audiokinetic

Sample Project

2021.1.14

Sample Project

Wwise 2021.1.14 Revision 4331
Copyright © 2023 Audiokinetic Inc. All rights reserved.

Table of Contents

Chapter 1. About the Sample Project

Wwise Sample Project

For the current release of Wwise, Audiokinetic has prepared a sample project demonstrating typical sound design examples. This sample project demonstrates how you can creatively use established and new features of Wwise. We encourage you to have a close look at this project to see how Wwise can manage game audio in one complete project. After you've gone through this project, go ahead and make your own projects to reflect your vision of game audio.

This documentation is meant to provide general guidelines about project design and Wwise functionality, and is not meant to replace other Wwise documentation. To learn more about Wwise, you can consult the online Help, video tutorials and other resources from the Audiokinetic website.

About the Project

This sample project exposes different examples of typical ways to use Wwise. Each of these examples has been integrated into different Work Units, each identified by name.

- **Footsteps** — A demonstration of how a hierarchy of Switch Containers can create interesting footstep sounds within a game.
- **Minigun** — An example of how a complex audio object can be easily managed with two Events and a game sync.
- **Car Engine** — A demonstration of how to use a Blend Container to reproduce the shifting sounds of a car engine.
- **NYC Ambience** — A demonstration of how Wwise allows you to create a complex ambience which evolves over a day and is influenced by weather.
- **SoundSeed Impact** — A demonstration of many uses of SoundSeed Impact.
- **Dialogue** — A demonstration of how to create dialogue that changes dynamically according to actions in game.
- **Ambisonics** — A demonstration of the surround sound of ambisonics through the binauralization of an Audio Bus using the Auro Headphone Effect.
- **Music** — These examples demonstrate different techniques for arranging and implementing interactive game music using pre-rendered audio files.
- **MIDI** — This MIDI music mixes and matches pre-rendered music and MIDI sequences, the MIDI Sequences target sampled instruments and a soft synthesizer.

Note

Materials sent by content providers were all originally high quality (48 kHz, 16 or 24 bit). However, to save download time and disk space, we have reduced the size of some of the sound assets.

Chapter 2. Project Details

Project Details

The following sections describe each of the project’s examples in detail. Feel free to experiment with these examples in Wwise. Before doing so, however, you may want to save a backup copy of the original sample project to your hard drive.

Bus Routing - Master-Mixer Hierarchy

The Master-Mixer hierarchy for this project is organized in two sections as if a 3D game was developed with a series of rooms where different reverbs would be applied at runtime:

- The “Environmental” bus regroups audio busses for 3D sounds that would normally be affected by reverberation (e.g. foley, weapons, vehicles, etc.) and auxiliary busses that have reverb or delay effects inserted. The audio busses can be seen as the 'dry' path and the aux busses as the 'wet' path.

  Note

  The sample project has been set as if it would be integrated in an actual 3D game where reverbs are dynamically assigned at runtime. Therefore, the objects in the Actor-Mixer hierarchy have been set to use the Game-Defined Auxiliary Sends and not the User-Defined Auxiliary Sends.

  Game-Defined Auxiliary Sends cannot be heard from Wwise. If you'd like to listen to these sounds with reverb, you can route the top parent User-Defined Auxiliary Sends objects from the Actor-Mixer hierarchy to the Aux busses.

- The “Non-Environmental” bus groups all busses that are not affected by runtime reverberation. In general, music and some voices such as commentators or narrators are the types of objects that are not affected by runtime reverb.

  Note

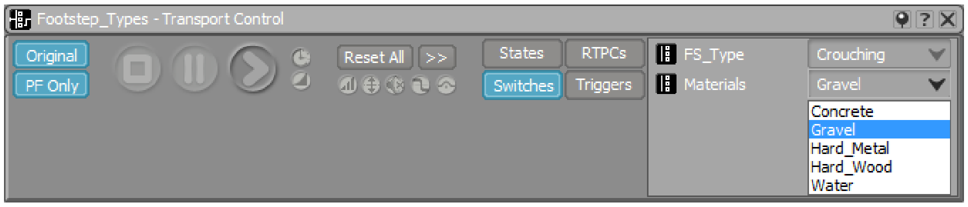
  The “Music” bus is flagged as “Mute for Background Music” to be compliant with technical certification requirements for Xbox One™, PlayStation® 4, iOS, and Android™.

Example: Footsteps

The Footsteps example demonstrates how you can create a simple hierarchy of Switch Containers to create variation for an otherwise repetitive sound.

To listen to the Footsteps example:

1. Load the "Footstep_Types" Switch Container or the "Play_Footstep" Event into the Transport Control.
2. In the Transport Control, click Switches and select the footstep type and material that you want to hear.



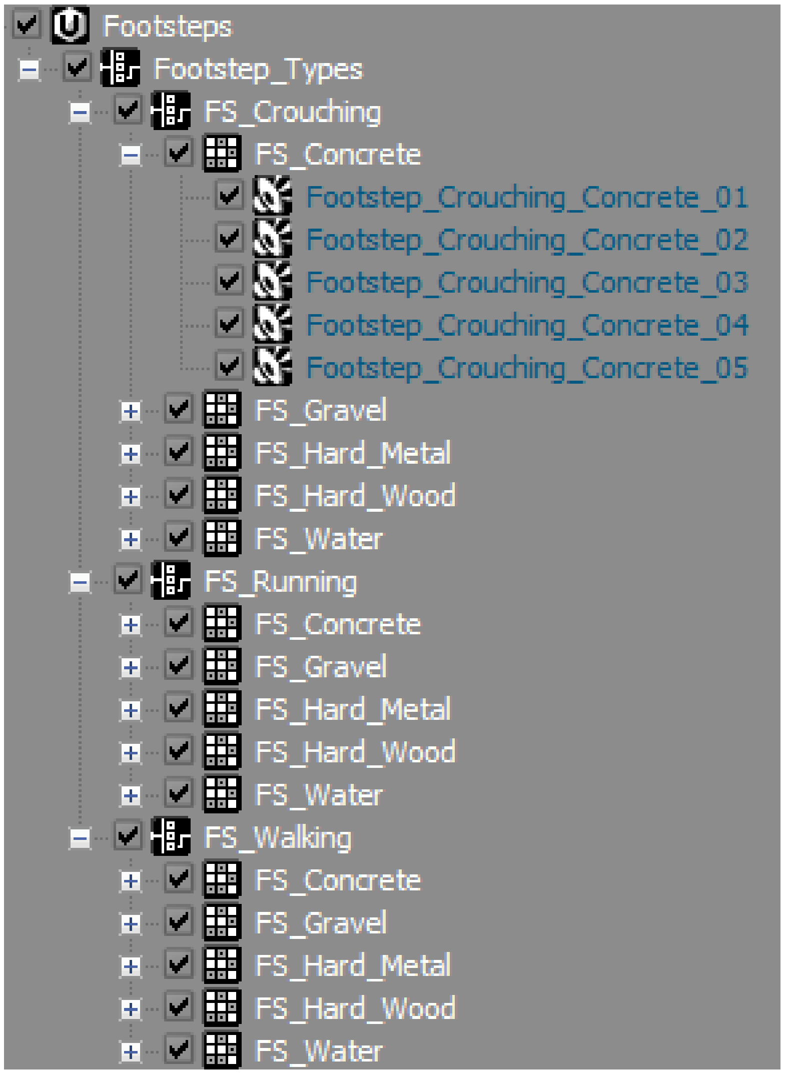
In this example, the parent Switch Container includes three footstep types:

- Crouching
- Walking
- Running

Each of these three footstep types contains another Switch Container that determines the surfaces on which the characters will walk:

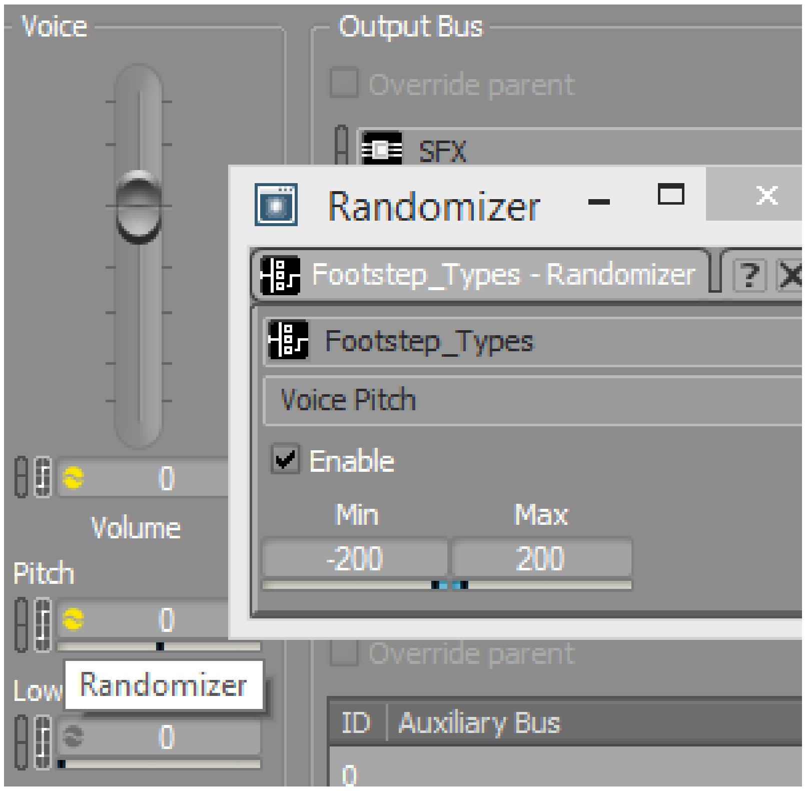
- Concrete
- Gravel
- Hard Metal
- Hard Wood
- Water

To create diversity for each footstep type to material association, a Random Container containing between four and six variations has been created for each variation. The complete hierarchy looks like this:



```
Footsteps
  Footstep_Types
    FS_Crouching
      FS_Concrete
        Footstep_Crouching_Concrete_01
        Footstep_Crouching_Concrete_02
        Footstep_Crouching_Concrete_03
        Footstep_Crouching_Concrete_04
        Footstep_Crouching_Concrete_05
      FS_Gravel
      FS_Hard_Metal
      FS_Hard_Wood
      FS_Water
    FS_Running
      FS_Concrete
      FS_Gravel
      FS_Hard_Metal
      FS_Hard_Wood
      FS_Water
    FS_Walking
      FS_Concrete
      FS_Gravel
      FS_Hard_Metal
      FS_Hard_Wood
      FS_Water
```

To further avoid repetition, volume and pitch is randomized each time a new footstep is played. To see the random ranges, double click on the volume or pitch randomizer icon on the top-level Switch Container named “Footstep_Types”.



