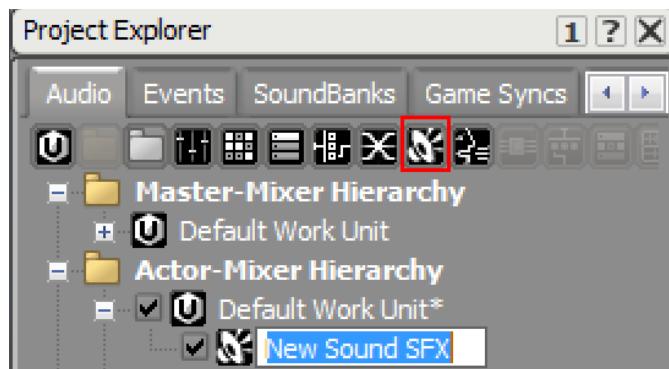
- **RTPC**: Real Time Parameter Control can be used in conjunction with a Game Parameter set by the game or internally from Wwise as part of a Wwise Meter effect plug-in to output a game parameter to modify properties of the synth.
- **Modulator Envelope**: Modulator Envelopes can be used to control the Attack, Decay, Sustain, Release and other Envelope behaviors of any property that can be modified using RTPC.
- **Modulator LFO**: Modulator LFOs can be used to modify the properties of Wwise Synth One as a RTPC which can produce wildly varied dynamics.



### Designer Note

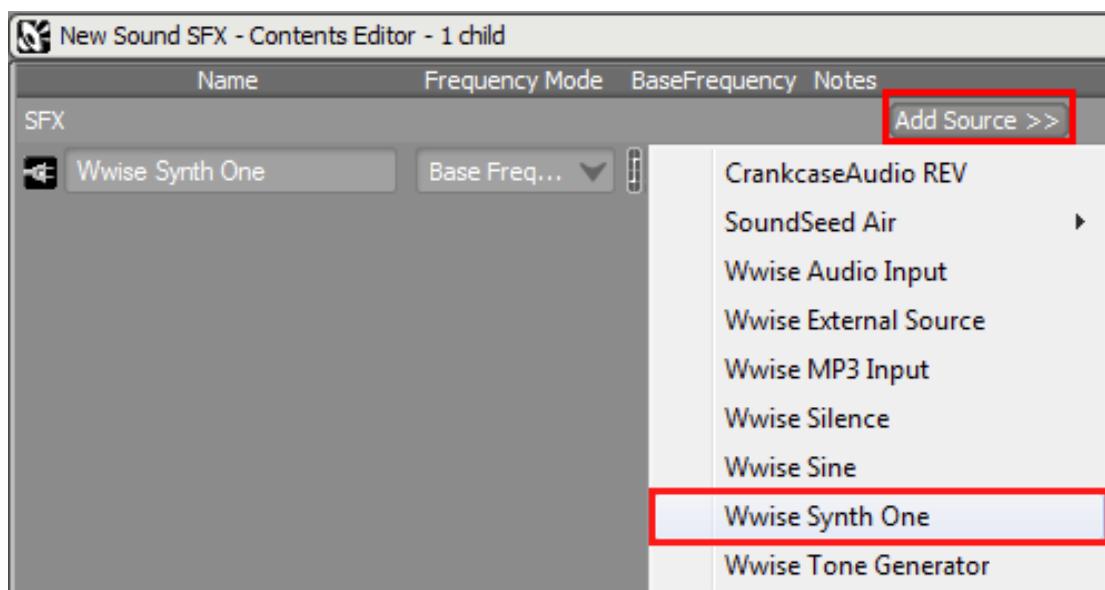
For more information on Modulator LFOs and Modulator Envelopes, see the 'Modulators' section in the Making Magic chapter.

The Wwise Synth One plug-in can be added as an input source to a Sound SFX in the same way that the Silence plug-in was added in Chapter 1. To begin, add Sound SFX object to the default work unit by clicking the Sound SFX icon in the Project Explorer toolbar; a new Sound SFX is created. Alternatively, Sound SFX can be created from the contextual menu or by using shortcut keys.



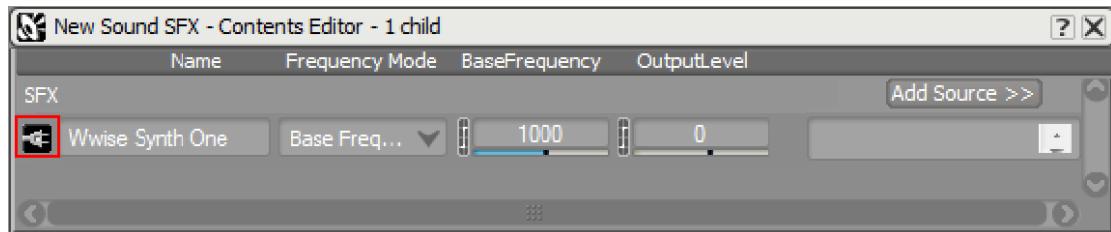
Creating a Sound SFX from the Project Explorer toolbar

After double-clicking to select the new Sound SFX, you can add the Wwise Synth One plug-in from the Add Source menu in the Contents Editor.

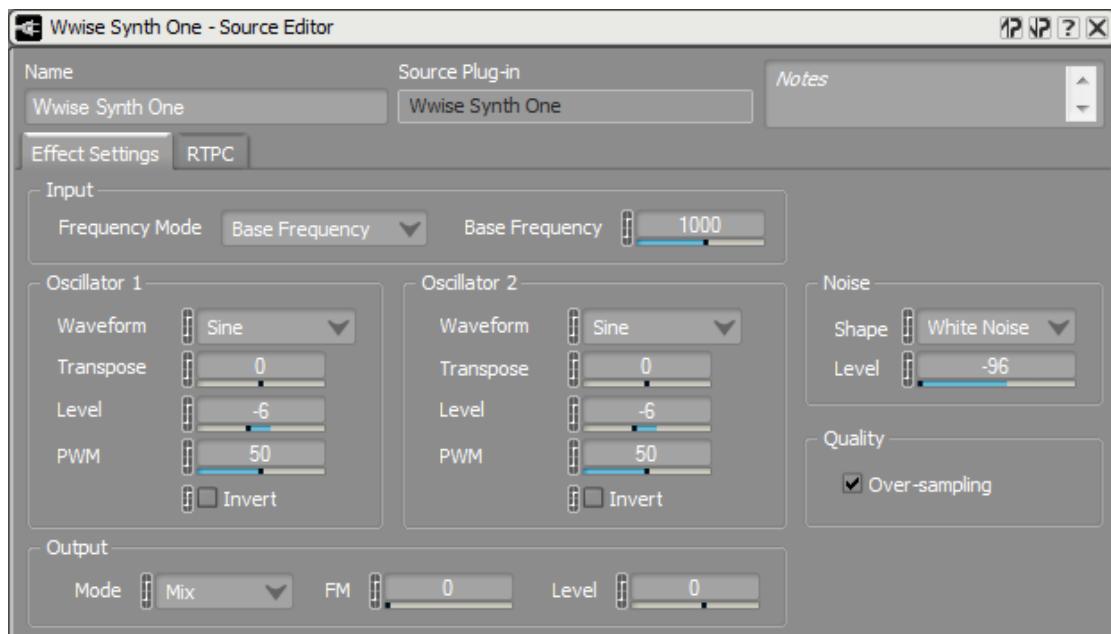


Creating a Wwise Synth One source plug-in for a sound object

The Source Editor can be accessed in the Contents Editor by double-clicking the plug-in icon.

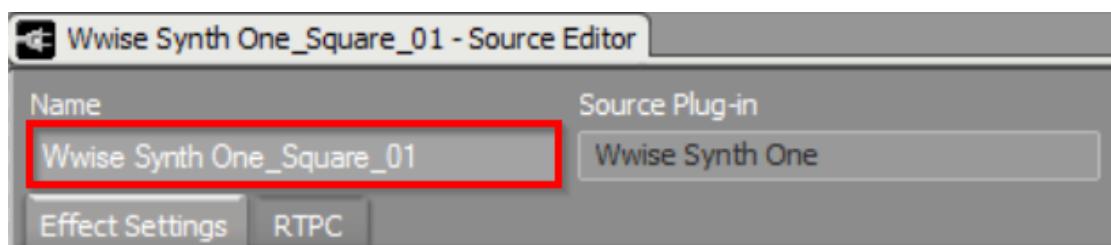


Source plug-in icon for Wwise Synth One



Wwise Synth One - Source Editor

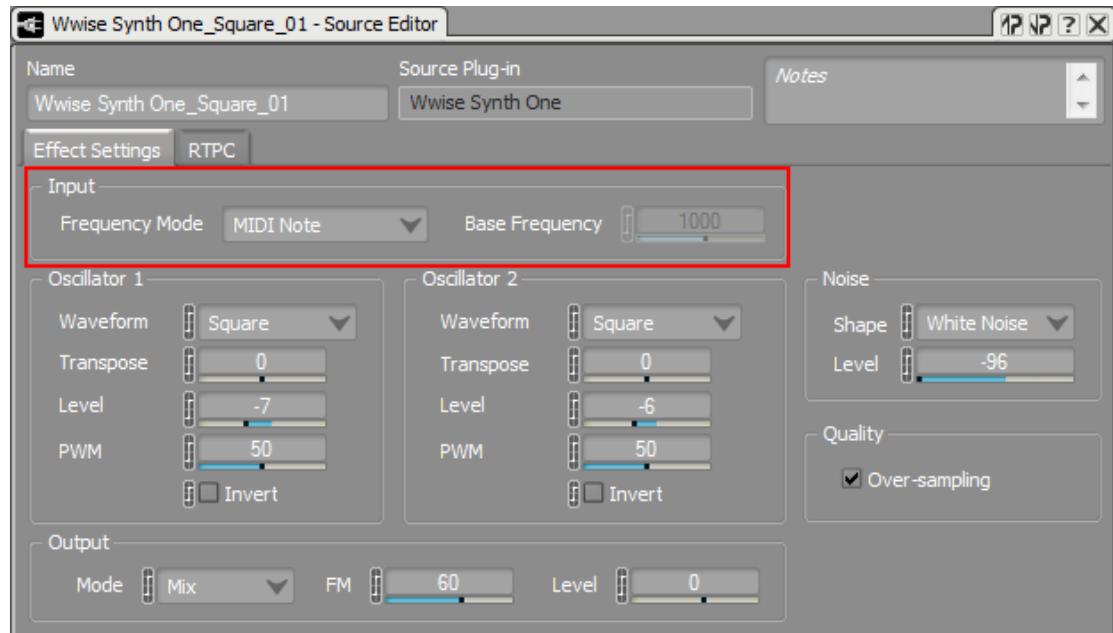
The name of the Wwise Synth One source plug-in can now be renamed in the Source Editor to reflect its intended use.



Renaming the Wwise Synth One plug-in in the Source Editor

The synth can be auditioned by selecting the Sound Object in the Actor-Mixer Hierarchy and pressing 'Play' (or Spacebar).

By default, the Input Frequency Mode is set to Base Frequency and uses the Base Frequency property value in Hz as an input for the Oscillators. To use the Wwise Synth One plug-in in conjunction with a MIDI file, the Input must be set to MIDI Note.



Setting the Input Frequency Mode to MIDI Note in the Source Editor

Without much effort, a basic instance of the Wwise Synth One plug-in can be ready for use in conjunction with a MIDI file (or files).

## Section Summary

The process and methodology of synthesis is a well-established part of creative music making. The inclusion of Wwise Synth One as part of the runtime functionality of the Wwise audio engine opens up a wide-range of capabilities to harness the power of realtime sound synthesis. The ability to modify properties dynamically unlocks a veritable treasure-trove of interactive possibilities.

## Connecting MIDI & Sound

The compositional task has its underpinnings in the graphical representation of notes and behaviors. Whether on manuscript (staff) paper, a "piano roll", or beneath the surface of a DAW as MIDI notes, the notation serves as a mechanism to communicate the intended performance to be played back by performers or their mechanical/ digital equivalent. Operating below the surface of most computer-based composition tools, MIDI has provided a consistent and standardized way to represent a composer's musical intention. Wwise acts as an interpreter for this intention and can be used to play back and participate in the interaction between the composition and the "game as performer" modifying properties of the composition in realtime.

A MIDI file from a DAW must be exported with special considerations for use in the Wwise Authoring Application. Specifically, any tempo and MIDI channel configurations should be assigned in correlation to their expected use. Wwise will, by default, use the tempo and MIDI channel assignments exported in the MIDI file to control these aspects.

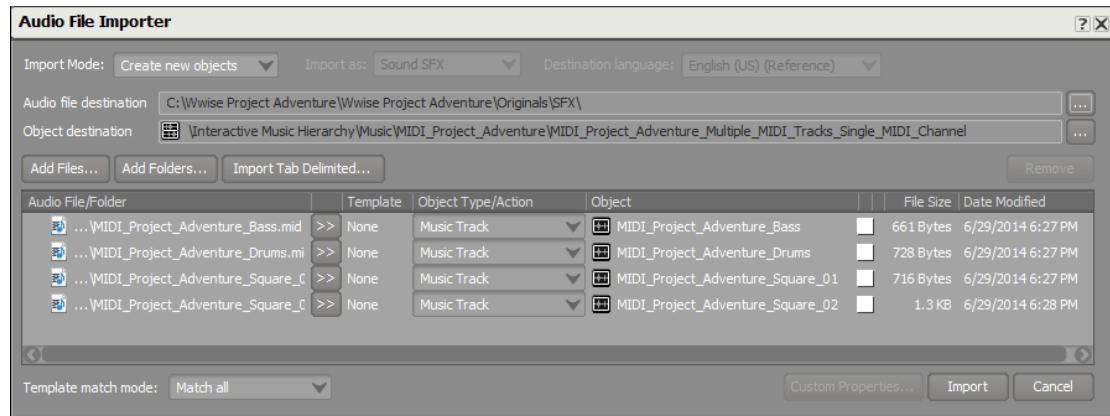
### Importing the Individual MIDI Tracks

The instrument tracks have been exported from a DAW as four individual files which represent the different instruments:

- MIDI\_Project\_Adventure\_Square\_01.mid
- MIDI\_Project\_Adventure\_Square\_02.mid
- MIDI\_Project\_Adventure\_Bass.mid
- MIDI\_Project\_Adventure\_Drums.mid

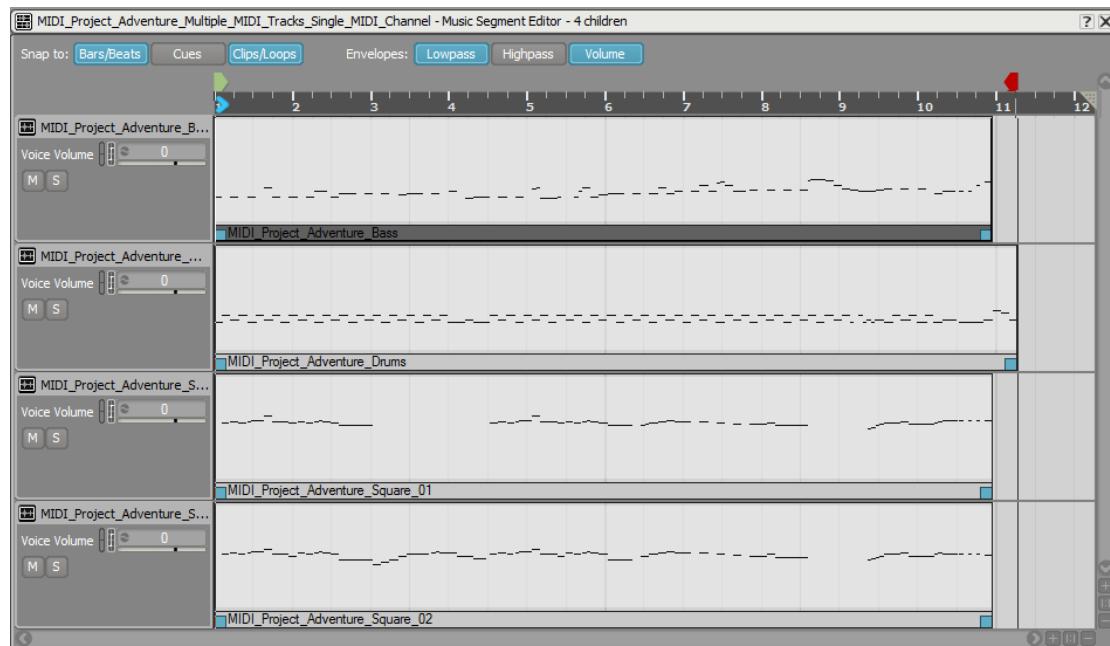
Each file has been output at a tempo of 140BPM, all channels (Omni), and full Velocity.

MIDI files can be imported as Music Segments or as tracks within an existing Music Segment by either using the Audio File Importer or by dragging and dropping them into the Interactive Music Hierarchy. After files have been added as tracks to the segment, they can be accessed through the Music Segment Editor like any other music track.



The Audio File Importer showing multiple MIDI files to be imported as Music Tracks

After MIDI files have been added as tracks to the segment, you can access them through the Music Segment Editor, where any combination of audio and MIDI tracks can be arranged and edited within the Music Segment.



Editing tracks in the Music Segment Editor

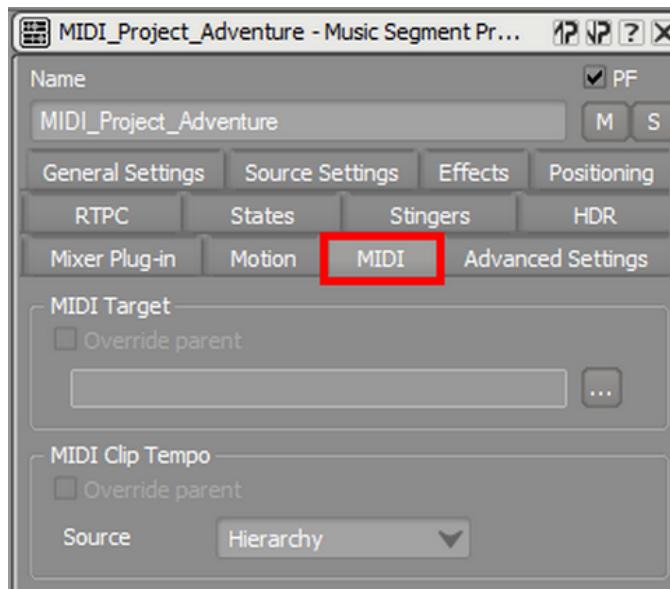
MIDI Tracks within a Music Segment exhibit the same behavior and functionality as an audio file. (as explained in Chapter 7. Adventures in Music - The Layered Approach - Creating the Ambient Music Segment)

Each track needs to be associated with a Sound Object that is used to play back information communicated from the MIDI file.

## Music Segment MIDI Properties

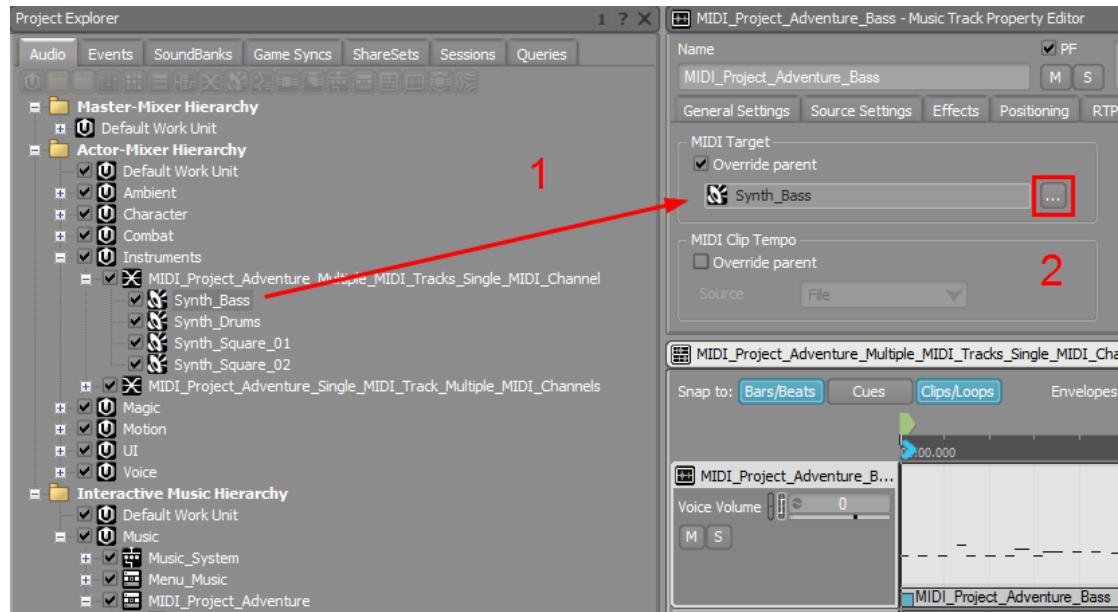
The MIDI Target and MIDI Clip Tempo can be defined for either the Music Segment (Parent) or Music Track (Child) in the MIDI tab in Music Segment Property Editor or Music Track Editor.

The MIDI Target is specified as the Sound Object through which all MIDI Events are routed. This property can target a specific Sound Object or a Blend Container including multiple Sound Objects, each with their own individual MIDI properties.



The MIDI tab in the Music Segment Property Editor

The MIDI Target can be assigned to a Music Segment or Music Track by navigating to a Sound Object through the Project Explorer - Browser or by dragging and dropping a Sound Object from the Actor-Mixer Hierarchy onto the MIDI Target field.



Assigning the MIDI Target using drag and drop  
(1) or using the Project Explorer - Browser (2)

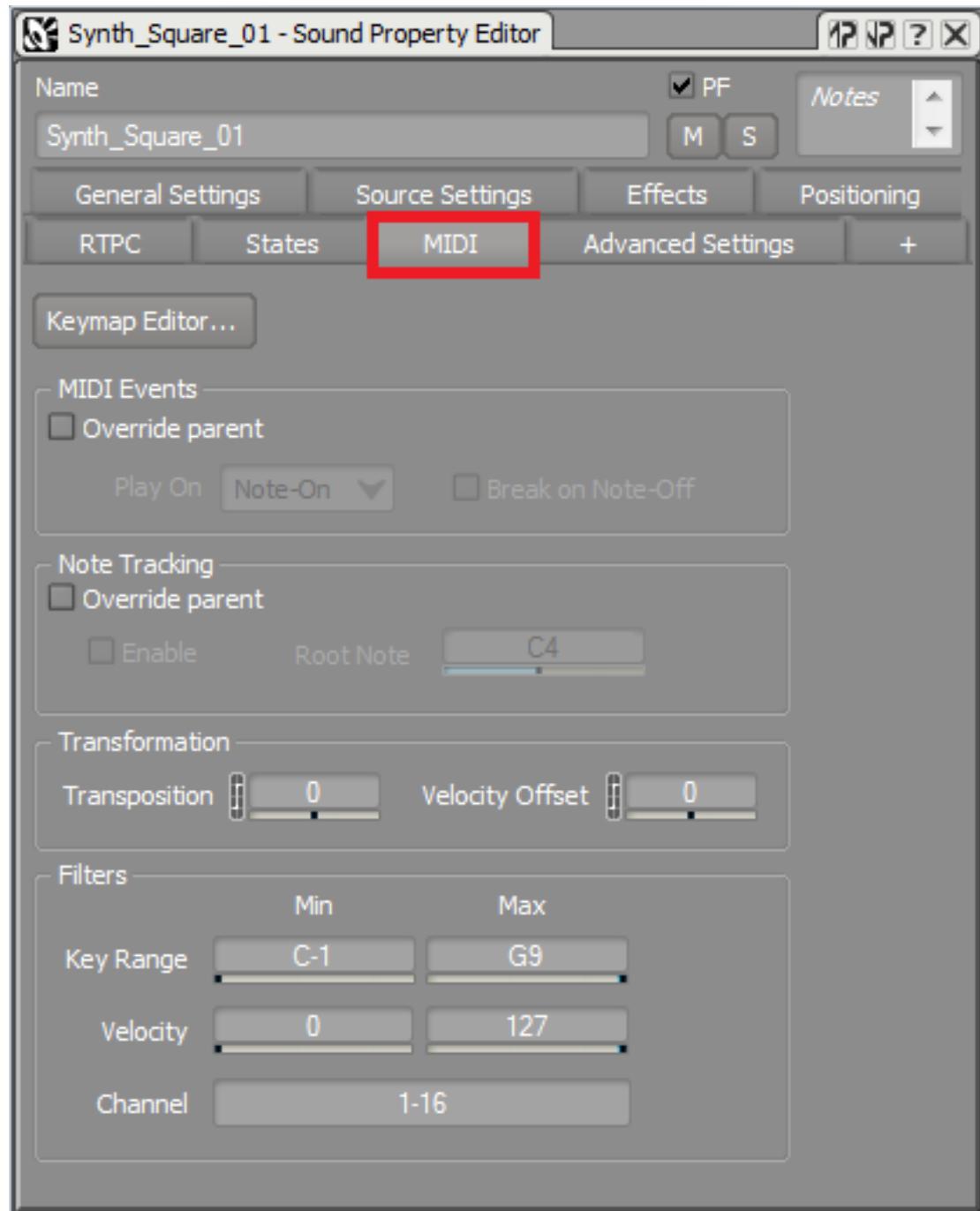
Wwise can use the Tempo map to control the tempo (exported as part of the MIDI file) or use the tempo specification set for the sound object in the Interactive Music Hierarchy.

The imported MIDI tracks use the MIDI Clip Tempo information from the (MIDI) file to control their tempo.

The composition can now be auditioned by selecting the Music Segment in the Interactive Music hierarchy and pressing 'Play' (or Spacebar).

### **Sound Object MIDI Properties**

Additional MIDI properties are located in the MIDI Tab for the parent Sound Object. These properties can be used to: control the behavior of incoming MIDI Events, transform the transposition or velocity offset of MIDI data before it is executed, or filter incoming MIDI data. These techniques can be particularly useful when building instruments comprised of multiple samples, synthesizers, or any combination to be targeted by a MIDI file.

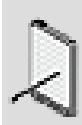


Sound Object properties in the MIDI Tab of the Sound Property Editor

Sound Object MIDI Properties include:

- **Keymap Editor** - Inspects the object in the MIDI Keymap Editor.
- **MIDI Events**

- **Override parent** - Determines whether the MIDI events controls are inherited from the parent or defined at the current level in the hierarchy. When this option is not selected, the MIDI Events controls are unavailable.  
If the object is a top-level object, this option is unavailable.
- **Play On** - Determines what type of MIDI note event will cause the object to play.
  - **Note-On** - Plays the object on Note-On.
  - **Note-Off** - Plays the object on Note-Off.
- **Break On Note-Off** - If Play On is set to Note-On, this property determines whether the playing object stops looping upon reception of a note-off. If so, the playback of looped sounds or continuous containers is stopped, while allowing the current object(s) to finish playing.
- **Note Tracking:**
  - **Override parent** - Determines whether the Note Tracking controls are inherited from the parent or defined at the current level in the hierarchy. When this option is not selected, the Note Tracking controls are unavailable. If the object is a top-level object, this option is unavailable.
  - **Enable** - If selected, the node's playback is pitch-shifted. The amount of pitch-shifting is determined by the note of the received MIDI event and the value of Root Note.
    - Default Value: false
  - **Root Note** - The root note of the node's sound. This value is used in conjunction with a received MIDI note to determine the pitch-shifting of the node's sound.
    - Default Value: C4
- **Transformation:**
  - **Transposition** - The offset applied to the MIDI event's note. The transposition is applied before the Key Range filters.
    - Default Value: 0
  - **Velocity Offset** - The offset applied to the MIDI event's note velocity. This applies to MIDI note events only. The offset is applied before the Velocity filters.
    - Default Value: 0
- **Filters**
  - **Key Range** - Filter applied to received MIDI note event's note. A received MIDI note event is ignored if its note is not within the Min-Max range.
    - Default Min: C-1 Default Max: G9



### Note

**Note:** The mapping of numerical MIDI notes to octaves is specified via the user preferences; refer to: [Wwise Help > Wwise Reference > Projects > User Preferences](#) for more details.

- **Velocity** - Filter applied to received MIDI note event's velocity. A received MIDI note event is ignored if its velocity is not within the Min-Max range.
  - Default Min: 0 Default Max: 127
- **Channel** - Filter applied to received MIDI note event's channel. A received MIDI note event is ignored if its channel is not in the filter.
  - Default: 1-16

## Section Summary

As another feature in the expanding Wwise Authoring Application, MIDI can be employed as the right technique for the right situation. Leveraging small sample-sets, synthesis, and the power and flexibility of MIDI can help balance out the use of resources across resident memory (size on disk or RAM) and processing (CPU). Depending on the constraints of your target platform and the desired aesthetic, MIDI can be a powerful ally towards maximizing resources and creating a dynamic audio experience.

## MIDI Summary

Like a hero fallen from grace, the return of MIDI comes with the knowledge of past battles and the remembrance of the way it used to be. Some of this knowledge can be immediately put to use navigating through the dawning of a new era for MIDI; experience with: creating sample-sets, MIDI channel assignment, velocity, and MIDI Events. Harnessing the inherent efficiency, dynamism, and flexibility of MIDI heralds the return of a well tested methodology. However, some parts of MIDI still hold negative connotations that need to be minimized; "bad" sounding, difficult workflow, and general prejudice work against the fundamental structure that MIDI makes available.

**Throughout this chapter we have:**

- Imported MIDI Files
- Added MIDI Files to a Music Track
- Made Modifications to the MIDI Target
- Adjusted the MIDI Clip Tempo
- Looked at the synthesizer functionality of the Wwise Synth One
- Talked about preparing music content for integration.

**Stepped through the process of:**

- Importing MIDI files.
- Setting up the Wwise Synth One plug-in.
- Creating a multitrack composition comprised of:
  - Four MIDI tracks representing different instruments.
  - Four Wwise Synth One synths representing different instruments.

**We've also touched on:**

- Advanced functionality of the MIDI format.

**Throughout this section we have created the following objects:**

- A Music Segment containing a MIDI file with individual channels, each targeting instances of Wwise Synth One which represent the different instruments.
- A Music Segment containing four individual MIDI files, each targeting instances of Wwise Synth One which represent the different instruments.

---

## Chapter 9. Mastering the Mix

Overview .....	194
Routing with Audio Busses .....	195
Routing with Auxiliary Busses .....	197
Using Auxiliary Sends .....	199
User-Defined Auxiliary Sends .....	200
Game-Defined Auxiliary Sends .....	202
States and Mix Snapshots .....	204
Auto-Ducking vs. Side-Chaining .....	206
Auto-Ducking .....	206
Side-Chaining .....	207
Mixing with RTPC .....	207
Using Effects in the Master-Mixer .....	209
Visualizing the Mixing Desk .....	210
Mixing Techniques for Attenuations .....	212
Mix Summary .....	213
Mixing Additional Resources: .....	214

## Overview

Heading into the final mix is not unlike entering into battle with a fire breathing dragon. By the time you arrive at the scene of the last stand against the bloodthirsty creature, you've hopefully equipped yourself with every skill and weapon in your arsenal in order to bely the beast into submission. As the signal flow winds its way through the labyrinth of routing possibilities, you know you have the flexibility to reshape the audio path and the power to commandeer the ultimate mix.

From a single instance of a sound played back by an audio engine, to the potentially massive pile-up of sounds happening in a single frame, the race is on to make sure that the right sounds are heard throughout the game. Whereas the Mix title has been appropriated from the music and film world where it represents a time locked and non-interactive representation of sound, its use in game audio tends to be a bit more nebulous. Using a combination of dynamic mix-related methodologies and available techniques to balance the final sound output has become one of this generation's greatest challenges.

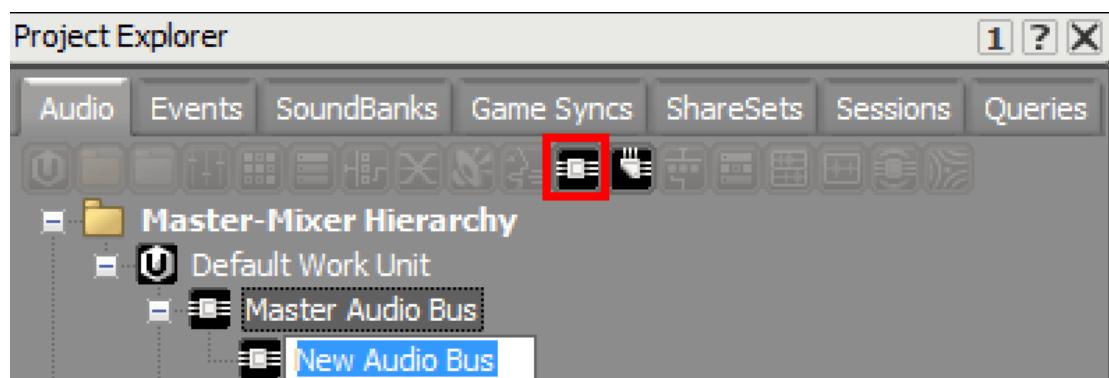
This chapter will take you through the process of:

- Routing with Audio Busses
- Routing with Auxiliary Busses
- Using Auxiliary Sends
- States and Mix Snapshots
- Auto-ducking and side-chaining
- Mixing with RTPCs
- Using effects in the Master-Mixer
- Visualizing the mixing desk
- Mixing techniques for attenuations

## Routing with Audio Busses

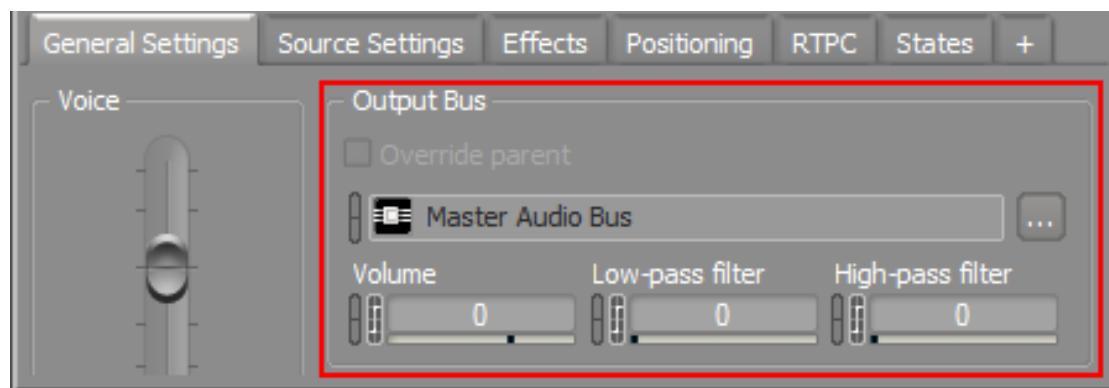
Anyone who has worked with hardware or software mixers in the past can appreciate the Master-Mixer hierarchy in Wwise with its customizable interface for routing audio signals from sound objects throughout the project. You can establish a comprehensive and flexible representation of the signal flow by adding and organizing audio and auxiliary busses.

Fundamentally, audio busses are a way to balance volumes, add effects, apply RTPC, and state-based changes. Audio busses can be added to the Master-Mixer hierarchy by selecting an existing audio bus and then clicking the audio bus icon in the Project Explorer toolbar. A new audio bus is created as a child of the selected audio bus.



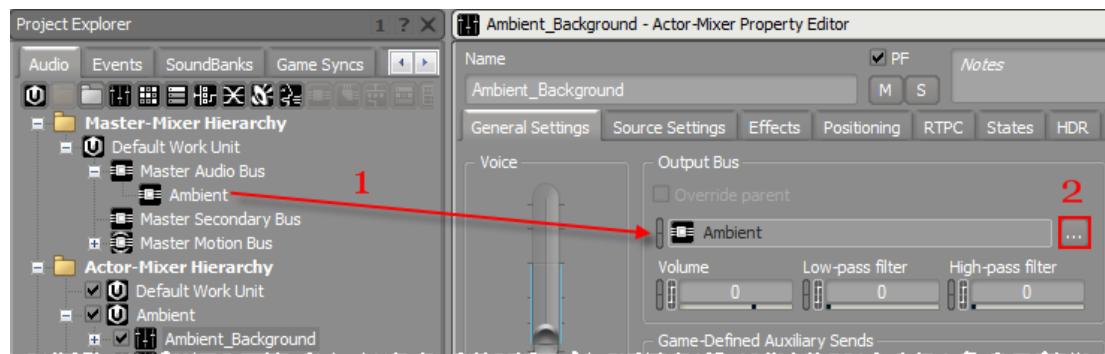
Creating an audio bus from the Project Explorer toolbar

After an audio bus has been created, it can be assigned to any sound object via the General Settings tab in the Property Editor. Audio busses can be assigned to a sound object by either navigating to the Bus through the Project Explorer - Browser, or by simply dragging and dropping an audio bus from the Master-Mixer onto the Audio Output Bus field.



Audio output bus and settings in the Property Editor

Audio busses can be configured and reconfigured to suit the developing needs of any project. Dragging and dropping audio busses within the Master-Mixer hierarchy also maintains any assignments for routing that have been established for a sound object.



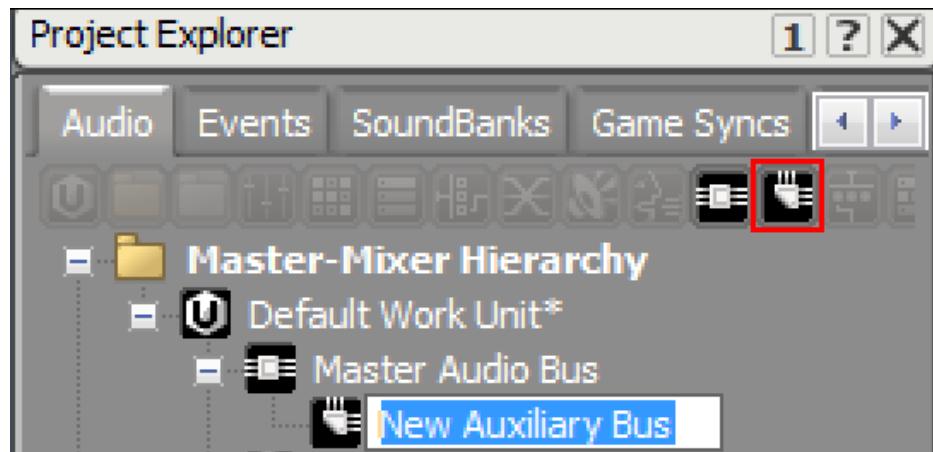
Assigning the audio output bus of a sound object using drag and drop (1) or using the Project Explorer - Browser (2)

In the previous image, once the “Ambient” audio output bus has been assigned, any sound output from the “Ambient\_Background” Actor-Mixer is routed through the “Ambient” audio bus. The output bus volume and audio output bus low pass filter settings control the amount of volume or low pass filter from the sound object that is passed to the audio output bus. Furthermore, the properties set for the audio bus in the Master-Mixer hierarchy govern the final output.