Footstep_Types

Voice Pitch

✓ Enable

Min	Max
-200	200

Example: Minigun

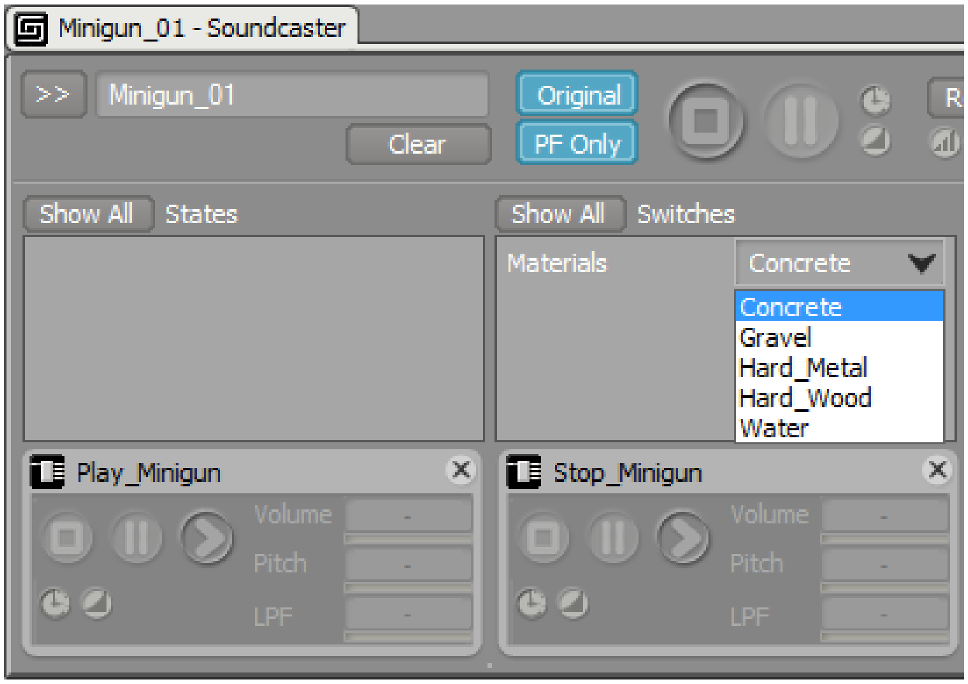
The Minigun example demonstrates how the sounds of a complex firearm can be simulated in a game by using Wwise.

To listen to the Minigun example:

1. In the Sessions tab of the Project Explorer, go in the Soundcaster Sessions section, expand the Minigun Work Unit and double-click the Minigun Soundcaster session.

The session is loaded into the Soundcaster.

2. You can then do any of the following:
   - Play the “Play_Minigun” Event.
   - Play the “Stop_Minigun” Event.
   - Select a material from the list to hear the shells falling on different materials.



Minigun Details: Events

To understand the construction of the Minigun example, take some time to examine the Play and Stop Minigun Events. Look at the delays that have been inserted on the different Actions. Don’t hesitate to unselect the platform (PF) check boxes on both Events to be able to listen to the three different stems (barrel, fire, and shells) independently.

For the Stop_Minigun Event, the Break action has been used to stop the “Fire” and “Shells” random continuous containers that use the “trigger rate” transition type. The Break actions stop the containers to trigger new fire or shell sounds without cutting the tail of the sounds that have already been triggered.

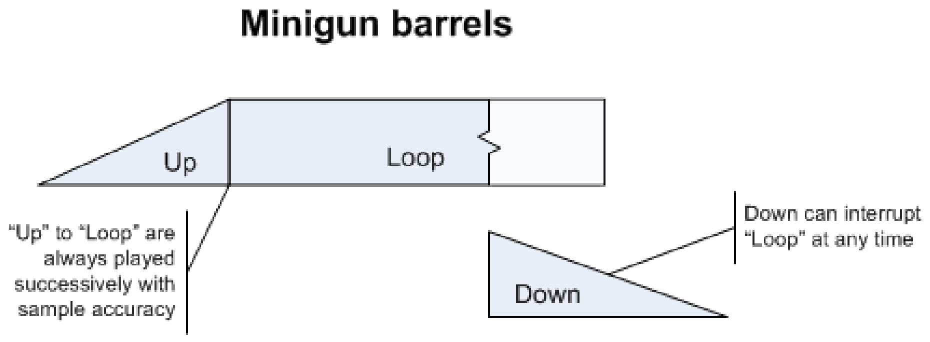
Note

The Events in the Minigun example have been simplified for this sample project. In a real game, refinements would be made to produce a more realistic simulation. For example, the Play_Minigun Event would be split into two separate Events for the barrel and fire sounds, so that each could be controlled more closely.

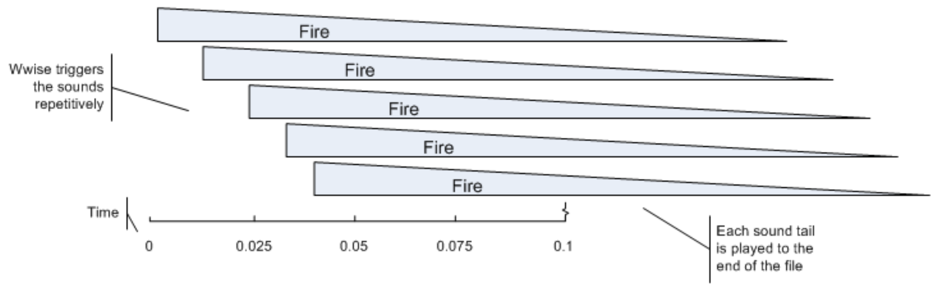
Minigun Details: Sounds

The minigun sounds are divided into three categories:

- **Barrel sounds:** The minigun barrel movements are represented by three sounds: “Up”, “Loop,” and “Down”. The “Up” and “Loop” sounds have been integrated into the “Minigun_Barrel_Start” sequence container using a sample accurate transition. The “Down” sound is played when the “Stop_Minigun” Event is triggered.


Minigun barrels
Up
Loop
"Up" to "Loop" are always played successively with sample accuracy
Down
Down can interrupt "Loop" at any time

- **Fire sounds:** The minigun discharge sounds are included in the “Fire” Random Container. This container is set to trigger a new fire sound every 0.05 seconds in an infinite loop. The Trigger rate transition option ensures that a new sound is triggered at the frequency defined in the Duration field without interrupting the previously played sounds. A limit of five instances per game object has been set in the Fire sounds Advanced Settings to restrict CPU usage.


Fire
Fire
Fire
Fire
Fire
Wwise triggers the sounds repetitively
Time
0
0.025
0.05
0.075
0.1
Each sound tail is played to the end of the file

Note

The trigger rate is not always rhythmically perfect when the “Fire” container is played back within Wwise when the transport is set to play the original audio files. This happens because original files are streamed from the hard drive. This is a playback limitation within Wwise only; when played back from a game, the timing of each fire sounds is sample accurately perfect and constant.

You can get better timing from Wwise by unselecting the **Original** button in the transport. Beforehand, make sure you convert the audio files (select 'Convert All Audio Files...' from the Project menu) as you'll hear nothing otherwise.

- **Shell sounds:** The sounds of the minigun’s brass hitting the ground is simulated by five Random Containers set to infinitely trigger a new shell sound every 0.1 seconds. These objects are inserted into a Switch Container to play the right Random Container based on the actual material onto which the shells will fall.

  Note that the frequency of the trigger rate for the shells is slower than the fire rate. This decision was made simply because the slower rate sounded better.

  A limit of five instances per game object has been set in the Shells sounds Advanced Settings to restrict CPU usage.
- **Using MIDI**: For optimal precision, you can use MIDI to achieve a sample-accurate trigger rate. However, this is an advanced option and requires use of the SDK. For more information, refer to Simulating Rapid Gunfire.

Example: Car Engine

Car Engine demonstrates two examples of how car engines can be reproduced using Wwise. The first example uses the "old-school" approach of pitch shifting a series of recording with a blend container. The other example uses REV, a source plug-in develop by Crankcase Audio that uses granular synthesis and runtime DSP to reproduce car engines.

Porsche 911 - Blend Container

To create this simulation, two main operations were performed. First, a pitch curve was created for RPM for each sound to link pitch to RPM. Next, these sounds were placed into a blend container to define which would play within defined RPM ranges.

To listen to the car engine example:

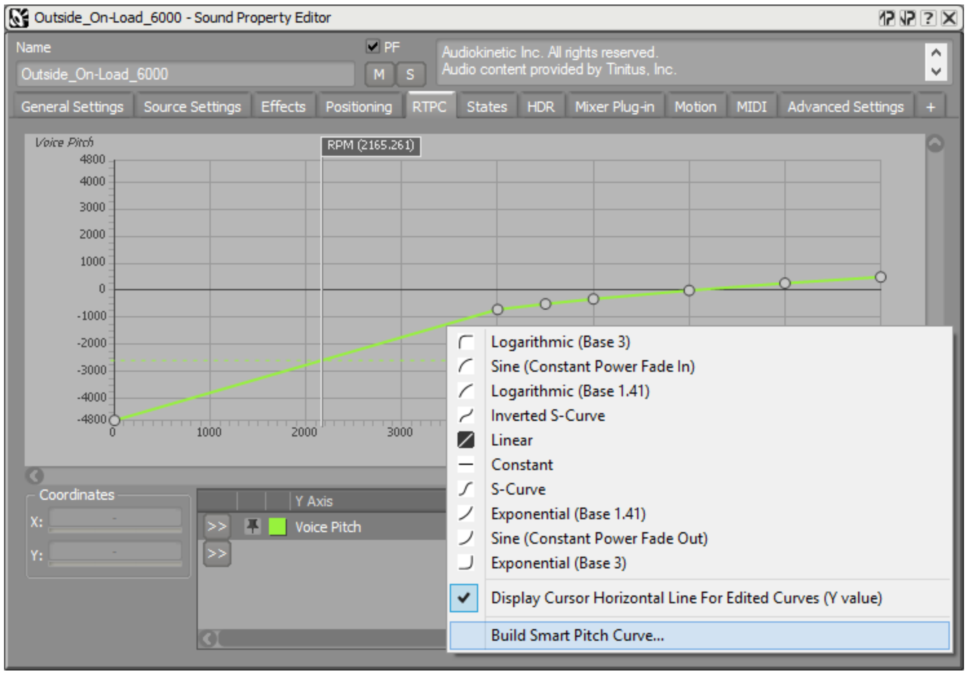
1. Load the “Engine” blend container or the “Play_Porsche_911” Event in the Transport Control.
2. Click RTPCs to modify the Engine_Load and RPM game parameters.
3. Begin with the Engine_Load at “1” and the RPM at “1000”, then move the RPM value slowly.

Note

To hear even better audio results, you can use the CarSim application located where you installed Wwise in Samples \SoundFrame\CarSim.

Car Engine Details: Matching Pitch to RPM Using RTPCs

Each of the engine sounds (except idle) has a pitch curve attached to RPM game parameter. To speed up the process, Wwise allows you to automatically draw a pitch curve based on the actual RPM at which a file was recorded. To use it, right-click any graph segment and select Build Smart Pitch Curve. For more information on this or any other feature, refer to the Wwise Help.



You might have noticed that while certain engine sounds will not play at certain RPM values, each sound’s pitch curve still extends for the full range of possible RPM values. In this case, because the curves extend past values that will be used, points have been removed to save runtime memory.

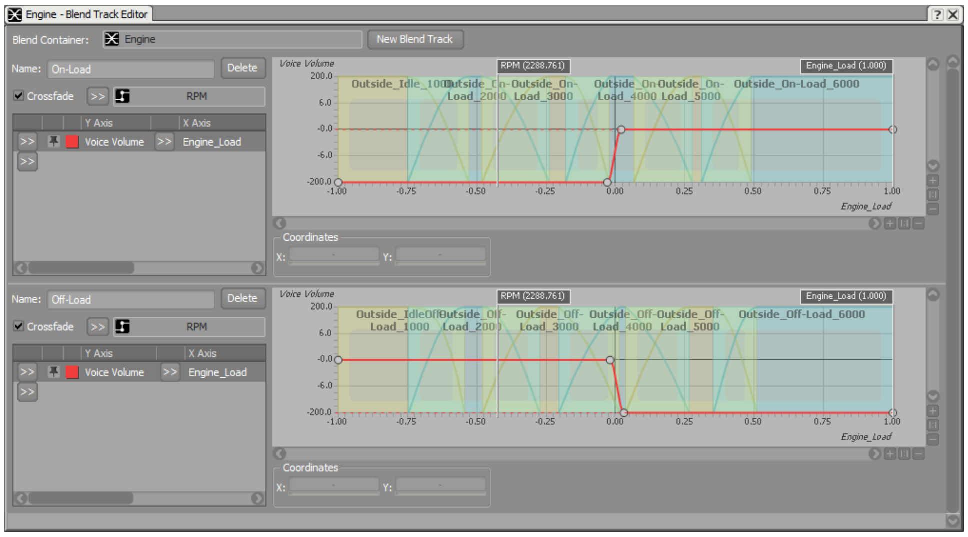
Car Engine Details: Using Blend Containers

The car engine simulation consists of six looping sounds revolving at different RPM on-load values (on-load meaning that the gas pedal is pressed) and six other sounds matching the same RPM off-load (off-load meaning the gas pedal is released). These sounds were inserted into the blend container on two blend tracks named On-Load and Off-Load. A crossfade region was defined between each of these sounds to ensure a smooth transition between them for when the RPM game parameter changes in real time.

A volume curve attached to the game parameter "Engine_Load" was created for each blend track. This curve, which is displayed in red, defines which blend track is heard based on the current Engine_Load value:

- From -1 to ~0: the Off-Load track is heard.
- From ~0 to 1: the On-Load track is heard.

Therefore, as the game updates the current Engine_Load game parameter, the corresponding curve is heard.



Engine - Blend Track Editor

Blend Container: Engine

Name: On-Load
Crossfade: ✓ RPM

Y Axis	X Axis
Voice Volume	Engine_Load

Name: Off-Load
Crossfade: ✓ RPM

Y Axis	X Axis
Voice Volume	Engine_Load

Chevrolet Camaro - Crankcase Audio REV

Crankcase Audio REV is a source plug-in using granular synthesis and runtime DSP.

Note

Mac Authoring Tool: Users cannot use Crankcase Audio REV.

To listen to Camaro SS - CrankcaseAudio REV example:

1. Select the sound Camaro SS - CrankcaseAudio REV in the Project Explorer.

2. From the Content Editor, double-click the source plug-in "Chevrolet Camaro SS" to display the Source Editor view.
3. Play the sound.
4. Click on the **Drive** button in the REV source plug-in and move the cursor left and right to 'drive' the vehicle.

Note

This example shows the plug-in doing the full car engine simulation. That may suit certain games just right. For games needing more control over the car simulation, it's possible to control the car throttle, RPM, gear and velocity using game parameters. Again, refer to the plug-in documentation if you'd like more details on this.

Example: New York City Ambience

The New York City Ambience example shows how you can create a complex and evolving soundscape. This example uses a blend container to alter game sounds based on two parameters: Time of day and Rain Intensity.

To listen to the New York City Ambience example:

1. Load the “24h New York City Ambience” Blend Container or the “Play_24h_New_York_City_Ambience” Event into the Transport Control.
2. In the Transport Control, click RTPCs.
3. Set “Rain_Intensity” and “Time” to 0.
4. Start the playback and move the “Time” game parameter very slowly to the right. “Time” ranges from 0 to 240, so ten units represent one hour.
5. Move the “Rain_Intensity” game parameter slowly to hear the rain falling on the city.

   The ambient sounds subtly disappear as the rain becomes stronger. The Rain_Intensity game parameter also modifies the volume and low pass filter properties of many of the objects inserted on certain blend tracks.

A single blend container containing nine tracks has been used for this example. Each of the blend tracks contains one or several sounds or Random Containers. Before looking at the Blend Track Editor, you can look at the behavior settings of each Random Container to understand how each of these has been set up. The transitions options should be of particular interest.

Example: SoundSeed Impact

Game audio content creators strive to create realistic and immersive audio soundscapes for games. Compelling audio content requires many sound variations to help reduce repetition which can contribute to several negative aspects of gameplay, including player fatigue and a loss of realism and player immersion.

Unfortunately, the need for many sound variations results in a high memory consumption to accommodate multiple sound asset variations. SoundSeed Impact allows you to create many unique sound variations at runtime using a single audio source file, thus reducing the impact on storage and memory.

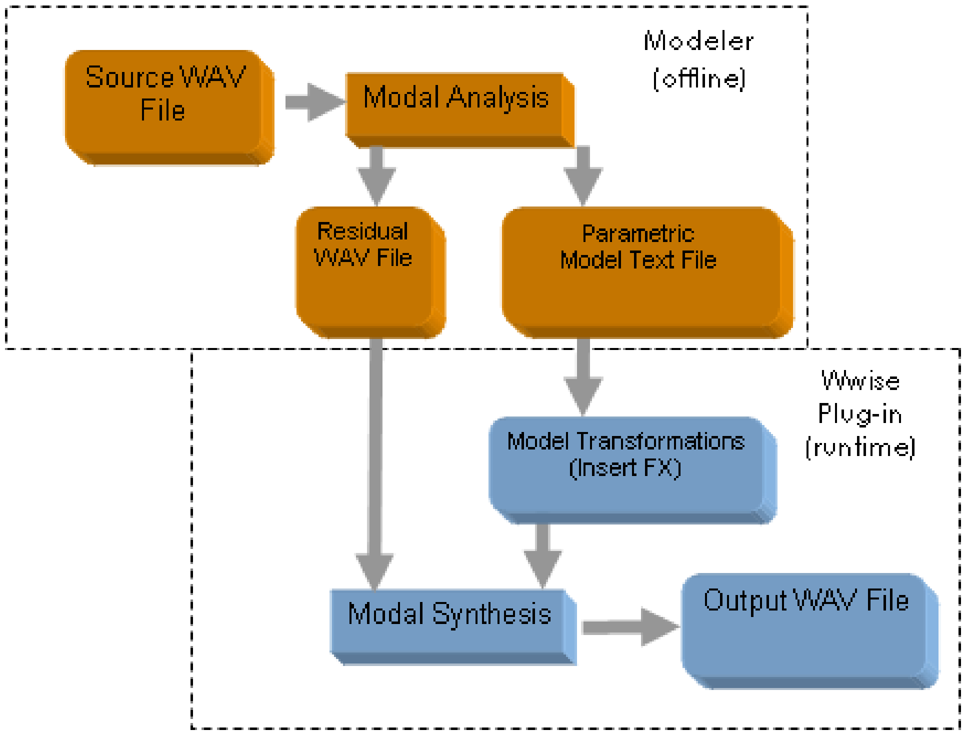
SoundSeed Impact models a wide range of resonant impact type sounds, which means you can use it to create many different sounds in your game. Furthermore, the compromise between audio quality and performance can be controlled at runtime along with other model parameter transformations that allow you to dynamically control a sound's characteristics in real time.

The main advantages of using SoundSeed Impact are to:

- Increase sound variety through synthesis of multiple sound instances from a single template sound.
- Reduce memory requirements by storing compact parameter sets instead of entire audio files for each variation.
- Increase interactivity by controlling synthesis parameters based on context-specific (possibly physics related) run-time parameters.

Technology Overview

The SoundSeed Impact technology has two distinct processing stages: offline analysis using the SoundSeed Impact Modeler, and runtime sound rendering using the SoundSeed Impact Wwise plug-in.



Offline Processing

Offline analysis is performed using the SoundSeed Impact Modeler, which is a standalone external application. During offline analysis, you detect and remove resonance information from the source audio file. Each mode of resonance is characterized by a series of parameters, including frequency (Hz), magnitude (dB) and bandwidth (Hz). The offline analysis stage generates the following components:

- A residual WAV file, which is based on the source WAV file, but with the detected resonances removed.
- A parametric model text file, which is a very compact .SSM (SoundSeed Model) file that contains information in the form of various parameters, about the resonances.

Runtime Processing

During the runtime processing stage, the information in the resonance file is re-introduced with the residual file using efficient frequency filtering. This creates a new sound that retains the characteristics of the original analyzed file. What makes it so powerful is that you can modify the resonance parameters within the text file during re-synthesis to generate an unlimited number of variations that are slightly or largely different from the analyzed sound.

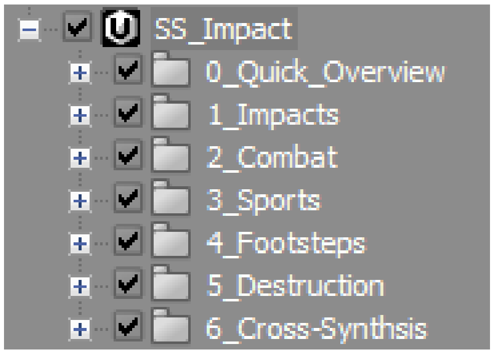
Runtime processing is performed in Wwise using the SoundSeed Impact effect plug-in. Using both the residual file and the .SSM model file, the Wwise SoundSeed Impact plug-in allows you to do the following:

- Synthesize the output sound using the transformed model. The resonances are reintroduced through time-domain signal filtering of the residual sound.
- Modify the characteristics of the detected resonances by transforming model parameters.

  For more detailed information on using the SoundSeed Modeler or the Wwise plug-in, consult the Audiokinetic web site for additional resources.