## Routing with Auxiliary Busses

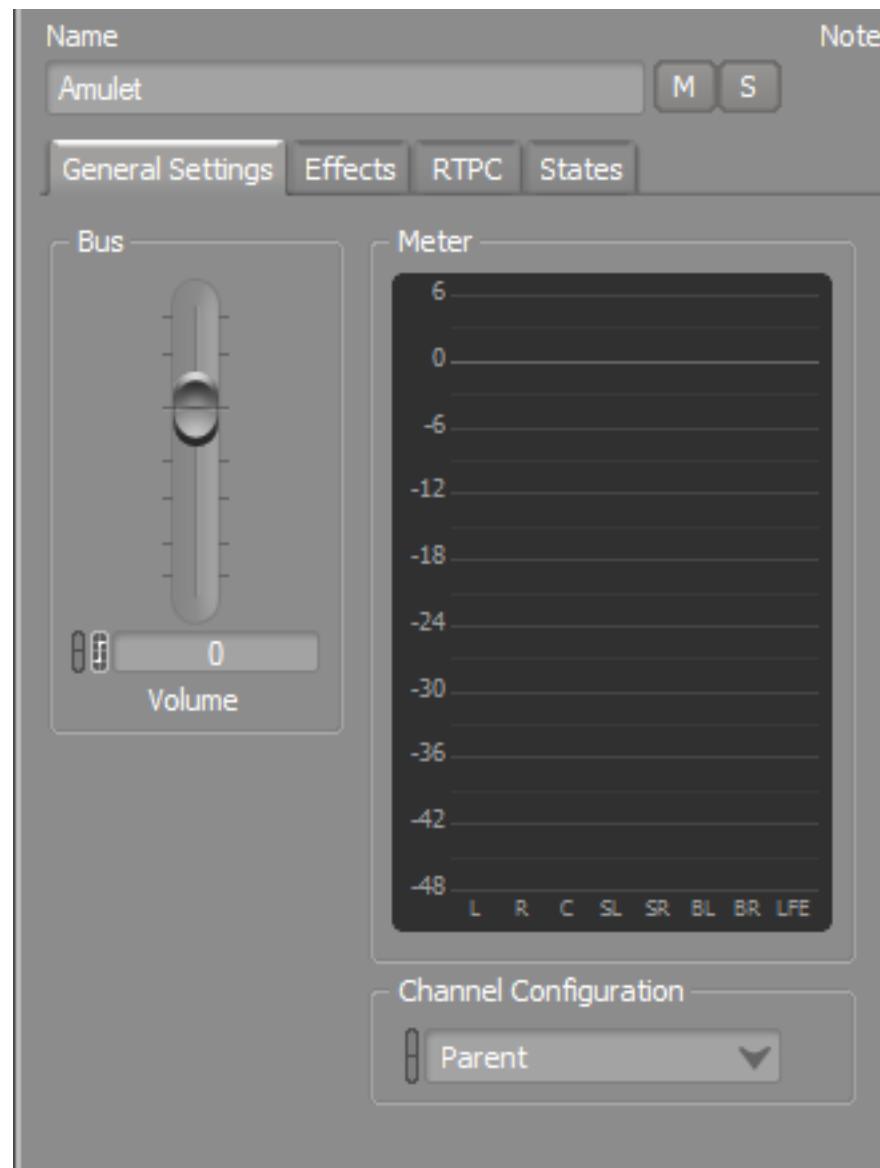
An auxiliary bus is organized within the Master-Mixer hierarchy in the same way as a standard audio bus. Each auxiliary bus can manage up to four effects which can be enabled or disabled programmatically or with game parameters. Auxiliary busses can also be positioned within the Master-Mixer hierarchy as children of other audio or auxiliary busses, allowing for more than four effects in a series.

The process begins by adding an auxiliary bus to the Master-Mixer hierarchy by selecting an existing audio or auxiliary bus and then clicking the auxiliary bus icon in the Project Explorer toolbar. A new auxiliary bus is created as a child of the selected audio or auxiliary bus.



Creating an auxiliary bus from the Project Explorer toolbar

Each auxiliary bus includes metering and allows you to set volume, add effects, RTPC, as well as apply changes based on state.



General Settings tab of the auxiliary bus in the Property Editor

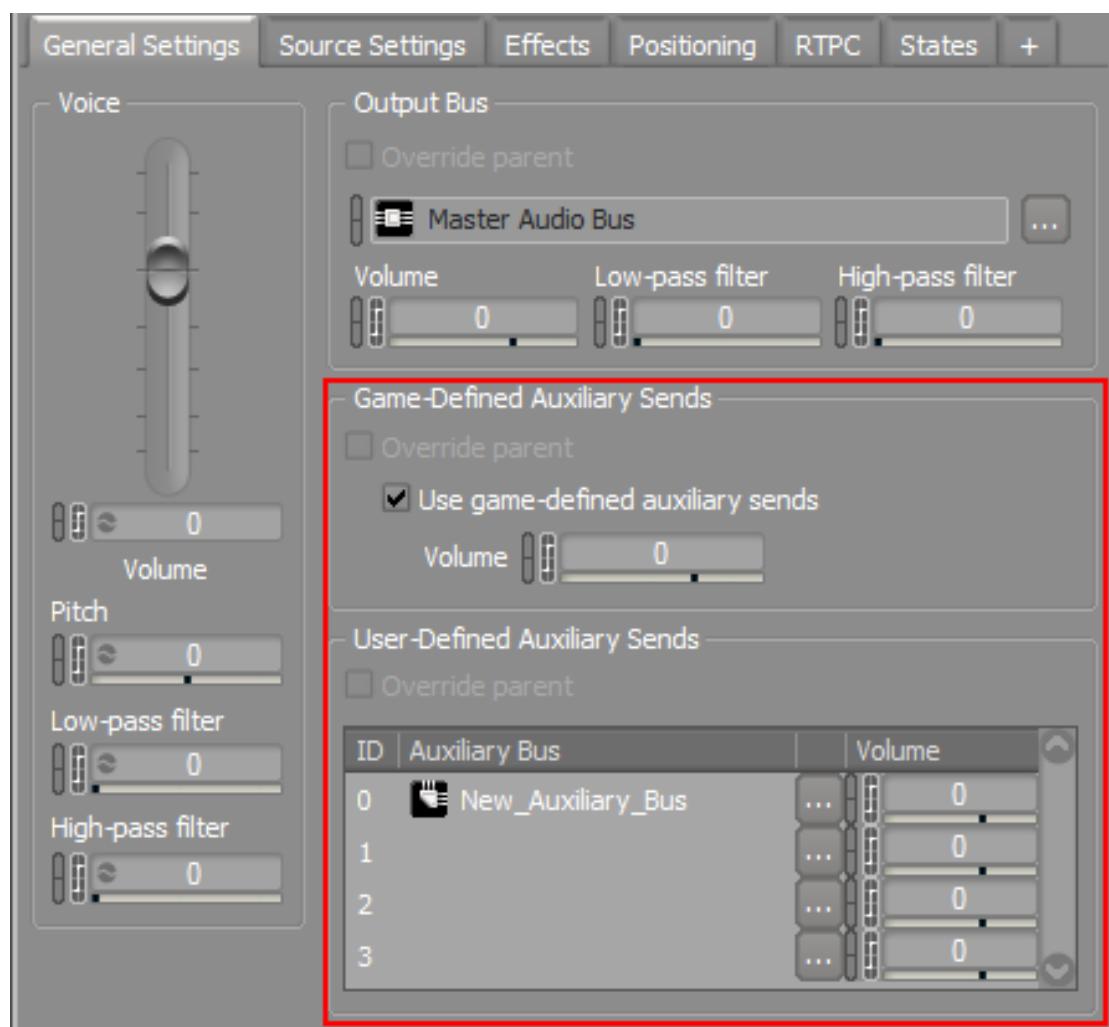
## Using Auxiliary Sends

Traditionally, auxiliary sends are used to route an audio signal to a set of effects that are then applied to any audio signal as a whole. In other words, an auxiliary send is a convenient way to route the output of a sound object in the Actor-Mixer Hierarchy to an auxiliary bus within the Master-Mixer Hierarchy.

There are two types of auxiliary sends in Wwise:

- User-defined auxiliary sends
- Game-defined auxiliary sends

A sound object can use one or both types of auxiliary send from the General Settings tab in the Property Editor. The different types of auxiliary send behave in exactly the same way. Sends from a sound object can be mixed and matched between the two types within the authoring application.



Defining auxiliary sends in the General Settings tab of the Property Editor

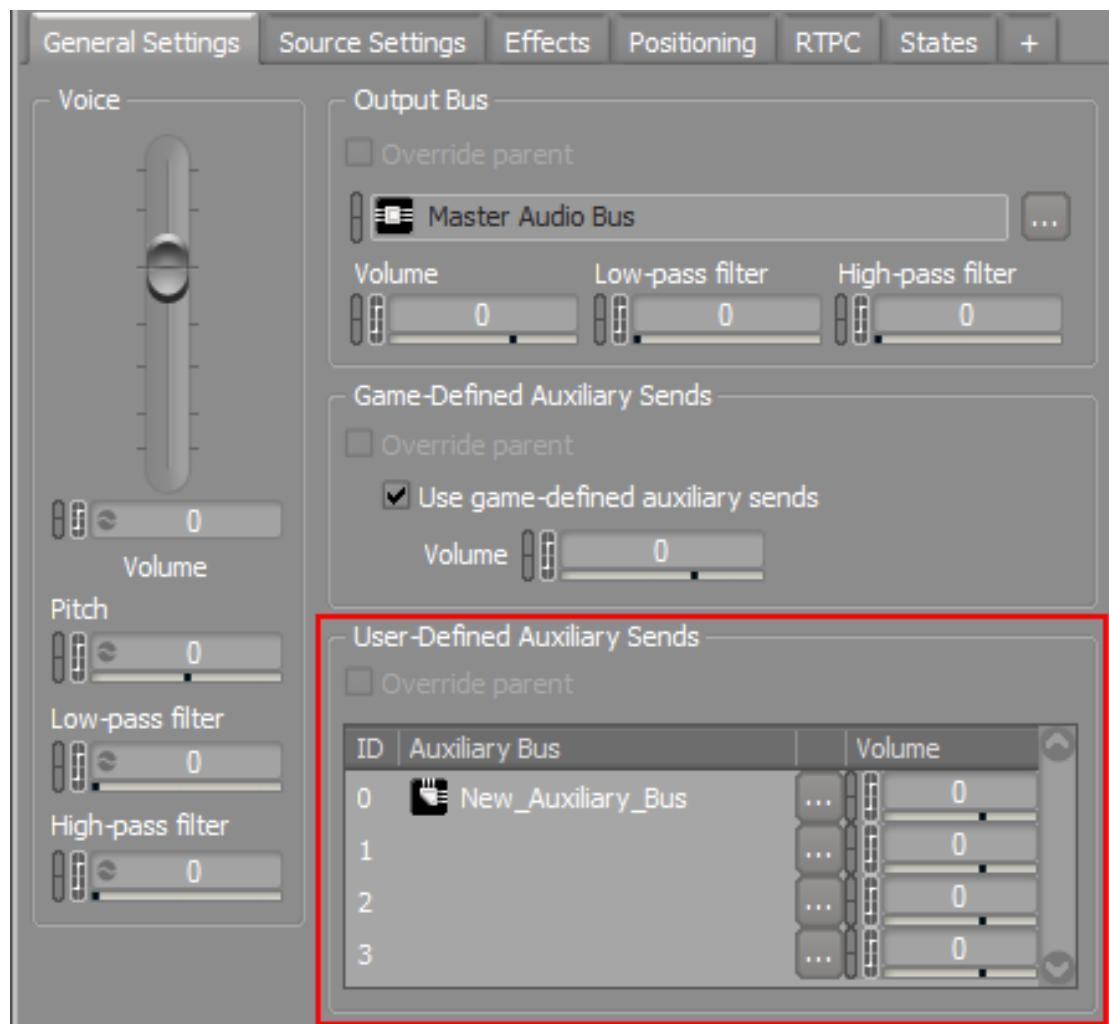
The auxiliary send is controlled in two different ways:

- User-defined: Per sound object in Wwise.
- Game-defined: Per game object using the SDK API.

The send volume related to an auxiliary bus within the Master-Mixer hierarchy is independent of the audio output bus volume and routing. This allows for the creative application of effects and other interactive mixing techniques.

## User-Defined Auxiliary Sends

Up to four user-defined auxiliary busses can be assigned to any sound object in the Actor-Mixer Hierarchy. The send volume is the level or amplitude of the audio signal that is sent to an auxiliary bus that can also be parameterized using RTPC. User-defined auxiliary busses are assigned in the General Settings tab of the Property Editor.

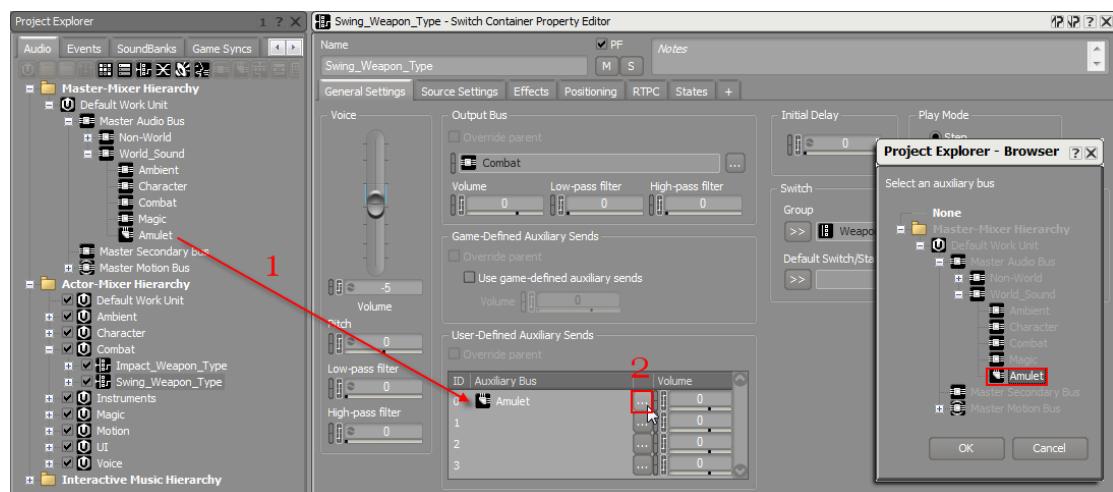


User-defined auxiliary sends in the Property Editor

In this example we will create a user-defined auxiliary send that is used to apply a magical effect to certain sound types in the project when our hero is in danger. The auxiliary bus will contain a chain of effects that utilize the "Danger" game parameter to adjust various properties. Modifying the sound of weapon swings and magic will be used as an indicator of danger and will enhance the dynamic aspects of battle.

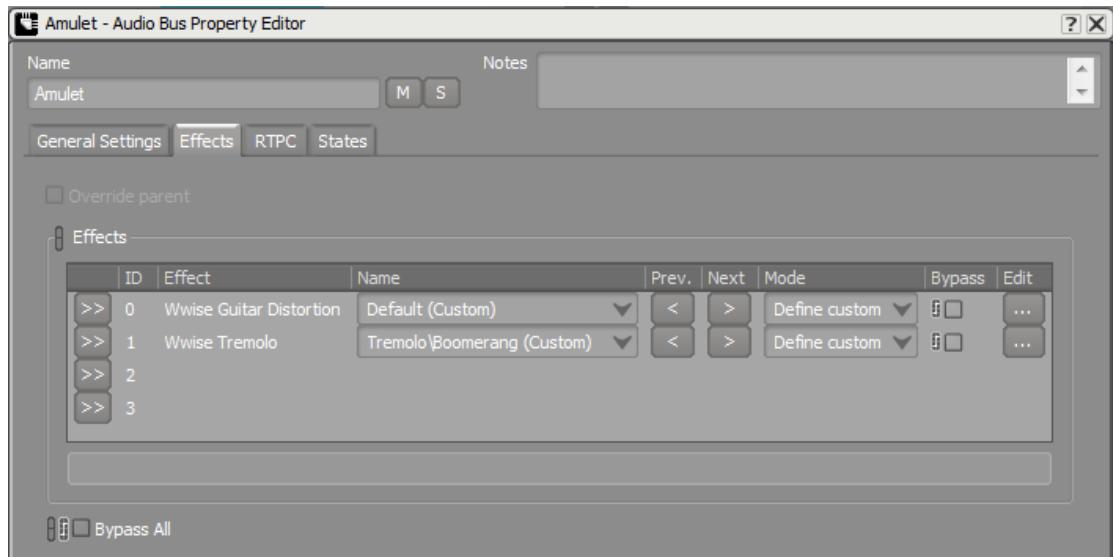
The source of our hero's power stems from an ancient amulet, and so we will name the newly created auxiliary bus "Amulet". In this example we'll be routing the output of the "Swing\_Weapon\_Type" switch container and of the "magic\_blast\_fire\_distance\_blend" blend container to the Amulet auxiliary bus.

Any existing auxiliary bus can be assigned to sound objects by either navigating to the bus through the Project Explorer - Browser, or by simply dragging and dropping the auxiliary bus from the Master-Mixer hierarchy onto the "Auxiliary Bus" field in the User-Defined Auxiliary Sends section of the Property Editor.



Assigning the auxiliary bus of a sound object using drag and drop (1) or using the Project Explorer - Browser (2)

Once a sound object routes a signal to an auxiliary bus, the send volume can be adjusted. Adding and adjusting effects can now be applied to the "Amulet" auxiliary bus to communicate a sense of magical change when our hero is in danger. In this example, a custom "Wwise Guitar Distortion" has been added as the first effect and has been set to increase the wet/dry mix based on the "Danger" game parameter. Additionally a "Wwise Tremolo" has been added as the second effect, with the LFO depth controlled by the "Danger" game parameter.



A chain of effects on the “Amulet” auxiliary bus



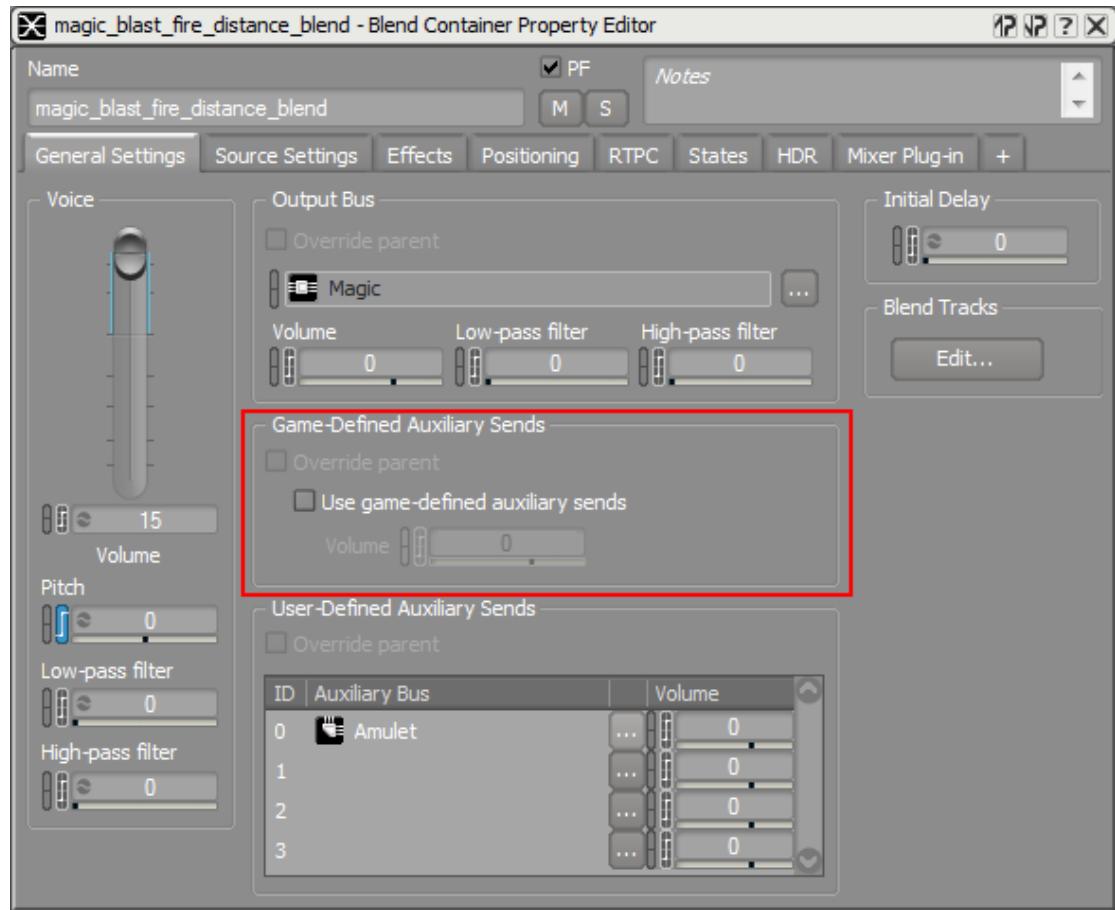
### Designer Note

The user-defined auxiliary bus routing cannot be changed at runtime, but the game can use the `SetBusEffect()` and `SetActorMixerEffect()` functions to set different effects ShareSets to the auxiliary bus. This method allows for the application of environment acoustics in your game, but with a finer granularity (per audio object and additionally per game object).

By controlling auxiliary sends you can apply creative and runtime-conscious special effects throughout the project.

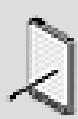
### Game-Defined Auxiliary Sends

In addition to setting user-defined auxiliary busses, up to four game-defined auxiliary busses can be set by the game engine or managed programmatically outside of the authoring application for each game object. Auxiliary busses defined by the game can be used for reverbs, game-state dependant effects, interactive mixing, or any other game-defined usage. Additionally, the sound designer has control from the user interface to enable and offset the game-defined auxiliary send volumes on a per sound basis from the General Settings tab in the Property Editor.



Enabling game-defined auxiliary sends in the Property Editor

Game objects can be routed to a combination of game-defined and user-defined auxiliary busses for a total of eight independent auxiliary targets (four game-defined and four user-defined).

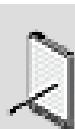


### Designer Note

The final send volume is the combination of the game-defined auxiliary send volume from the UI and the SetGameObjectAuxSendValues() SDK function set by a programmer.

## States and Mix Snapshots

Another powerful practice for manipulating an interactive mix is by using states, or in this context, mix snapshots. States can be directly related to, and commonly referred to as, game states such as: combat, stealth, idle. They can also be used to define spaces such as a forest, hallway, or dungeon and, furthermore, can be abstracted to define any circumstance under which you may want to change the sound of the game. States are usually defined in the game engine and triggered within Wwise, where you can combine multiple states simultaneously.

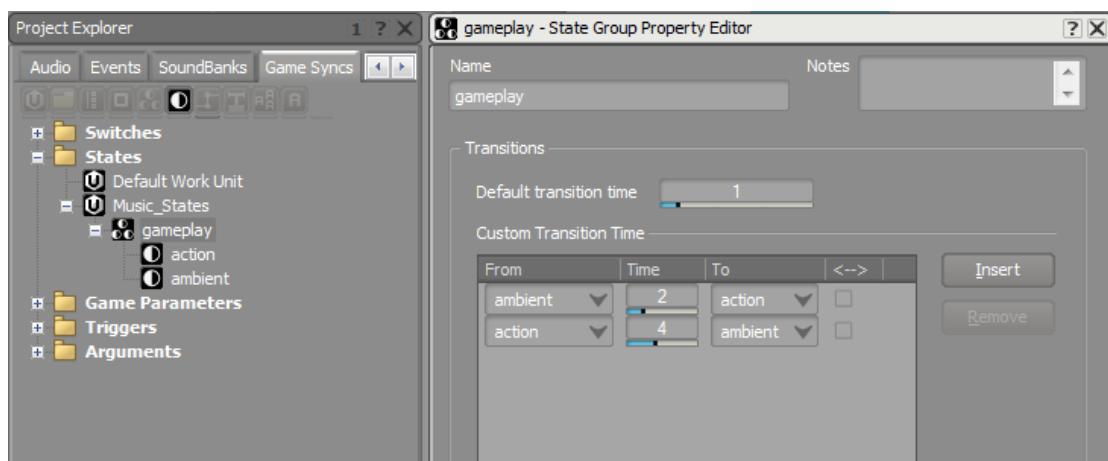


### Designer Note

When an object is registered to multiple states, a single property can be affected by multiple value changes. In this scenario, each change of value is added up together. For example, when two states in two different state groups have a volume change of -6 dB, and both become active simultaneously, the resulting volume will be -12 dB.

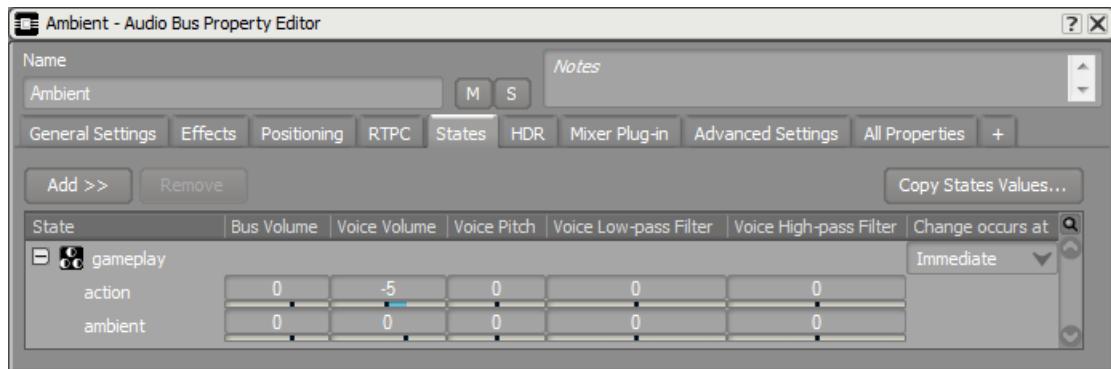
Due to the non-linear progression and randomness inherent in most games, it is often desirable to plan for a mix that can respond to events in the game dynamically as opposed to a static mix that does not change based on circumstances within the game. By using state changes to modify the properties of different mix busses, you can create a system for dynamically mixing the game. In a sense, it is like programming an artificial intelligence that is able to make changes to the mix in accordance with the rules specified by the sound designer.

States are defined in the game syncs tab of the Project Explorer and include settings for defining a default transition time or custom transitions based on changes between specific states.



Defining states and setting default and custom transition times

Once states have been established, properties that can be affected by state changes are available for any audio object or audio bus by adding a state in the States tab.



### State based volume reduction of the ambient bus when in the action state

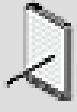
For more information on states:

[Wwise Help > Interacting with the Game > Working with States > Working with States](#)

[Wwise Help > Interacting with the Game > Working with States > Assigning States to Objects and Busses > Example: Creating a Temporary Loss of Hearing Effect](#)

## Auto-Ducking vs. Side-Chaining

While mix states give explicitly defined control over properties, you can also automate volume reduction based on an incoming signal from another audio bus. This process is often referred to as ducking.

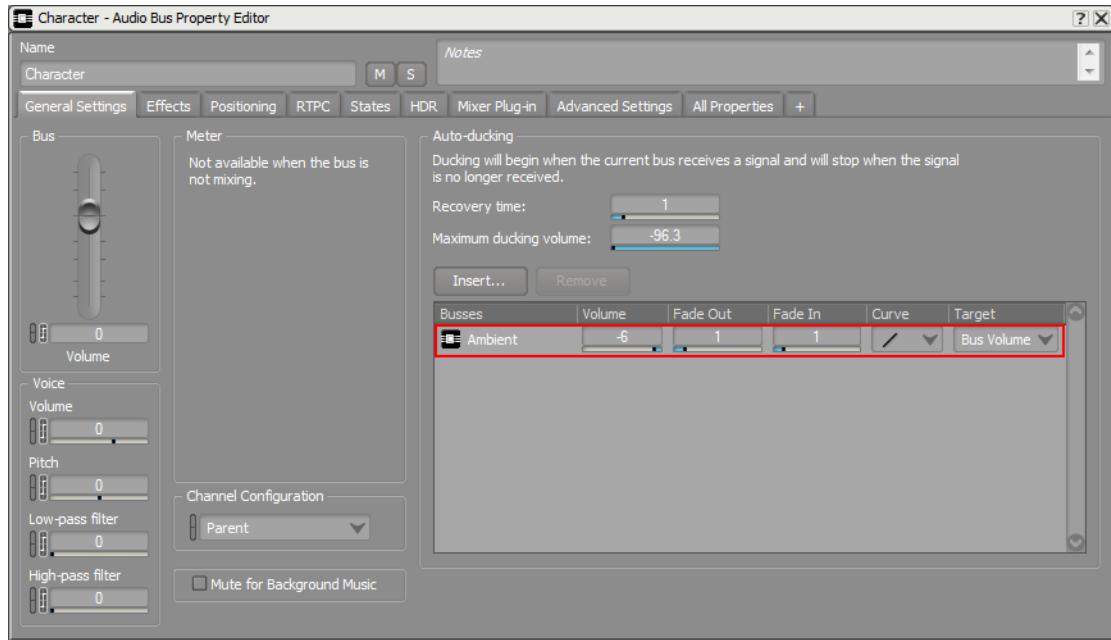


### Designer Note

Ducking allows you to automatically lower the volume level of all objects passing through one bus in order for another simultaneous bus to have more prominence.

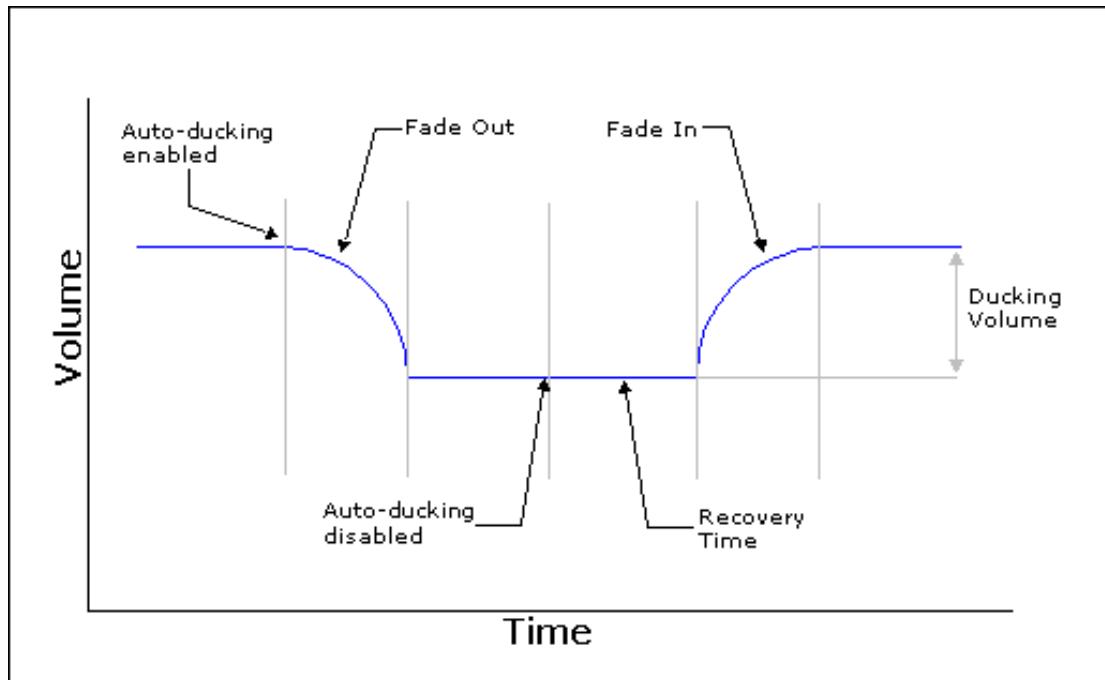
### Auto-Ducking

Wwise allows for Auto-duking through the General Setting tab on any audio bus. Simply insert any other bus into the Auto-duking window and define the volume attenuation and fade in/out properties.



**Character audio bus automatically ducking Ambient audio bus by -6 dB over a period of one second using a linear curve**

Auto-duking begins when the selected bus receives any signal and applies the set attenuation to any inserted busses for the duration of a signal, including silence.



## Side-Chaining

Another approach to ducking is through the use of Side-Chaining. Side-chaining consists of monitoring the level of an audio signal and using it to manipulate another audio signal. A concrete example of side-chaining occurs in radio broadcasting where a DJ's voice automatically ducks the music volume.

When using Side-Chaining as part of mixing, you are essentially taking volume information from input signal with the Wwise Meter effect and using it to affect other parts of the mix.



### Designer Note

More information about side-chaining:

[http://www.audiokinetic.com/download/documents/Wwise\\_SideChaining\\_Tutorial.pdf](http://www.audiokinetic.com/download/documents/Wwise_SideChaining_Tutorial.pdf)

## Mixing with RTPC

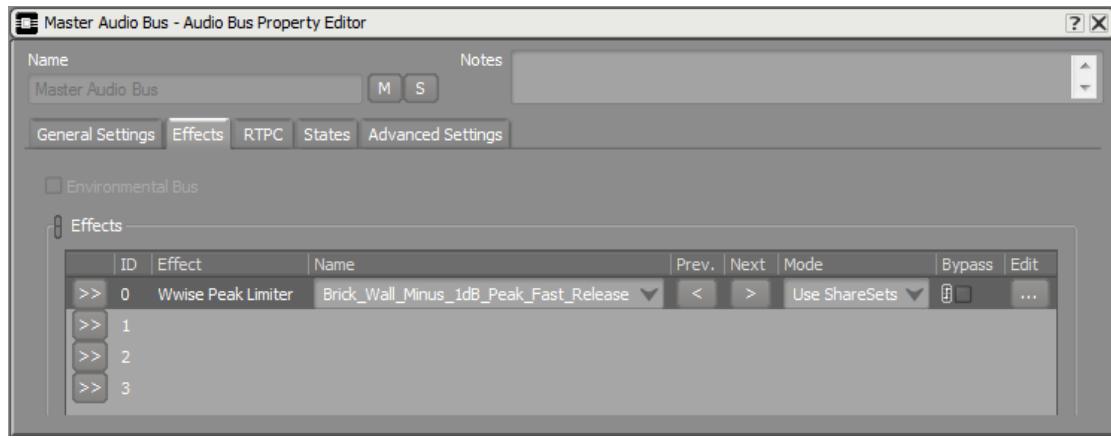
Real Time Parameter Control (RTPC) can also be leveraged to provide dynamic mixing opportunities using parameters from the game to drive volume changes throughout the project. A common parameter driven mix technique is to apply a low pass filter to the in-game environmental mix as the player's health decreases. This can easily be accomplished with a health game parameter used as an RTPC on one or many audio busses.

Remember that Danger Game Parameter we created in order to increase the intensity for the layered music system? The same game parameter could be used to reduce the volume of sounds or audio busses that demand less attention during combat. Things like ambience or character movement sounds could have an RTPC curve applied to reduce their volume in the mix and make way for the sounds of combat.

Having multiple ways to interact with the different aspects of the mix means that when difficulties arise there will be several ways to approach a solution. Mixing with RTPC is another effective tool in the toolbox of interactive and dynamic mixing.

## Using Effects in the Master-Mixer

The use of common DSP is well defined by decades of linear music and sound production. EQ, Delay, Distortion, Compression, among others have found common use in most Digital Audio Workstations for decades, but have only recently (within the last 10 years) become efficient enough to use at runtime in order to affect the mix. It's not just the use of fixed settings that makes this area of growth so exciting, but the dynamic and interactive nature of manipulating DSP using parameters from the game to produce special effects that would be impossible to achieve using other techniques.



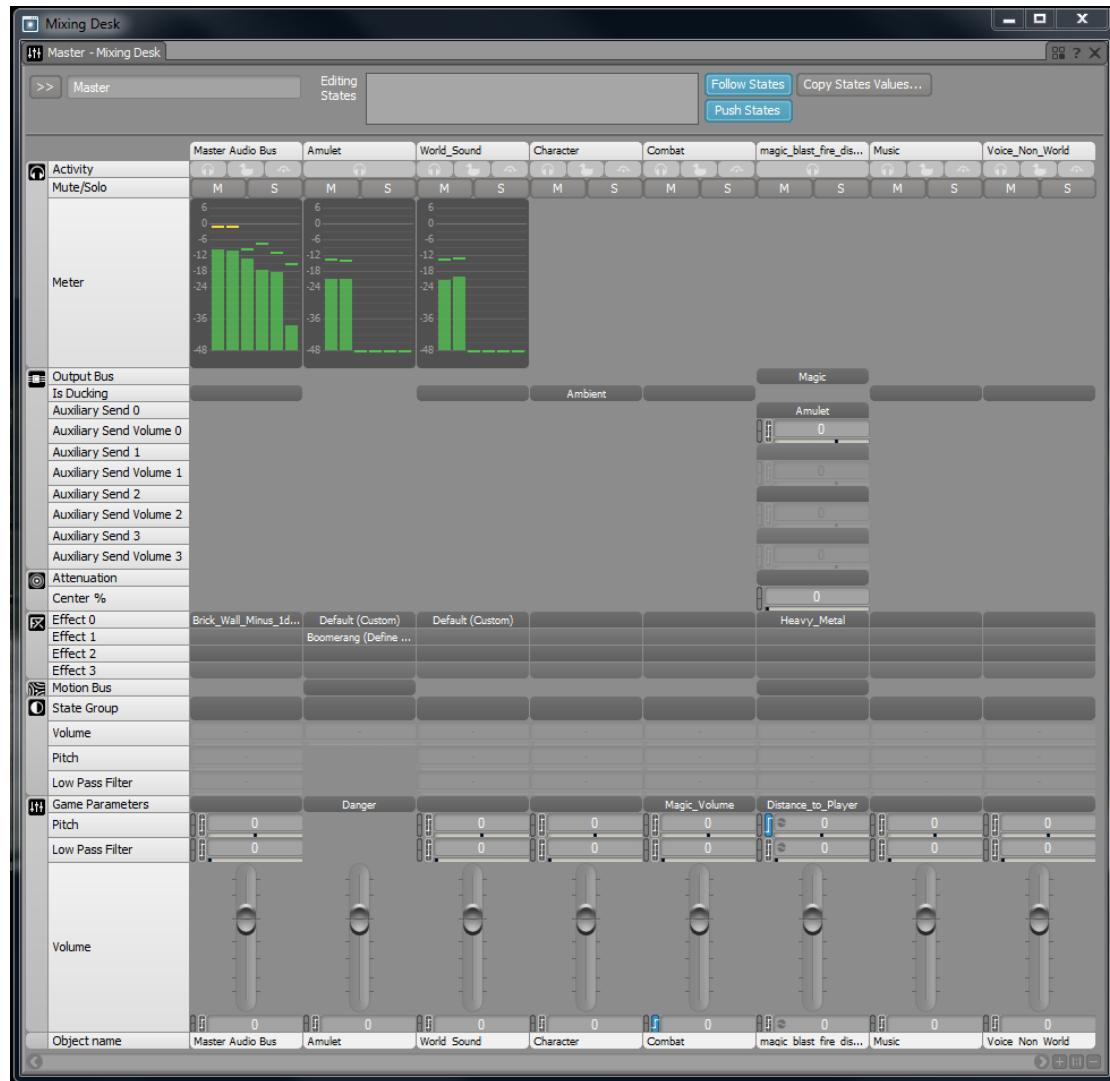
Applying a limiter to the Master Audio Bus

Effects used at runtime will always use up CPU power, but being aware of the different costs can help you use them more efficiently. If an effect (or effects) are applied to only a few mono instances of a sound object, it is more efficient to apply these at the Actor-Mixer level. If many simultaneous sounds need to be processed by a given effect, it would be more efficient to apply the effect(s) on an audio bus that will be mixed before applying the effect (in stereo or 5.1 depending on the speaker configuration).

## Visualizing the Mixing Desk

The Mixing Desk is a flexible and powerful interface that groups a variety of properties into one view, allowing you to fine-tune the audio mix in real time. You can add any sound object, audio or auxiliary bus to the Mixing Desk, and then define routing, apply effect and attenuation ShareSets, edit state properties, and modify the properties of individual objects and busses.

After your audio and motion are integrated into the game, you can connect Wwise to the game and profile information in real time as the game is being played. Adjustments and mixing decisions can be made in real time in game directly from the authoring application. Activity of each object within the Mixing Desk can also be viewed, including when a voice is playing, if a bus is being ducked, and whether effects are being bypassed either manually or programmatically. Each audio object can also be muted or soloed, allowing you to adjust the individual objects within the audio mix.