SoundSeed Impact: Details

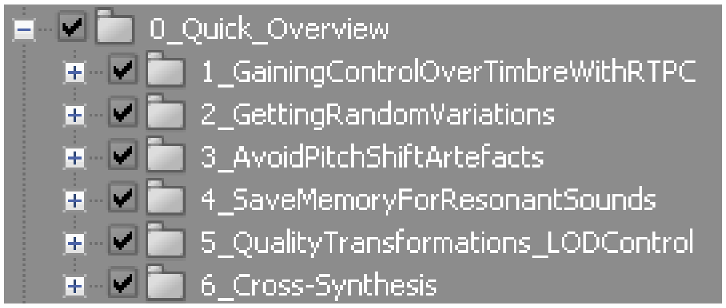
The Sample Project features some practical examples on how SoundSeed Impact can be used. The SS_Impact Work Unit of the Sample Project is broken down into the following folders:


SS_Impact
0_Quick_Overview
1_Impacts
2_Combat
3_Sports
4_Footsteps
5_Destruction
6_Cross-Synthsis

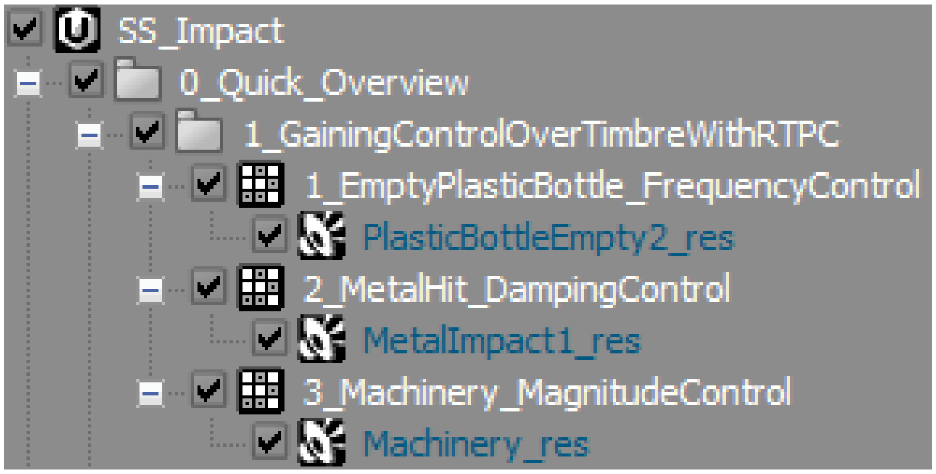
Be sure to check the "Notes" field of the various sound elements to read additional information.

Quick Overview

This folder contains a series of subfolders that give you a general overview of using SoundSeed Impact in your game productions.

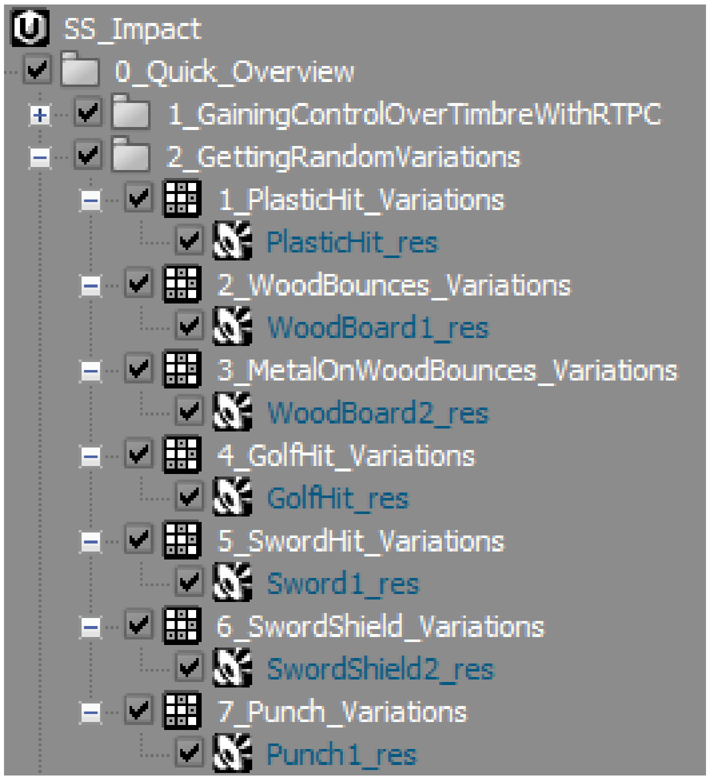

0_Quick_Overview
1_GainingControlOverTimbreWithRTPC
2_GettingRandomVariations
3_AvoidPitchShiftArtefacts
4_SaveMemoryForResonantSounds
5_QualityTransformations_LODControl
6_Cross-Synthesis

- 1_GainingControlOverTimbreWithRTPC—this folder contains examples that show how to control the SoundSeed Impact Synthesis parameters in real time using RTPCs.

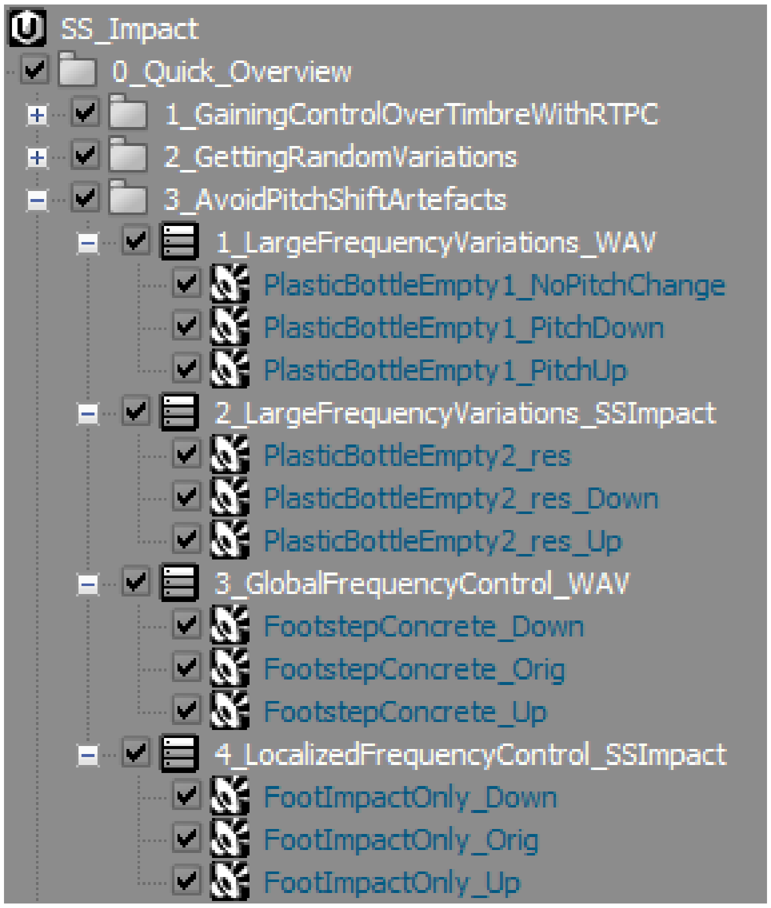

SS_Impact
0_Quick_Overview
1_GainingControlOverTimbreWithRTPC
1_EmptyPlasticBottle_FrequencyControl
PlasticBottleEmpty2_res
2_MetalHit_DampingControl
MetalImpact1_res
3_Machinery_MagnitudeControl
Machinery_res

- 1_EmptyPlasticBottle_FrequencyControl—demonstrates the effect of frequency transformation in isolation. Play with the "PlasticBottle_Frequency" RTPC slider in the Transport Control to affect the perceived size of the bottle without getting pitch shift artifacts or duration changes.
- 2_MetalHit_DampingControl—demonstrates the effect of bandwidth transformation in isolation. Play with the "MetalHit2_Resonance" RTPC slider in the Transport Control to see how changes to the bandwidth stretching property can affect the resonance. Reduce the value to 0 for more damping, or increase it to 100 for less damping. Ctrl+click the slider to return it back to the original value of 50. Notice how the sound does not get truncated when using less damping as it would when using a multi-band EQ.
- 3_Machinery_MagnitudeControl—play for a bit and then increase the magnitude variation RTPC slider in the Transport Control to 100 and listen how a random weight on each resonance creates a different impact every time. These changes are akin to different weight or strike position.

• 2_GettingRandomVariations—this folder contains examples that show how virtually an unlimited number of playback variations can be achieved from a single source file. Play the Random Container for each example.

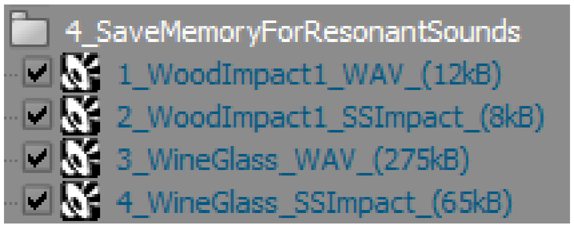


- 3_AvoidPitchShiftArtefacts—this folder contains examples that demonstrate how pitch shifting is often not possible for large pitch changes due to the undesirable time-scaling it introduces. These examples also show how frequency variations can contribute greatly to sonic variations.



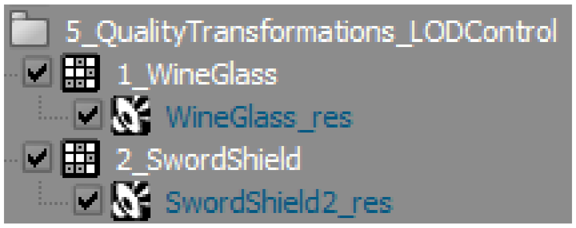
SS_Impact
0_Quick_Overview
1_GainingControlOverTimbreWithRTPC
2_GettingRandomVariations
3_AvoidPitchShiftArtefacts
1_LargeFrequencyVariations_WAV
PlasticBottleEmpty1_NoPitchChange
PlasticBottleEmpty1_PitchDown
PlasticBottleEmpty1_PitchUp
2_LargeFrequencyVariations_SSImpact
PlasticBottleEmpty2_res
PlasticBottleEmpty2_res_Down
PlasticBottleEmpty2_res_Up
3_GlobalFrequencyControl_WAV
FootstepConcrete_Down
FootstepConcrete_Orig
FootstepConcrete_Up
4_LocalizedFrequencyControl_SSImpact
FootImpactOnly_Down
FootImpactOnly_Orig
FootImpactOnly_Up

- 4_SaveMemoryForResonantSounds—this folder contains examples that demonstrate the possible runtime memory savings that are possible for highly resonant sounds.



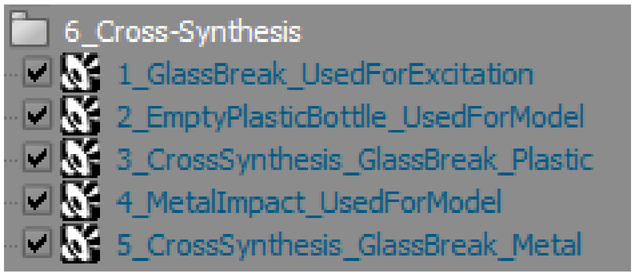
4_SaveMemoryForResonantSounds
1_WoodImpact1_WAV_(12kB)
2_WoodImpact1_SSImpact_(8kB)
3_WineGlass_WAV_(275kB)
4_WineGlass_SSImpact_(65kB)

- 5_QualityTransformations_LODControl—this folder contains examples that show how an RTPC can be attached to the "Model Quality" parameter to reduce the number of harmonics effectively synthesized to change the sound quality. This can also be used as a level of detail control in the 50-100 range. For example, a value of 50 only uses half the processing power and may be suitable for background sounds.