Visualizing the mix using the Mixing Desk view

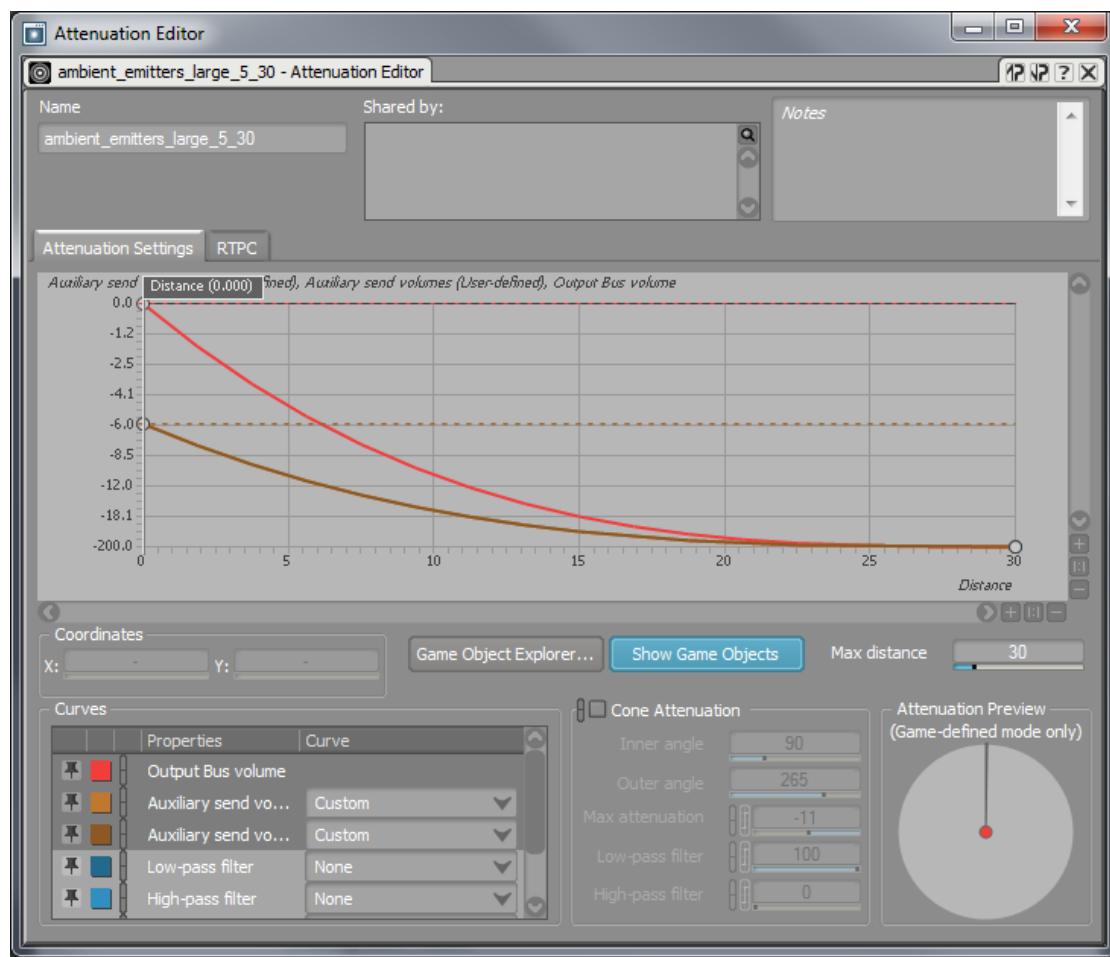
A context menu is also available for each property setting in the mixer strip. These menus contain a set of commands specifically related to the selected property. For example, when you right-click one of the Effect slots (0-3), you can edit the properties of the inserted effect, set a new effect, bypass the effect, and so on. To access the contextual menus, simply right-click the individual property settings in the mixer strip.

The Mixing Desk setup is saved within a Mixing Session ShareSet. This allows you to create different mixing sessions for different components of the game. It also means that you can set up a mixing session and then continue to fine-tune the audio mix at a later time.

## Mixing Techniques for Attenuations

The use of attenuations to balance the resulting mix is one of the best contributors to a natural sounding game. Simple things like extending the maximum distance while changing the curve depth or shape can have a tremendous effect, allowing the sounds to be heard from greater distance at a lesser volume. Alternatively, reducing the maximum distance can help to clean up an overactive mix and gain some clarity between elements.

Additionally, the auxiliary send volumes of an attenuation are directly related to the amount of signal sent to both game and user-defined busses. Ensuring the correct balance between the output bus volume and the game/user-defined send volumes for a sound or group of sounds can help reveal detail and definition in the mix.



Adding a custom auxiliary send volumes curve to an attenuation

## Mix Summary

A sound designer equipped with the tools to enable a comprehensive dynamic mix can rest easier when the time comes to complete the game's final mix. With tools modeled after those commonly used in the linear world, and further imbued with the ability to react interactively, a doorway to the next level in dynamic mixing is opened.

**Throughout this chapter we have:**

- Discussed signal-flow and routing using audio busses.
- Established the use of user-defined auxiliary sends.
- Established the use of game-defined auxiliary sends.
- Discussed the use of states as mix snapshots.
- Compared auto-ducking with side-chaining.
- Mixed with RTPC.
- Discussed the benefits of using effects in the Master-Mixer.
- Shown ways to visualize the mix using the Mixing Desk.
- Illustrated techniques for mixing with attenuations.

**Stepped through the process of:**

- Creating audio busses.
- Creating auxiliary busses.
- Setting up states.
- Applying auto-ducking to audio busses in the Master-Mixer.

**Also touched on:**

- The relationship between wet and dry reverb volumes in the attenuation editor.

**Throughout this section we have created the following objects:**

- An “Amulet” auxiliary bus used to affect magic and weapon swing sounds based on the “Danger” game parameter.

## Mixing Additional Resources:

[Video Tutorial - Mixing Desk](#)

[Video Tutorial - Dynamic mixing using States](#)

[Video Tutorial - Wwise Side-Chaining](#)

[Wwise Help > Working with Wwise > Getting to Know the Property Editor > Overview > Advanced Settings > Using Advanced Settings and Dynamic Mixing](#)

[Wwise Side-Chaining Tutorial PDF](#)

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Managing the Priority of Sounds and Motion > Limiting Object Playback Instances](#)

[Wwise Help > Interacting with the Game > Working with States > Working with States](#)

[Wwise Help > Interacting with the Game > Working with States > Assigning States to Objects and Busses > Example: Creating a Temporary Loss of Hearing Effect](#)

---

# **Chapter 10. HDR Audio Wizardry**

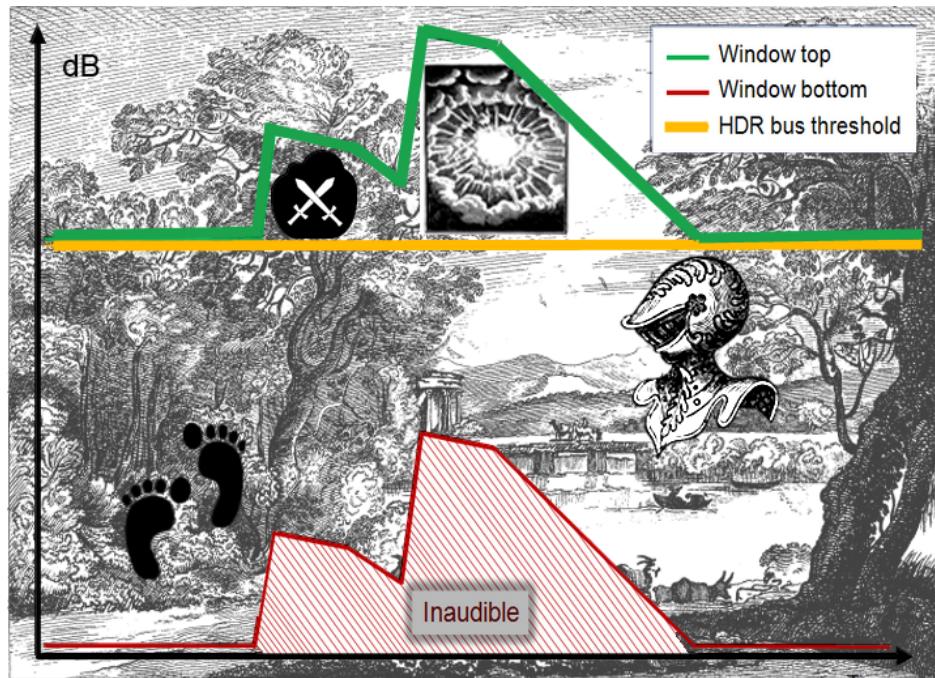
Overview .....	216
Implementing HDR Audio in Wwise .....	218
Setting up an HDR Audio Mix .....	219
Setting up the HDR Audio Dynamic Range Window .....	219
Enabling HDR Audio in the Master-Mixer Hierarchy .....	221
Setting up HDR Audio Dynamics Properties .....	221
Using HDR Audio in the Actor-Mixer Hierarchy .....	224
Enabling Envelope Tracking .....	225
Editing a Waveform Envelope .....	227
Enabling Source Normalization .....	228
Using Make-up Gain .....	229
Using the Voice Monitor to Understand HDR Audio .....	231
Opening the Voice Monitor View .....	231
Auditioning Sounds in the Soundcaster .....	232
Capturing Data from Wwise .....	233
HDR Audio Summary .....	236

## Overview

High atop the misty mountain peak, the wizened Wwizard stares out across the battle raging across the valley. The maelstrom churns below as the loudest sounds of cold steel-on-steel impact, blasts of magical power, and the release of catapults drift up from the ensuing assault. As each one of these moments takes precedence, the soundscape clears to reveal and accentuate the madness unfolding. Whether we are conscious of it or not, a finely tuned mechanism is at work assigning relevance to the loudness of what we hear and shaping the way we inherently experience the world through sound. So how can the idea of prioritizing loud sounds fit with today's gaming?

High Dynamic Range (HDR) is a new mixing paradigm available within Wwise to augment the already existing suite of dynamic mixing tools. HDR allows for authoring a system that responds to the dynamic loudness of sounds by giving priority to sounds that are authored the loudest. With this relativistic mixing system, the clarity of intention from the authoring perspective is ensured to make appropriate space in the mix for “important” sounds to be heard.

Think back to the scene that we just described. As the sounds of battle ensue, you instinctively focus on the loudest sounds (combat) and quieter sounds (footsteps) are ignored. In an HDR system, these loud sounds represent the top of a user-defined “window” that moves dynamically to ensure that the loudest sounds are in focus. As this window moves upward, sounds below the window bottom are removed from the mix. When one side emerges triumphant in the battle below and eyes turn toward the misty mountain top, the sound of horse hooves pounding the earth swell with renewed amplitude, becoming the loudest, and most threatening focus. As the HDR window threshold returns to its rest position, the original amplitude of the horses is represented.



An illustration of the HDR window shifting in response to increasing loudness of sounds over time.

This chapter will take you through the process of:

- Implementing HDR audio in Wwise
- Setting up an HDR audio mix
- Setting up the HDR audio dynamic range window
- Enabling HDR audio in the Master-Mixer Hierarchy
- Setting up HDR audio dynamic properties
- Using HDR audio in the Actor-Mixer Hierarchy
- Enabling envelope tracking
- Editing a waveform envelope
- Enabling source normalization
- Using Make-up Gain.
- Using the Voice Monitor to understand HDR audio
- Opening the Voice Monitor View
- Auditioning Sounds in the Soundcaster
- Capturing Data from Wwise

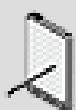
## Implementing HDR Audio in Wwise

HDR in Wwise can be enabled for any parent Audio Bus in the Master-Mixer Hierarchy. Once enabled, the Audio Bus acts as a converter between Sound Object volumes at the input of the HDR bus and full (device) scale at the output of the same bus. All sounds that are routed to it are handled relative to each other within the HDR bus, with the output of quieter sounds constantly modified according to the properties of the system.

The controls of an HDR bus are similar to that of an audio compressor. The properties of Threshold, Ratio, and Release Time are used to modify the behavior of the project-specified dynamic range window. At run-time, the authored system dynamically maps this wide range of levels to a volume range that is more suited to your sound system's output.

In real life, the audible dynamic range, defined by the loudest possible sound and the threshold of human hearing, is several times wider than the dynamic range offered by speakers at game play levels. The role of the Wwise HDR system is to collapse or "compress" this real life dynamic range into roughly 40 dB (70 dB SPL for TV/Music listening minus 30 dB SPL for the room noise level).

The process is a sort of behavioral compression. It affects your mix by making soft sounds inaudible as soon as loud sounds play, and then making them audible again when the soft sounds play alone. The relative levels of sounds between one another remain intact and add clarity to the mix by playing fewer sounds.



### Designer Note

In prior literature, HDR audio systems are presented as having SPL values directly assigned to each individual sound. Wwise removes the notion of SPL, and instead focuses on relative mixing. Hence you will not find a SPL slider anywhere in Wwise; only relative decibel values are used. If you wish to use real-life SPL values into the system, then chose a value that will act as a reference, and perform the necessary subtraction to find the corresponding relative dB level. For example, you may decide that 100 dB SPL is your reference at 0 dB. Then a sound at 80 SPL should have its volume slider set to -20 dB, and a sound at 130 dB SPL should have its volume slider set to +30 dB.

## Setting up an HDR Audio Mix

In addition to working with the features available within the authoring application, using the HDR feature allows you to place the loudest sounds front-and-center in the mix. After you have determined the importance of various sound types you can situate them within an evolving mix during production that places them in relative importance to other sounds or sound types throughout the HDR system. The resulting mix adheres to the volumes set for each sound and is constantly balanced by the shifting dynamic range window.

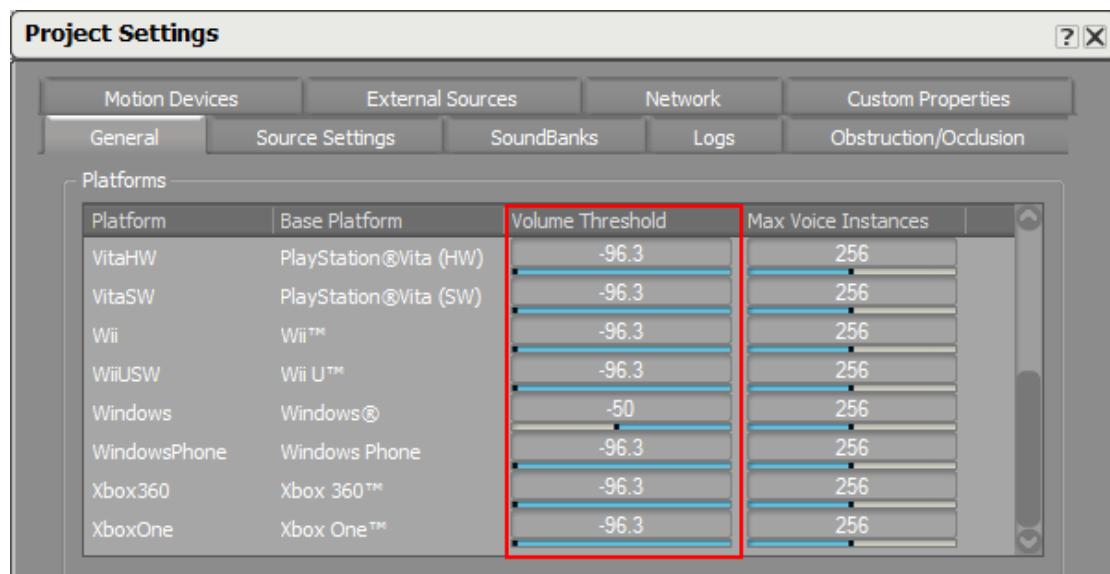
The first step is defining the dynamic range (Volume Threshold) for the project. Then, an Audio Bus is enabled for HDR processing and dynamics properties are adjusted to fine-tune the way the window responds to signals within the system.

Each auxiliary bus includes metering and the ability to set the volume, add effects, RTPC, and apply changes based on state.

### Setting up the HDR Audio Dynamic Range Window

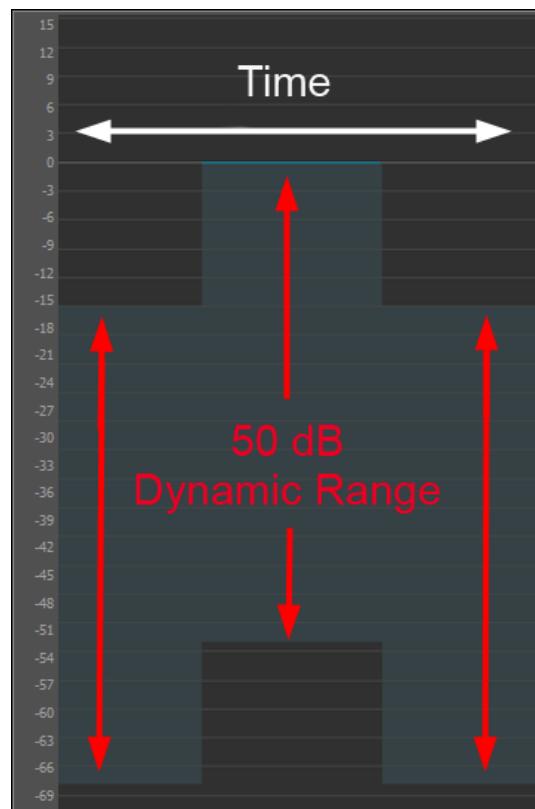
Getting set up for HDR in Wwise begins with setting the Volume Threshold (window width) that encapsulates the expected dynamic range for a specific target platform. This Volume Threshold applies to the dynamic range on the output of the HDR system.

To begin setting up an HDR system using the provided Wwise Project Adventure, set the Volume Threshold in the Project Settings to -50 dB:



Setting the Volume Threshold in the Project Settings

The Volume Threshold value from the Project Settings defines the difference between the window top and window bottom. This means that when the amplitude of the loudest sound is played back at HDR bus volume, any sounds below the Volume Threshold (window bottom) set for the platform will either be killed or use their virtual voice settings according to the advanced settings defined in the object properties of each structure. When the window shifts, the dynamic range specified for the window is maintained as it rises and falls adjusting to the loudest sound played back at the HDR bus volume.

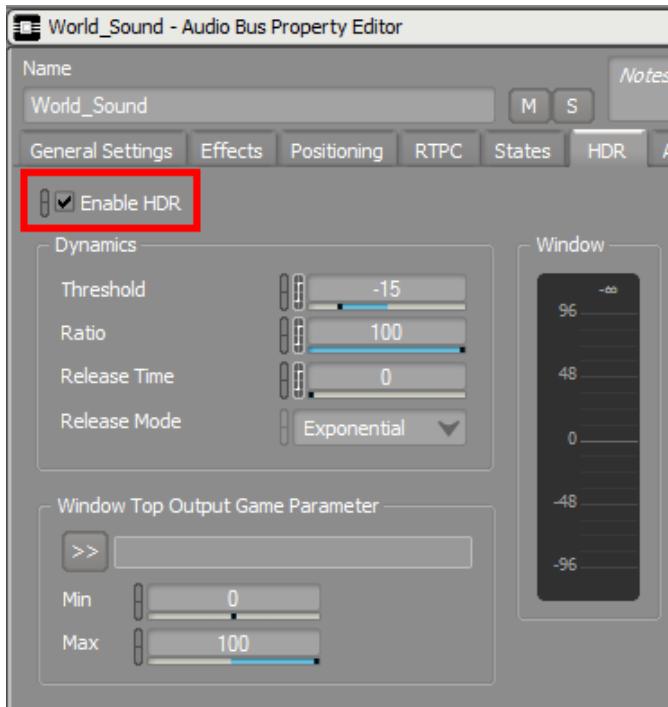


Visualization of a 50 dB Dynamic Range  
Volume Threshold Window shifting over time.

This Volume Threshold allows sounds that are within the specified volume range to be heard at the output. As the window moves, the volume of sounds that are playing is modified by the value of the loudest sound. This process ensures that sounds that are authored the loudest are heard and quieter sounds (below the window bottom) are removed from the mix.

## Enabling HDR Audio in the Master-Mixer Hierarchy

HDR, which can be enabled at any parent level Audio Bus, takes into account any sounds routed through child busses within the hierarchy. While only one bus in a given hierarchy can be set as an HDR Audio Bus, other Audio Busses outside of an HDR-enabled Audio Bus can be used as additional HDR busses.

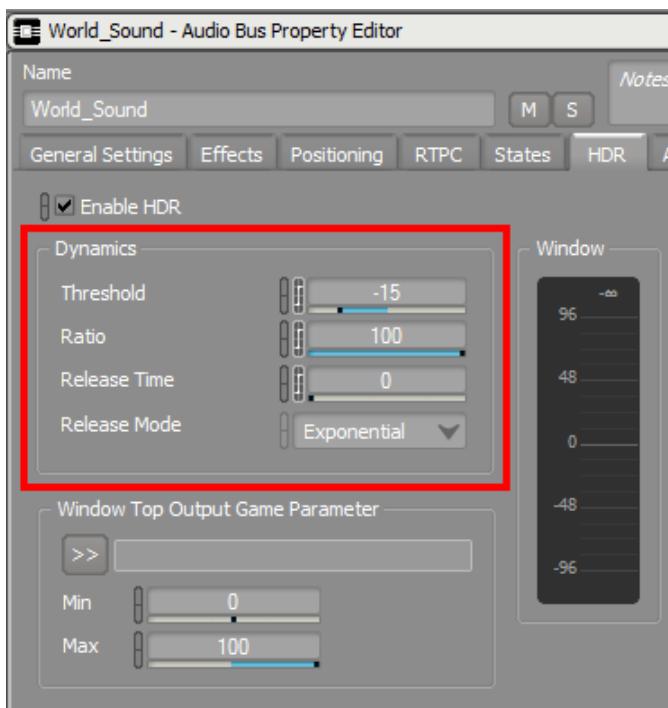


Enabling HDR in the Master-Mixer Hierarchy

### Setting up HDR Audio Dynamics Properties

Dynamics properties govern the way the HDR system reacts when sounds routed to the HDR Audio Bus interact with each other. As sounds with different volumes are passed through the HDR Audio Bus, their amplitude dynamically adjusts the global volume based on familiar properties like Threshold, Ratio, and Release Time.

To test the HDR system using the Wwise Project Adventure, enable HDR on the “World\_Sound” Mix Bus and set the Threshold property to -15 dB, leaving the other Dynamics settings at their default values.



Dynamics controls for an HDR enabled bus.

### Threshold

The Threshold defines the minimum input level (in dB) above which the HDR window top will engage.

### Ratio

This control has a similar behavior to the ratio control in an audio compressor. The HDR window top attenuates peaks that exceed the threshold while reducing the volume of quieter sounds in proportion to the ratio. For example: two sounds, one peaking at 20 dB and the other peaking at 40 dB above threshold, come out at the same level of 0 dBFS, as long as they are not played at the same time. The difference between the two is that the former will result in an attenuation of -20 dB to sounds below threshold, while the latter will result in an attenuation of -40 dB.

At lower ratios, say 4, a sound peaking at +20 dB comes out at +5 dB, while a sound peaking at +40 dB comes out at +10 dB. The attenuation that results on sounds below threshold in these examples is -15 dB and -30 dB respectively. Using lower ratios is therefore useful to gain back "global" dynamic range for sounds above threshold that are otherwise

taken away by the HDR system. The drawback is that sounds may peak above threshold, so you need to keep sufficient headroom after the HDR bus to avoid clipping. This can be done by setting the HDR bus volume to a value lower than 0 dB (for example, -10 dB).



### Release Time

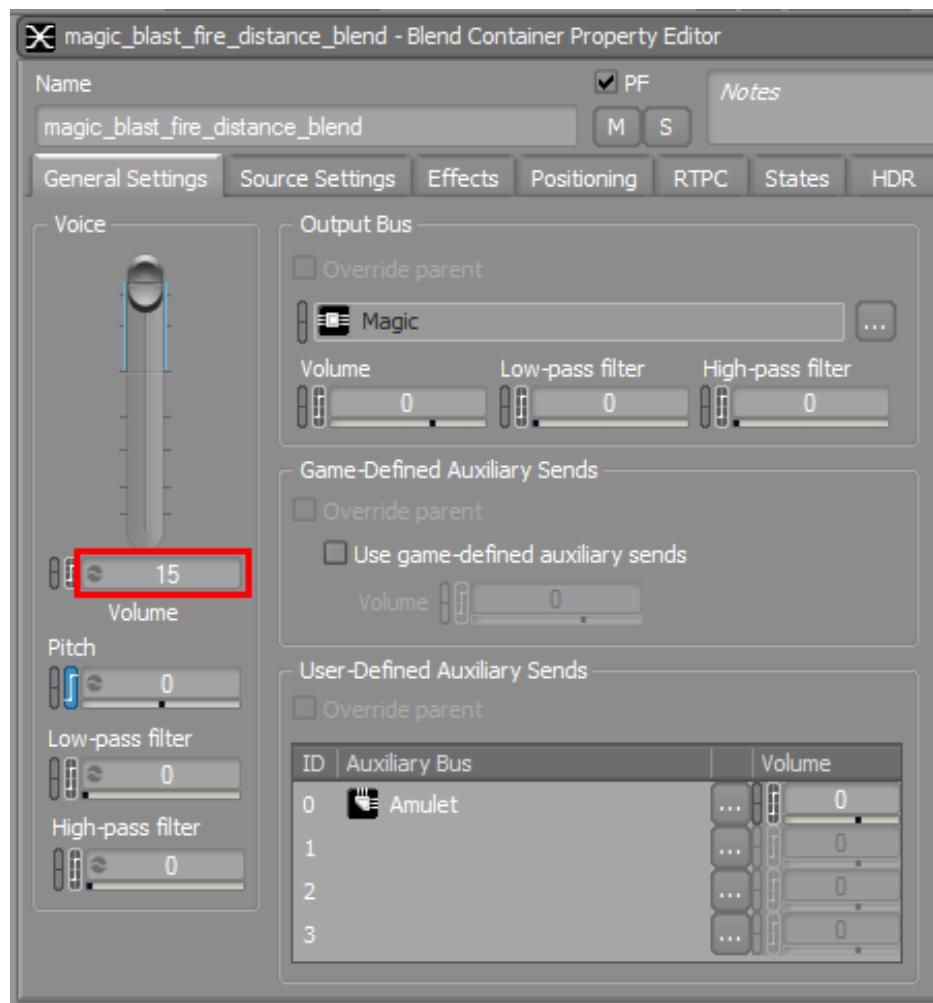
The Release time defines the rate at which the HDR window falls back to rest when the target is below the current value. In Linear Mode, it is the time in seconds it takes to fall by approximately 10 decibels. In Exponential Mode, it is the time in seconds it takes to reach approximately 0.37 (1/e) of the difference between the target and the current value. To know which mode to choose, you need to decide which sounds best based on your source material and game type.

## Using HDR Audio in the Actor-Mixer Hierarchy

Each sound object has additional HDR functionality within the Actor-Mixer Hierarchy. From the HDR tab in the Property Editor, special considerations can be addressed that affect the way sounds behave when processed within the HDR system.

In the Wwise Project Adventure Actor-Mixer Hierarchy, begin by setting volume values for groups of sounds based on the provided Work Units:

- Ambient: -20 dB
- Character: -10 dB
- Combat: -5 dB
- Magic: 15 dB



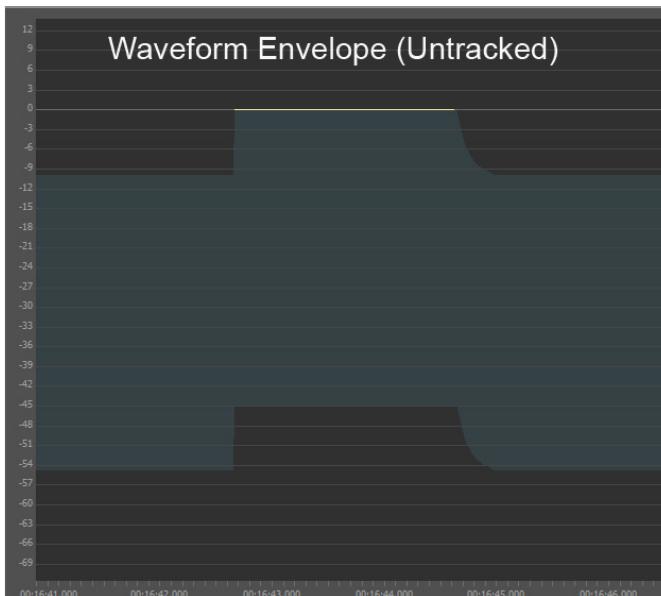
Setting the Volume for the “magic\_blast\_fire\_distance\_blend” Blend Container

These values represent a relativistic mix that represents magic sounds as the loudest, followed by: combat, character, and ambient sounds in decreasing amplitude. This simple group-based mix allows you to audition different sound types within the HDR system to see how playback affects each of them.

### Enabling Envelope Tracking

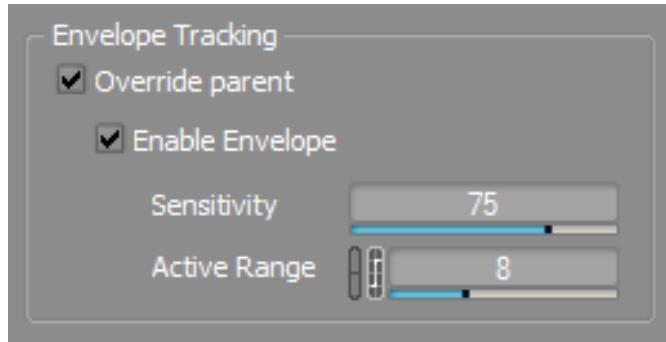
From high atop the mountain, the threshold engages the HDR system as the first magic blast resonates across the battlefield. The volume of the magic blast has been authored as the loudest sound in the project, which swiftly engages the window top and removes脚步 sounds from the mix. Regardless of your position during battle, the jolt of the blast signifies danger and all the sounds - nature, footsteps, and fallen soldiers - are removed from your purview while you focus on the magnitude of sound.

As the window top follows the magic blasts envelope, the cold sound of steel-on-steel begins to cascade across the ravaged field of warriors. Authored below the relative volume of a magic blast, each impact rings out with a sharp attack that continues peaking the window top. The ratio property directly relates to the attenuation of sounds below threshold. This leaves ambiance out of the mix as the battle cries rise up towards another assault.

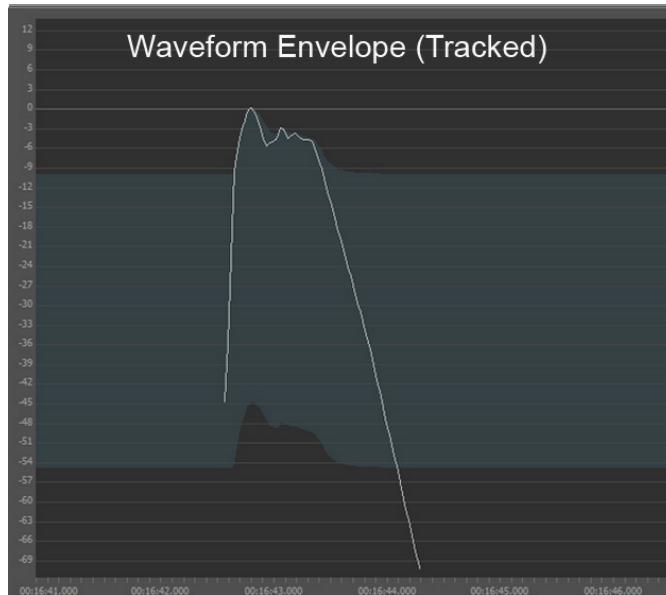


Untracked waveform envelope shifting the HDR window top

When Envelope tracking is enabled, an offline analysis of the waveform automatically calculated by Wwise is used to adjust the window top in relation to the waveform's envelope. This can be very useful for loud sounds that drive the window top over the Volume Threshold. Because the process of tracking the waveform envelope has a small memory cost compared to the default envelope behavior, it is recommended for use with loud sounds or sounds with a long envelope that changes in volume.

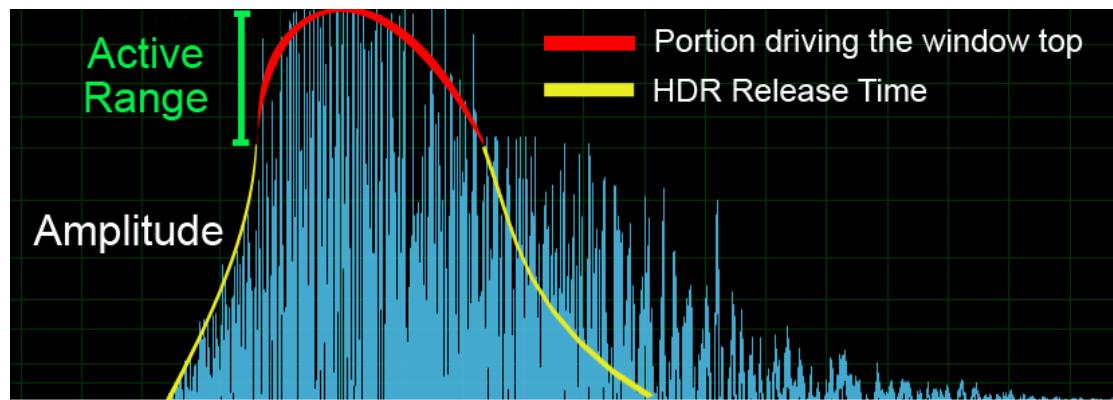


Envelope Tracking properties for Sensitivity and Active Range.



Tracked waveform envelope shifting the HDR window top.

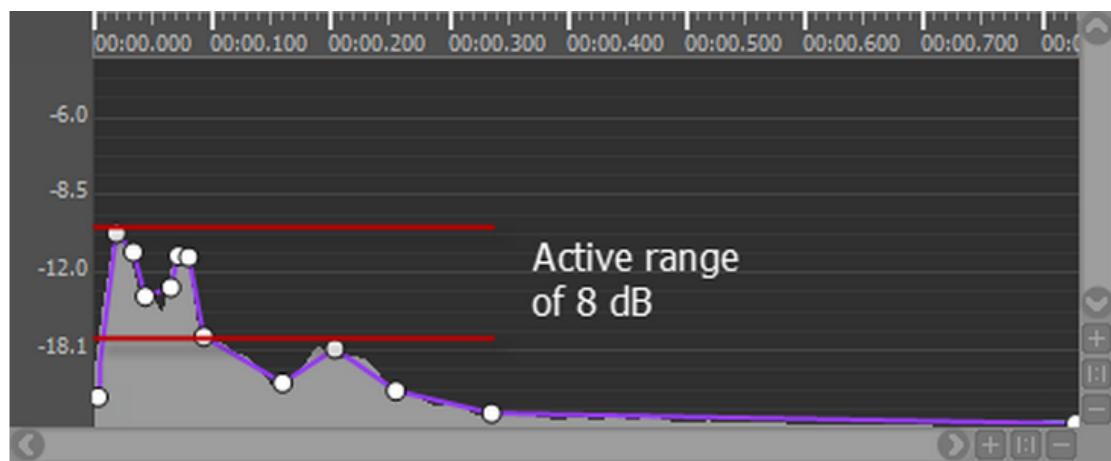
The active range defines the area of a voice from its peak (in decibels) in which the HDR dynamics are active. This region of interest is based on its analyzed envelope: it is "active" as long as the current envelope level is above "peak level" minus "active level". When it is not active, the HDR dynamics ignore the content of the sound and the release time set in the HDR bus is applied.



A chart describing the relationship between the Active Range and HDR Release Time in relation to the envelope of a sound.

### Editing a Waveform Envelope

You can further adjust individual waveform envelopes from the Source Editor for specific sound files.

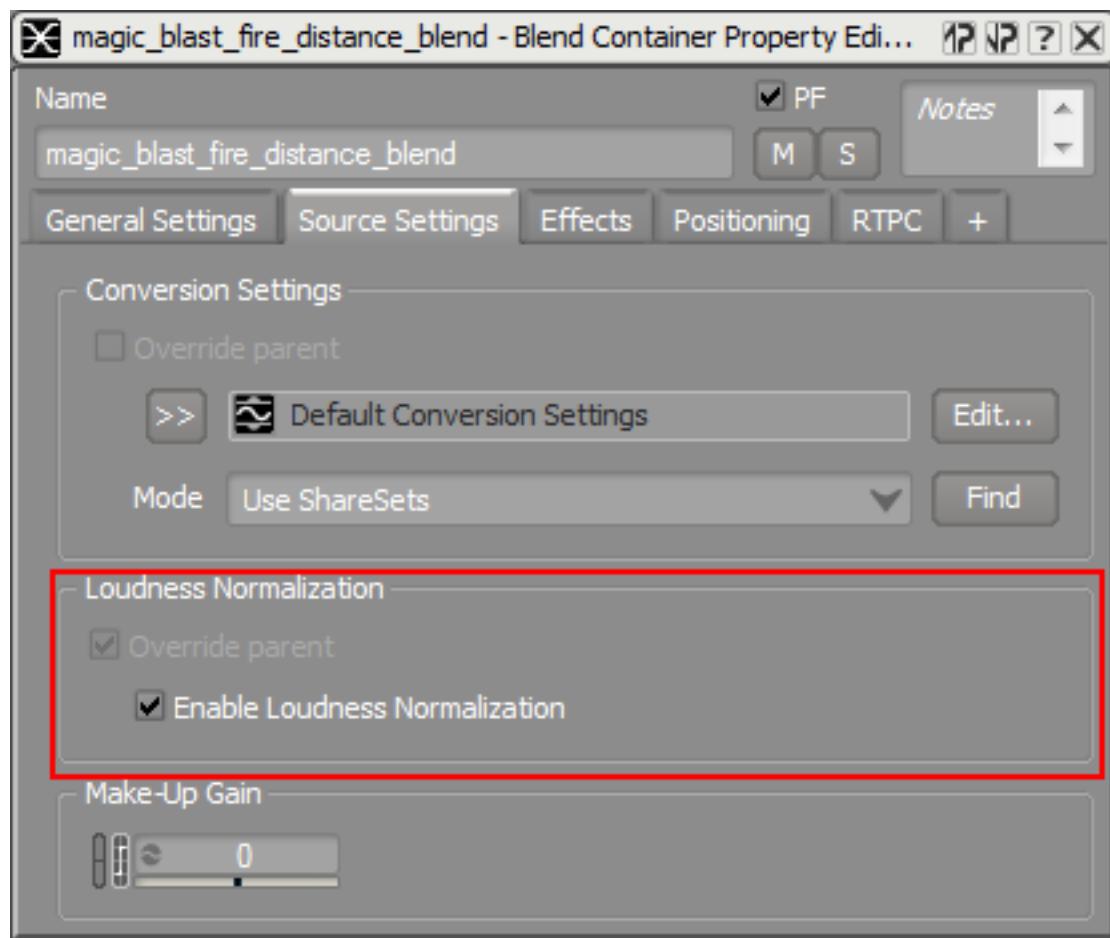


Editing the audio source envelope to isolate the interesting audio signal that should be part of the 8 dB active range defined in the Object properties HDR tab.

The Envelope Editing Sensitivity value directly relates to the number of data-points in the editor that are represented and tracked when Envelope Tracking is enabled. Reducing the sensitivity reduces the level of detail that is tracked by the HDR system at runtime. Data-points can be modified by manually modifying their position in the Source Editor.

## Enabling Source Normalization

Non-destructive Source Normalization can be enabled at either the Parent or overridden at the child level of the Actor-Mixer Hierarchy for any sound or group of sounds. Enabling loudness normalization provides normalization of any sources by applying an automatic gain calculated from the measured loudness of the source recording.

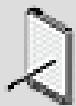


Enabling Loudness Normalization in the Source Setting tab of a Sound Object

To hear the results of Loudness Normalization, enable Loudness Normalization in the Source Settings for Sound Objects contained within these Work Units:

- Ambient
- Character
- Combat
- Magic

Historically the practice of normalization has varied widely across different development pipelines and audio engine methodologies. Enabling Loudness Normalization as part of your mixing strategy gives you a reliable way to ensure that two sounds playing at the same volume (e.g. -10 dB) will be perceived as playing at exactly the same level. In this way the sound content is uniformly prepared in a way that inspires confidence during mixing.

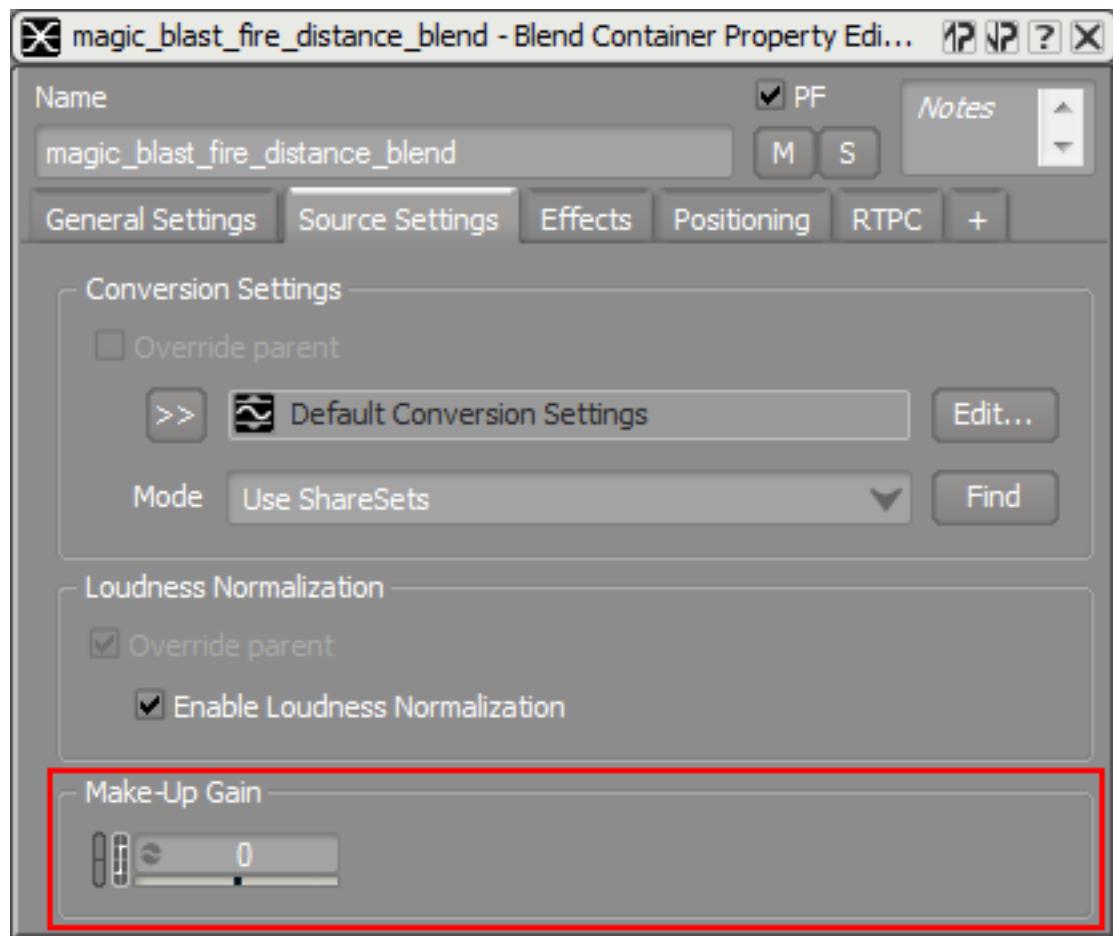


### Designer Note

Loudness Normalization is applied to source files at a target value of -23 dB. Wwise analyzes the wave data, stores its loudness measurement proportional to its RMS value, and then at run-time, applies gain to the sound such that its loudness will be equal to -23 dB. For example, a sound turns out to have a loudness of -35 dB; at run-time, we "normalize it" by applying a gain of +12 dB.

## Using Make-up Gain

Make-up Gain defines an amount of gain applied after the HDR processing. This gain does not influence the HDR dynamics, in the way that adjustments made with standard volume sliders do. Make-up Gain can be used to offset sound levels relative to the HDR window. It can also be used to adjust loudness normalization.



#### Using Make-Up Gain to offset volume levels for a Sound Object

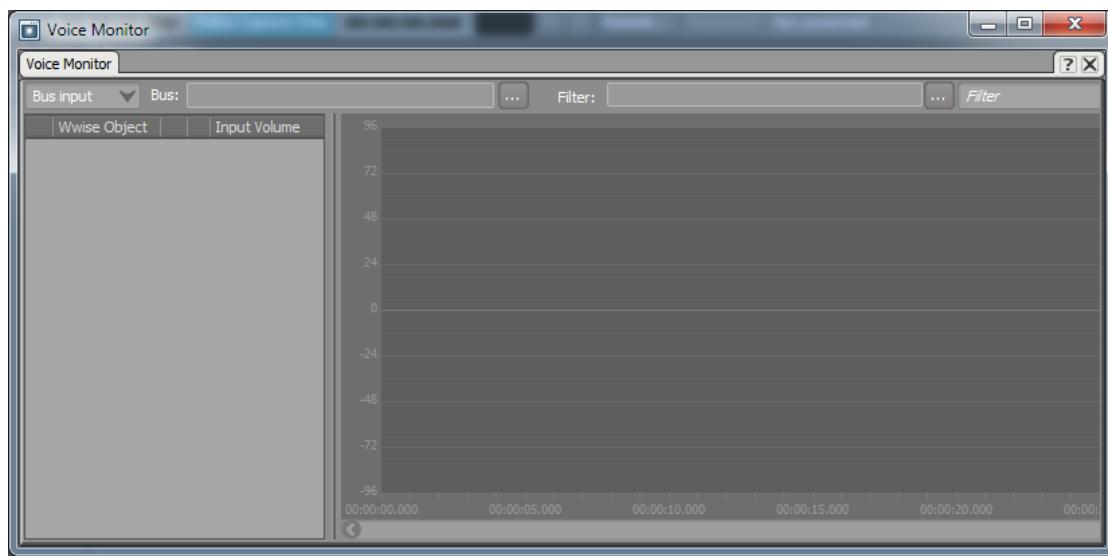
The ability to control the volume of a source without affecting the HDR response is a powerful tool that gives precise control over the mix. The abstraction of volume values becomes a natural process as the mix is iterated during production. Once volumes for sounds are set within the HDR system, this additional volume control ensures that the mix can be fine-tuned without disturbing the overall composition.

## Using the Voice Monitor to Understand HDR Audio

Being able to see the effect of the moving window on sounds routed through the HDR bus allows for visualization that helps to clarify the process. Using the Voice Monitor view illustrates the window movement, the volume of sounds, and how the envelope of the loudest sound affects the HDR system.

### Opening the Voice Monitor View

Begin by opening a Voice Monitor view from the Views Menu in the Menu Bar.



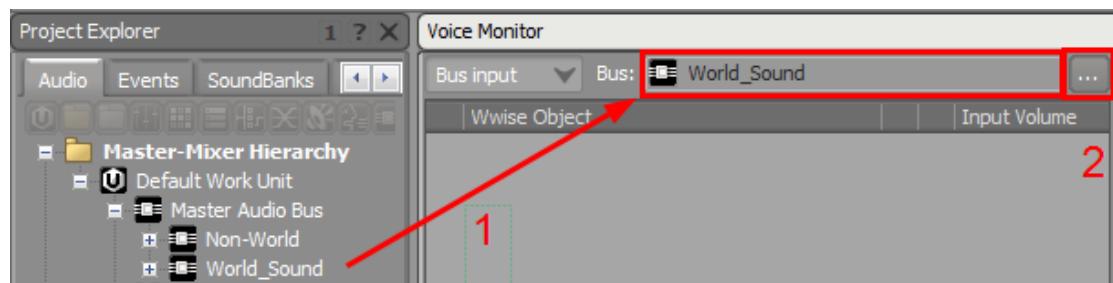
Opening the Voice Monitor View

To see and understand the different behaviors of HDR in action, set the monitoring mode to either: Bus Input, Bus Output, or All voices.



Viewing the different monitoring modes in the Voice Monitor

A HDR Bus can then be assigned to the Voice Monitor using either drag and drop or the Project Explorer Browser.



Assigning a mix bus can be done using drag and drop (1) or using the Project Explorer - Browser (2)

After these properties of the Monitor View have been assigned, any audio running through the selected bus is monitored when capturing from the authoring application or from a remotely connected game running on any development platform.

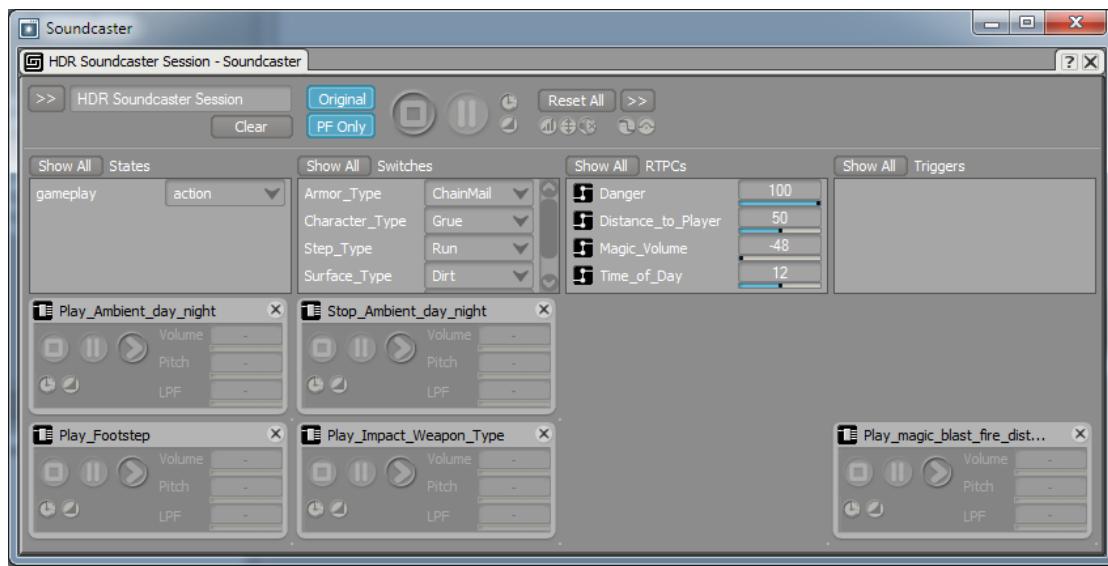
### Auditioning Sounds in the Soundcaster

In the Wwise Project Adventure, open the Soundcaster View from the Views Menu and select the default “Soundcaster Session” from the list of sessions:



Selecting the default Soundcaster Session within the Soundcaster

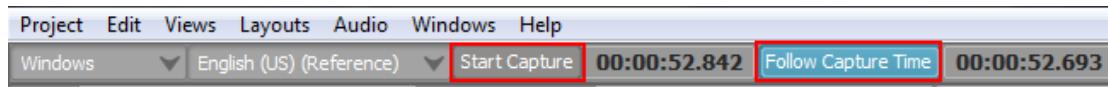
The Soundcaster Session contains a set of Events that can be used to audition sounds within the Project and additional Events can be added to further experiment with the different categories of sound.



The default Soundcaster Session included in the Wwise Project Adventure

### Capturing Data from Wwise

Now that the Soundcaster is ready for auditioning sounds within the HDR system, use Wwise to capture data from the project. To capture data in the authoring application, click Start Capture. Use the Follow Capture Time button to follow along with the capture time to see what happens in real-time.

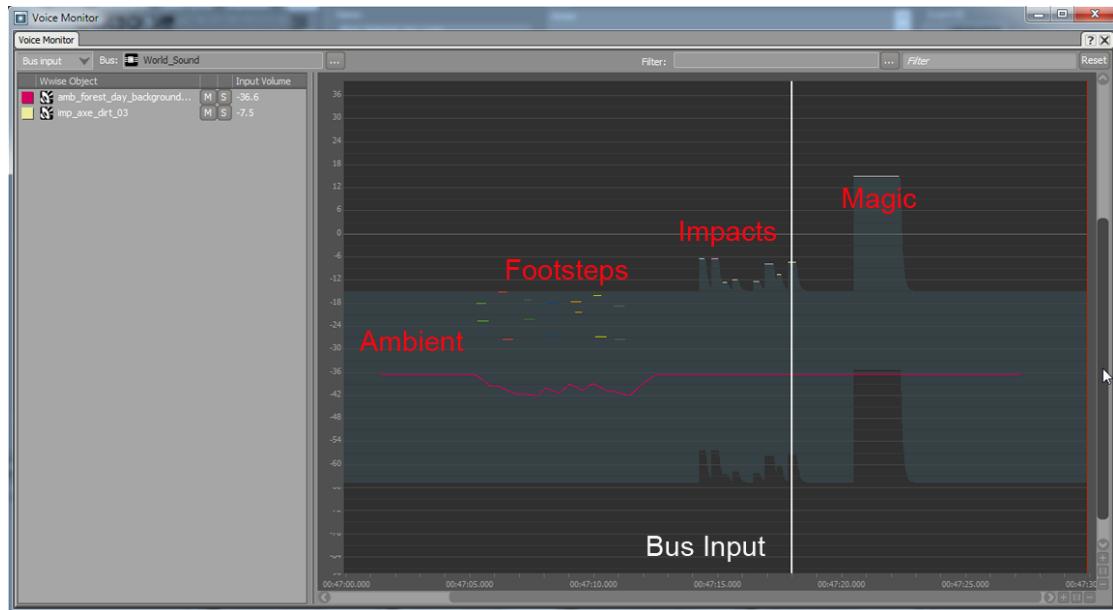


### Selecting Start Capture and Follow Capture Time in the Voice Monitor

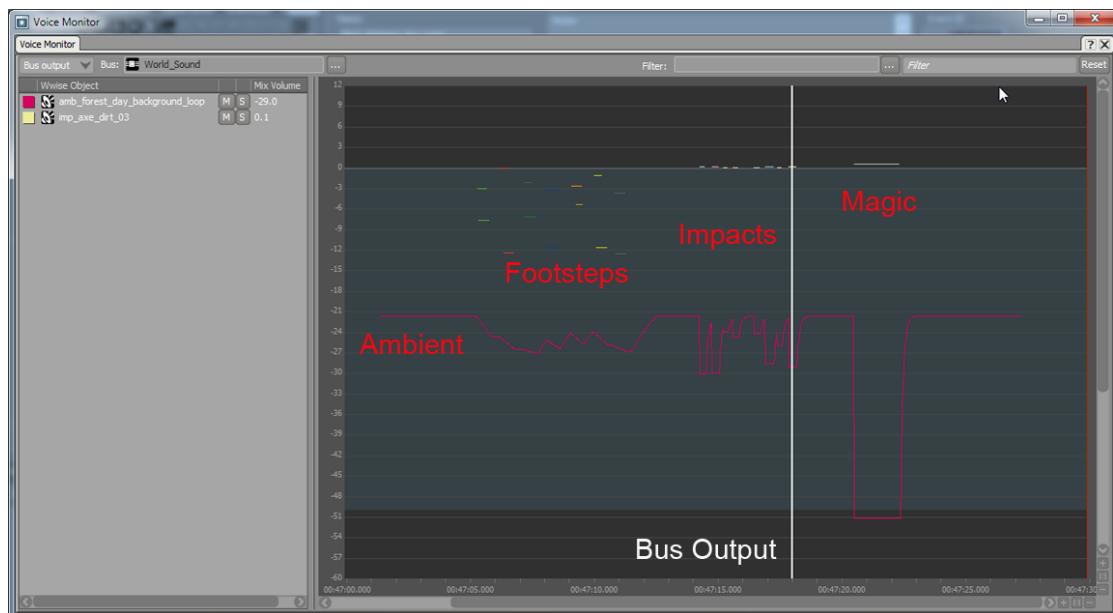
Once capturing has begun, any sounds passing through the selected bus are displayed according to their current volume. Sounds can be auditioned from the Transport, Soundcaster, or from a locally or remotely connected game running on any development platform.

Begin auditioning by selecting the Play icon for the Play\_Ambient\_day\_night Event. From this point, other Events can be auditioned and their effect on the HDR window can be seen represented in the Voice Monitor. It is worth enabling the envelope for the magic sounds to see the difference between settings. Adjusting the HDR Threshold or Ratio will give additional insight into how dynamic range can be modified within the HDR system.

Here's a look at a captured sequence of Events using the Voice Monitor:



Monitoring the Bus Input during a capture using the Voice Monitor



Monitoring the Bus Output during a capture using the Voice Monitor

Wwise has implemented an end-to-end solution for authoring and auditioning the HDR system, the results of which can be seen in the Voice Monitor. Being able to test out different values offline within the authoring application and immediately visualize their effect on the mix bring unprecedented clarity to the inner working of the created system. This ability to not only hear the resulting mix but see how sounds affect each other is the key to making optimal mix decisions and knowing how they affect the final output.

## **HDR Audio Summary**

Because HDR is applied selectively to a parent bus within the Master-Mixer hierarchy, it need not be the only technique used for dynamically mixing the game. HDR is one of six mixing techniques within the authoring application that include: set-volume mixing, state-based (snapshot) mixing, auto ducking, RTPC, side-chaining, and high dynamic range mixing. Striking the right balance between different mixing systems allows you to creatively orchestrate your interactive mix and help solve mix-related problems, create customized mixing solutions, and address any project needs that may arise during production.

**Throughout this chapter we have:**

- Described the theory behind High Dynamic Range audio within Wwise.
- Given background for the application of HDR using the Wwise Project Adventure as an example.
- Provided an fictitious scenario which illustrates the fundamentals of HDR audio.

**Stepped through the process of:**

- Setting up the HDR Audio Dynamic Range Window.
- Enabling HDR Audio in the Master-Mixer Hierarchy.
- Setting up the HDR Audio Dynamics Properties.
- Enabling HDR Audio in the Actor-Mixer Hierarchy.
- Enabling Envelope Tracking.
- Editing a Waveform Envelope.
- Enabling Source Normalization.
- Using Make-up Gain
- Using the Voice Monitor to Understand HDR Audio.
- Capturing Data from Wwise to view HDR Audio in the Voice Monitor.

**Also touched on:**

- A fictitious scenario illustrating the fundamentals of HDR audio.

**Throughout this section we have created the following objects:**

- The Project Settings Volume Threshold
- Master-Mixer “World\_Sound” Audio Bus
- Volume and Loudness Normalization for Sound Objects within the following Work Units:
  - Ambient
  - Character
  - Combat
  - Magic

---

# Chapter 11. Getting Set Up for Adventure

Overview .....	238
Work Unit Management .....	239
Establishing a Naming Convention Early .....	239
Logical Grouping of Work Units .....	239
Creating Work Units with Sharing in Mind .....	239
Batch Renaming .....	240
Grouping Objects in the Actor-Mixer Hierarchy .....	247
Setting the Audio Channel Configuration .....	249
Speakers vs Headphones Panning Rules .....	249
Creating Simulations with the Soundcaster .....	252
Project Settings .....	254
Project Settings - General Tab .....	254
Workgroup Plug-in Configuration .....	254
Audio File Locations .....	256
Default Conversion Settings .....	259
Defining the Sample Rate Automatic Detection Settings .....	259
Platform Manager .....	260
Obstruction and Occlusion .....	265
Setting Obstruction and Occlusion .....	267
Motion .....	270
Generating a Motion Source from an Existing Audio Signal .....	271
Generating Motion Using a Motion FX Object .....	272
Customizing Layouts .....	274
Layout Docking .....	275
View Docking .....	277
External Sources .....	279
SoundBanks and SoundBank Generation .....	280
Creating a SoundBank .....	280
Populating and Managing the Contents of SoundBanks .....	281
Excluding Elements from a SoundBank .....	282
A New Approach for SoundBank Management .....	283
Conversion Settings .....	284
The SoundBank Definition File .....	284
Using the Integrity Report .....	286
Using the File Packager .....	287
Downloadable Content (DLC) .....	288
Setup Summary .....	289
Setup Additional Resources .....	290

## Overview

Like any well planned adventure, being prepared for the trials and tribulations you'll face along the way starts with what you bring with you. A well thought out plan of attack can make for smooth travel along the road to development.

This chapter will take you through the process of:

- Managing Work Units
- Batch renaming objects
- Grouping objects in the Actor-Mixer Hierarchy
- Adjusting Project Settings
- Defining the default conversion settings
- Using the Platform Manager
- Defining workgroup settings
- Setting the audio file locations
- Managing obstruction and occlusion
- Working with motion
- Customizing layouts
- Understanding and generating SoundBanks
- Defining conversion settings
- Using the Integrity Report
- Using the File Packager
- Preparing for downloadable content (DLC)

When building a hierarchy there are a couple of key considerations to keep in mind.

## Work Unit Management

Work Units are the structural base in the organization of all your project's objects. Managing Work Units well is, therefore, key to maintaining an optimal project structure. The following paragraphs detail strategies to help in this regard.

### Establishing a Naming Convention Early

Establishing a naming convention for work units, sound objects, and audio files can help identify the type of asset and possibly its orientation within the project at a glance. In larger projects, being able to visually distinguish different types of assets based on naming alone can really speed up the workflow. Throughout this project we've attempted to adopt a naming standard that is consistent and transparent.



#### Designer Note

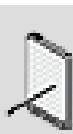
Don't worry! If the project becomes a mess of different naming conventions over the course of development, the Batch Rename utility can be used to clean up inconsistencies and unify naming.

### Logical Grouping of Work Units

While it's easy to begin dragging and dropping WAV files directly into the Actor-Mixer Hierarchy, when planning for a larger project you and your project would benefit from some additional thought on the matter. A simple hierarchy may consist of a single work unit that encompasses all of the game objects. Alternatively, you might choose to have a work unit for each type of sound, for example: Ambiance, Characters, Combat, UI, Voice, and so on. These Work Units can contain Game Objects of any type and can be further organized within folders or Actor-Mixers.

### Creating Work Units with Sharing in Mind

It's important to note that work units represent individual files within the Wwise project folder. These files are readable via a text editor and contain the information, properties, and relationships specified within the tool. Hierarchies of nested work units can be created in Wwise and organized in physical folders and subfolders. If Work Units are present in a subfolder for a specific category (such as the Actor-Mixer Hierarchy), they will be loaded with the project. This results in a finer granularity of files available for source control which simplifies development in a multi-user environment.

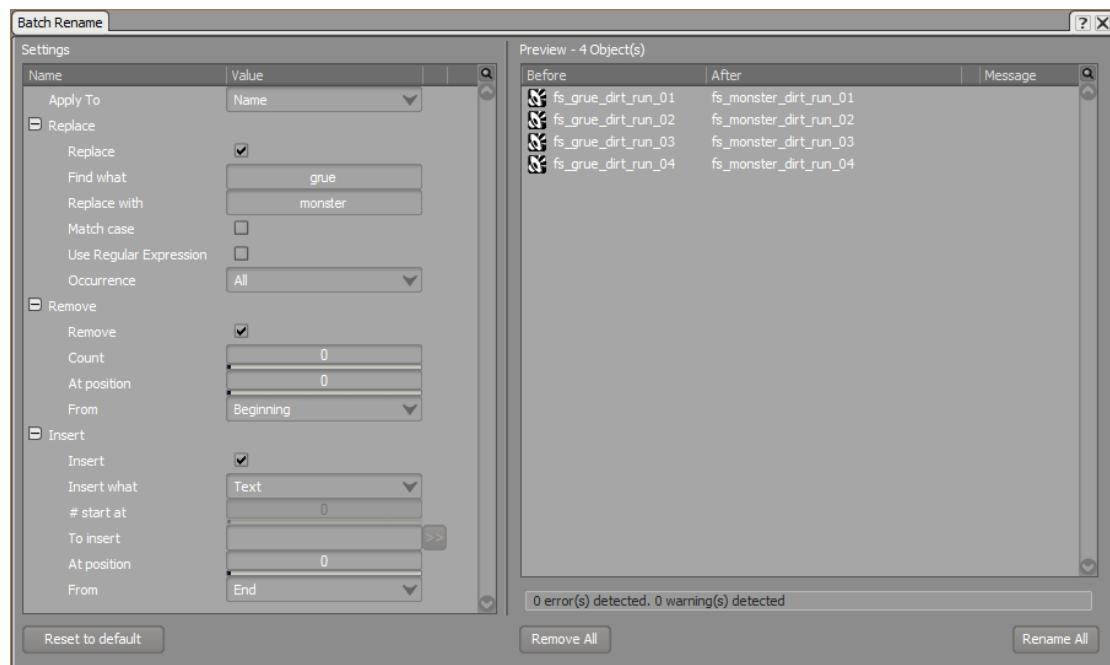


## Designer Note

When working with a large audio team, it is often necessary to create enough work units to allow for a single person to work exclusively with a given work unit for a duration of time. If you are using the integrated Wwise Source Control solution, it is important to make sure you create a hierarchy that will work well with the file permissions established by the programmer.

## Batch Renaming

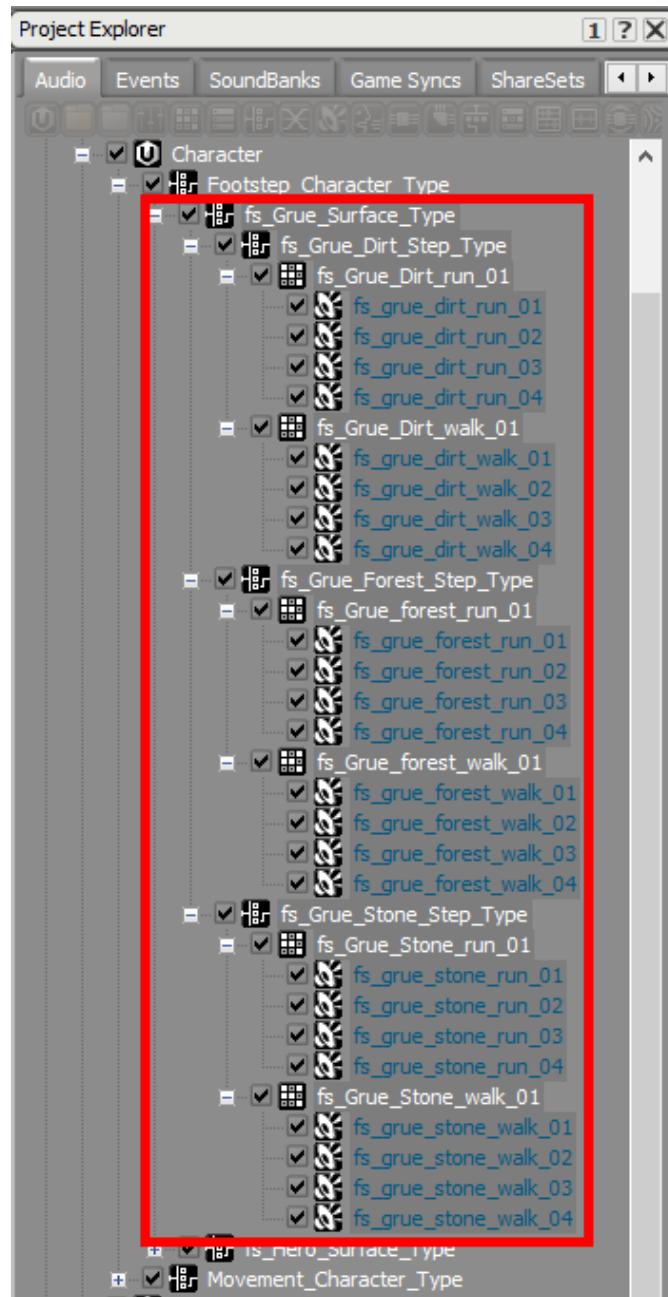
There is often a point during development when the reorganization of a Wwise project can help prepare for the next phase of development. Whether it's due to project iteration or a changing asset naming standard, a clean and clear project organization can quickly communicate an abundance of information. No place is this made more transparent than in the naming standard that is adopted throughout. From WAV file to container, Event to SoundBank, and places in-between, how things are named can help swiftly navigate the idiosyncrasies of a project. One of the tools that can help unify a project's naming is the Batch Rename utility.



**Batch Rename view**

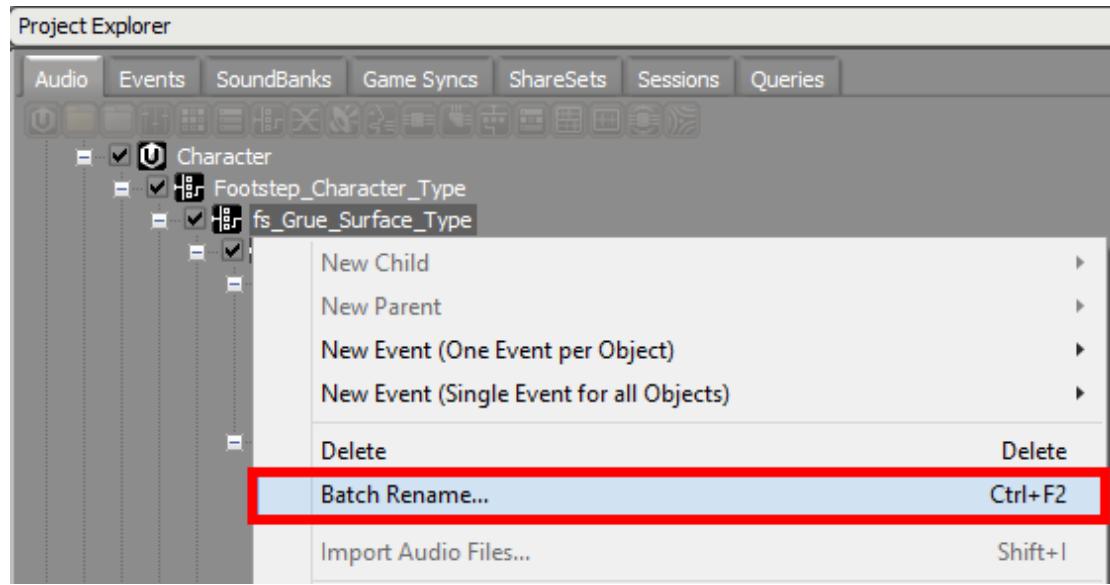
From a simple "find & replace" to a more complicated naming that involves invoking regular expressions, batch renaming can be done across any nameable assets in the Wwise project. Any combination of Sound Objects can be selectively renamed and modified in order to adhere to a standard.

To change the name of all Character footstep sound objects, multiple select the sound objects in the Project Explorer.



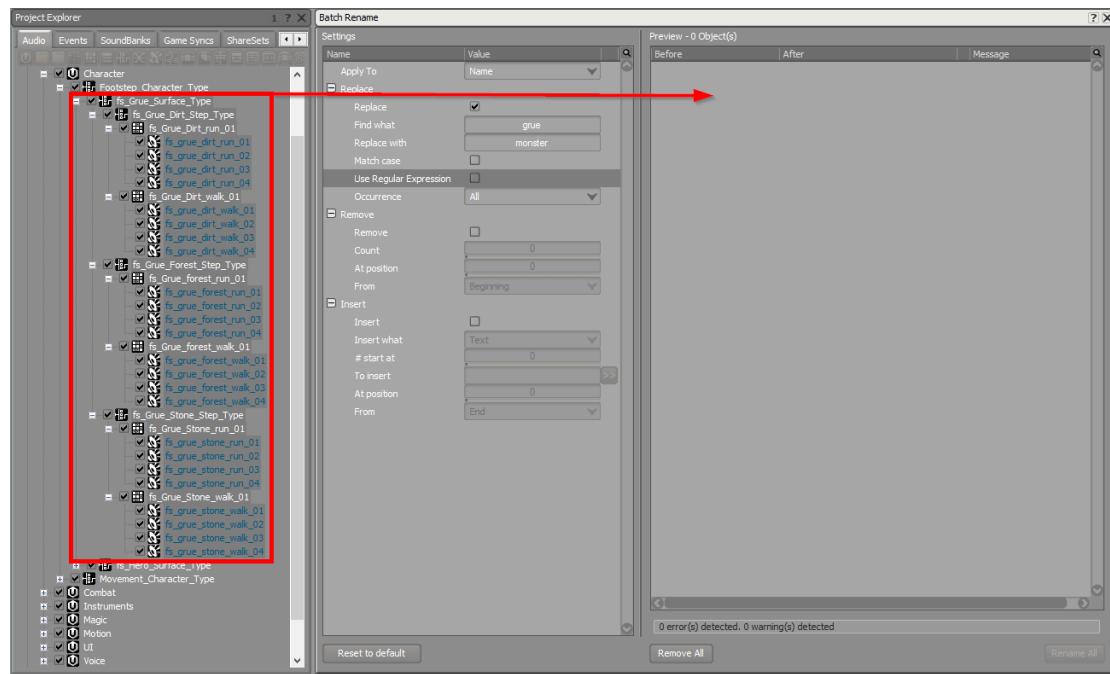
Selection of objects in the Project Explorer for the Batch Rename

Once Sound Objects have been selected, the Batch Rename view can be accessed by right-clicking and selecting "Batch Rename..." from the list of options.



Batch Rename shortcut menu option

Alternatively, any combination of Sound Objects can be dragged and dropped from the Project Explorer to the Batch Rename Preview panel.



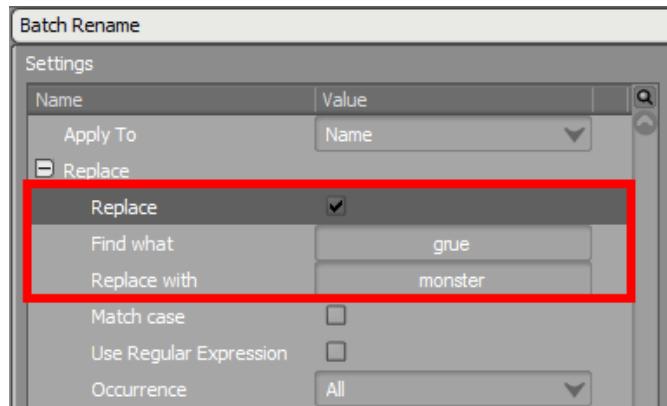
Dragging objects from the Project Explorer to the Batch Rename view

Once Sound Objects have been added to the Preview panel, individual selections can be removed by right-clicking and selecting "Remove from List" from the contextual menu, or by pressing the "Delete" key.

Preview - 33 Object(s)		
Before	After	Message
fs_Grué_Dirt_run_01	fs_Grué_Dirt_run_01	
fs_grue_dirt_run_01	fs_grue_dirt_run_01	
fs_grue_dirt_run_02	fs_grue_dirt_run_02	
fs_grue_dirt_run_03	fs_grue_dirt_run_03	
fs_grue_dirt_run_04	fs_grue_dirt_run_04	
fs Grué Dirt Step Type	fs Grué Dirt Step Type	
	Remove from list	

### Removing objects from the Batch Rename view

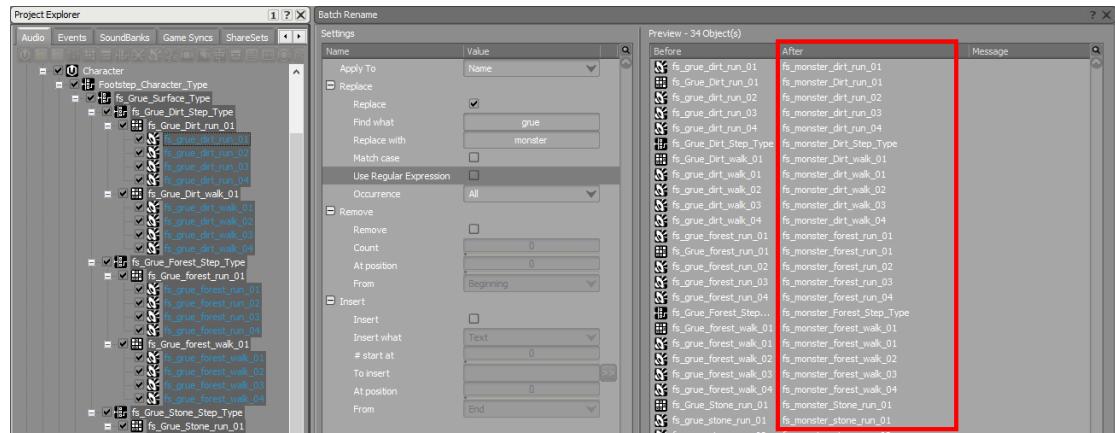
In this example the name of the creature will be changed from "grue" to "monster" across all of the Character movement Sound Objects. This can be accomplished by enabling Replace, entering "grue" in the "Find What..." field and "monster" in the "Replace With..." field.



### Using the Replace section of the Batch Rename

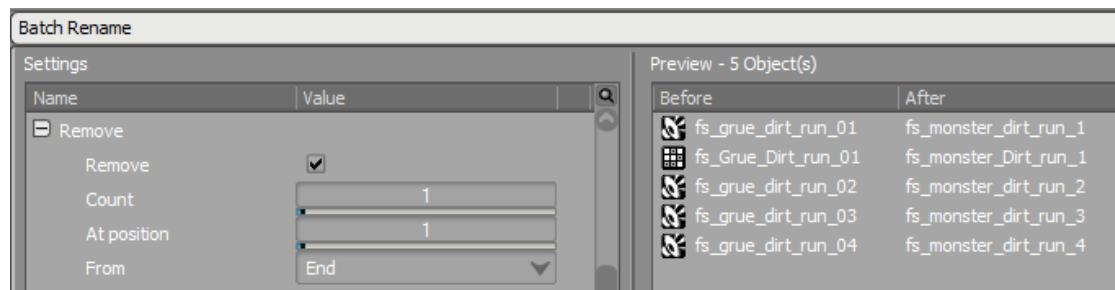
Additional Rename functionality includes the ability to Match Case, Use Regular Expressions, and specify which occurrences of the matching text to replace (All, First, Last).

Changes specified in the Batch Rename Settings panel will be applied to the selections in the Preview panel and reflected in the "After" column.



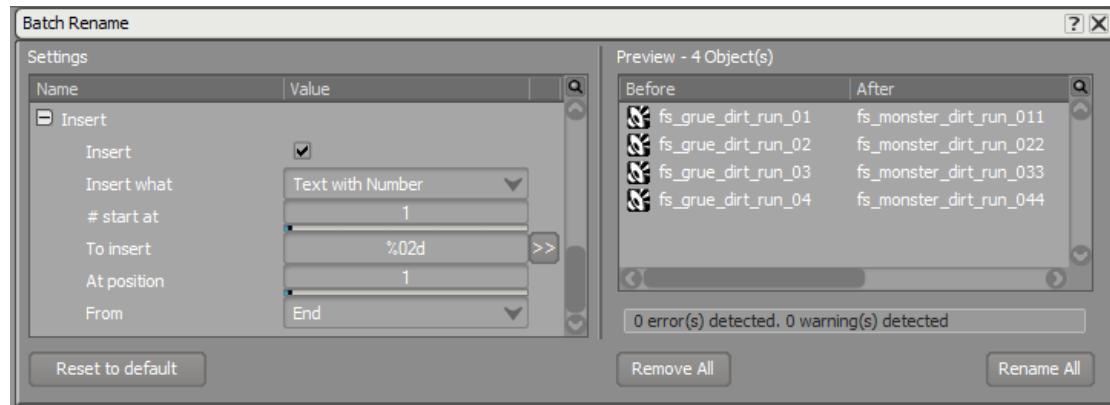
Preview of modifications seen in the After column

Enabling Remove, via the checkbox, allows for the removal of characters from a specified position within the name of a Sound Object. In this example, one character is being removed from the first position from the end of the name.



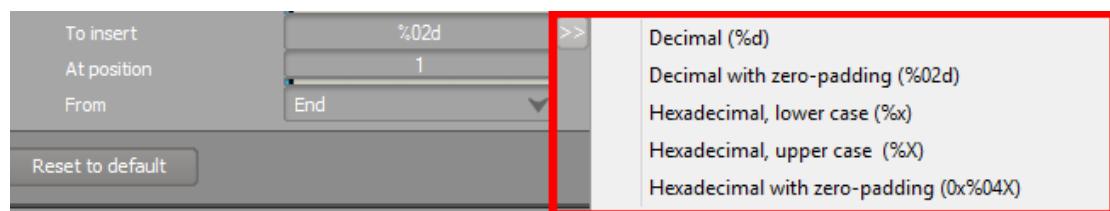
Commonly used number patterns available in the Batch Rename

Using the Batch Rename utility to insert decimal and hexadecimal data into a name allows for incremental numbering of Sound Objects. When enabled, the Insert functionality includes the insertion of either "Text" or "Text with Number" at the designated position and location. In this example, a decimal with zero-padding starting at the number 1 is being added 1 position from the end.

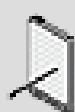


**Adding a decimal with zero-padding starting at the number  
1, 1 position from the end of the Sound Object name**

The "Text with Number" functionality includes the ability to insert a number defined according to the C++ printf format number pattern, along with any applicable text adjustments. Commonly used number patterns are available in the selector list.