5_QualityTransformations_LODControl
1_WineGlass
WineGlass_res
2_SwordShield
SwordShield2_res

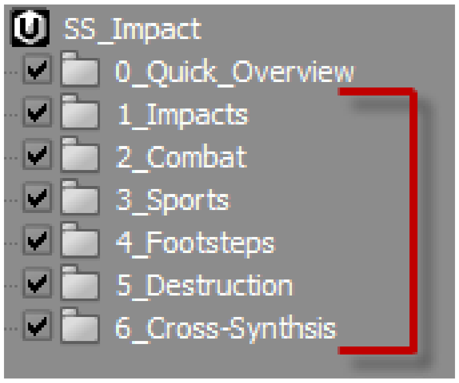
- 6_Cross-Synthesis—this folder contains examples that show some creative uses of SoundSeed Impact. In these examples, data model files from different sources are combined with residual files.



6_Cross-Synthesis
1_GlassBreak_UsedForExcitation
2_EmptyPlasticBottle_UsedForModel
3_CrossSynthesis_GlassBreak_Plastic
4_MetalImpact_UsedForModel
5_CrossSynthesis_GlassBreak_Metal

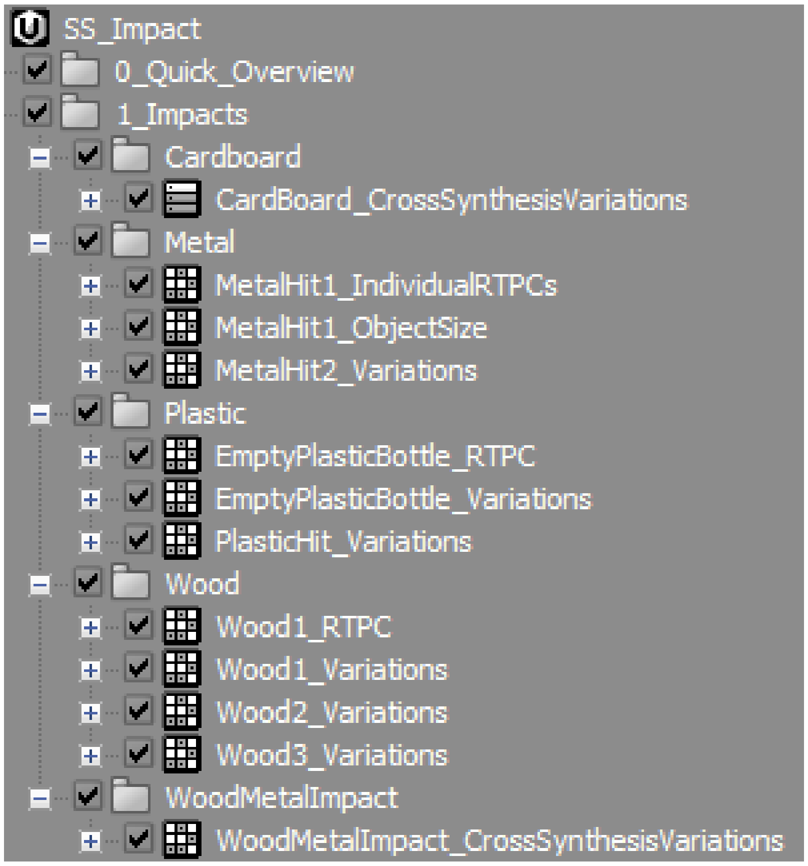
Detailed Examples

The remaining folders contain more elaborate examples of typical uses of SoundSeed Impact.



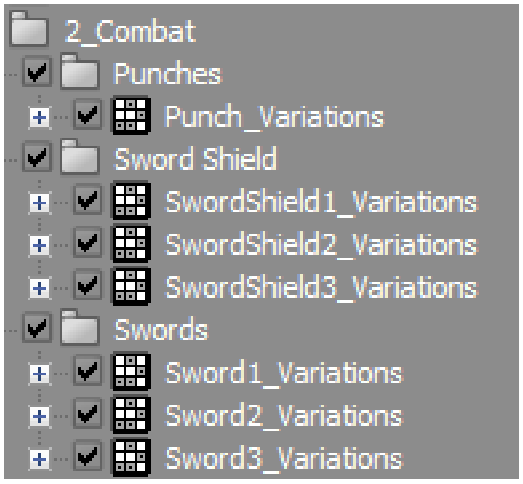
SS_Impact
0_Quick_Overview
1_Impacts
2_Combat
3_Sports
4_Footsteps
5_Destruction
6_Cross-Synthsis

- 1_Impacts—this folder contains examples of transformations using a variety of different materials.



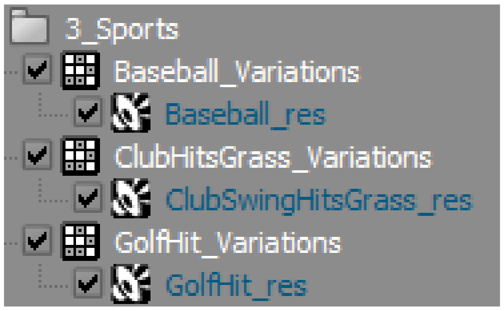
SS_Impact
- 0_Quick_Overview
- 1_Impacts
  - Cardboard
    - CardBoard_CrossSynthesisVariations
  - Metal
    - MetalHit1_IndividualRTPCs
    - MetalHit1_ObjectSize
    - MetalHit2_Variations
  - Plastic
    - EmptyPlasticBottle_RTPC
    - EmptyPlasticBottle_Variations
    - PlasticHit_Variations
  - Wood
    - Wood1_RTPC
    - Wood1_Variations
    - Wood2_Variations
    - Wood3_Variations
  - WoodMetalImpact
    - WoodMetalImpact_CrossSynthesisVariations

- 2_Combat—this folder contains examples of transformations possible for combat class type sounds. Play the Random Containers to hear the different sound variations.

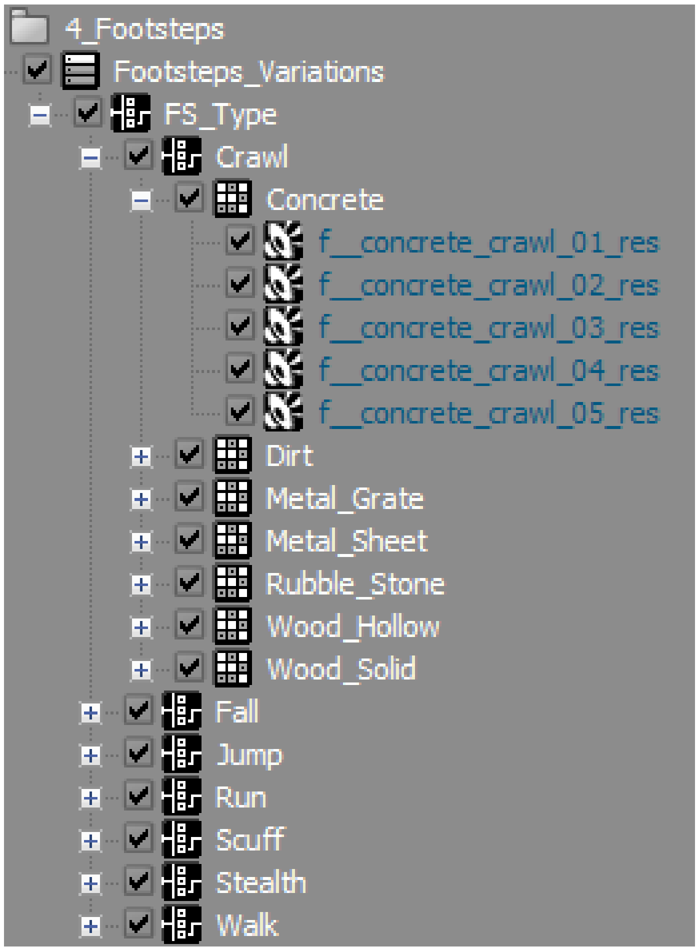


2_Combat
- Punches
  - Punch_Variations
- Sword Shield
  - SwordShield1_Variations
  - SwordShield2_Variations
  - SwordShield3_Variations
- Swords
  - Sword1_Variations
  - Sword2_Variations
  - Sword3_Variations

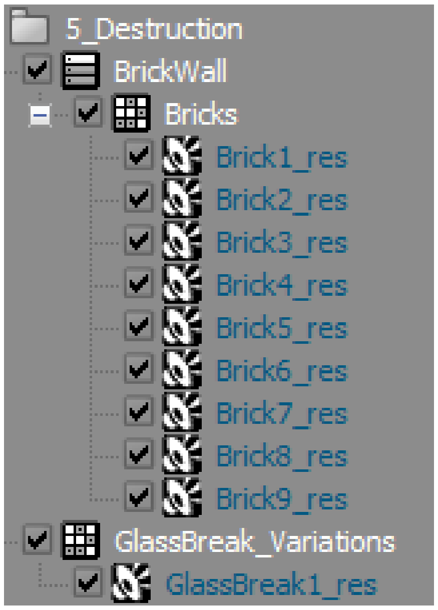
- 3_Sports—this folder contains examples of transformations possible for sports type sounds. Play the Random Containers to hear the different sound variations.



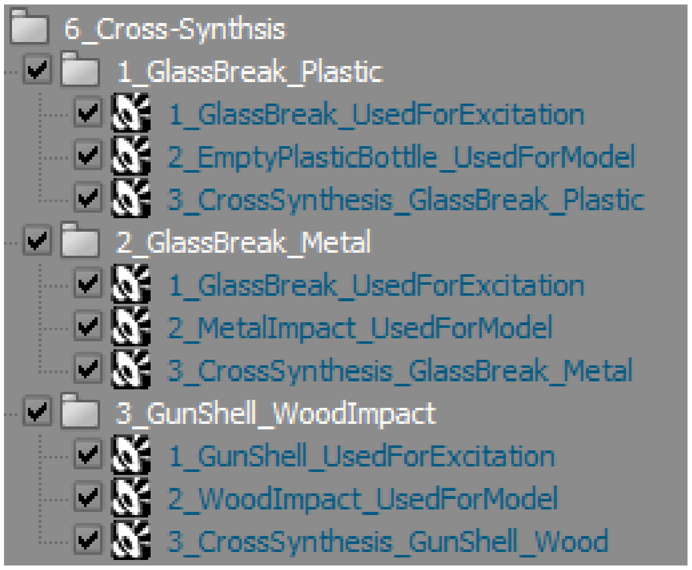
- 4_Footsteps—this folder contains a dramatic example of the potential of SoundSeed Impact. Here we can see a series of nested Switch Containers, Random Containers and residual sounds/SoundSeed impact plug-ins. Be sure to select the **Switches** button in the Transport Control so that you can switch between the available footstep types and surface materials during playback.



- 5_Destruction—this folder contains an example of the destruction of a brick wall. It demonstrates how SoundSeed Impact can be used to obtain more variations on individual grains in a granular synthesis setting. Additionally you can gain control over the sound of the brick wall destruction by changing, for example, the brick resonance RTPC control.



- 6_Cross-Synthesis—this folder contains other examples of the creative use of cross-synthesis using SoundSeed Impact.