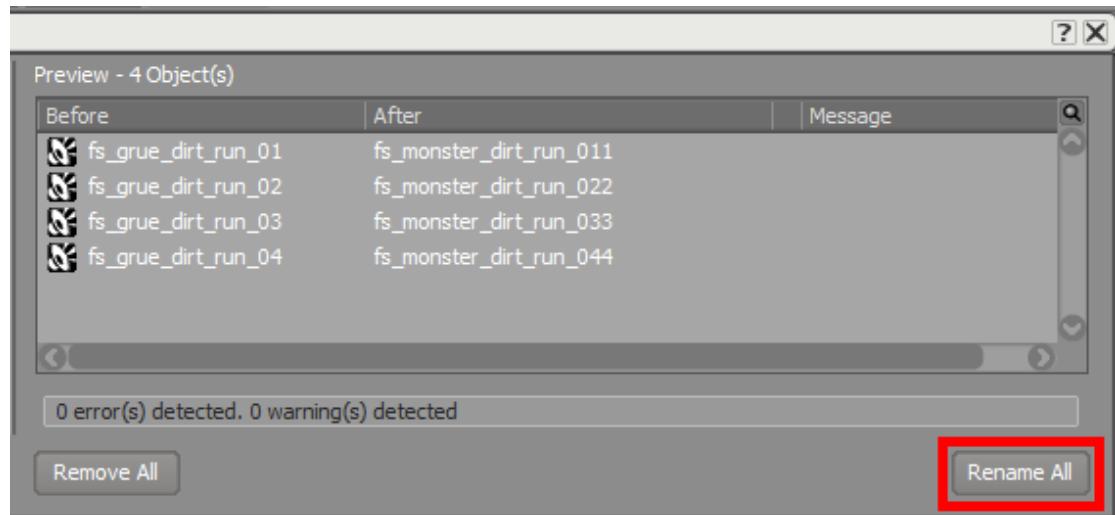
**Batch Rename - Removing characters from a Sound Objects name**



### Designer Note

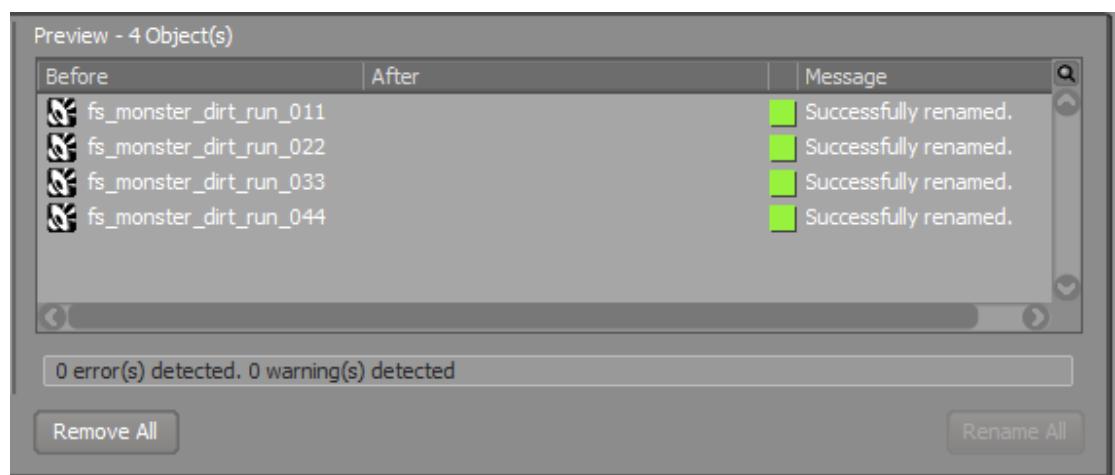
For more information about Batch Rename properties refer to the contextual help icon available in the top-right corner of the Batch Rename view.

Once all of the Batch Rename functions have been previewed, clicking the "Rename All" button will execute the selected functions and apply them to the Sound Objects in the Preview panel.



Pressing Rename All executes the Batch Rename operation

Messages regarding the status of a rename Preview or execution are output in the "Message" column and indicate: success, warning, or error.



Batch Rename message showing successful renaming

Batch renaming throughout the course of a project is limited only by the organizational methodology and naming standard that is adopted. While a naming standard begins with the WAV filename as it's imported at the beginning of the implementation pipeline, the continued application of naming standards across Sound Objects within the project helps maintain the fluid navigation of objects using easily identifiable naming conventions. The benefits of a well named and coherently organized project can help workflow clarity and allow collaborators to easily identify different parts of a project at a glance.

## Grouping Objects in the Actor-Mixer Hierarchy

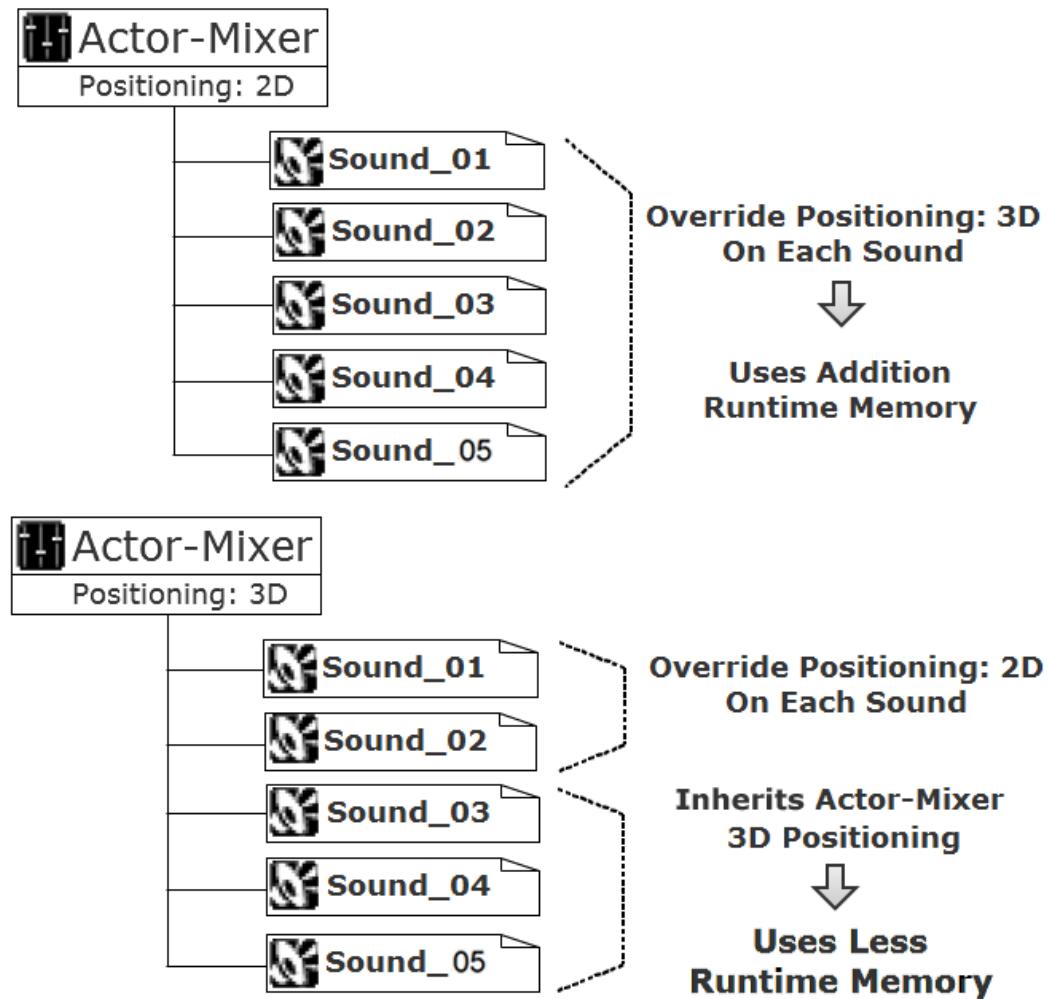
The Actor-Mixer is the ultimate memory and CPU saver because some of the Actor-Mixer's properties, such as positioning and RTPCs, are shared by all of its child objects. When considering how to organize different sound objects, think about grouping within Actor-Mixers to:

- Share property settings so they are processed only once.
- Limit overrides to avoid processing the overrides for each object.

To optimize memory usage, consider grouping objects into Actor-Mixers to share the following properties:

- Positioning
- RTPCs
- States
- Randomizers

Let's say you have an Actor-Mixer containing 10 sounds and you want to set the sound positioning to 3D. You could set the sounds individually to 3D by using the override parent option for each sound. However, doing it this way uses 10 times more memory at run time than if you had set the Actor-Mixer positioning properties to 3D. Now if you wanted some of the sounds to be 2D, you would still be optimizing memory if you set the Actor-Mixer's positioning to 3D game-defined. In this case you would override the Actor-Mixer and apply 2D to the specific sounds because 2D sounds do not require additional memory.



Illustrating the memory saving benefits of hierarchy property inheritance

While the Actor-Mixer is usually your best choice, in certain situations, you can decide to apply properties in containers to optimize memory consumption. If you are only applying positioning to specific objects within a container, for example, footstep sounds in a random container, you could save memory by applying the positioning properties to the container and not to the parent Actor-Mixer. If, however, you want all the objects in the structure to share the positioning properties, you would apply these at the Actor-Mixer level.

## Setting the Audio Channel Configuration

### System Default Channel Configuration

- From the menu bar, click **Audio > System Default Channel Configuration**
- By default, Wwise uses the speaker setup configuration from the Windows Control Panel. Select this option to choose value selected in the Windows Control Panel.

### Stereo Channel Configuration (Speakers)

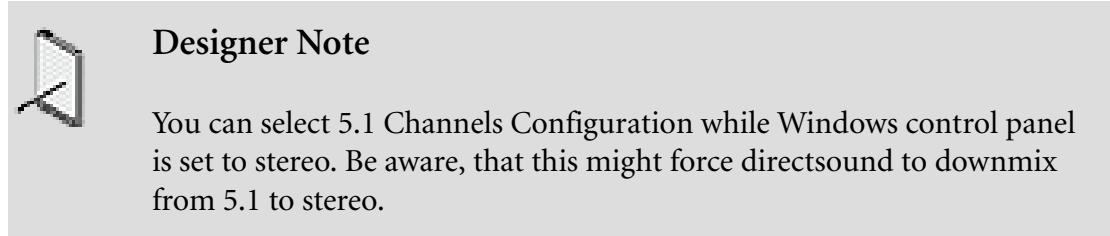
- From the menu bar, click **Audio > Stereo Channel Configuration (Speakers)**
- For more information on panning rules (speakers, headphones), refer to [Speakers vs Headphones Panning Rules](#) in the next section.

### Stereo Channel Configuration (Headphones)

- From the menu bar, click **Audio > Stereo Channel Configuration (Headphones)**.
- For more information on panning rules (speakers, headphones), refer to [Speakers vs Headphones Panning Rules](#) in the next section.

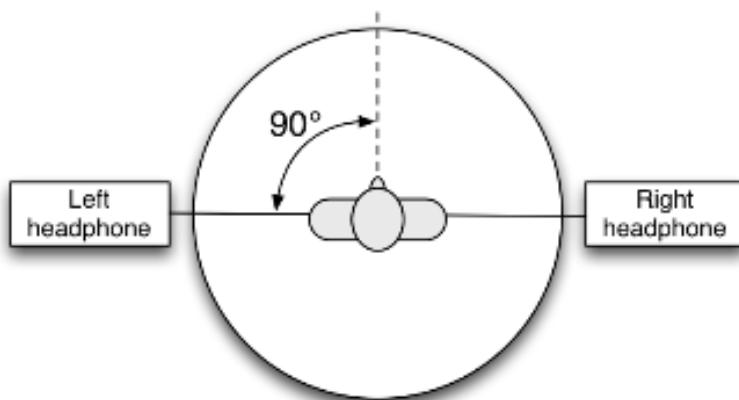
### 5.1 Channel Configuration

- From the menu bar, click **Audio > 5.1 Channel Configuration**

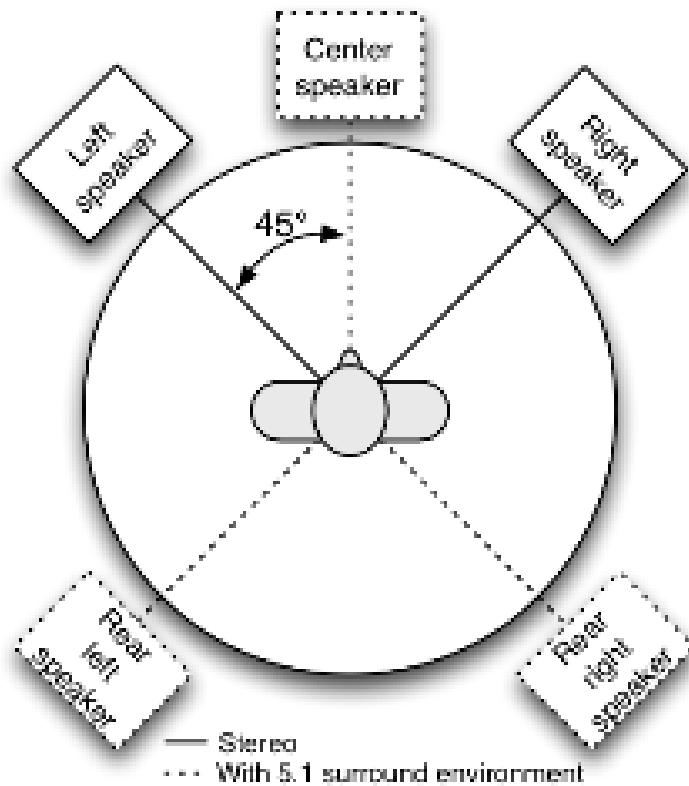


### Speakers vs Headphones Panning Rules

In Wwise there are two different panning rules: Headphones and Speakers. By default, all platforms use the speaker panning rule with the exception of the handheld consoles that use the headphone panning rule. The difference in between the two modes is subtle but helps to provide a realistic and accurate audio experience depending on your listening set-up. This setting can be auditioned in Wwise but also can also be set in the game at run-time.



Headphone Panning Rule



Loudspeaker Panning Rule

To audition the two modes:

- From the menu bar, click **Audio > Stereo Channel Configuration (Speakers)**
- From the menu bar, click **Audio > Stereo Channel Configuration (Headphones)**



### Programmer Note

You can also set the panning rule in the game, please refer to AK::SoundEngine::SetPanningRule in the sound engine documentation.

## Creating Simulations with the Soundcaster

At any point in the development process you might find it helpful to build a simulation using the Wwise objects and events you have been working on. To accomplish this, Wwise has created a simulation environment called the Soundcaster where you can play back sound, music, and motion structures asynchronously. This means you can control what plays and when. This can be very handy for testing events, mixing in real time, and so on. The Soundcaster is a powerful tool that can be used for:

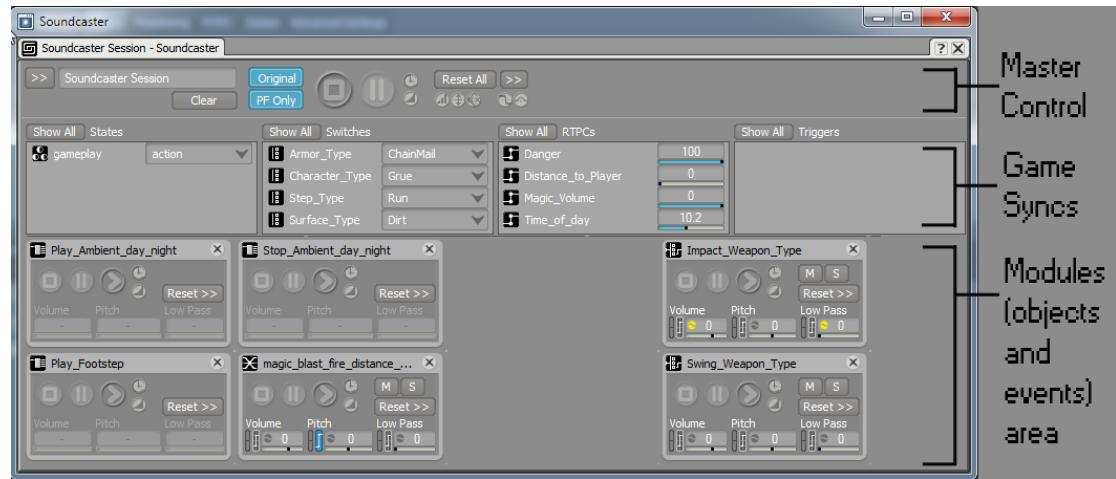
- Prototyping and experimenting.
- Developing a proof of concept.
- Auditioning sounds, music, and motion simultaneously.

Since you can simulate in Wwise alone or by remotely connecting to a game, the Soundcaster provides you with many different uses for your simulation. For any simulation you can choose to:

- Selectively audition the audio or motion for each platform.
- Audition pre-converted audio files.
- Profile your audio and/or motion as it is playing back.
- Mix and test your audio and motion in Wwise by manually simulating the game action.
- Profile your audio and motion in game and in Wwise.
- Experiment with the sounds, music, and motion objects associated with a game object.
- Mix and test your sounds, music, and motion in game.

The Soundcaster consists of three areas:

- Master controls
- Game syncs
- Objects and events



Soundcaster showing the three main areas of simulation interaction

Using the different areas of the Soundcaster, you can work with its mixing and playback functionalities when you build your simulation.

For more information on using the Soundcaster:

[Wwise Help > Finishing Your Project > Creating Simulations](#)

[Video Tutorial - Wwise Quick Tip - Transport and Soundcaster](#)

## Project Settings

There are several aspects of the Project Settings that define the default behavior for sound throughout the project.

### Project Settings - General Tab

Through the General tab you can set the volume threshold and the maximum voice instances for each platform.

Platform	Base Platform	Volume Threshold	Max Voice Instances
3DS	Nintendo 3DS™	-96.3	256
Android	Android™	-96.3	256
iOS	iOS	-96.3	256
Linux	Linux	-96.3	256
Mac	Mac®	-96.3	256
PS3	PlayStation®3	-96.3	256
PS4	PlayStation®4	-96.3	256
VitaHW	VitaHW	-96.3	256

Setting platform volume threshold and voice instances

#### Volume Threshold:

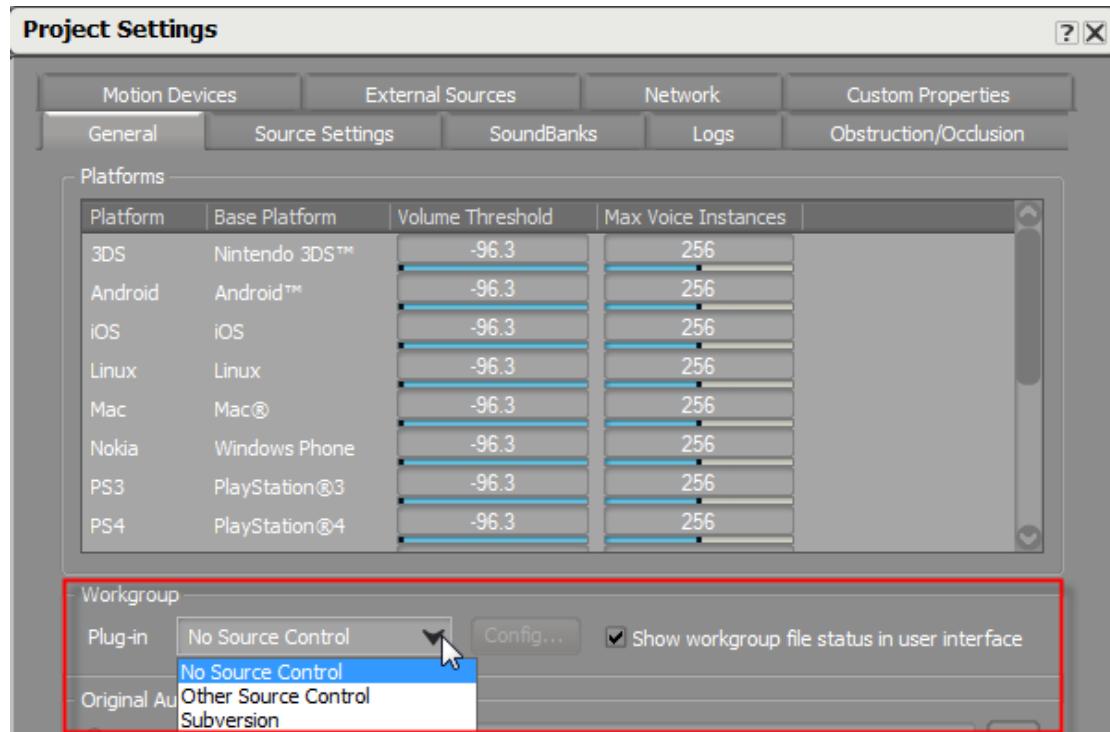
The volume threshold is the default volume level below which voices are managed by behaviors defined in the Advanced Settings tab of the Property Editor, essentially determining the volume when voices will either be killed or sent to virtual voice.

#### Max Voice Instances:

This value defines the maximum number of simultaneous voices that can be active at the same time in the entire project. Virtual voices do not count as active voices. Over this limit, voices with the lowest priority will adopt their virtual behavior. If priorities are equal, older sounds will be considered having a higher priority.

## Workgroup Plug-in Configuration

The Workgroup plug-in allows you to specify a source control solution and specify the settings. In your project development environment, you may already be using a source control system, such as Perforce or Subversion, to effectively manage your assets and other types of project files.



### Specifying a source control plug-in for managing project files from Wwise

The following files within the project can be managed by a source control system:

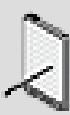
- **Wwise project file** - the WPROJ file.
- **Wwise work units** - the WWU files, including the Default work units.
- **Originals folder** - the folder that contains the original sound files that were imported into Wwise.
- **Generated SoundBanks** - the SoundBank files generated for each platform and language.



#### Designer Note

The .cache folder located in the project directory is a local working folder for Wwise. Its contents should not be added to a source control system because it may cause unexpected behaviors in Wwise.

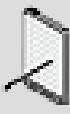
Throughout the development of your game, you can view the status of your project file (WPROJ), work unit files (WWU), and audio files in the File Manager. If you are using Perforce, Subversion, or another Workgroup plug-in, you will be able to perform source control functions directly in Wwise. All Wwise project files, including the individual work units, are XML-based, which means you can use your source control system to easily manage these files as well.



### Designer Note

For more information about using a Workgroup plug-in in Wwise, refer to "Managing Project Files Using a Workgroup Plug-in" in the help documentation.

When you are working as part of a workgroup and are using a source control system to manage the files in your project, you should always be aware that others are working on the same project and that there may be merge conflicts that need to be resolved. This is why it is important to sync and merge your work often and to communicate frequently with your team members about the work you are doing.



### Designer Note

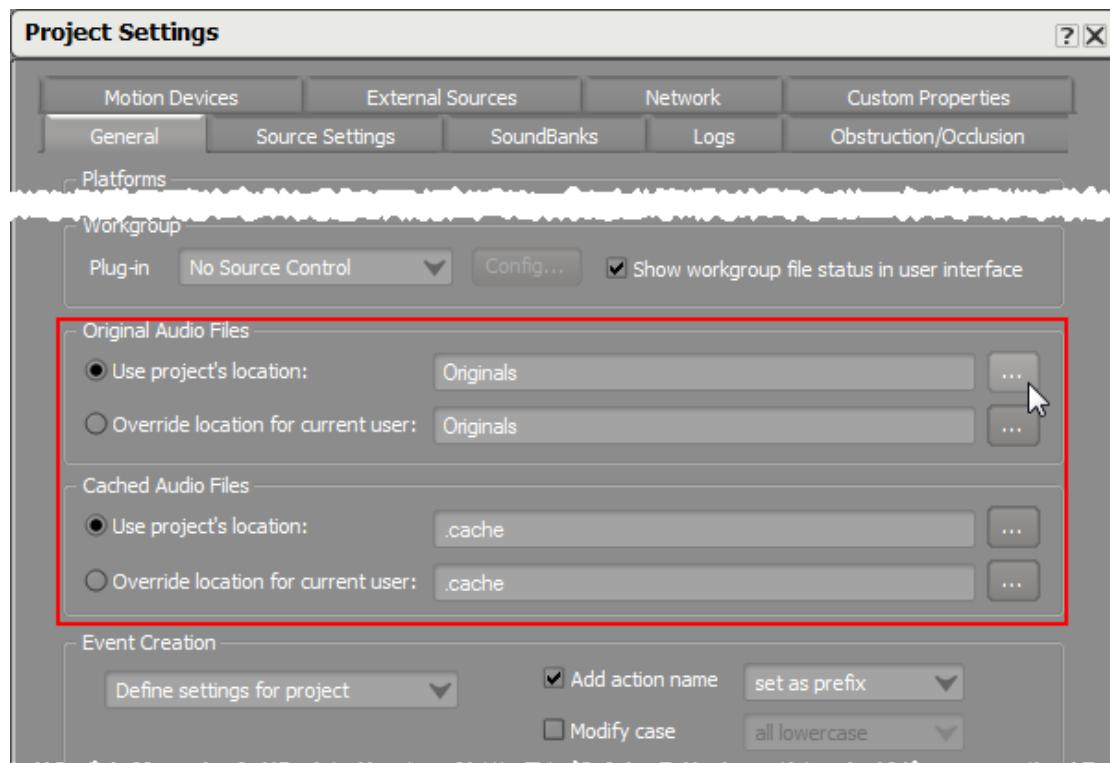
For a complete list of best practices, refer to "Workgroup Tips & Best Practices" in the help documentation.

**For more information on workgroups:**

[Video Tutorial - Workgroup management in Wwise using Perforce](#)

## Audio File Locations

A custom location for original and cached audio files can be set for the project as well as an override location for the current user.

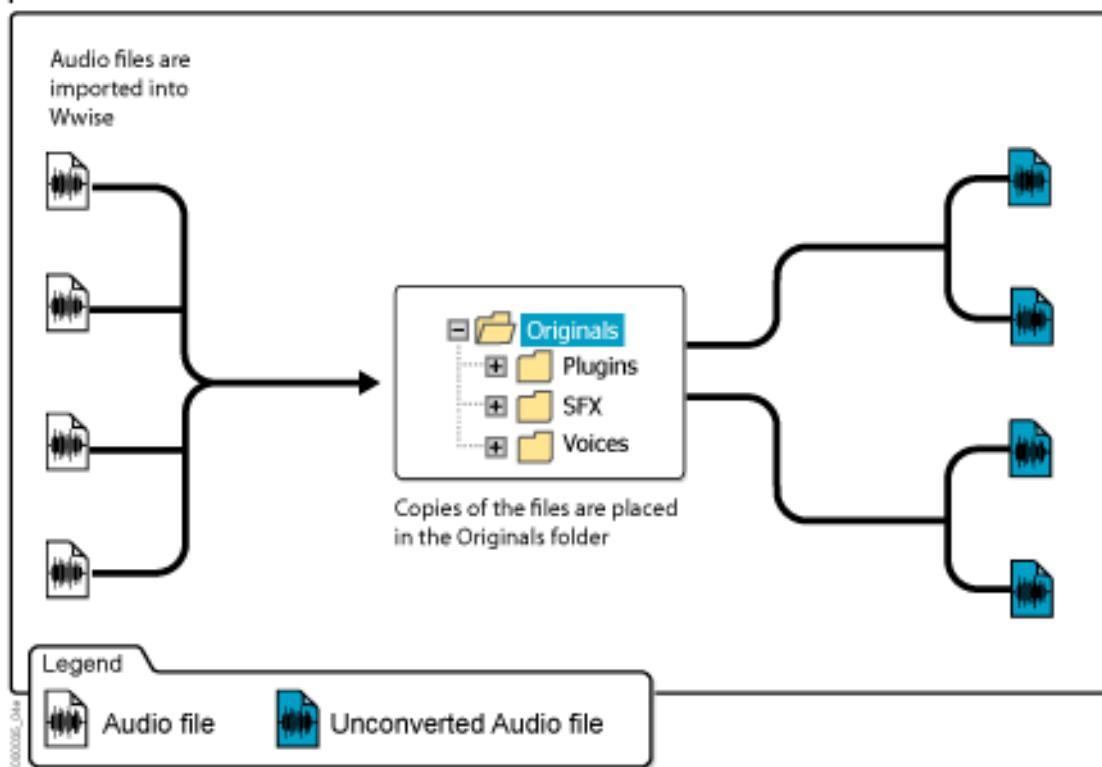


### Defining locations for original audio files used by the project

Overriding the user location can be useful in the following types of situations:

- You temporarily do not have access to the Originals/Cached folder.
- You do not have permission to alter the contents of the Originals/Cached folder.
- You need to create a temporary location for the Originals/Cached folder without changing the location of the project's Originals/Cached folder.

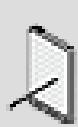
When you import a file into the project, a copy of the original asset is stored in the Originals project folder. Since these assets are usually shared by several people on the team, this folder can be located anywhere on the network and can easily be managed by a source control system.



The asset versions created for the various game platforms are stored locally in each user's project cache folder. This allows each user to manage their own platform versions and to experiment with different conversion settings.

The project's cache folder contains intermediate data generated by Wwise during audio file conversion and SoundBank generation. Its location is initially set to '.cache/' inside the project's directory when a new project is created.

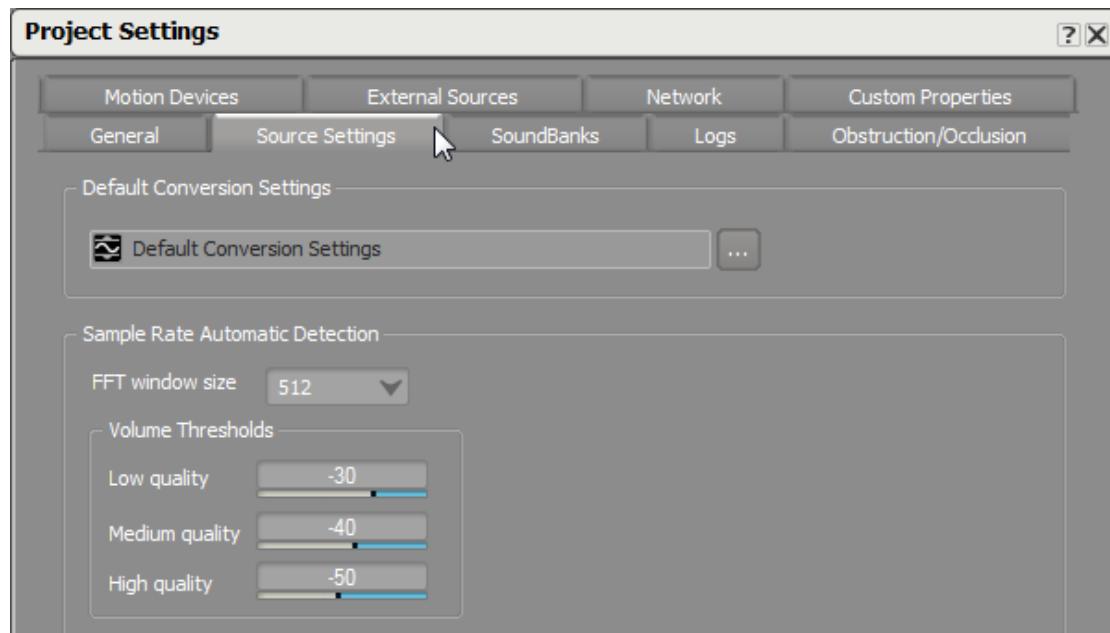
Both types of files can be played back in Wwise. The original pre-converted files will be played back whenever the Original control is activated in the Transport Control or Soundcaster. When the Original control is not active, Wwise will attempt to play the converted file, if one exists.



### Designer Note

There are some restrictions to playing converted files in Wwise. When you convert an audio file for a particular platform, it is converted to meet the specific hardware requirements of that platform. As a result, you may not be able to play back these converted files in Wwise when a platform, other than Windows, is selected.

## Default Conversion Settings



### Default Conversion Settings

The Default Conversion Settings in Project Settings specify the name of the ShareSet that will be used as the project's default. When a new object is created, the Default Conversion Settings are used only if the new object is a top-level parent object. If the new object is a child of another object, it will inherit the conversion settings assigned to the parent.

### Defining the Sample Rate Automatic Detection Settings

As part of the Project Settings, you can define the size of the Hanning window used by the FFT algorithm as well as the threshold levels for three different quality settings: High, Medium, and Low. These threshold settings are used when you select Auto High, Auto Medium, or Auto Low as the sample rate conversion method.

For more information about conversion settings:

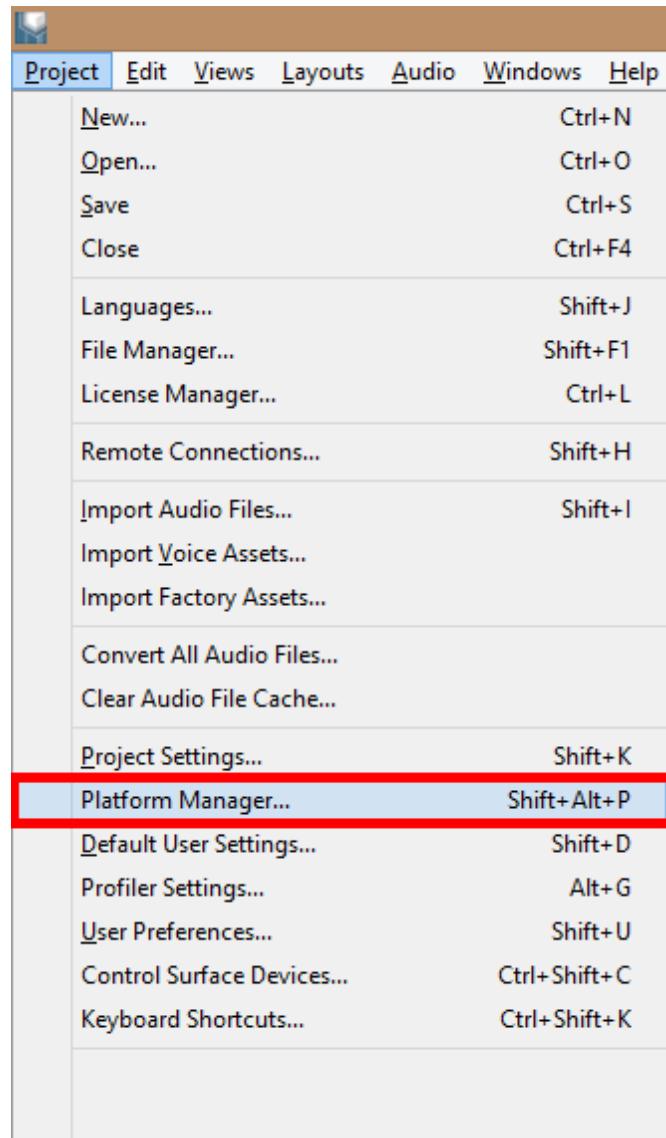
[Wwise Help > Setting Up Your Projects > Working with Projects > Defining your Project Settings > Defining the Conversion Settings for Your Project](#)

## Platform Manager

The Platform Manager can be used to customize existing platforms and create new ones that can be used to non-destructively tailor project settings depending on the selected platform. The ability to have multiple platforms share the same Base Platform gives greater control over situations where a project requires distinct configurations using the same Base Platform.

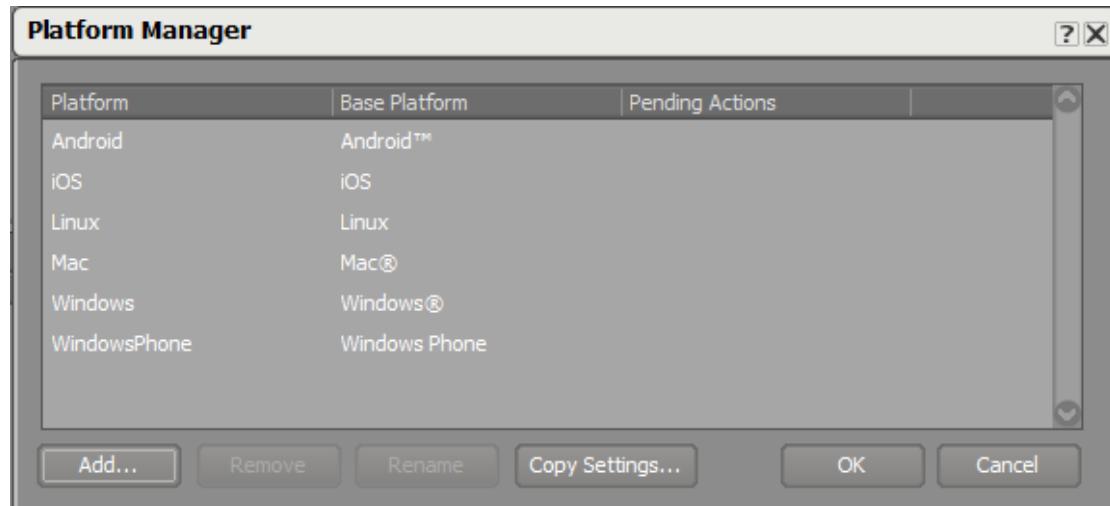
The uses of customizable platforms will vary depending on the development environment. It's not uncommon in today's games-as-a-live-service to have multiple configurations in a constant state of iteration. The potential to have a development, staging, and live platform all building to the same base platform allows development to work consistently within the same project without the need for unique workflows for each branch of development.

The Platform Manager can be accessed through the Project menu.



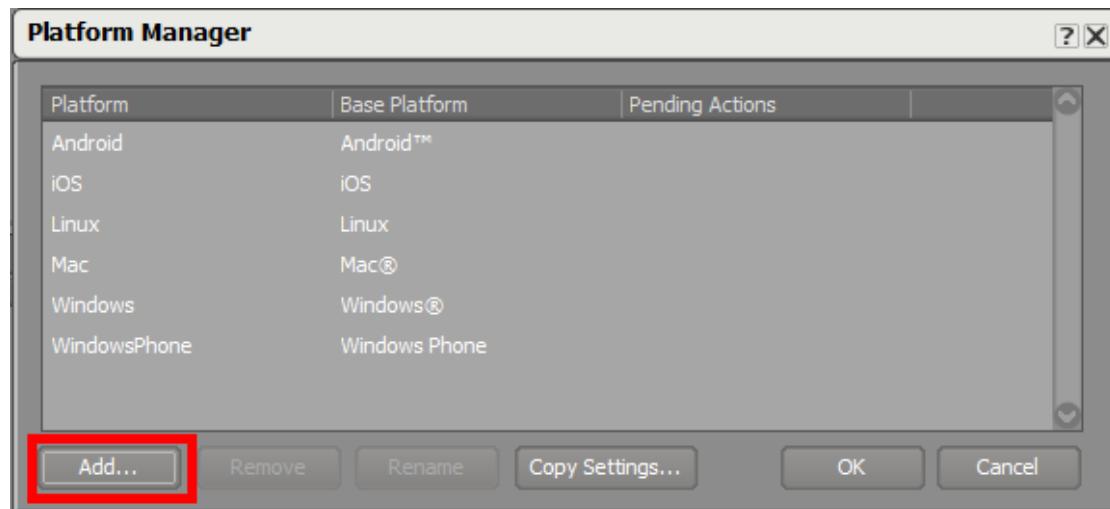
### Selecting the Platform Manager in the Project menu

The Platform Manager shows all of the potential platforms available in the project and their Base Platform association.



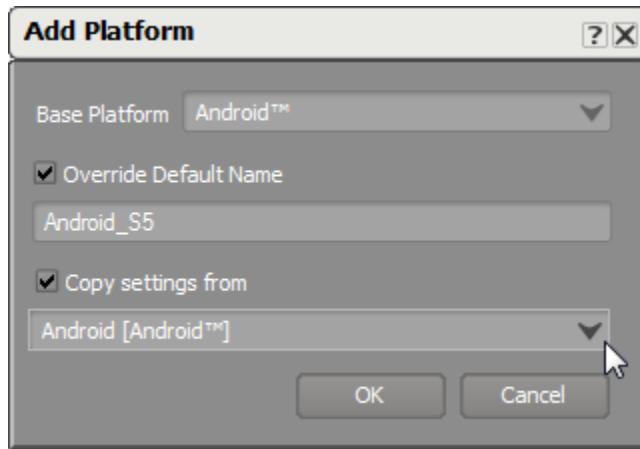
The Platform Manager showing available platforms and their associated Base Platform

New platforms can be added from the Platform Manager by clicking the Add... button.



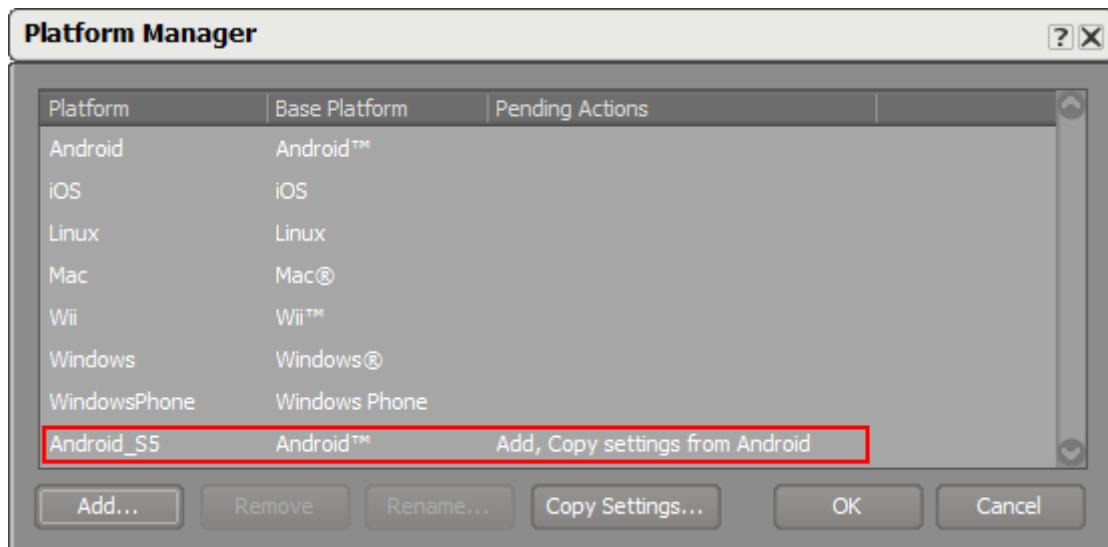
Adding new platforms by clicking Add...