6_Cross-Synthsis
1_GlassBreak_Plastic
1_GlassBreak_UsedForExcitation
2_EmptyPlasticBottlle_UsedForModel
3_CrossSynthesis_GlassBreak_Plastic
2_GlassBreak_Metal
1_GlassBreak_UsedForExcitation
2_MetalImpact_UsedForModel
3_CrossSynthesis_GlassBreak_Metal
3_GunShell_WoodImpact
1_GunShell_UsedForExcitation
2_WoodImpact_UsedForModel
3_CrossSynthesis_GunShell_Wood

Example: Dialogue

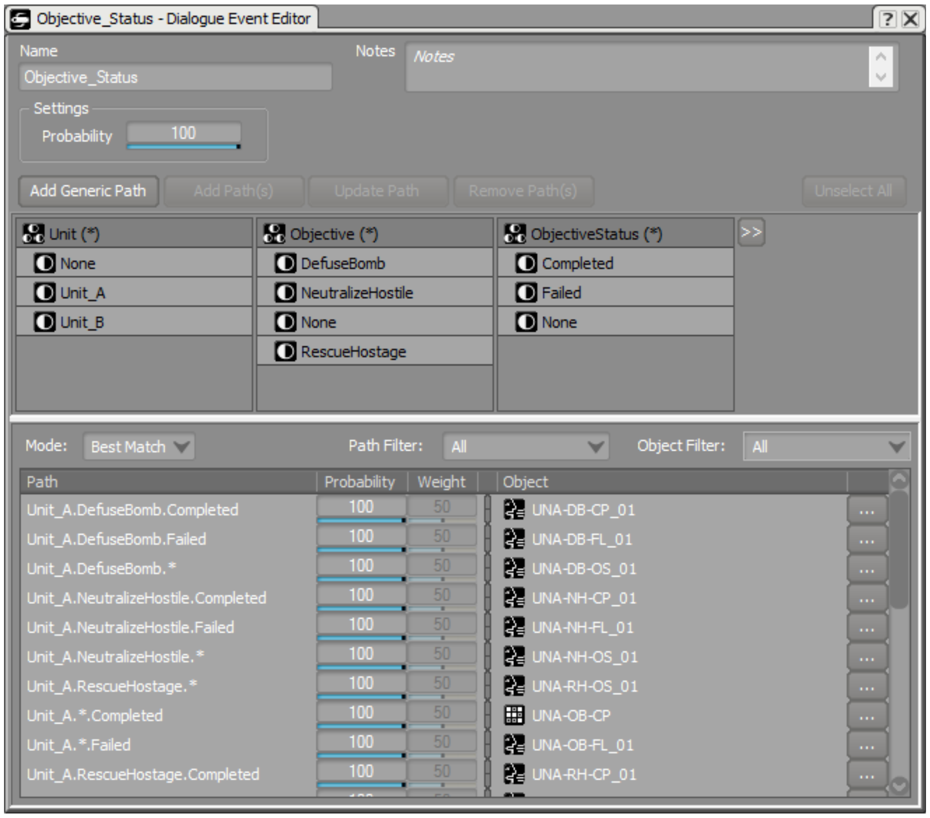
The Dialogue example demonstrates how you can use Wwise's dynamic dialogue features to organize dialogue that adapts to gameplay situations as they happen. Dynamic dialogue uses a set of rules within a decision-tree structure to determine which piece of dialogue to play at any particular moment in game.

In the sample project's game, the main player character has to work with NPCs to accomplish mission objectives. The main player can radio the NPCs, who in this case are two on-duty police officers. Through their radio responses, the two officers report back with details of their current assignment and status. For example, the officers can report that they are rescuing a hostage, then later report on whether their rescue was successful.

To listen to the Dialogue example:

1. In the Events tab of the Project Explorer, locate the Dialogue Work Unit in the Dynamic Dialogue section and double-click the Objective_Status Dialogue Event.

   The Objective_Status Dialogue Event is loaded into the Dialogue Event Editor.



Objective_Status - Dialogue Event Editor

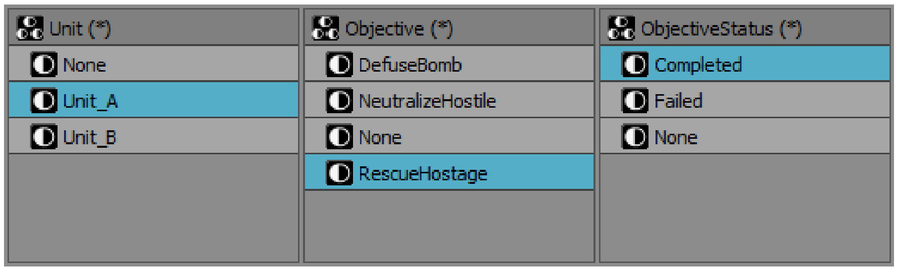
Name: Objective_Status
Notes: Notes
Settings
Probability: 100

Unit (*)	Objective (*)	ObjectiveStatus (*)
None	DefuseBomb	Completed
Unit_A	NeutralizeHostile	Failed
Unit_B	None	None
	RescueHostage	

Mode: Best Match
Path Filter: All
Object Filter: All

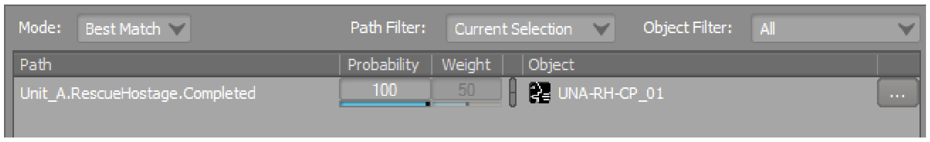
Path	Probability	Weight	Object
Unit_A.DefuseBomb.Completed	100	50	UNA-DB-CP_01
Unit_A.DefuseBomb.Failed	100	50	UNA-DB-FL_01
Unit_A.DefuseBomb.*	100	50	UNA-DB-OS_01
Unit_A.NeutralizeHostile.Completed	100	50	UNA-NH-CP_01
Unit_A.NeutralizeHostile.Failed	100	50	UNA-NH-FL_01
Unit_A.NeutralizeHostile.*	100	50	UNA-NH-OS_01
Unit_A.RescueHostage.*	100	50	UNA-RH-OS_01
Unit_A.*.Completed	100	50	UNA-OB-CP
Unit_A.*.Failed	100	50	UNA-OB-FL_01
Unit_A.RescueHostage.Completed	100	50	UNA-RH-CP_01

2. In the Dialogue Event Editor, select an argument from each column as follows:
   - In the Unit column, select either Unit_A or Unit_B.
   - In the Objective column, select either DefuseBomb, NeutralizeHostile, or RescueHostage.
   - In the ObjectiveStatus column, select either Completed or Failed.



Unit (*)	Objective (*)	ObjectiveStatus (*)
None	DefuseBomb	Completed
Unit_A	NeutralizeHostile	Failed
Unit_B	None	None
	RescueHostage	

   Each argument you choose is highlighted
3. In the Argument Path Filter list, select Current Selection.



Mode: Best Match Path Filter: Current Selection Object Filter: All

Path	Probability	Weight	Object
Unit_A.RescueHostage.Completed	100	50	UNA-RH-CP_01

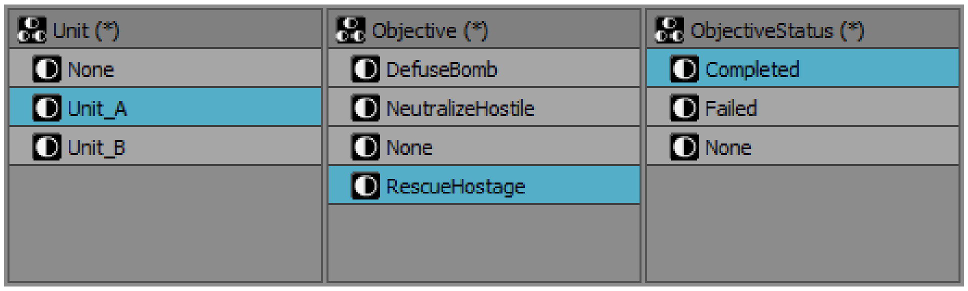
   The selected argument path and its associated object are displayed.
4. In the Transport Control, click Play to hear the object.

   The object associated with the selected argument path is played.
5. To listen to other argument paths in the dialogue example, repeat Steps 3 to 5.

Dialogue Details: Understanding Dynamic Dialogue

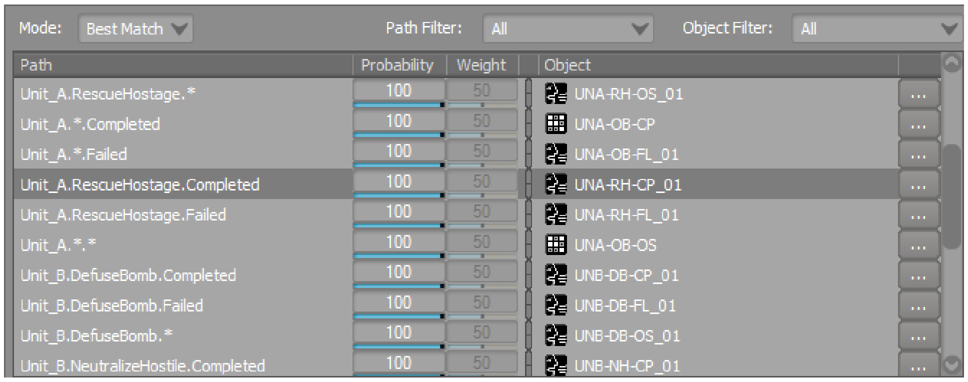
Dynamic dialogue is a flexible tool for creating responses on the fly. These responses are determined through a decision-tree structure that selects audio based on multiple game conditions. The example of dynamic dialogue provided with the sample project shows how to implement dialogue functionality in a game that uses game variables to trigger the particular sentences. Dynamic Dialogue is particularly well suited to create play-by-play for sports games, but any game genres could see advantages to use this system. It can also be used for SFX in situations where large hierarchies are needed like footsteps structures for example.

In this project, in the Objective_Status Dialogue Event example, a series of arguments and arguments values has been created to reflect multiple game situations and outcomes. These arguments create a matrix of argument paths that can be triggered at any moment by the player's radio call to the police officers.



Unit (*)	Objective (*)	ObjectiveStatus (*)
None	DefuseBomb	Completed
Unit_A	NeutralizeHostile	Failed
Unit_B	None	None
	RescueHostage	

In this project, in the Objective_Status Dialogue Event example, a series of arguments and arguments values has been created to reflect multiple game situations and outcomes. These arguments create a matrix of argument paths that can be triggered at any moment by the player's radio call to the police officers.



Mode: Best Match Path Filter: All Object Filter: All

Path	Probability	Weight	Object
Unit_A.RescueHostage.*	100	50	UNA-RH-OS_01
Unit_A.*.Completed	100	50	UNA-OB-CP
Unit_A.*.Failed	100	50	UNA-OB-FL_01
Unit_A.RescueHostage.Completed	100	50	UNA-RH-CP_01
Unit_A.RescueHostage.Failed	100	50	UNA-RH-FL_01
Unit_A.*.*	100	50	UNA-OB-OS
Unit_B.DefuseBomb.Completed	100	50	UNB-DB-CP_01
Unit_B.DefuseBomb.Failed	100	50	UNB-DB-FL_01
Unit_B.DefuseBomb.*	100	50	UNB-DB-OS_01
Unit_B.NeutralizeHostile.Completed	100	50	UNB-NH-CP_01

In this game, each time a police officer responds to the player, the game triggers the "Objective_Status" Dialogue Event while setting the actual argument values. Consequently, the object associated with the argument path, either a voice object or a container holding sound objects, is heard. This example doesn't take advantage of the Probability and Weighting tools, but you can use them in your game to have more control over which piece of audio is played, if any.

Fallback Mechanism

When you create dynamic dialogue for a game, you may realize that your game can't always provide all the information you need at all times. In this example, when the player radios an officer for his status, the officer NPC might be between actions. To deal with this possibility, the dynamic dialogue system includes a fallback mechanism. The fallback mechanism allows you to trigger generic dialogue for situations where an argument path does not have an associated object, or when an argument value in the argument path is not specified by the game.

For example, the sample project contains the following fallback lines:

- Unit_B.*.Completed: "Mission accomplished. Nice job everyone."

- Unit_B.*.Failed: "Let's do better next time, OK?"

Note

When an argument is used to create a fallback argument path, the argument is represented by an asterisk in the path name. In the example above, the asterisk represents the argument: "Objective".

Compare these to specific lines such as:

- Unit_B.RescueHostage.Completed: "We got the hostage."
- Unit_B.DefuseBomb.Failed: "We can't stop the bomb. Everyone out!"

The fallback mechanism is a really important aspect because it provides a "safety net" for the scriptwriter. These lines could be delivered in cases in which the game does not specify a particular objective. They could also be useful in a situation where new objectives are added to the game after the voice recordings are completed. Because no specific dialogue exists for the new objectives, the fallback dialogue will be selected automatically.

Example: Ambisonics

The Ambisonics Work Unit in the Actor-Mixer Hierarchy includes three examples of moving ambient sounds to demonstrate the spatial audio you can expect with ambisonics output. The multichannel setup necessary to do ambisonics justice (you must at least have some height channels) is not a commonly available Sound Engine Audio System. Consequently, these examples are routed to an "Ambisonics" Audio Bus and its parent "Binauralizer" Audio Bus, which uses the Auro Headphone Effect to generate binaural output. Listening to this with just a pair of headphones will faithfully render the spherical directional power of ambisonics according to the selected **Channel Configuration**, which could be set to first, second, or third order ambisonics.

Note

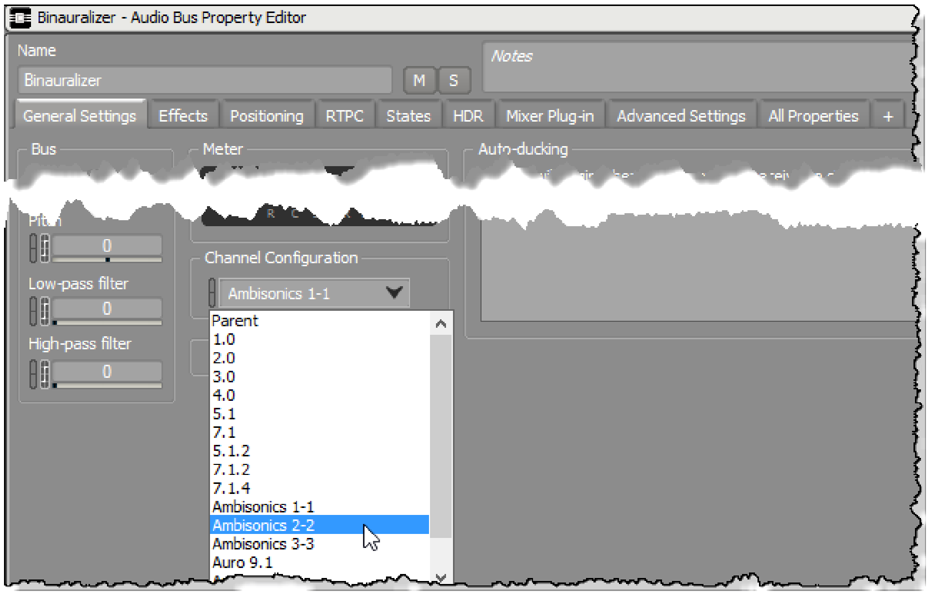
Feel free to open the Effect Editor and tweak the Auro Headphone settings to optimize your personal experience. In practice, your perceived output will vary according to the content.

To listen to different ambisonic channel configurations:

1. Select the "Binauralizer" Audio Bus, found in the Master-Mixer Hierarchy under the Master Audio Bus's Environmental Audio Bus.

It opens in the Property Editor.

2. In the General Settings tab of the Property Editor, select **Ambisonics 2-2** for second order output (9 channels) or **Ambisonics 3-3** for third order output (16 channels).



3. With one of the examples loaded in the Transport Control, press play. Note the differences between the different orders of ambisonic channel configuration outputs.

Tip

Regardless of what you select in the Project Explorer, you can keep the same sound object loaded in the Transport Control by pinning it. Just click the **Pin** icon found in the upper right corner of the view to the left of the **Help** icon.

Each of the following ambisonic examples uses different types of source files arranged to demonstrate a variety of ways to produce an ambisonic output.

Ambisonics Example: Ambisonic City 3D Ambiance

This ambisonic example is just a single first order (4-channel) ambisonic recording imported in Wwise. It gives an idea of how well ambisonics can generate ambient sounds with the traffic rhythms of passing cars and a blaring horn interjection.

As with the other ambisonic examples, users can adjust the “Binauralizer” Audio Bus to listen to the differences in precision when it's set to a second or third order channel configuration. However, with a first order source file, the output will be the same.

Ambisonics Example: Helico Passby Overhead

The “Ex 2 - Helico Passby Overhead” Sound FX is only a mono source file. However, we have set it to have a 3D User-defined path, which helps to simulate the overhead directional sound of a helicopter flying by. More importantly, the 3D positioning allows the ambisonic output to be audibly more precise in higher orders.

You may wish to play with the User-defined path to help you better appreciate the differences in precision.

To adjust the User-defined path:

1. Select the “Ex 2 - Helico Passby Overhead” Sound FX, found in the Actor-Mixer Hierarchy under the “Ambisonics” Work Unit.
   It opens in the Property Editor.
2. In the Positioning tab of the Property Editor, click **Edit...** next to the **User-defined** option of the **Position Source** group box.
   The Position Editor opens.
3. Within the Position Editor, adjust the path as desired by clicking points and dragging them to the desired location. Add points by double-clicking in the Graph view. Refer to the contextual help to learn more about how you can adjust the user-defined path.

You should play the “Ex 2 - Helico Passby Overhead” Sound FX in the Transport Control using different ambisonic channel configurations. The higher order ambisonics configurations deliver more precise sound than the lower order ones.

Ambisonics Example: Ambisonic City Ambiance

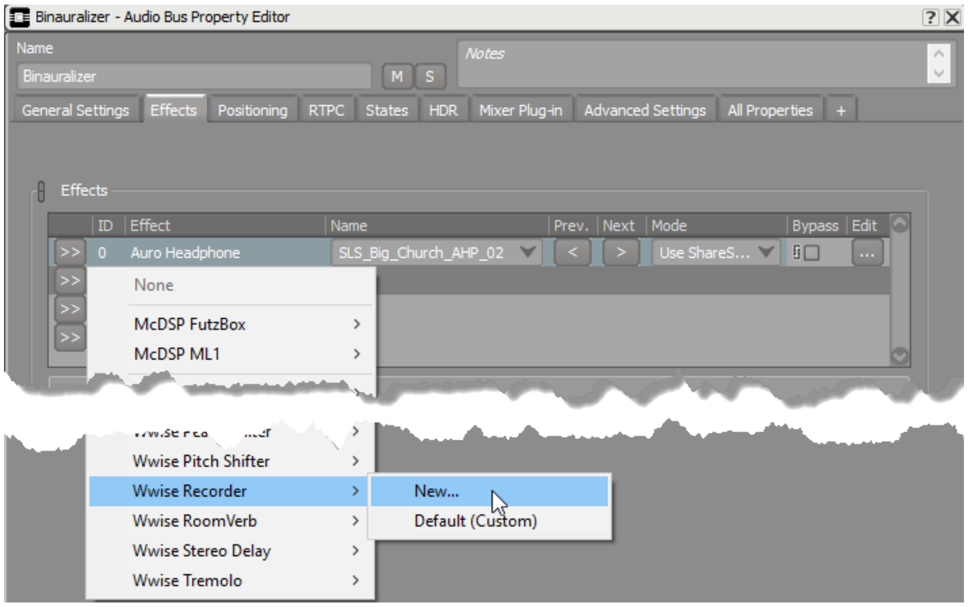
The “Ex 3 - Ambisonic City Ambiance” Blend Container is a mix of multiple Random Containers with mono source files, giving a variety of city atmosphere sounds, and the single first order ambisonics file of our first ambisonics example.

Tip

To help hear the difference in ambisonic orders, mute the first order ambisonics file.

Like the other examples, the channel configurations can be adjusted and played back in the Transport Control to hear how well different orders of ambisonic output produce an encompassing spatial audio. Because there are many other sounds to output to the higher order channels, you will also notice a more significant difference in outputting to second and especially third order. That is, in addition to the four basic channels of the source first order file, you will hear with more precision the additional mono sounds of the Blend Container's city atmosphere.

This can be recorded to a single ambisonics file, for later import or for sharing, using a Wwise Recorder plug-in Effect placed on the "Binauralizer" Audio Bus.



See more about using the Wwise Recorder in the Wwise Plug-ins section of the help.

Example: Music

The Music Work Unit in the Interactive Music Hierarchy includes two examples of game music to demonstrate both a basic and an advanced integration of interactive music. These interactive music examples show how you can create music that evolves through three distinct States representing the level of intensity of the gameplay:

- **Stealth:** No threat in sight. The player moves quietly through the environment.
- **Stress:** The player is in danger and should hide or escape to avoid trouble.
- **Fight:** Action sequence where the player is attacked by bad guys and fights for his life.

Game Music Details: Looking at Interactive Music

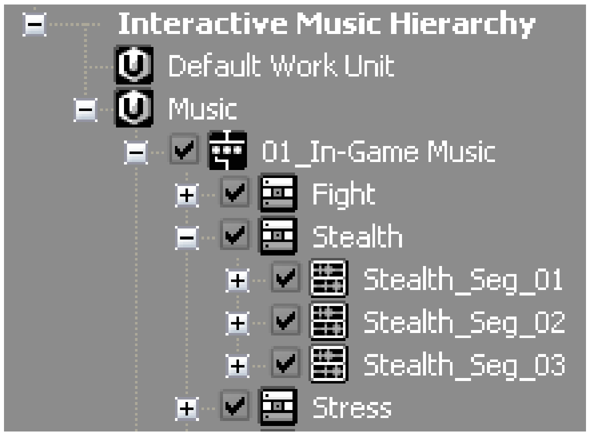
The interactive music view allows you to examine game music at different levels of detail. The following procedure should show you how to take advantage of Wwise's interface when looking at one particular project element.