After specifying the Base Platform, Wwise enters the default associated name. Although not generally recommended because of third-party dependencies, you can optionally override the default name and enter a unique one for the new platform that will be used for the project and copy settings from an existing platform.



**Adding a unique platform with a distinct name from the default for a particular Base Platform**

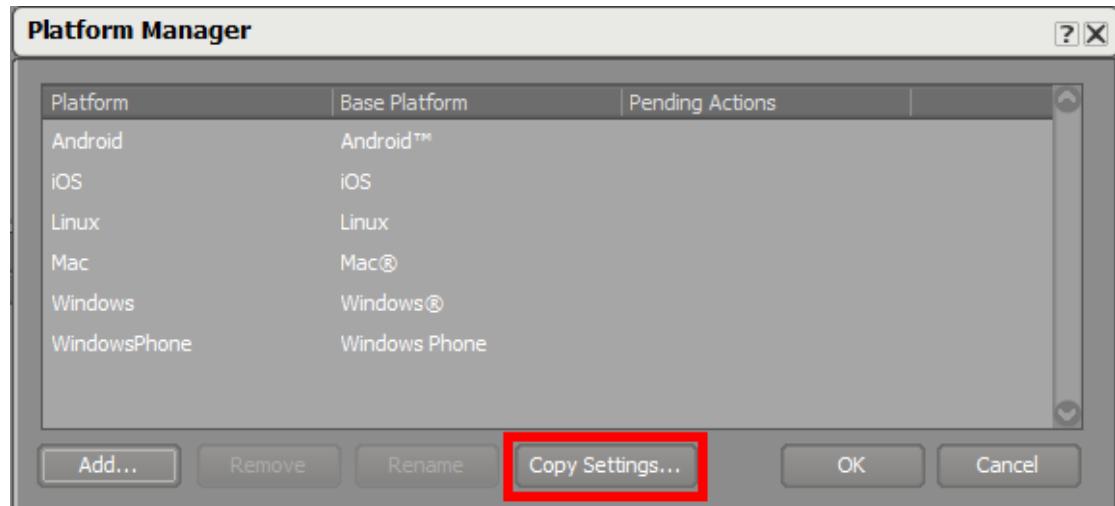
Once new platform(s) have been added, the Platform Manager will show the pending actions that are required to establish the new Platforms as part of the Project.



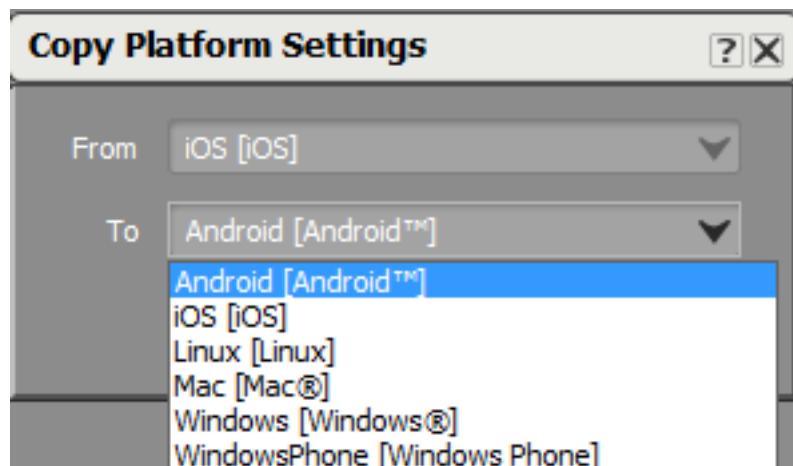
**Platform Manager showing pending actions for the addition of the new platform**

Clicking OK will confirm the addition of the new platform to the project. To complete the operation, the project must be reloaded, which will clear the clipboard contents and undo history.

Once a platform has been established, settings can be copied between existing platforms using the Copy Settings button.



Copying settings between existing platforms



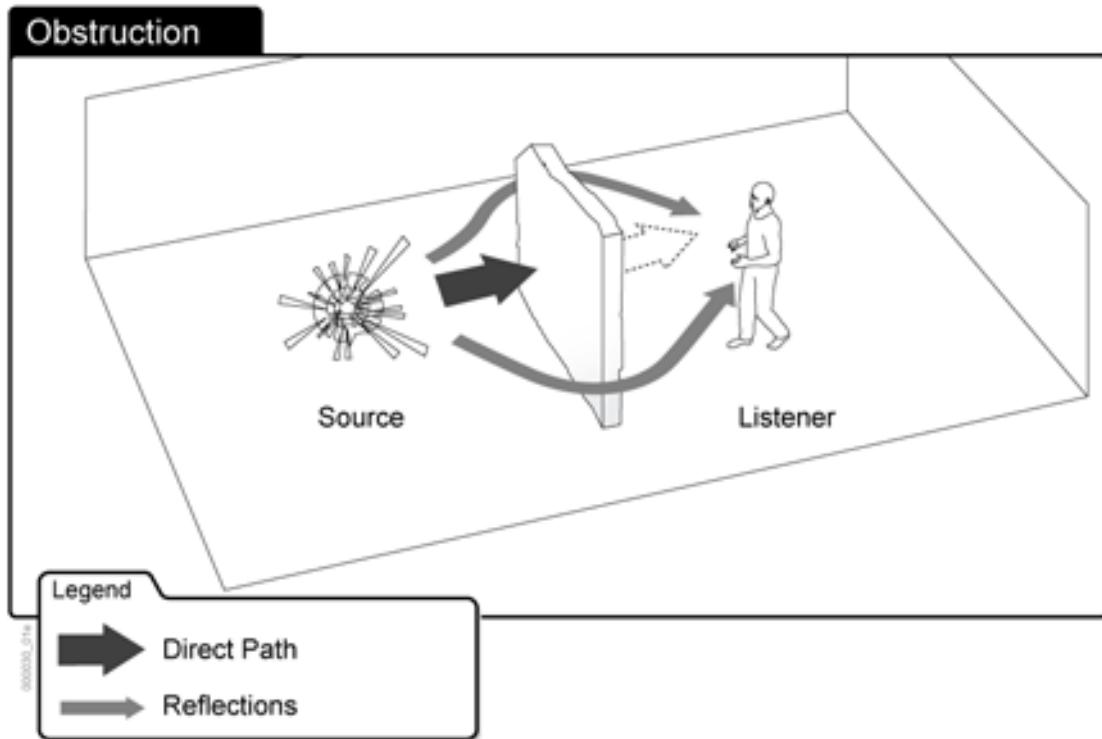
Copying Platform settings from one platform to another

The flexibility of the Platform Manager puts the decision of how to structure the build pipeline in the hands of content creators and adds value to the audio development pipeline.

## Obstruction and Occlusion

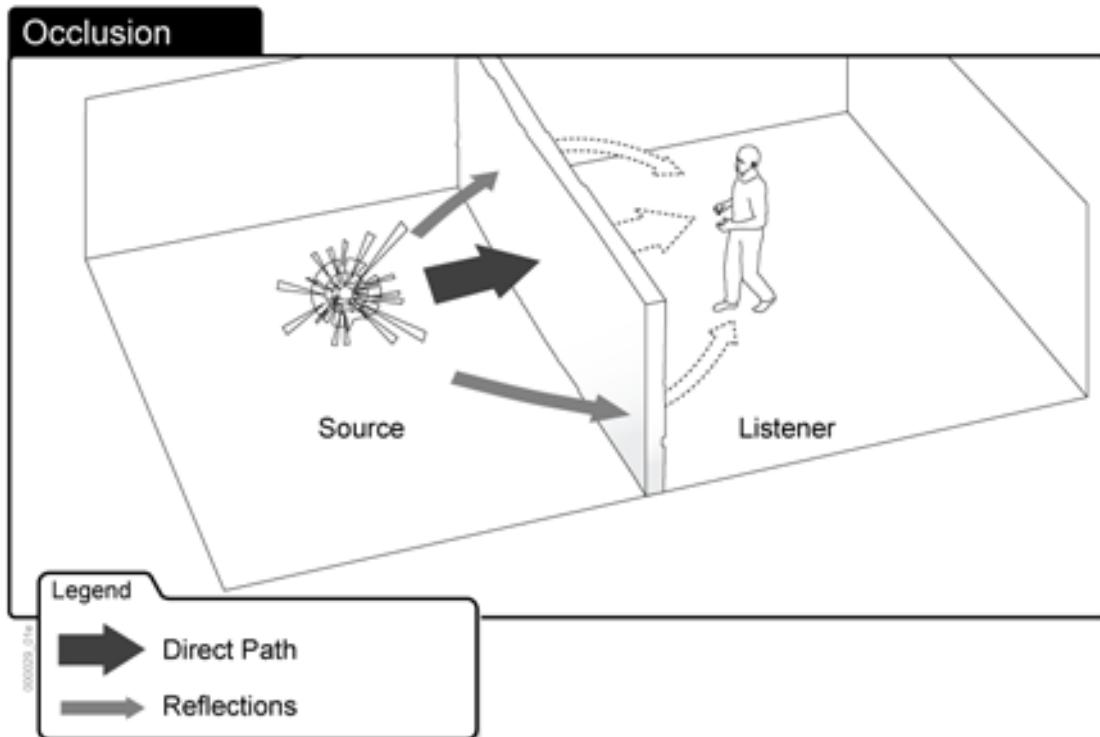
A typical condition of most game environments is having a game object become either obstructed by another object (such as a wall or beam) or occluded in a room where the listener can only hear a few muffled sounds leaking through the walls.

The following diagrams illustrate examples of occlusion and obstruction.



Example of an Obstruction

Obstruction can be modeled by applying a volume control and/or a Low Pass Filter (LPF) affecting only the direct path of the signal. The environmental reflections are unaffected.



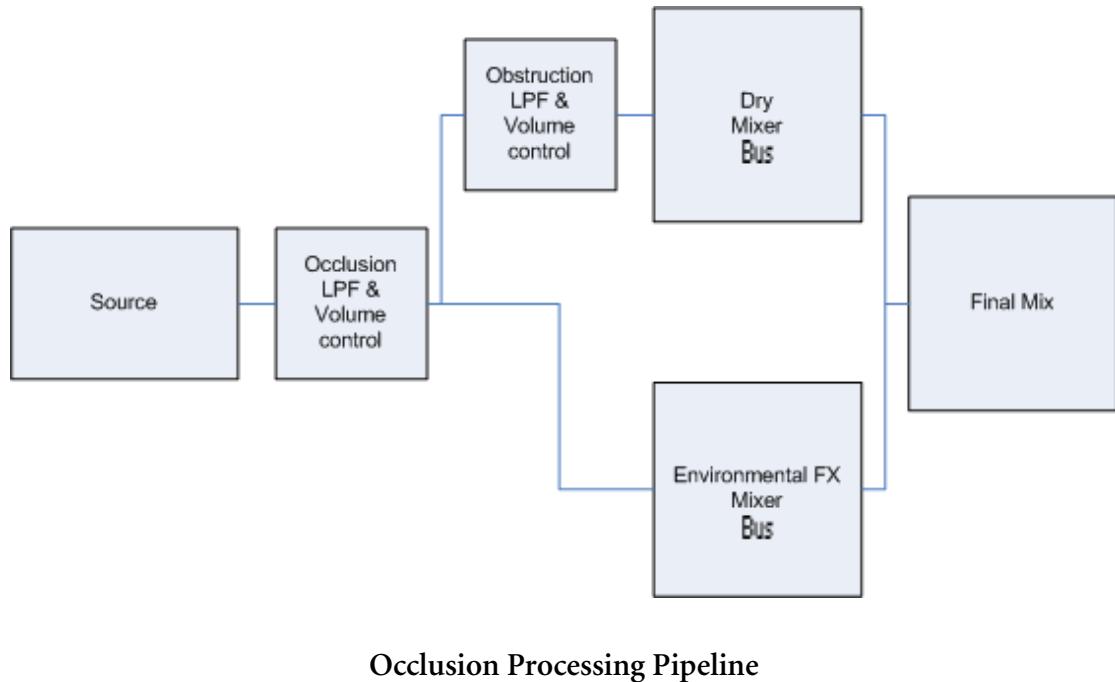
### Example of an Occlusion

Occlusion can be modeled by applying a volume control and/or an LPF affecting both the direct path and the environmental reflections of the signal.

#### Designer Note

Obstructions and occlusions can happen simultaneously and are modeled with the direct path being affected by both the obstruction and occlusion value. However, the reflection path is affected by the occlusion value only.

Here is a diagram of the obstruction/occlusion processing pipeline inside the sound engine:



Occlusion Processing Pipeline

### Setting Obstruction and Occlusion

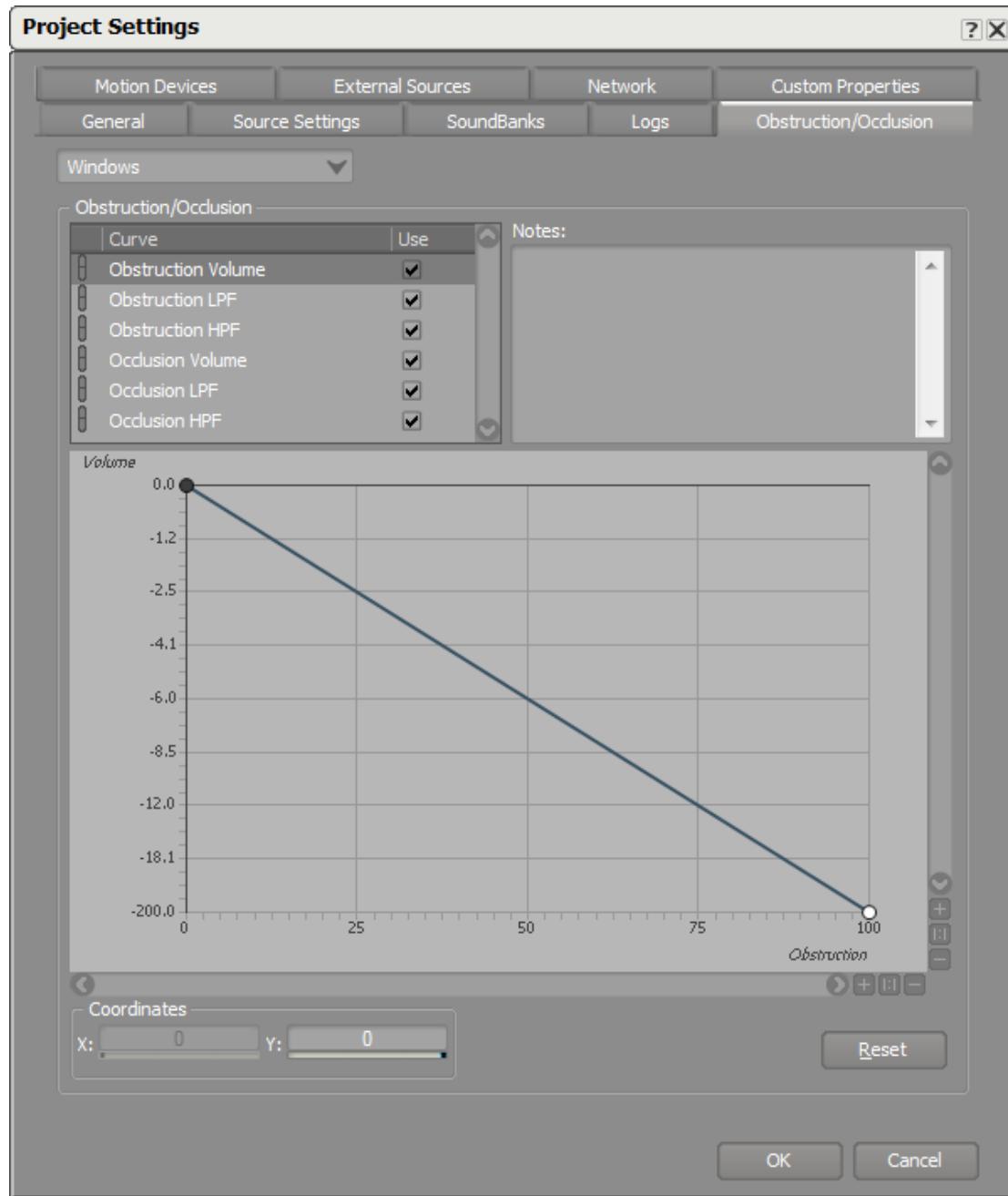
The game engine is responsible for determining the obstruction and occlusion values, which it does using the position of the objects and listeners in relation to the game's geometry.



#### Programmer Note

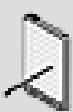
The obstruction and occlusion values for each game object affecting each listener must be passed down to the sound engine from the game programmatically.

In the Obstruction/Occlusion tab of the Project Settings dialog box, you can enable and define the default volume and Low Pass Filter (LPF) curves for obstruction and occlusion for each active platform in the project. Sound designers can also enable or disable the usage of any curve to best suit their performance and realism needs.



### Customizing Obstruction and Occlusion Curves in the Project Settings

Using the curve of this snapshot, setting an obstruction value of `1.0f` (100%) will produce a volume change of `-50 dB` on the source object.



### Designer Note

It is a good practice to always define curves in a linear fashion to minimize CPU and memory usage in the project. Keep curves as simple as possible to begin with and customize only as needed.

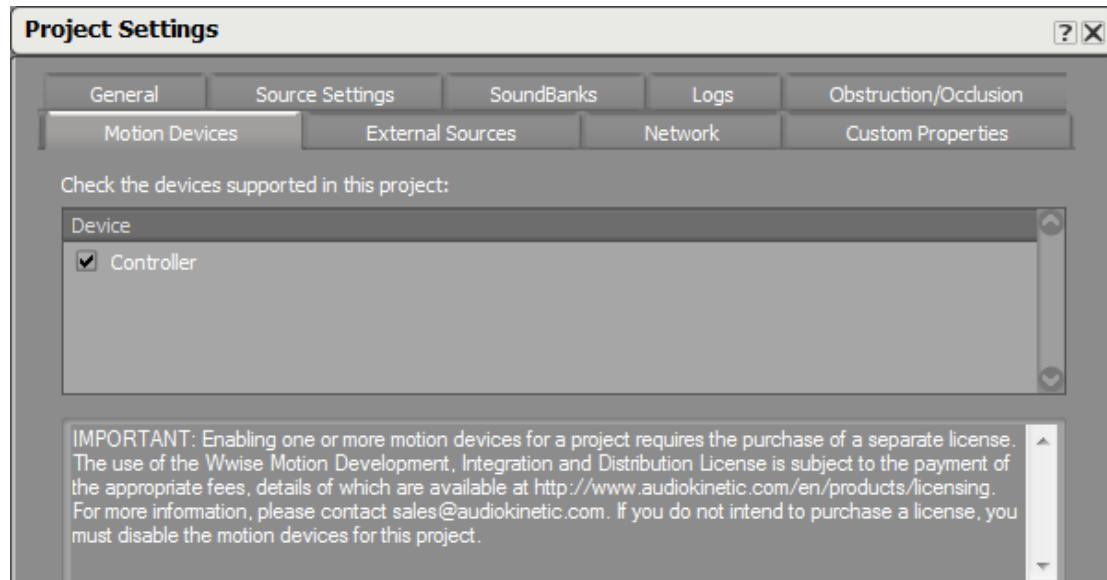
## Motion

It begins with an earth shaking rumble as the dragon emerges from an underground lair, followed quickly by the pounding of gargantuan footsteps racing towards the final showdown. With the expectation of a soundtrack to match the intensity of this scenario, it seems a perfect fit for using sound to drive controller vibration.

Wwise offers a complete pipeline for creating and integrating motion in your game. By implementing a comprehensive pipeline solution for motion, similar to the one that exists for building audio, Wwise allows you to:

- Create sophisticated and realistic motion effects with a very short learning curve.
- Integrate motion easily into a game without significantly affecting the performance of the game or sound engine.
- Use the same features as audio to build and integrate motion.
- Create motion effects for the same type of device on various platforms without additional work.
- Add or remove the motion component easily based on your game requirements.

Before creating motion effects in the project, the types of motion devices need to be enabled within the Project Settings in order to create motion effects.



### Enabling motion devices in the Project Settings

After a motion device is enabled, you can do the following in the project:

- Create sources for Motion FX objects using media files or motion generators.
- Include motion data for the selected device within SoundBanks.

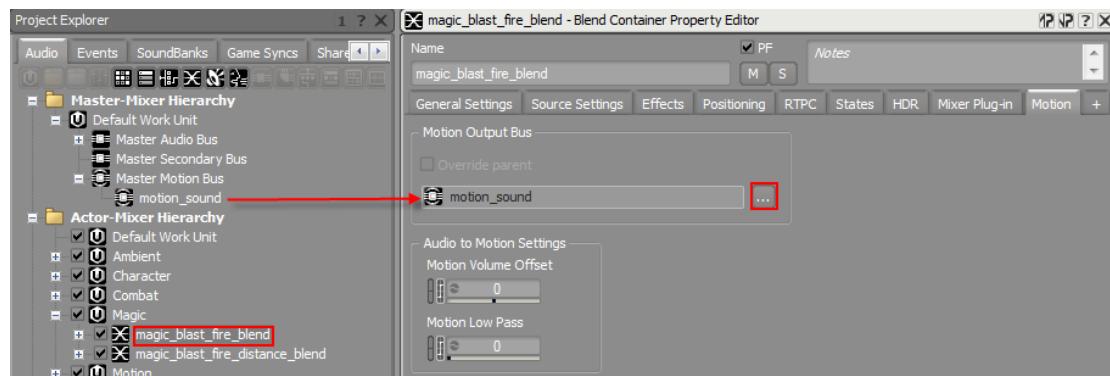
At a very basic level, motion data is generated in Wwise from a source. This source can either be an existing media file, a new media file, or a Motion generator plug-in. After you have decided which motion devices will be supported by the game, you must then decide which method you will use to generate motion.

In Wwise, there are two different methods for creating a motion source:

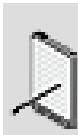
- Using an existing audio signal.
- Using a Motion FX object.

### Generating a Motion Source from an Existing Audio Signal

When you convert an existing audio signal into a motion source, the audio signal is split in two at run-time after both RTPCs and effects have been applied. The split is done so as not to affect the original sound. Since audio has a much larger spectrum than motion, the higher frequencies are filtered out using a low pass filter. The signal is then re-sampled using a much lower sample rate to create the motion source.



Enabling routing to a Motion Bus from an existing sound object



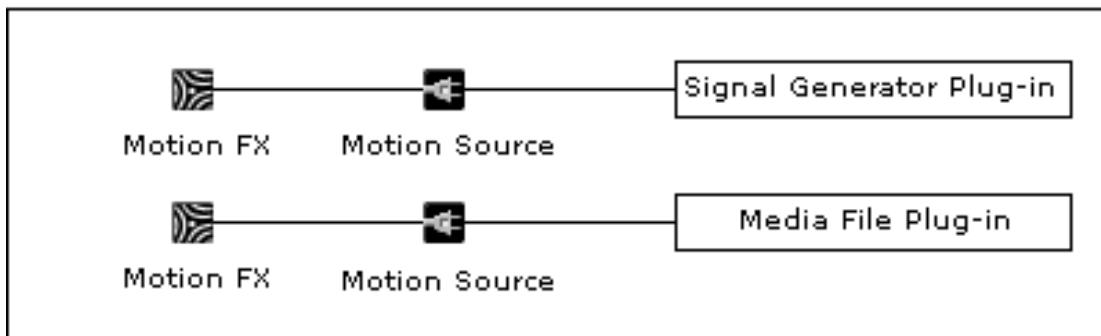
#### Designer Note

The LFE channel is ignored when generating motion from an existing audio source.

Since the motion source is generated from an existing audio source, the motion is tied to the audio playback in game. This means that the motion source does not require a separate event to be triggered in game. It also means that the motion source is affected by the same properties, behaviors, game syncs, and so on, as the audio object.

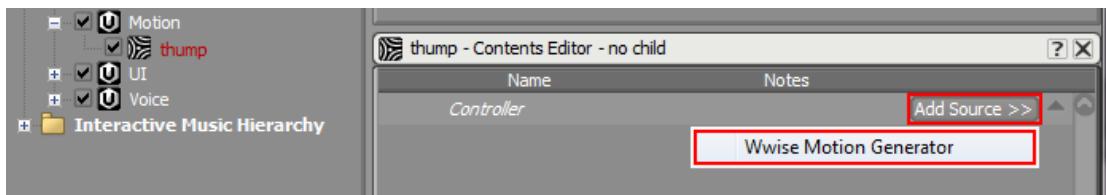
## Generating Motion Using a Motion FX Object

Another way to generate motion is by creating special Wwise objects, called Motion FX objects. These objects, like sound objects, contain a source. The motion source can be created from a Motion Generator plug-in.



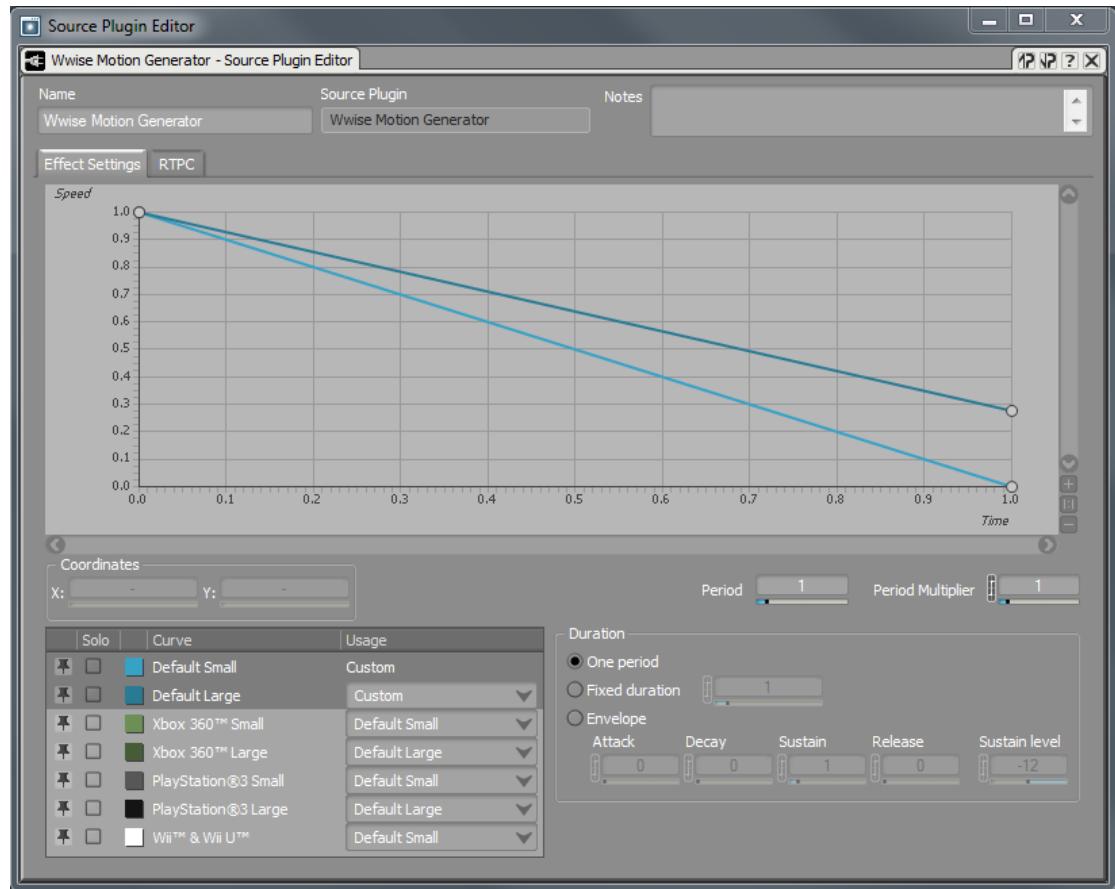
When using Motion FX objects, you can build sophisticated motion structures, using containers and Actor-Mixers, to define the properties and behaviors of the motion effects. Since these motion effects are not necessarily tied to the audio in the game, they can be triggered at any point in game by their own events.

After enabling the use of Motion FX in the Project Settings, you can begin adding Motion FX objects with a Motion Generator source.



Adding a Motion Generator source to a Motion FX object

The Motion Generator source plug-in gives access to the authoring of motion effects based on duration and speed, in addition to envelope properties. A default setting can be used to define small and large curves for Motion FX, which are inherited for all dual-motor controller types. These default settings can be overridden on a per controller basis in order to fine-tune the motion experience.



**Motion Generator source plug-in editor**

Of course, there will be situations where one method will be preferential over another. For a further discussion on the benefits of each of these methods, refer to [Creating Motion for Your Game](#) in the Wwise help documentation.

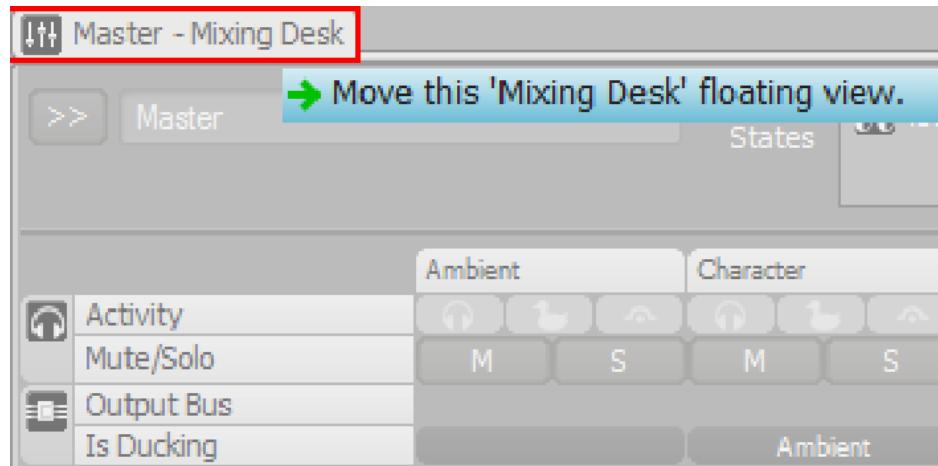
For additional information about Wwise Motion:

[Video Tutorial - Wwise Motion](#)

## Customizing Layouts

Multiple layout views are available from the menu bar to help you navigate different aspects of the Wwise project workflow. Layouts can be modified by adding or removing any of the views available to streamline your workflow.

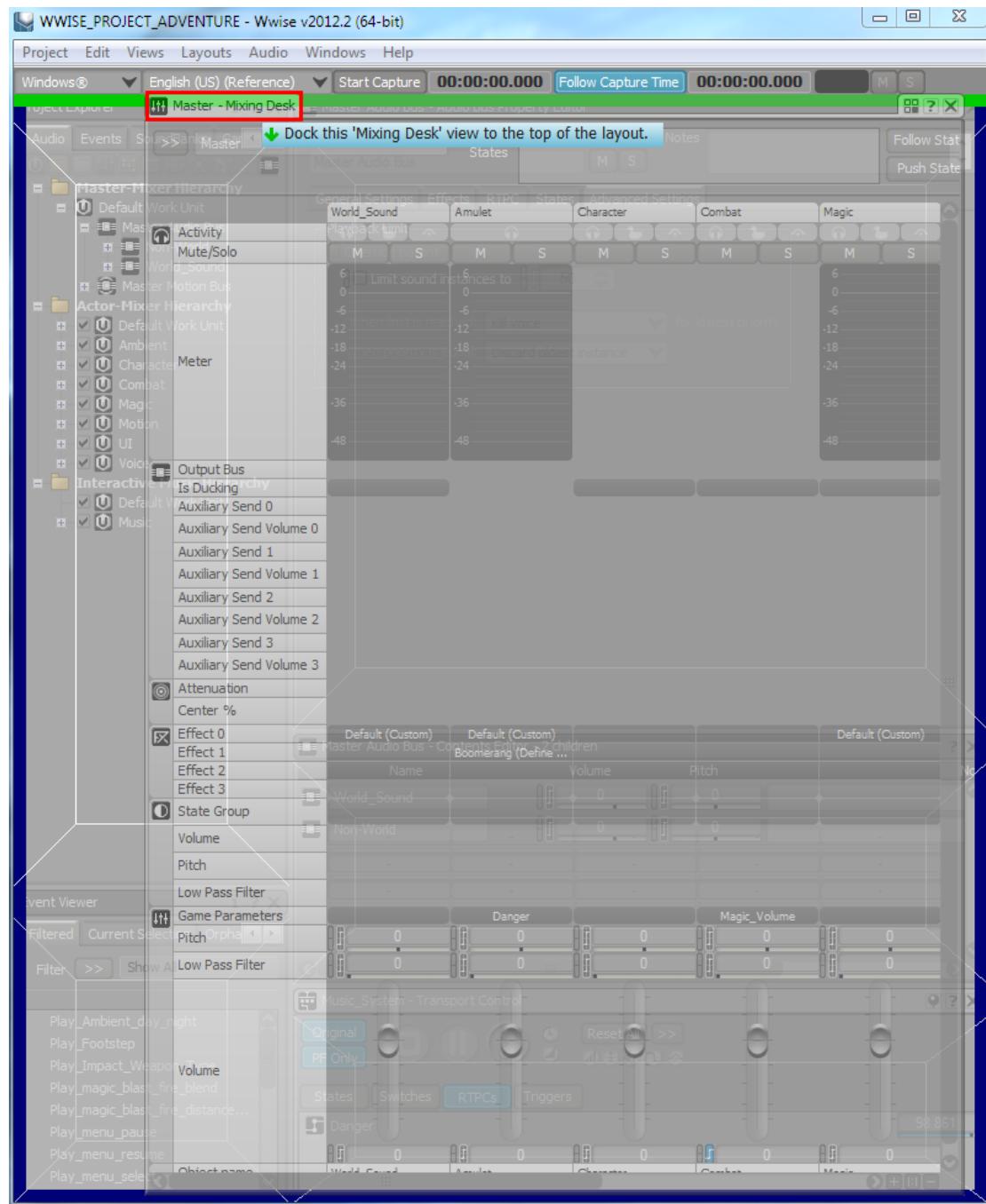
Clicking and dragging on the view's title bar will transform it to a floating view and show possible docking locations.



Clicking and dragging the view's title bar in order to move it

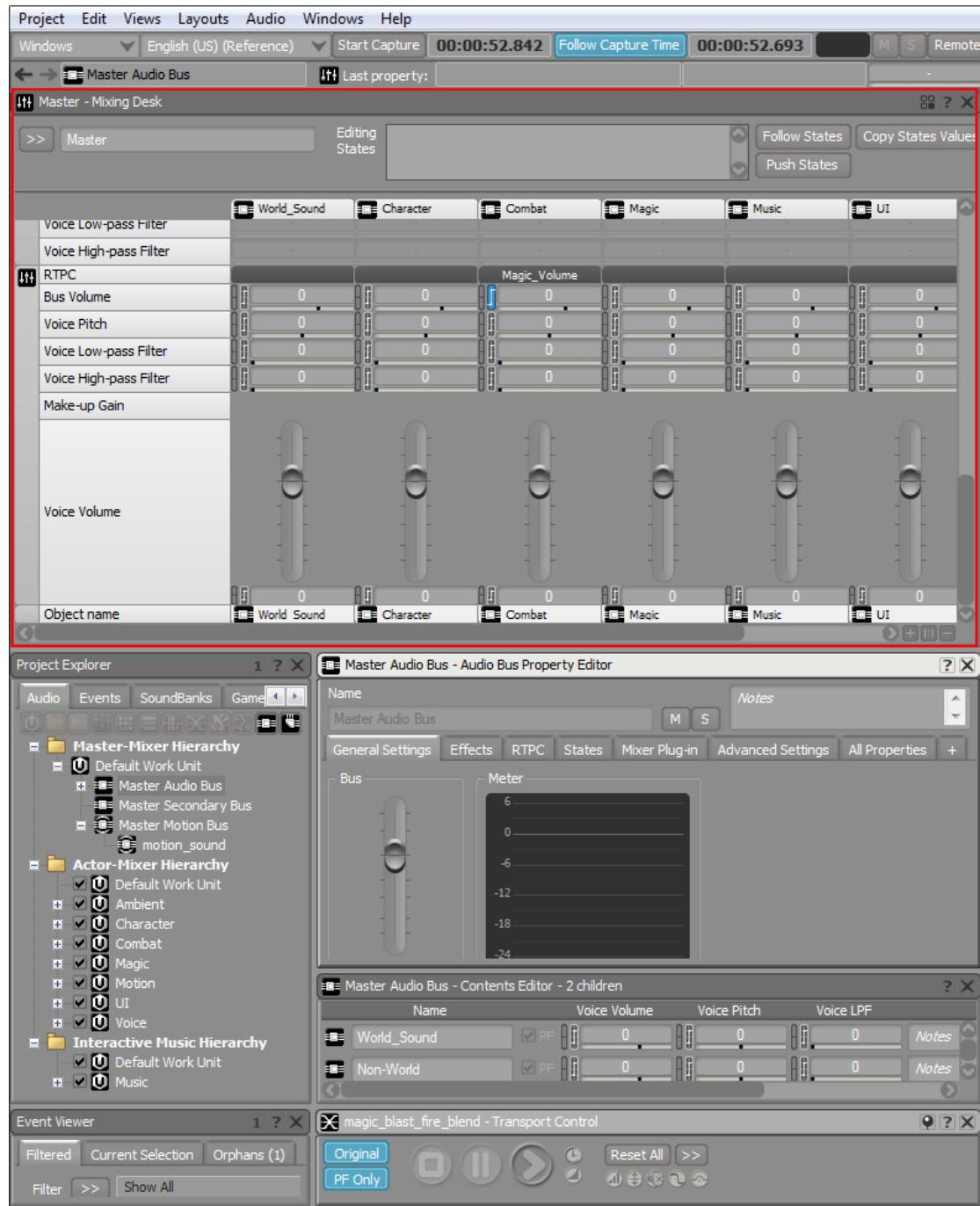
Views can be docked on the four sides of both the layout and each individual docked view.

## Layout Docking



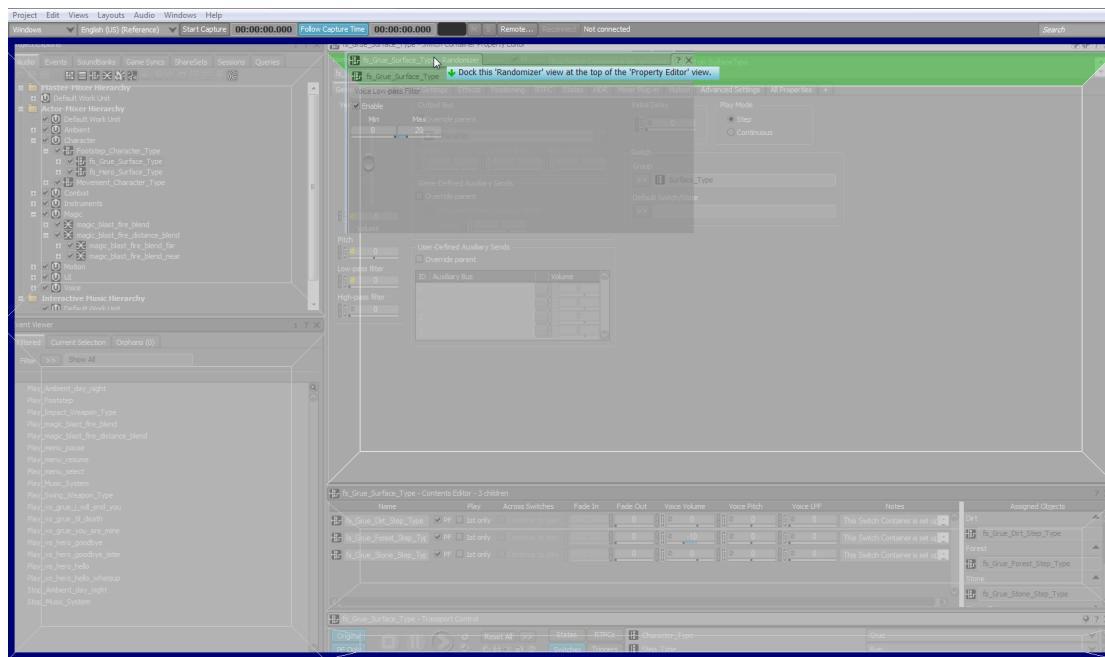
Docking locations in blue, along the border of the layout window, turn green when hovered over

## Getting Set Up for Adventure

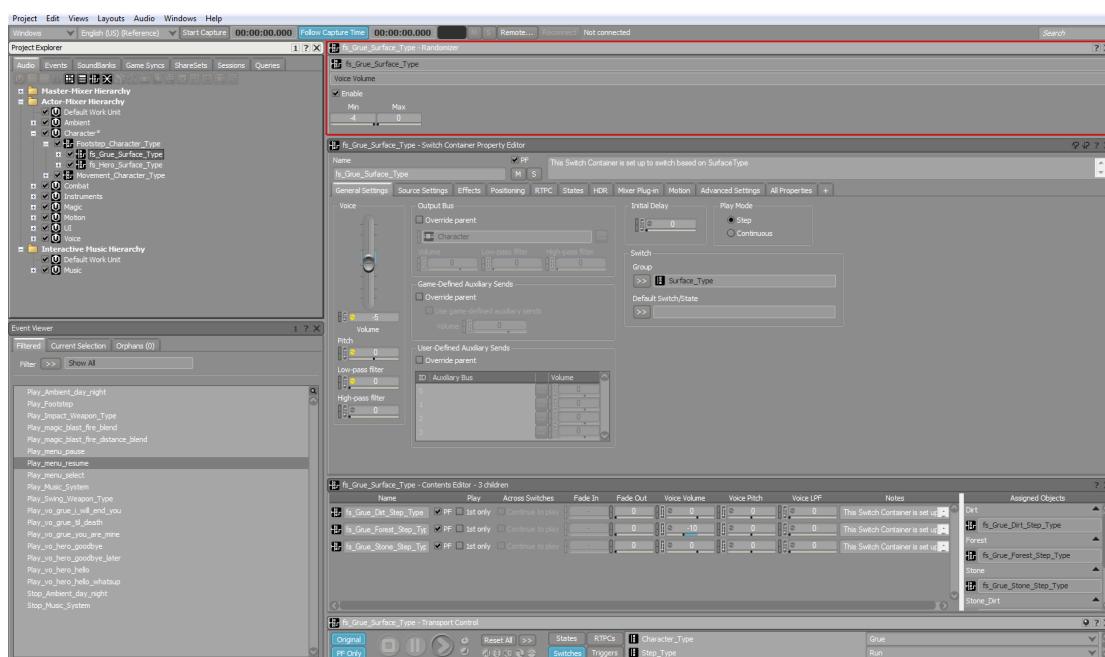


An example of the Mixing Desk view docked at the top of the layout

## View Docking

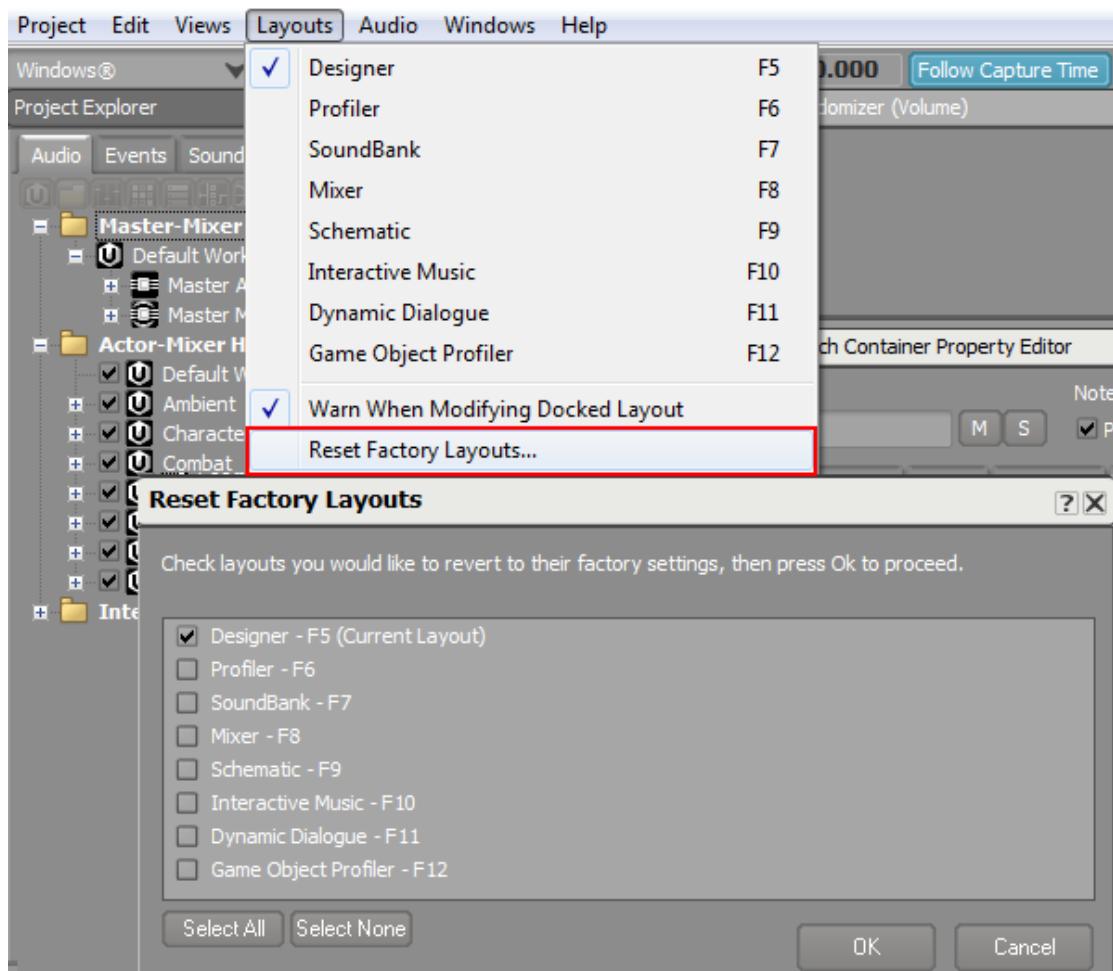


Docking locations on the four sides of each docked view turn green when hovered over



An example of the Randomizer view docked at the top of the Property Editor view

Layouts can be selectively reverted to the default settings at any time from the Layouts menu on the menu bar.



### Resetting the factory layout settings for the Designer layout

Customizing layouts to maximize screen real estate in a multi-monitor setup is a great way to put the views you need in place for a streamlined workflow.

**For more information about layouts:**

[Video Tutorial - Managing Layouts](#)

## External Sources

External Sources are a special type of audio source that you can put in a Sound object in Wwise. It indicates that the real sound data will be provided at runtime. This is very useful when managing a large amount of dialogue lines that would otherwise need a Sound and an Event for each, which would then need to be included in SoundBanks. It is also very useful if the dialogue lines are already managed through another system such as an AI-driven speech generator.

Depending on how you manage the dialogue in the project, there may be additional runtime memory savings because the External Source plug-in allows you to play dialogue without having to load many voice assets into memory at any given time.

The External Source plug-in works as follows:

- A Sound Voice 'template' is created in Wwise using the External Source plug-in. This template represents a series of audio files with common properties.
- The External Source plug-in can take full advantage of the power and flexibility of the project hierarchy, by placing it within containers, Actor-Mixers, applying states, RTPCs, effects, and so on.
- A play event is created that calls the external source.
- The location and conversion settings of the pool of external audio assets that can be used by the External Source plug-in are defined in the External Sources List file. This file is a very simple XML file that contains the location of the external audio assets that need to be converted along with the conversion settings you want to use. The location of this file is defined in the Project Settings dialog box in Wwise.
- At runtime, the game calls the External Source and then associates the template with one of the external audio files. The actual audio file that is played is left entirely up to the programmer. It is important to note that the management of the source audio files is done external to the Wwise sound engine. While this involves more work, it also gives you more flexibility.

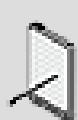


### Designer Note

Most of the work related to this feature is performed by the audio programmer within the SDK. For more information, refer the Wwise SDK documentation.

## SoundBanks and SoundBank Generation

SoundBanks keep all of your audio files and information about how they are played back bundled together in files that can be easily loaded and unloaded with the demands of the game. Regardless of the systems employed on the game side for managing available memory and resources, Wwise SoundBanks can be generated to support most implementations.



### Designer Note

A SoundBank can contain any number of events, Wwise objects, and/or converted media files. At particular points in the game, one or more of these project elements will be loaded into a game's platform memory in anticipation of specific sounds or motion objects being triggered.

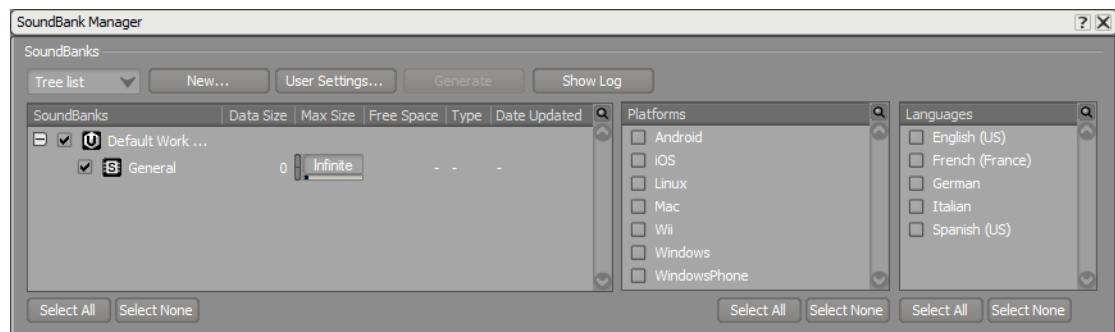
SoundBank creation is better handled through the SoundBank Layout (F7) that gives you access to the SoundBank Manager and SoundBank Editor.

### Creating a SoundBank

To begin with, we'll create a new SoundBank called General. Start by selecting the default work unit from the SoundBanks tab in the Project Explorer. This enables you to add a new SoundBank from the Project Explorer toolbar by clicking on the SoundBank icon. Alternatively, a SoundBank can be created from the contextual menu for a SoundBank work unit or by using shortcut keys.



Creating a General SoundBank

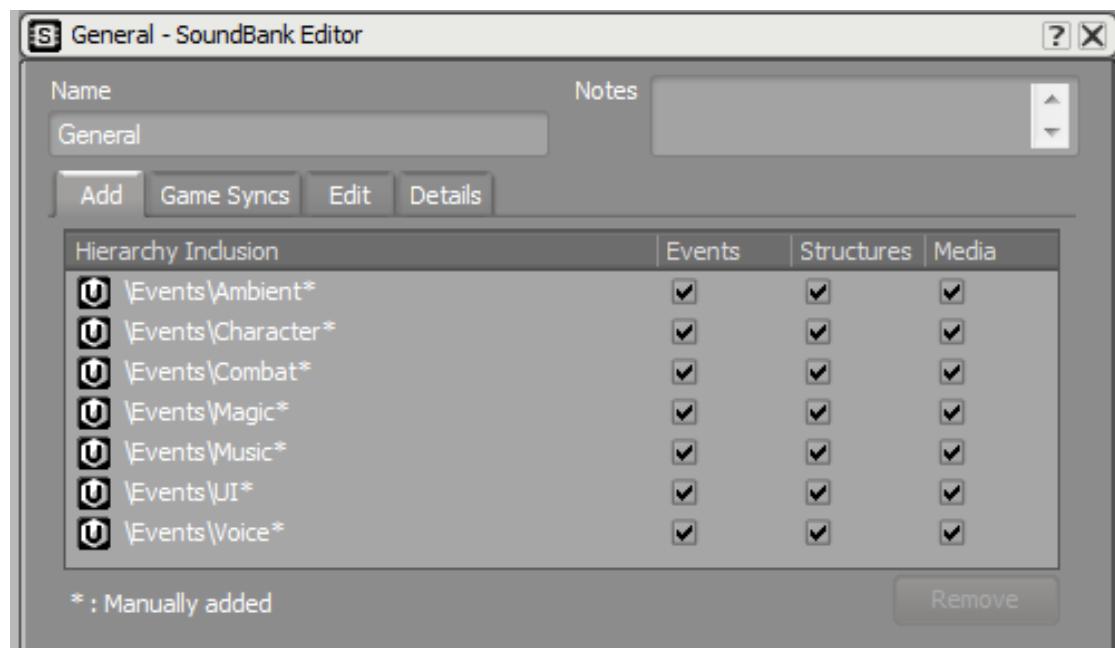


SoundBank information displayed in the SoundBanks Manager

The SoundBank Manager displays the list of SoundBanks that have been created. It also displays information about each SoundBank, including its current size, the amount of free space left, and the type or contents of the SoundBank. Before generating your SoundBanks, you can update them, define custom user settings, and specify for which platforms and languages the SoundBanks will be generated.

### Populating and Managing the Contents of SoundBanks

When you double-click a SoundBank in the SoundBank Manager, information related to the selected SoundBank is automatically displayed in the SoundBank Editor.



SoundBank Editor showing manually added events

The SoundBank Editor, which is where you populate and manage the contents of your project's SoundBanks, is divided into four different tabs:

**SoundBank Editor - Add Tab** - displays only the actual events, hierarchies, work units, and folders that were added to the SoundBank. Any corresponding child objects that are also automatically added to the SoundBank are only displayed on the Edit tab. On the Add tab, you determine what types of information and/or media will be included in the SoundBank per hierarchical element.

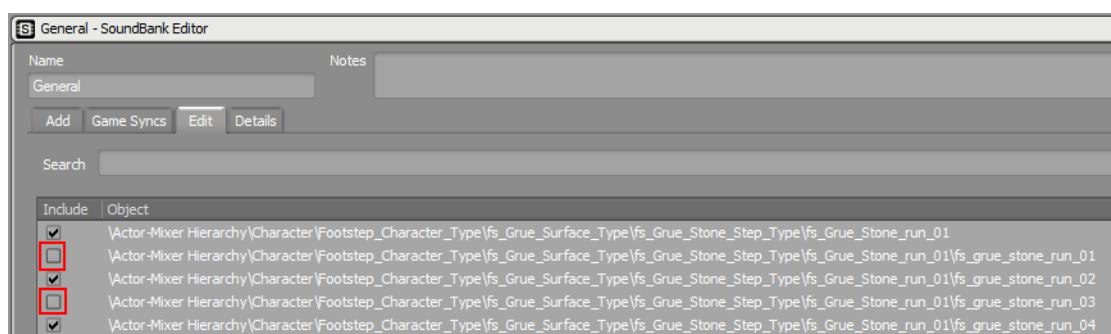
**SoundBank Editor - Game Syncs Tab** - displays a list of game syncs, except game parameters and arguments, referenced by the events and sound structures that have been included on the Add tab. On this tab, you can filter out sound structures, events, and media files based on their relationship with a particular game sync.

**SoundBank Editor - Edit Tab** - displays a detailed list of each individual event, object, and media file, including all child objects that are associated with the hierarchical project elements on the Add tab. You can filter the list by language and object type and then deselect any project elements that you want to exclude from the SoundBank.

**SoundBank Editor - Details Tab** - displays detailed information about all aspects of the SoundBank, including memory size, file size, SFX versus Voice size, as well as the number of missing and replaced files, if any.

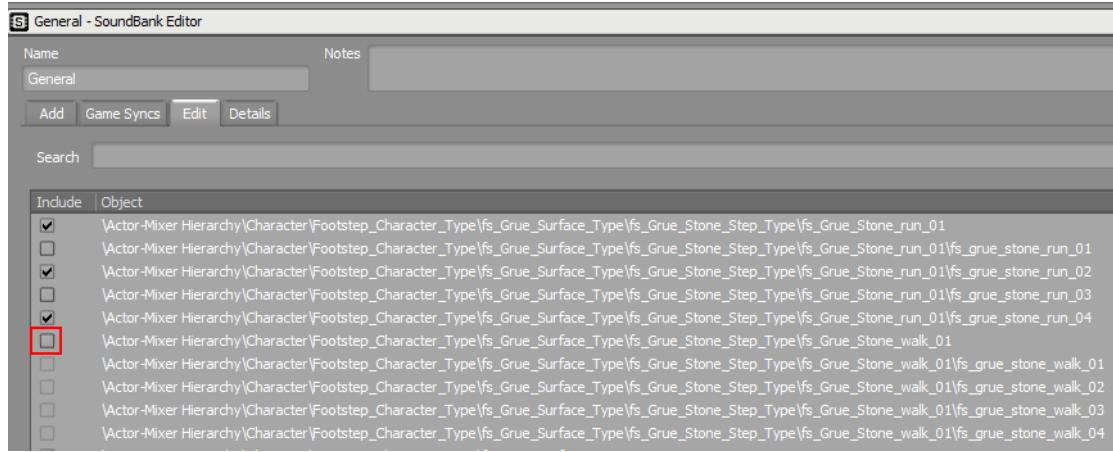
### Excluding Elements from a SoundBank

Specific elements within a SoundBank can be selectively excluded by clearing the corresponding check box.



#### Manually excluding individual sound objects from a SoundBank

Child sound objects can also be removed from the SoundBank by excluding the parent object.



### Manually excluding parent and child sound objects from a SoundBank

Once the SoundBanks have been populated they can be generated and used by the game.

## A New Approach for SoundBank Management

In an attempt to be as flexible as possible and to meet the requirements of almost any type of game, Wwise has introduced a new approach for managing the SoundBanks in your game. This new approach does not invalidate the usefulness of the original method (explained later in this section), but simply gives you more control and flexibility so that you can better manage the requirements of your games.

The new approach offers three major improvements over the existing traditional method:

- You can populate your SoundBanks with not only events, but also structural content (sounds, containers, Actor-Mixers, and so on), work units, and folders.
- You can determine what types of information will be included in the bank. This means that you can populate a bank with only media, structural data, event content, or any combination of the three. For example, you may want to create a bank that contains only the media associated with a specific event.
- You can include or exclude specific items from a SoundBank.

The main advantage of this new approach is that it allows media content to be split into multiple memory banks. For example, let's say the music for the entire game is started using one single event. Using the traditional method, you would add the event to the bank, which automatically adds all the corresponding in-memory sounds and pre-fetched media, including the pre-fetch for the song that only plays at the end of the game. Storing all the media in memory for the entirety of the game seems very inefficient. Using the new approach, however, you can better manage your memory by splitting the music media into multiple banks so that it would be loaded only when sounds are likely to be played.

By splitting the media into multiple banks, you can also prioritize the media that is to be loaded. For example, in an environment with limited memory, you will want to load only the most important media. Non-critical media could be stored in a separate bank that would be loaded into memory only when there was enough room. Previously, both critical and non-critical media files were contained in the same bank. If the bank was too large to load into memory, none of the sounds would play, including the critical ones.

The Wwise SDK Bank Management Samples describe the different methods that can be used to generate and integrate the banks in your game. In a single game, you can use one or a combination of the different methods. Since every game is different, the method or methods you choose will depend on the specific requirements of your game. Remember that all solutions will work, but the strategy you choose should take into consideration the memory usage, the I/O access, and the ease of integration in game. Each method has its advantages and drawbacks, so in most situations, it will be a question of balance between memory usage and ease of integration.

**Additional SoundBank integration details:**

[Wwise SDK - Windows > Sound Engine Integration Walkthrough > Integrate Wwise Elements into Your Game > Integrating Banks > Integration Details - Banks](#)

### **Conversion Settings**

Conversion settings are managed as ShareSets and created based on the needs of your project and the requirements of each active platform. Many of your choices here can have a big impact on the performance and quality of your audio project. After applying conversion settings ShareSets to the objects in your project, you can go back and adjust your ShareSets at any time to achieve the best possible quality within the constraints of the platform and the game. When you import audio files you can also speed up the process by re-using ShareSets.

**For more information about ShareSets:**

[Wwise Help > Finishing Your Project > Managing Platform and Language Versions > Authoring Across Platforms > Converting Audio Files > Creating Audio Conversion Settings ShareSets](#)

[Video Tutorial - Conversion Settings ShareSets](#)

### **The SoundBank Definition File**

Outside of the manual generation of SoundBanks, another method that is becoming common is the use of the SoundBanks Definition File as a way to automatically generate SoundBanks based on level, character, object, material or any information from the game. A definition file can be generated automatically by external applications, such as a game level editor, or manually by creating a tab delimited text

file that lists all the events in your game, classified by SoundBank. When SoundBanks are generated, Wwise packages all the Actor-Mixers, Containers, Sounds, and so on that are used by the events contained within each SoundBank. If some of the Sounds in a SoundBank are Sound Voices, then a different version of the SoundBank is generated for each language supported by the Wwise project.

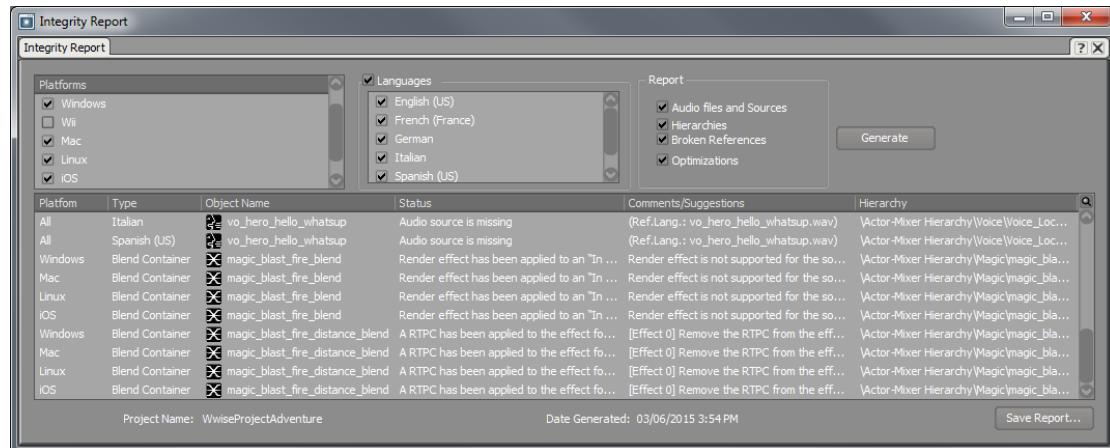
**For more information on SoundBanks:**

[Video Tutorial - Creating and Managing SoundBanks](#)

[Video Tutorial - Building SoundBanks](#)

## Using the Integrity Report

The Integrity Report serves as a map to the underworld, that is, if the underworld were populated by errors and inconsistencies instead of treasures and foes. The Wwise Integrity Report is where a report can be generated that contains information about the project, including errors and how to resolve them.



A view of the Integrity Report listing information, possible errors, and information on resolution

The Integrity Report lists errors such as:

- Missing media files
- Missing audio or motion sources
- Plug-in problems
- Missing events in SoundBanks

By double-clicking an error in the Error list, you can open a corresponding Wwise dialog box where you can resolve the error, or receive further information about how to handle it.

You can also filter the Integrity Report to display only the types of information that you specify, such as details about the following:

- Platforms
- Languages
- Audio files and sources
- Hierarchies
- References
- Optimizations

Often the integrity report can highlight areas of a project that need additional consideration or a deeper understanding.

## Using the File Packager

The File Packager is a stand-alone utility that groups the SoundBanks and/or streamed media files for a Wwise project into one or more file packages to be used for a specific platform. File packages can also help you better manage language versions as well as downloadable content that is made available post release.

File packages can be created that include any of the following:

- SoundBank files only
- Streamed media files only
- SoundBank files and streamed media files

All information about a Wwise project's SoundBanks and streamed media files can be retrieved by importing the SoundBanksInfo.xml file into the File Packager. The SoundBanksInfo.xml file is created automatically by Wwise each time the SoundBanks are generated.

The File Packager can be used to create your file packages manually, or the process can be automated by using a command line to run the File Packager as part of SoundBank generation. This command line can be defined at the project level or as a custom SoundBank user setting.

For more information about the File Packager:

[Wwise Help > Finishing Your Project > Managing File Packages](#)

[Wwise SDK > Samples \(Using File Packages\)](#)

[Wwise SDK > Sound Engine Integration Walkthrough > Integrate Wwise Elements into Your Game > Integrating Banks > Integration Details - Banks > Loading Banks](#)

[Wwise Help > Finishing Your Project > Managing SoundBanks](#)

[Video Tutorial - File Packager](#)

## Downloadable Content (DLC)

It has become a common development strategy for many games to plan for content that will become available after a game has been released. Often, this strategy is included in the schedule for creating a game and, if properly planned for, can be a seamless experience for adding additional content.

One key aspect of developing content to be distributed after release is: All DLC content must be created using the same Wwise Project that was used for the main release. Also, to ensure compatibility, the same Wwise version must be used for the Main and DLC releases. A basic scenario for managing DLC can be found in the Wwise Help and also includes considerations and limitations involved with preparing your project.

It is important to devote time to understand the process involved with preparing for the eventuality of downloadable content. With the right approach, the ability to modify implementation and include additional audio can enable new creative potential after the initial release.

For more information on Downloadable Content:

[Wwise Help > Finishing Your Project > Managing File Packages > Downloadable Content Overview](#)

## Setup Summary

While the myriad of setup options and considerations can seem overwhelming at a glance, the ability to prepare and modify the project to meet the needs of a particular development methodology ensures that the functionality is available. Understanding the fundamental concepts that run throughout the project can help prepare a road worn veteran for the task at hand. Armed with the tools to forge your own way on the path, a limitless vista of potential awaits.

Throughout this chapter we have discussed:

- Work Unit Management
- Establishing a Naming Convention Early
- Batch Renaming Objects Within Hierarchies
- Logical Grouping of Work Units
- Creating Work Units with Sharing in Mind
- Grouping Objects in the Actor-Mixer Hierarchy
- Creating Simulations with the Soundcaster
- Project Settings
- Workgroup Plug-in Configuration
- Audio File Locations
- Default Conversion Settings
- Defining the Sample Rate Automatic Detection Settings
- Obstruction/ Occlusion
- Setting Obstruction and Occlusion
- Motion
- Customizing Layouts
- Layout Docking
- View Docking
- External Sources
- SoundBanks and Bank Generation
- A New Approach for SoundBank Management
- Conversion Settings
- The SoundBank Definition File
- Using the Integrity Report
- Using the File Packager
- Downloadable Content (DLC)

## Setup Additional Resources

[Video Tutorial - Workgroup Management in Wwise Using Perforce](#)

[Video Tutorial - Wwise Motion](#)

[Video Tutorial - Managing Layouts](#)

[Wwise Help > Finishing Your Project > Managing Platform and Language Versions > Authoring Across Platforms > Converting Audio Files > Creating Audio Conversion Settings ShareSets](#)

[Video Tutorial - Conversion Settings ShareSets](#)

[Video Tutorial - Creating and Managing SoundBanks](#)

[Video Tutorial - Building SoundBanks](#)

**For more information about the File Packager:**

[Wwise Help > Finishing Your Project > Managing File Packages](#)

[Wwise SDK > Sound Engine Integration Walkthrough > Integrate Wwise Elements into Your Game > Integrating Banks > Integration Details - Banks > Loading Banks](#)

[Wwise Help > Finishing Your Project > Managing SoundBanks](#)

[Video Tutorial - File Packager](#)

[Wwise Help > Finishing Your Project > Managing File Packages > Downloadable Content Overview](#)

---

# **Chapter 12. Workflow Optimization**

Overview .....	292
Including or Excluding Platforms .....	293
Linking or Unlinking Properties .....	295
Rendering Effects .....	296
Understanding the Different Types of Profiling in Wwise .....	297
Connecting to the Game .....	298
Capturing Data Using the Profiler .....	299
Profiling Sound In-Game .....	300
Instance Limiting, Prioritization, and Virtual Voices .....	304
Playback Limits .....	305
Setting a Playback Limit per Game Object .....	305
Setting a Playback Limit on an Audio Bus .....	305
Global Playback Limit .....	306
Setting Playback Priority .....	307
Understanding Virtual Voice Behavior .....	309
Bridging the Game Engine Integration Gap .....	313
How Does SoundFrame Work? .....	313
Additional Game Engine Integration Techniques .....	314
Optimization Summary .....	315
Optimization Additional Resources .....	316

## Overview

Regardless of the game, genre, or platform you're working with, one concern that persists across all developments is the need to fit within the allotted memory and processing budget. Whether working within the tight constraints of mobile platforms, maintaining optimal CPU usage, or making sure the most important sounds get the memory needed for variation, the process of optimization is one that develops organically throughout development. It can be difficult to stay on target while the game is constantly changing, but there are some valuable resources available that can help keep things under control.

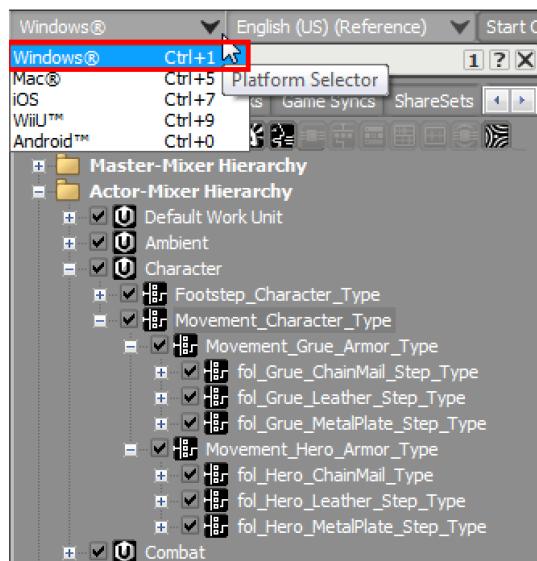
This chapter will take you through the process of:

- Including/excluding platforms
- Linking/unlinking properties
- Rendering effects
- Understanding the different types of profiling in Wwise
- Connecting to the game
- Capturing data using the Profiler
- Profiling the sound in-game sample rate automatic conversion
- Setting playback limits and priorities
- Understanding virtual voice behavior
- Integrating SoundFrame with the game engine

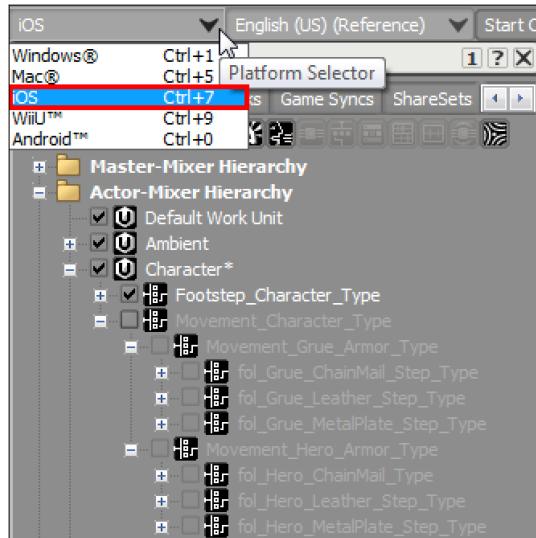
## Including or Excluding Platforms

One of the first strategies to make the trimming of sound content a testament to sanity is the ability to include/exclude sound objects non-destructively from one platform to another. By simply clearing the check box next to a sound object, the parent and any child objects are removed and excluded from the platform-specific build process.

For instance, removing all of the Movement Sounds from a platform with less memory in order to keep the game within its memory budget is as easy as removing the check from the parent sound object in the hierarchy:



Movement included for the Windows platform



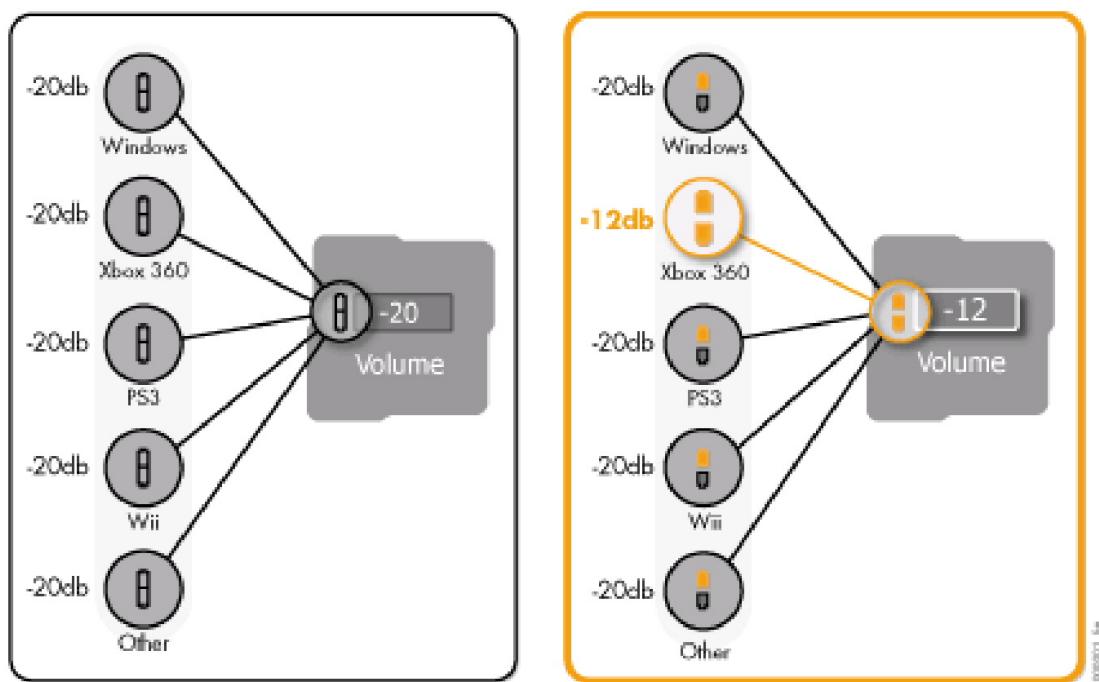
### Movement excluded for the iOS platform

If you're working on multiple platforms, you can use this technique to exclude sound objects or variations. Using the platform selector, you can work within each of your base platforms to maintain the necessary requirements.

## Linking or Unlinking Properties

By default, all sound and motion properties are set the same across all project platforms. These properties are said to be linked across platforms and can be selectively unlinked for any property on a per platform basis. This streamlines the creation of consistent projects across all platforms. Whether you're unlinking properties in order to apply different mix settings, to remove DSP effects for a specific platform, or any other reason, having a non-destructive workflow that enables this ability is a benefit to any multi-platform development.

Unlinking the properties of a property causes the link indicator to become orange.



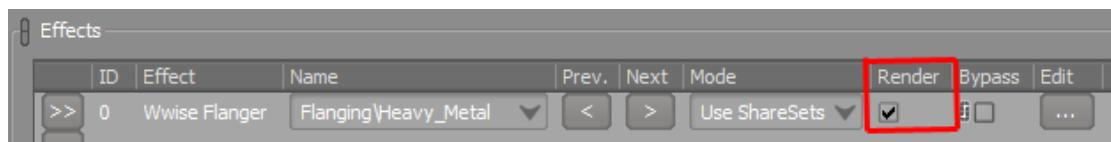
Properties are linked across all active platforms.

Volume is unlinked and customized for "XBox 360" platform, and the remaining platforms are partially unlinked.

When defining the properties for various platforms, you can easily tell if the properties are unlinked on another platform. The link/unlink indicator will be partly orange indicating that it is partially unlinked.

## Rendering Effects

Another place where you may be able to gain back valuable resources is in the rendering of any effects that are not actively modified by RTPC. Enabling rendering for an effect applies the effects settings to the audio files during SoundBank generation. While this adds to the overall size of SoundBanks, the reduction in CPU can be beneficial when in short supply.



Rendering effects



### Designer Note

Rendering time-based effects that may extend the duration of the file(s), such as delay, reverb, or time stretch, results in large files being generated in the SoundBank. This saves CPU at run time in exchange for additional memory usage.

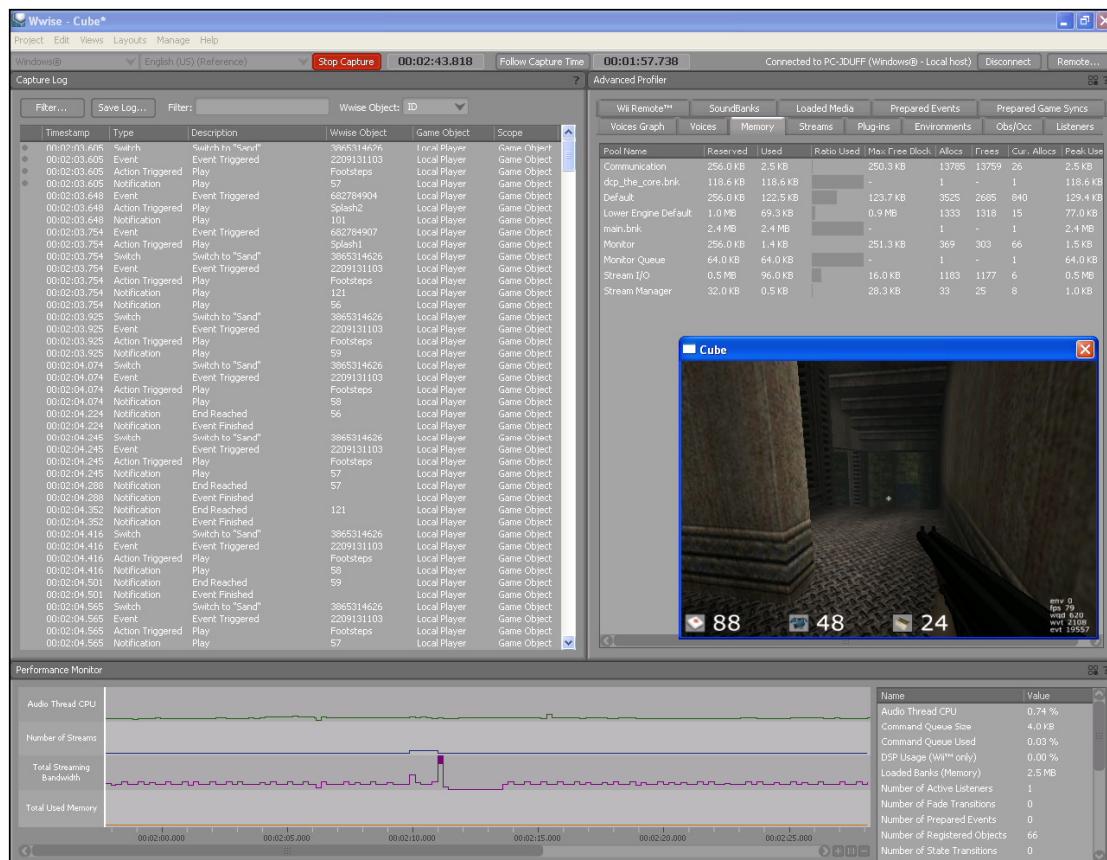
## Understanding the Different Types of Profiling in Wwise

The Wwise profiler is at the heart of most resource centered problem solving. There are few things related to the audio engine that you can't ascertain by connecting the running game to the Wwise authoring application and capturing key performance information in real time. The profiler is a consistent barometer for the state of audio in-game and gives access to a deep level of debug opportunities.

In Wwise you can perform two types of profiling:

- Game profiling
- Game object profiling

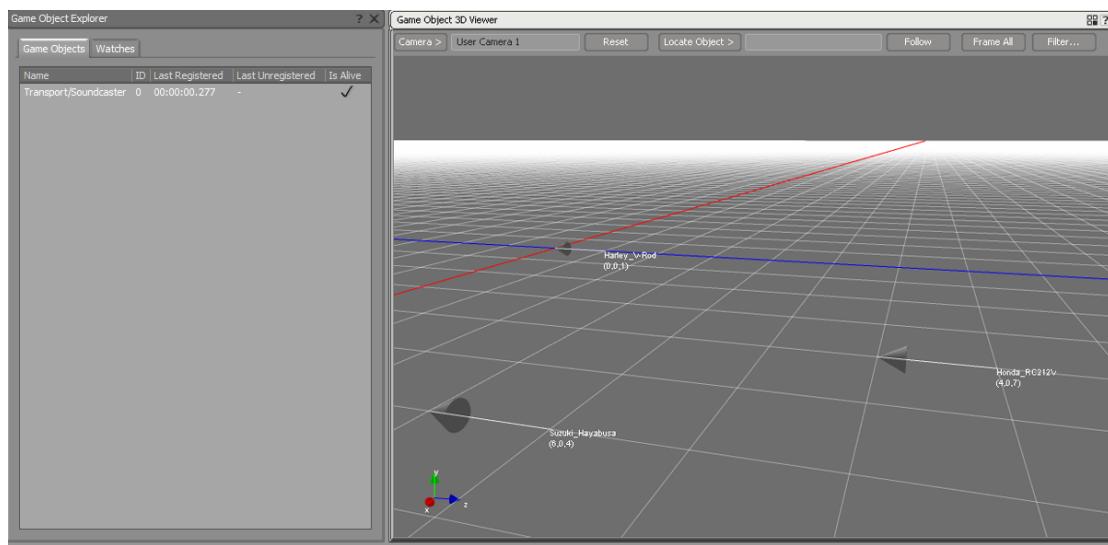
Game profiling focuses on performance requirements and demands, as viewed from the point of view of the sound engine and the various components that make up the project. It demonstrates in real time the cumulative effect the sound and motion in the project have on platform performance. It also allows you to examine the impact of individual voices.



Tracking performance using the Game Profiler with a game currently running

The Game Object Explorer is the starting point for studying game objects and listeners. Within this view, you can see all the registered game objects in your game, as well as control which game objects and listeners will be watched by Wwise. The game objects that you have selected for 'watching' become visible in the Game Sync Monitor, and both game objects and listeners show up in the Game Object 3D Viewer.

Game object profiling also analyzes the output of the sound engine, but from the point of view of individual game objects. Game object profiling tracks game objects so that you can observe their movements and behavior in real time. In this way, you can find out if certain game objects are problematic.

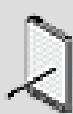


Tracking individual game objects in the Game Object Explorer 3D viewer

Game objects refer to discrete actors or entities that exist within a game. They are registered or created by the audio programmer for all objects or elements within the game that can emit a sound, such as player characters, non-player characters (NPCs), weapons, vehicles, etc. The game object profiling tools (the Game Object Explorer, Game Object 3D Viewer, and Game Sync Monitor) work together to examine game objects and listeners in a game or simulation.

### Connecting to the Game

To begin profiling or simulating sounds or motion FX in-game on a particular platform, you need to first connect to the PC or game console upon which the game runs. You can connect to any Wwise sound engine that is running and available on your local area network.



### Designer Note

When profiling, it is recommended that you connect to the Profile build configuration of the Wwise sound engine, even from the Debug build of your game, because the performance of the Debug configuration has not been optimized.

To help you find the PC or game console you are looking for, Wwise automatically searches for all PCs and game consoles on the same subnet of the network that are currently running a version of the Wwise sound engine. You can also connect to consoles or PCs outside your subnet by manually entering the IP address of the platform.

For more information on connecting to the game:

[Wwise Help > Finishing Your Project > Profiling > Connecting to a Local/Remote PC or Game Console](#)

### Capturing Data Using the Profiler

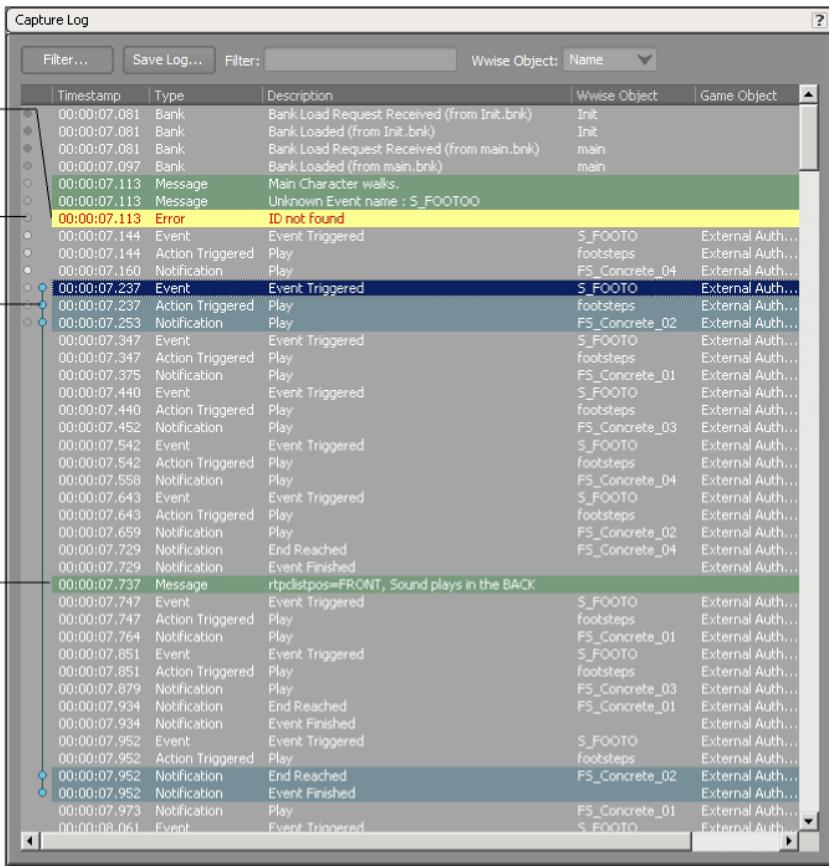
After connecting to a PC or game console, you can begin to profile the audio and motion FX in your game by capturing data directly from the sound engine. All the information coming from the sound engine is displayed in the Capture Log.

An entry is recorded in the Capture Log for the following types of information:

- Notifications
- Properties
- Banks
- Markers
- States
- Errors
- Events
- Switches
- Messages
- Actions
- Prepared Events

You can monitor each of these entries using the Performance Monitor and Advanced Profiler. These views contain detailed information about memory, voice, and effect usage, streaming, SoundBanks, plug-ins, and so on.

Wwise uses the following special indicators and color to help you quickly sort through the many entries that can appear in the Capture Log. The following illustration shows how the different indicators and colors are used in the Capture Log.



The screenshot shows the 'Capture Log' window with several log entries. Annotations on the left side explain the meaning of the different colored indicators:

- Any errors are highlighted in yellow.** (Yellow background)
- White/gray indicators show entries captured within 100ms of the Performance Monitor time cursor.** (Gray background)
- Blue indicators show which entries are related to one another. Related entries are also highlighted in teal blue.** (Teal blue background)
- All messages are highlighted in green.** (Green background)

The log entries include:

Timestamp	Type	Description	Wwise Object	Game Object
00:00:07.081	Bank	Bank Load Request Received (from Init.bnk)	Init	
00:00:07.081	Bank	Bank Loaded (from Init.bnk)	Init	
00:00:07.081	Bank	Bank Load Request Received (from main.bnk)	main	
00:00:07.097	Bank	Bank Loaded (from main.bnk)	main	
00:00:07.113	Message	Main Character walks.		
00:00:07.113	Error	ID not found Unknown Event name : S_FOOTOO		
00:00:07.144	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.144	Action Triggered	Play	Footsteps	External Auth...
00:00:07.160	Notification	Play	FS_Concrete_04	External Auth...
00:00:07.237	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.237	Action Triggered	Play	Footsteps	External Auth...
00:00:07.253	Notification	Play	FS_Concrete_02	External Auth...
00:00:07.347	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.347	Action Triggered	Play	Footsteps	External Auth...
00:00:07.375	Notification	Play	FS_Concrete_01	External Auth...
00:00:07.440	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.440	Action Triggered	Play	Footsteps	External Auth...
00:00:07.452	Notification	Play	FS_Concrete_03	External Auth...
00:00:07.542	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.542	Action Triggered	Play	Footsteps	External Auth...
00:00:07.558	Notification	Play	FS_Concrete_04	External Auth...
00:00:07.643	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.643	Action Triggered	Play	Footsteps	External Auth...
00:00:07.659	Notification	Play	FS_Concrete_02	External Auth...
00:00:07.729	Notification	End Reached	FS_Concrete_04	External Auth...
00:00:07.729	Notification	Event Finished		
00:00:07.737	Message	rtpdistpos=FRONT, Sound plays in the BACK		
00:00:07.747	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.747	Action Triggered	Play	Footsteps	External Auth...
00:00:07.764	Notification	Play	FS_Concrete_01	External Auth...
00:00:07.851	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.851	Action Triggered	Play	Footsteps	External Auth...
00:00:07.879	Notification	Play	FS_Concrete_03	External Auth...
00:00:07.934	Notification	End Reached	FS_Concrete_01	External Auth...
00:00:07.934	Notification	Event Finished		
00:00:07.952	Event	Event Triggered	S_FOOTOO	External Auth...
00:00:07.952	Action Triggered	Play	Footsteps	External Auth...
00:00:07.952	Notification	End Reached	FS_Concrete_02	External Auth...
00:00:07.952	Notification	Event Finished		
00:00:07.973	Notification	Play	FS_Concrete_01	External Auth...
00:00:08.061	Event	Event Triggered	S_FOOTOO	External Auth...

Capture Log showing indicator descriptions

### Profiling Sound In-Game

During an epic lower level dungeon raid, the frame rate slows as a burst of sound engulfs the player while a maelstrom of visual effects explodes onscreen. In order to identify the role that sound may have had in the decrease in frame rate, the game profiling and game object profiling tools could be used to analyze the performance.

You can use game profiling tools to analyze the following:

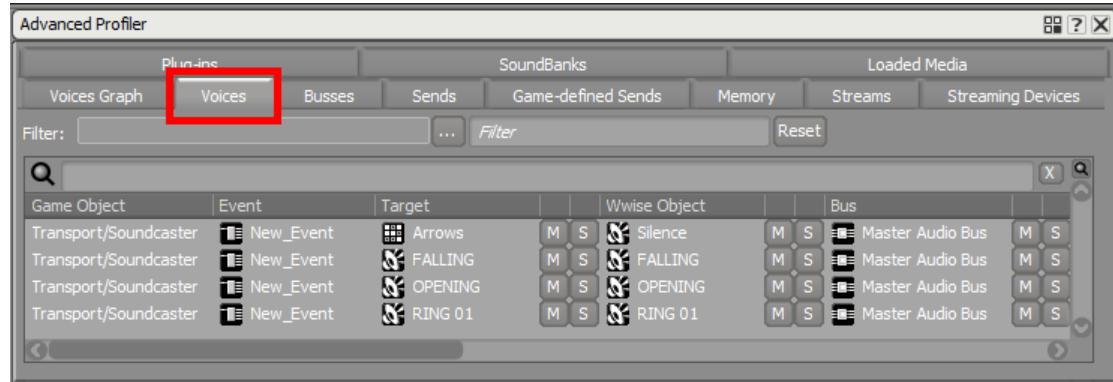
- How the many sounds associated with monsters, visual effects, and physics objects use the platform's streaming capabilities.
- How and when background noises, such as positional ambient sound, become virtual voices.
- Which effect plug-ins are applied to the different monster vocalizations and how these affect CPU usage.

You can use game object profiling tools to analyze the following:

- How the attenuation radius of the sounds for each monster interacts with that of each other monster.

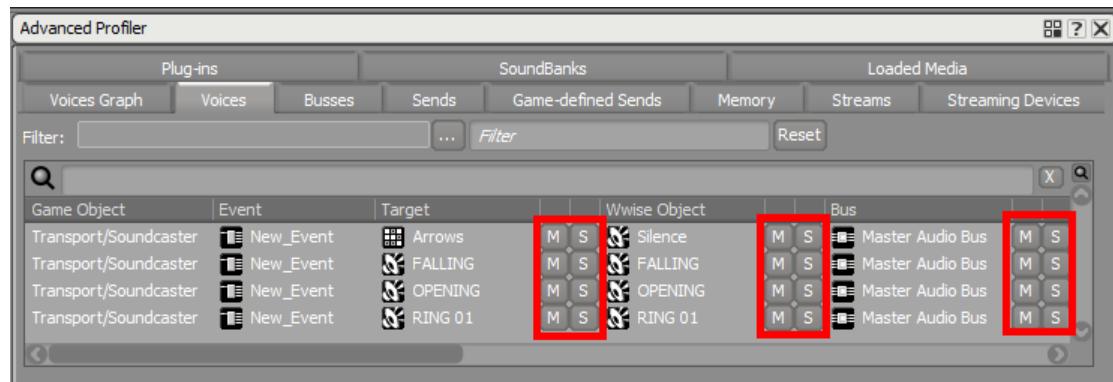
- Where game objects such as projectile magic visual effects move relative to one another and to the monsters.
- How an RTPC driving a side-chain affects the playback of sounds associated with combat.

In the Advanced Profiler, the relationship between the Game Object, Event, Target, and Wwise Object can be traced, adding further clarity to voices displayed in the Capture Log.



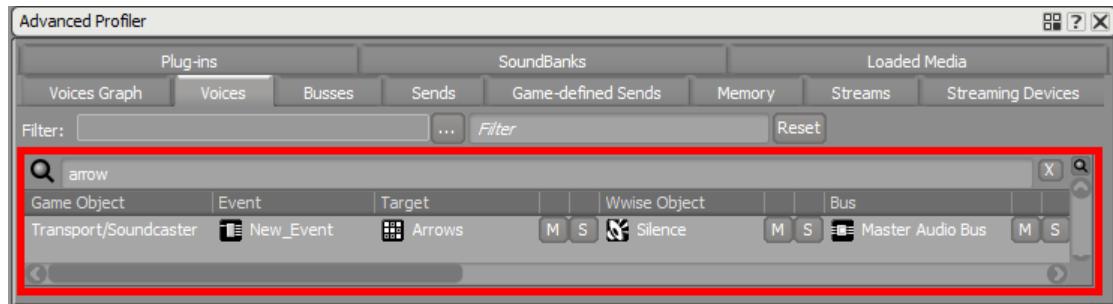
Advanced Profiler showing the Voices tab and current voices

Target Objects, Wwise Objects, and Busses displayed can be selectively muted and soloed to aid in the investigation and identification of sounds.



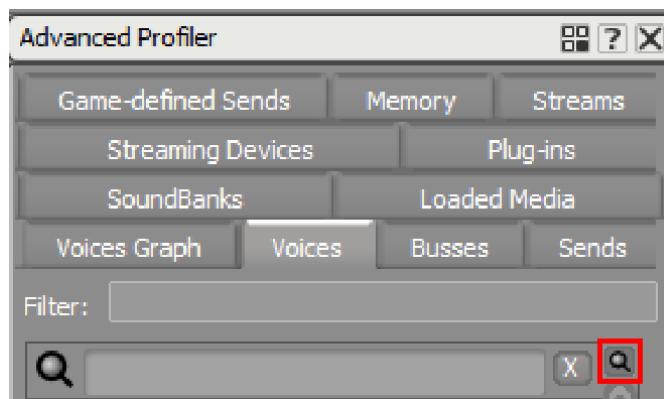
Voices Tab showing mute and solo buttons for Target, Wwise Object, and Bus

Additionally, each tab in the Advanced Profiler can open a search field where standard alphanumeric entries can filter out unmatching elements from the view.



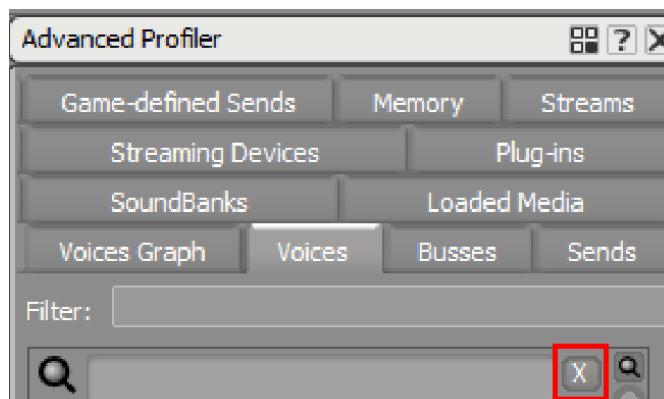
A view of the search field filtering down to all voices matching the word "arrow" in the Voices tab

Click the Search icon in the top right corner of the results window to open the search field to filter.



Opening the search field using the Search icon

Click the Close icon to the left of the Search icon to close the search field and remove the filter.



The Close button for the search field

The ability to swiftly visualize and navigate through different aspects of the project makes for great success in understanding the flow of actions being executed. In this way, the game profiling and game object profiling tools can give you a complete view of the game's soundscape in action.

**For more information about game profiling:**

[Wwise Help > Finishing Your Project > Profiling](#)

[Video Tutorial - Wwise Profiler Overview](#)

## **Instance Limiting, Prioritization, and Virtual Voices**

It's not until the project has been significantly completed that the effects of instance limiting, prioritization, and virtual voice behavior can be heard on the resulting sounds played back during gameplay. Understanding these concerns at the outset, and making preparations throughout development, can save the day at the end of production when resources for implementing a fundamental change can be at a premium.

## Playback Limits

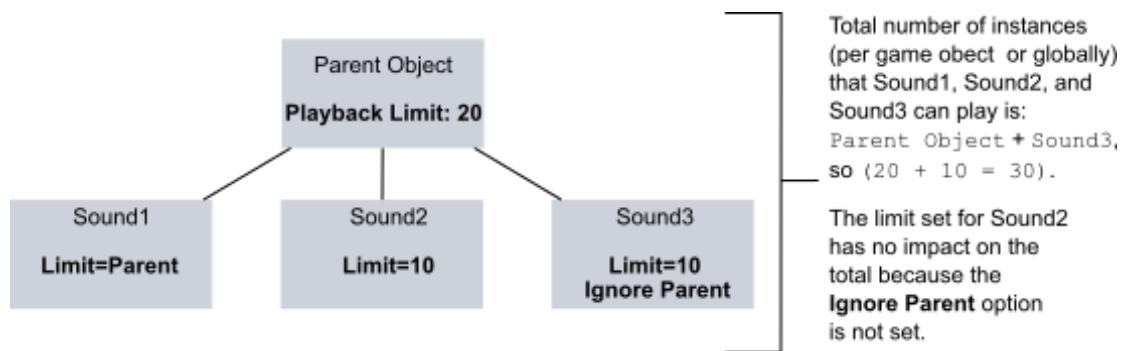
To deal with limited game resources or game-design constraints, you must optimize the sounds, music, and motion objects that are playing at any one point in the game. This can be accomplished using two different methods:

- Limit the number of sound, music, and/or motion instances that can be played per game object.
- Limit the overall number of sound, music, and/or music or motion instances that can pass through a bus.

### Setting a Playback Limit per Game Object

When either limit is reached, Wwise uses the priority setting of the object to determine which one to stop and which one to play. If objects have equal priority, there is the option to stop the newest or oldest instance that is playing.

When you set a playback limit at the Actor-Mixer or Interactive Music level, you control the number of instances within the same structure that can be played, per game object. If a child object ignores the playback limit set at the parent level, the total number of instances that can play is their sum. This means that if, for example, you have a parent with a limit of 20 and a child object set to **Ignore parent** with a limit of 10, the total possible number of instances is 30. If, however, the child were not set to **Ignore parent**, then the total possible number of instances would be the same as the parent.



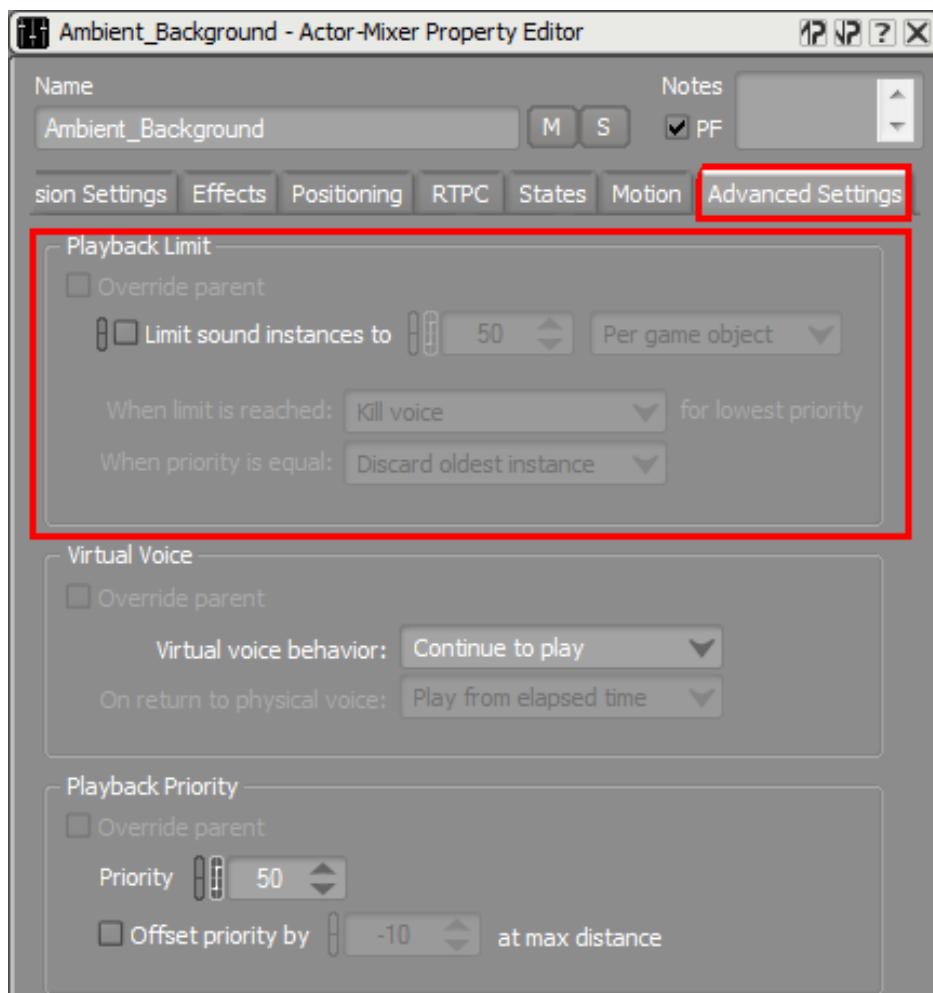
### Setting a Playback Limit on an Audio Bus

When you set the playback limit at the Master-Mixer level, the number of sound, music, and/or motion instances that can pass through the bus at any one time is specified. Since the priority of each object has already been specified at the Actor-Mixer or Interactive Music level, there is no playback priority setting for busses.

## Global Playback Limit

If the new sound, music, or motion object is not killed or sent to virtual voice at the Actor-Mixer or Interactive Music level, it passes to the second process at the Master-Mixer level. At this level, a global playback limit is used to restrict the total number of voices that can pass through the bus at any one time.

Managing playback limits can be done in the advanced setting of any sound object, Actor-Mixer, or audio bus in the Advanced Settings tab of the Property Editor.



Setting playback limits and behavior in the Advanced Settings tab

## Setting Playback Priority

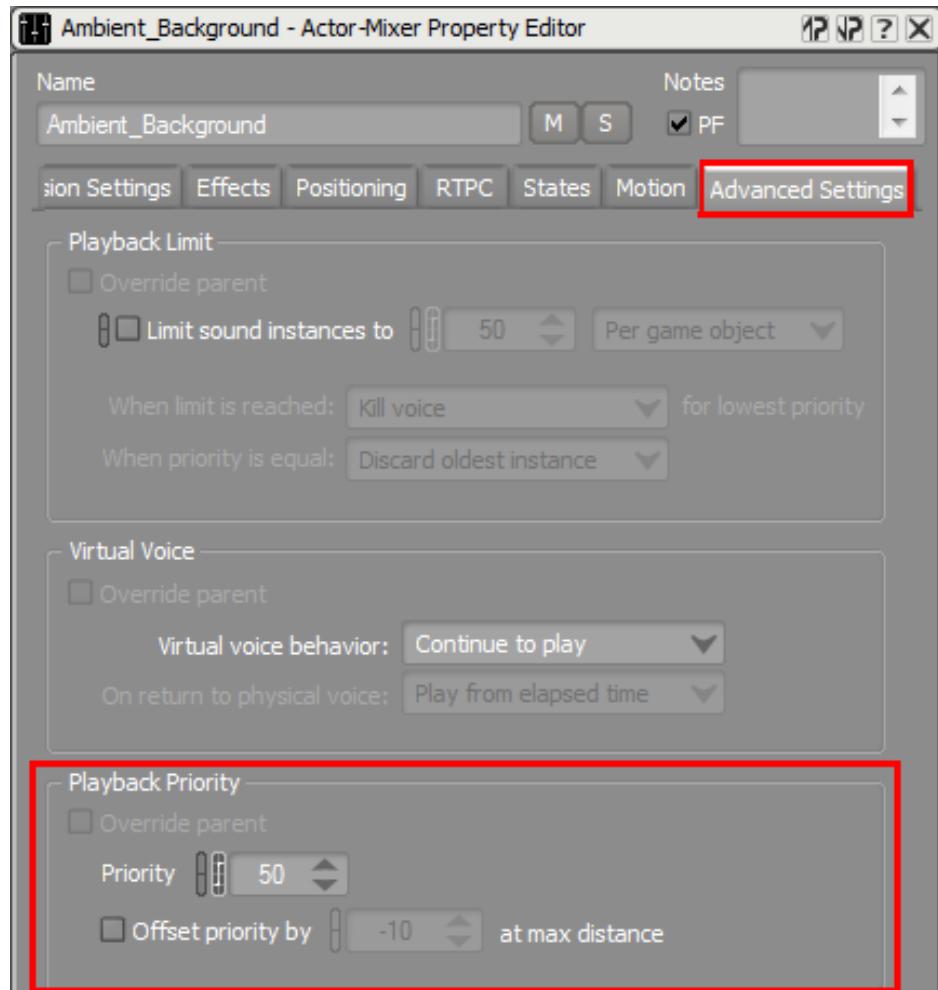
Priorities define the importance given to the sound or motion object in relation to other objects within the same Actor-Mixer structure. Establishing general priorities for parent containers early on at the top level of the Actor-Mixer based on sound type is a step in the right direction. Additionally, identifying critical sounds in need of a higher priority to ensure their playback is a matter of knowing the content and, where possible, organizing it appropriately to make things easier towards the end of a project.



### Designer Note

You can also alter the playback priority based on the distance the sound or motion object is from the listener. Wwise applies an offset to the priority using the Max distance values defined in the Attenuation Editor. The amount of offset applied will depend on the object's relative position to the listener. Wwise linearly interpolates the offset between the source point, where no offset is applied, and the attenuation max distance, where the full offset value specified is applied.

Managing priorities can be done in the advanced setting of any sound object, Actor-Mixer, or audio bus in the Advanced Settings tab of the Property Editor.



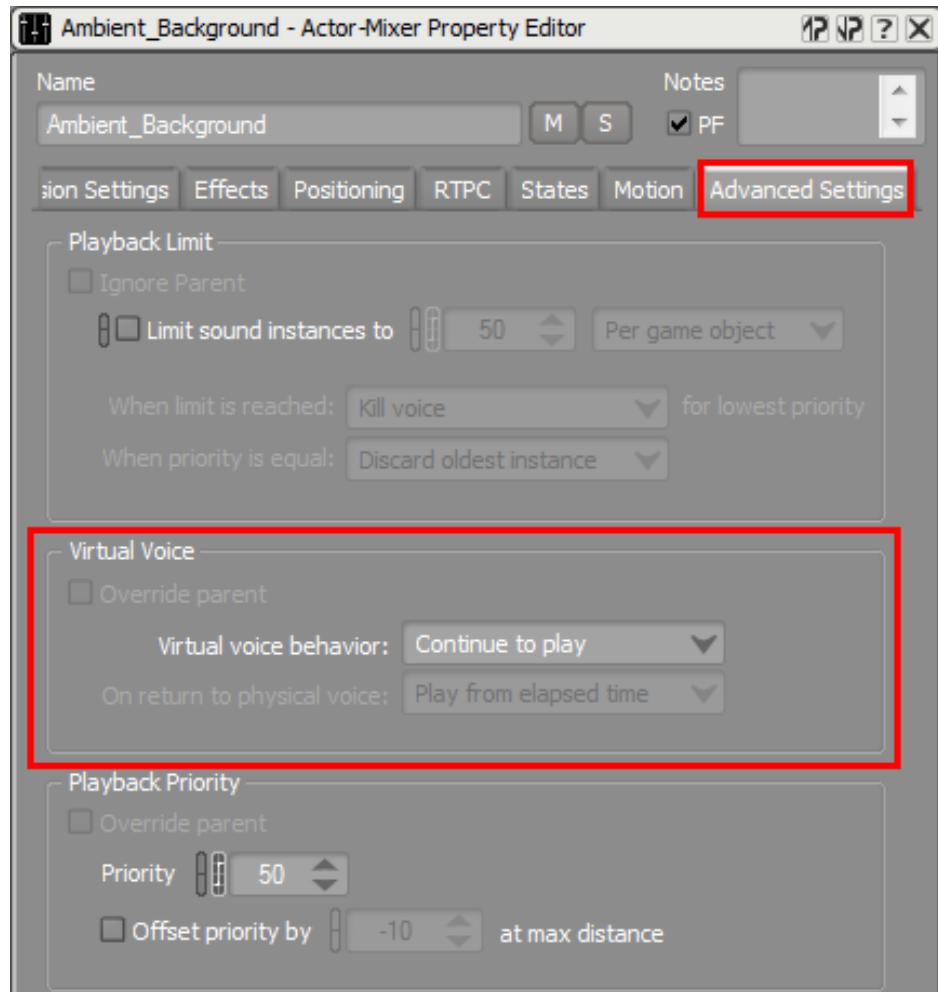
Setting playback priority in the Advanced Settings tab

## Understanding Virtual Voice Behavior

To maintain an optimal level of performance when many sounds are playing simultaneously, sounds below a certain volume level should not take up valuable processing power and memory. Instead of playing these inaudible sounds, the sound engine can queue them in the virtual voice list. Wwise continues to manage and monitor these sounds, but once inside the virtual voice list, they are no longer processed by the sound engine and won't take up one of the hardware's active voices.

When you select the virtual voices feature, sounds move back and forth between the physical and the virtual voice based on their volume levels when they are under threshold, or if their playback limit was exceeded. As the volume reaches the threshold set by the Wwise user in Project Settings, they are added to the virtual voice list and sound processing stops. As volume levels increase, as is the case when sounds move within the max distance radius, the sounds will move from the virtual voice list to the physical voice where they will be processed by the sound engine again.

Managing Virtual voice behavior can be done for any sound object, Actor-Mixer, or Audio Bus in the Advanced Settings tab of the Property Editor.



**Setting virtual voice behavior in the advanced setting tab**

There are three options to choose from when determining the move from physical to virtual voice:

- **Continue to play** - Continues playing the object as a physical voice even though it will no longer be heard.
- **Kill voice** - Stops playing the object. No fade out is applied with this option.
- **Send to virtual voice** - Sends the object to the virtual voice list.

When a sound or motion object is sent to the virtual voice list, certain parameters of the object are monitored by the sound engine, but no processing for audio or motion occurs.

When Send to Virtual Voice is set, there are three options that determine the behavior of sounds or motion objects that move from the virtual voice list back to the physical voice.

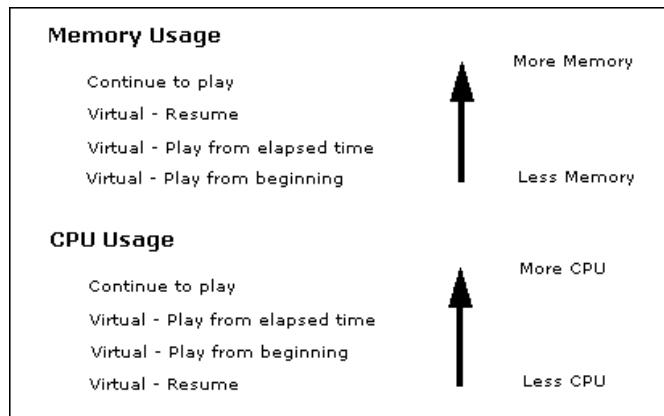
- **Play from beginning** - Plays the object from its beginning. This option resets the object's loop count.
- **Play from elapsed time** - Continues playing the object as if it had never stopped playing.

This option is not sample accurate, so sounds returning to the physical voice may be out of sync with other sounds playing.

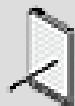
- **Resume** - Pauses the object when it moves from the physical voice to the virtual voice list and then resume playback when it moves back to the physical voice.

Each behavior has its own performance characteristics, as demonstrated in the following table:

Behavior	CPU cost	Memory cost
Play from beginning	Medium: Voice stops being serviced when virtual. Some extra operations are done when switching between virtual and physical.	Low: All internal processing buffers are flushed when virtual.
Play from elapsed time	High: Voice needs to be serviced at each buffer when virtual. Some extra operations are done when switching between virtual and physical.	Low: All internal processing buffers are flushed when virtual.
Resume	Low: Voice stops being serviced when virtual. No operations occur when switching.	High: All internal processing buffers are retained when virtual.



### Designer Note



Streamed sounds stop consuming I/O bandwidth while they are virtual. When the selected behavior is Play From Beginning or Play from Elapsed Time, the I/O buffer is flushed. This causes a delay before the sound is heard again when the voices switch from virtual to physical.

In short, taking control of instances and priorities can actually change the sound of your game. Special care and handling should be used when defining the properties at every level of both the Actor and Master-Mixer Hierarchies. Restrictively modifying these setting can prevent important sounds from being heard appropriately, in the same way that ignoring the setting can result in unwelcome behavior.

For additional information on Instance Limiting, Prioritization, and Virtual Voices:

[Video Tutorial - Voice Management](#)

[Wwise SDK > Going Further > Optimizing Memory Pools > Tips to Reduce Memory Usage](#)

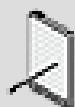
[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Managing the Priority of Sounds and Motion > Limiting Object Playback Instances](#)

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Managing the Priority of Sounds and Motion > Priority Tips and Best Practices](#)

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Managing the Priority of Sounds and Motion > Understanding How Wwise Prioritizes Objects](#)

## Bridging the Game Engine Integration Gap

The ability to bridge functionality between game and audio engine authoring applications is one of the most underestimated workflow improvements available to the development team. The interaction between tools that are linked by a shared set of information can be used to drive iteration as part of a sound designer's daily tasks. It is here that the SoundFrame technology aims to bring a suite of solutions to help quickly streamline communication between Wwise and other authoring applications in an intelligent and modular way.



### Designer Note

SoundFrame gives you access to most of the Sound Engine API. This allows you to enable event playback as well as modify states, switches, RTPCs, triggers, and environments in the application. This API lets you simulate real game scenarios directly in Wwise without requiring a working game engine or even having to generate SoundBanks.

Using the SoundFrame SDK, you can build plug-ins that can be integrated directly into your world building application, whether it be Unity, Unreal Editor (UnrealEd), Maya®, 3ds Max®, or any internal proprietary tool. This type of plug-in, which is built on top of the communication framework, allows you to perform many Wwise functions directly in your world building application, such as playing events, triggering game sync changes, and modifying positioning properties. You will also be able to integrate events at particular points in the animation, map switches to game textures, visualize attenuation radiiuses, and assign environmental reverb to zones, among many other things.

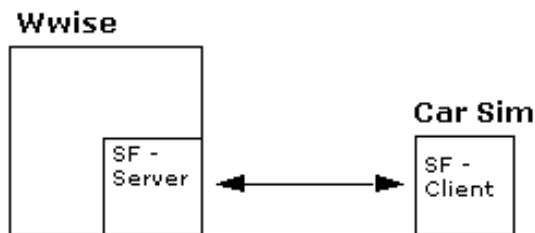
### How Does SoundFrame Work?

Applications and plug-ins created with the SoundFrame API work in a similar way to Soundcaster in Wwise. Like Soundcaster, you can re-create a variety of game scenarios by triggering events, sounds, and game syncs. The main difference is that the SoundFrame application or plug-in is outside of Wwise. In order for the two applications to communicate with each other, they both need to be installed and running on the same machine. SoundFrame establishes a bidirectional link with Wwise using a client-server type relationship. This type of communication allows you to validate sophisticated game scenarios quickly and efficiently.

### SoundFrame Plug-in



### SoundFrame Application

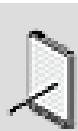


When sounds are triggered by a SoundFrame application or plug-in, they are played through Wwise. Since SoundFrame uses Wwise as its sound engine, it is free from any sound engine restrictions. This means that updates can be tested and validated live, which can save you a great deal of time at all stages of the development cycle.

By taking the simulation capabilities of Soundcaster one step further, the amount and types of audio simulations you can perform using SoundFrame are only limited by your own imagination.

### Additional Game Engine Integration Techniques

While the SoundFrame technology provides a framework that can allow rapid development of toolset integration with Wwise, there are those who may wish to take the task of game engine integration into their own hands. The Wwise XML Schema included with the SDK is provided as a roadmap for deeper integration which may be necessary in the development environment.



#### Programmer Note

The XML schema for Work Units is available in the Schemas folder in the Wwise installation (ObjectDataSchema.N.xsd - use the one with the highest number). This schema can be used to understand the format of Work Unit files, and it can also be used to validate the XML file generated. Looking at the WWU files saved by Wwise with simple projects will also allow you to see how information should be organized.

## Optimization Summary

Being aware of the many minute details of a Wwise project early on in development can contribute to a positive workflow methodology. By making many small changes over time you can avoid the common pitfalls that occur towards the end of a project. Ideally when you reach milestones within the production that can benefit from optimization, there has already been some thought on how best to manage limited resources. Through a combination of understanding and conscious implementation, the tools to realize great sound for any game are within reach.

Throughout this chapter we have discussed:

- Platform Inclusion / Exclusion
- Linking / Unlinking Properties
- Effects Rendering
- Understanding the Different Types of Profiling in Wwise
- Connecting to the Game
- Capturing Data using the Profiler
- Profiling the Sound In-Game
- Instance Limiting, Prioritization, and Virtual Voices
- Playback Limits
- Setting a Playback Limit Per Game Object
- Global Playback Limit
- Setting Playback Priority
- Understanding Virtual Voice Behavior
- Bridging the Game Engine Integration Gap
- How Does SoundFrame Work?
- Additional Game Engine Integration Techniques

## Optimization Additional Resources

[Wwise Help > Finishing Your Project > Profiling](#)

[Video Tutorial - Wwise Profiler Overview](#)

[Video Tutorial - Voice Management](#)

[Wwise SDK > Going Further > Optimizing Memory Pools > Tips to Reduce Memory Usage](#)

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Managing the Priority of Sounds and Motion > Limiting Object Playback Instances](#)

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Managing the Priority of Sounds and Motion > Priority Tips and Best Practices](#)

[Wwise Help > Using Sounds and Motion to Enhance Gameplay > Managing the Priority of Sounds and Motion > Understanding How Wwise Prioritizes Objects](#)

---

## **Chapter 13. Closing**

The Real Adventure Begins ..... 318

## The Real Adventure Begins

The development path is long, winding, and littered with fallen prototypes, past iterations, and a plethora of dead ends. As you head into battle, the best you can hope for is a trusty weapon by your side to help defend yourself against challenges as you progress through the forest of iteration. In game audio, you rely on the combined power of sound design and the magic of implementation to carry you across the battlefield to an eventual victory over the forces of evil, or bad sound. Wwise is a formidable ally to have on your side during the challenging task of development. On one hand a gleaming sword of efficiency and simplicity, on the other, a sorcerer's stone of possibility. Let the adventure begin!

---

**Thank You**

## **Special Thanks**

### **Audiokinetic**

The production of this document could not have been possible without vast expertise of the Audiokinetic team, especially Simon Ashby and Etienne Caron who were instrumental in validating the terminology and consistency of intent throughout the development of the documentation. Additionally, the final document presentation relied heavily on the capable technical wizardry of Bernard Rodrigue.

### **Editor**

Judy Lapalme moved swiftly and effortlessly to correct errant punctuation and help streamline clunky sentence structure.

### **Project Content**

All content was created exclusively for use in the accompanying Wwise Project Adventure by Bay Area Sound: Julian Kwasneski (Sound Design) & Jared Emerson-Johnson (Composer).

### **Continuity Experts (Beta Testers)**

Hrishikesh Dani, Luca Fusi, Jack Menhorn, Roel Sanchez, Michael Taylor, and Rob Bridgett.

---

## **About the Author**

## Damian Kastbauer

Damian Kastbauer is a Technical Sound Designer working to help bridge the gap between noise makers and game developers. Utilizing the functionality of game audio specific implementation authoring applications, his goal is to create dynamic sound interactions that leverage interactive techniques to make good sound content sound great.

He lives in Seattle, Washington with his sharp-witted and beautiful wife, two glorious daughters, a big hairy dog, and (the ghost of) a freaky cat. When not dreaming and talking about ways to bring the beauty of interactive audio to the people, he can be found: spending time with his family, building things with his hands, and making weird noises for fun with effect-pedals.

He can be reached at: [damian@lostchocolatelab.com](mailto:damian@lostchocolatelab.com).

Q: Is it: "Lost Chocolate LAB" or "Lost CHOCOLATE Lab"?

A: It depends on how you look at it. I've never lost a Labrador and I don't remember ever having a Laboratory...

Q: What is the future of game audio?

A: Everyone working together to increase the use of interactive audio to create unique and engaging experiences.