Note

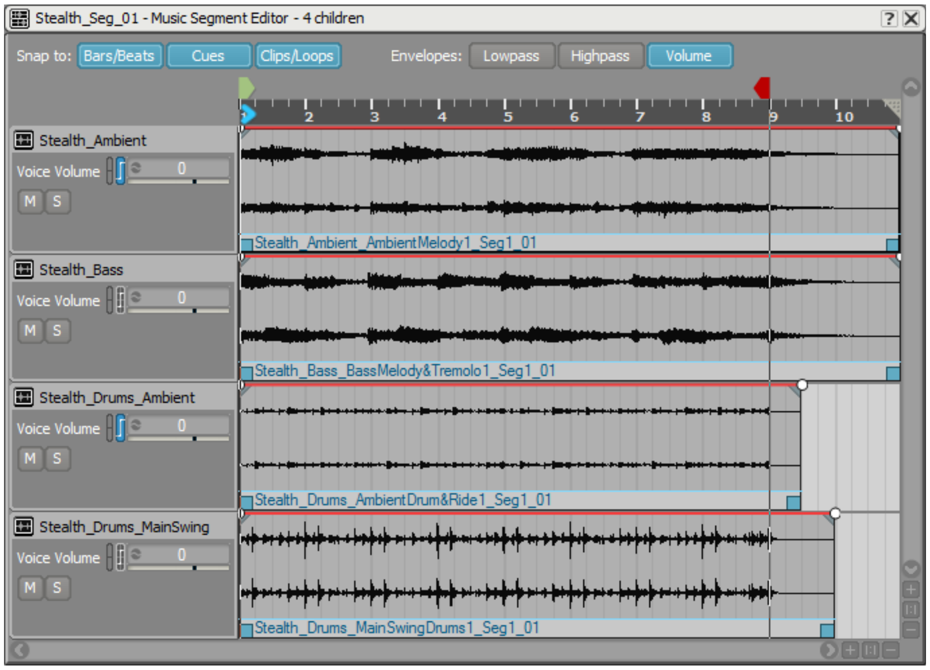
To view or work with interactive music objects, first select the Interactive Music layout by selecting Layouts > Interactive Music or pressing F10.

To examine the Stealth music:

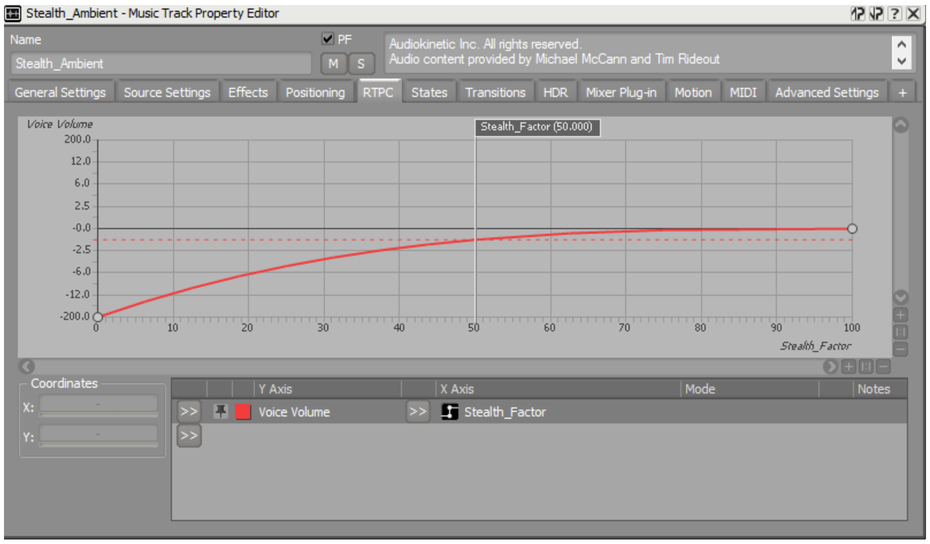
1. Double-click on the music segment named "Stealth_Seg_01" to reveal its properties and to see the four tracks in the Music Segment Editor.



2. In the Music Segment editor, hold "Z" while clicking in the Music Segment Editor track view to zoom out and see the four tracks.


Stealth_Seg_01 - Music Segment Editor - 4 children
Snap to: Bars/Beats Cues Clips/Loops
Envelopes: Lowpass Highpass Volume
Stealth_Ambient
Voice Volume 0
M S
Stealth_Ambient_AmbientMelody1_Seg1_01
Stealth_Bass
Stealth_Bass_BassMelody&Tremolo1_Seg1_01
Stealth_Drums_Ambient
Stealth_Drums_AmbientDrum&Ride1_Seg1_01
Stealth_Drums_MainSwing
Stealth_Drums_MainSwingDrums1_Seg1_01

3. The volume of the tracks “Stealth_Ambient” and “Stealth_Drums_Ambient” has a RTPC curve attached to it. Double-click on one of these tracks to reveal its properties in the Property Editor, and switch to the RTPC tab to see the curve that was created.


Stealth_Ambient - Music Track Property Editor
Name
Stealth_Ambient
PF
M S
Audiokinetic Inc. All rights reserved.
Audio content provided by Michael McCann and Tim Rideout
General Settings Source Settings Effects Positioning RTPC States Transitions HDR Mixer Plug-in Motion MIDI Advanced Settings +
Voice Volume
Stealth_Factor (50.000)
200.0
12.0
6.0
2.5
-0.0
-2.5
-6.0
-12.0
-200.0
0 10 20 30 40 50 60 70 80 90 100
Stealth_Factor
Coordinates
X:
Y:
Y Axis
X Axis
Mode
Notes
Voice Volume
Stealth_Factor

The volume of this track is influenced by the game-driven parameter “Stealth_Factor”.

Game Music Part 1: Learner’s Permit

The “01_In-Game Music” music Switch Container holds three music playlist containers, one for each state of the game.

- **Stealth Music:** The Stealth music is made up of multiple tracks, the contents of which are played according to the game's state and RTPC values. Effects can also be applied to these tracks.
- **Stress Music:** The Stress music is also comprised of multiple tracks which can be affected by states, RTPCs, or effects. Like the Stealth music, the stress music has also been inserted into a music playlist container to sequence the three segments.
- **Fight Music:** The Fight music takes another approach in terms of music composition. Instead of using multi-track segments, the fight music has simple two-bar long segment playing stereo files. This approach allows for elaborate combinations inside the music playlist container.
- **Transitions:** The transitions between Stealth and Stress use different fade-in and fade-out curves. However, they both use the "Same Time as Playing Segment" option as the destination sync point. This means that the destination segment starts playing at the exact moment at which the source music leaves.
  - **Outro_to_Stealth_or_Stress:** For the transitions from Fight to Stress and Fight to Stealth, a transition segment is used to bridge the source and the destination segments smoothly. This transition segment was needed to cover the tempo difference between the source music (120 BPM) and the destination music (90 BPM).

To listen to Game Music Part 1:

1. Load the Music Switch Container "01_In-Game Music" into the Transport Control.
2. In the Transport Control, click States.
3. Set the State to "a_Stealth".
4. Start the playback.
5. Change the State in the list to go from Stealth to Stress to Fight.
6. While in the Stealth state, you can click RTPCs to change the "Stealth_Factor" level and therefore modify the volume of certain tracks.

Game Music Part 2: In the Driver's Seat

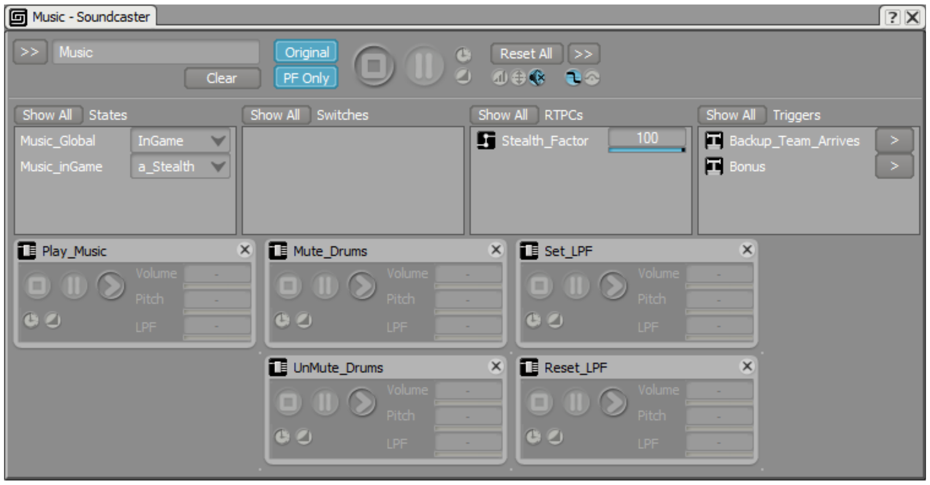
The example "02_In-Game Music" reuses the music from "01_In-Game Music" and adds more transitions segments, stingers, and an additional set of variables to transition music over different contexts. The following are advanced features included in this example:

- **An additional State Group for Music Transition:** A new state group is used to manage interactive music transitions between the In-Game music and the Menu music. It provides an example of how transitions can be set between different game contexts.

- **Stingers:** Stingers are brief musical phrases that are superimposed and mixed over the currently playing music.
  - **Bonus Stinger:** The “Bonus” stinger uses the segment “Stinger Bonus” for all objects except for the Fight music container, which uses the segment “Stinger Bonus - Fight”. “Stinger Bonus - Fight” has three subtracks set to play in sequence. This ensures that you hear a different music punch each time the Bonus stinger is triggered in the Fight state.
  - **Backup Team Arrives:** The “Backup Team Arrives” theme has been composed to play over the Fight music only. It uses the music segment “Stinger Backup Team”, which has been routed to a control bus that auto-ducks the “00_Fight” bus by -10 dB.
- **Additional Transition Segments:** The transitions found in the “02_In-Game Music” music Switch Container manage the transitions between the in-game music and the menu music. There is also a custom transition between Fight and Menu music that uses the dedicated transition segment "Fight to Menu Transition".
- **Transition from Fight to “Nothing”:** This transition uses a transition segment named “Death”. In this case, the “Nothing” state occurs at the main character’s death.
- **Stealth to Stress Transition:** The transitions between Stealth and Stress have been enhanced by adding rules to the Transition Matrix to get more interesting results. For example, if segment “Stealth_Seg_02” is playing when a transition to the Stress state occurs, the specific destination segment will be “Stress_Seg_02” instead of the first segment of the Playlist as the default behavior proposes. Six rules have been defined to cover all possible transitions between the three Stealth segments and the three Stress segments.

To listen to Game Music Part 2:

1. In the Sessions tab of the Project Explorer, double-click the Soundcaster session named “Music”.



Music_Global: InGame
Music_inGame: a_Stealth
Stealth_Factor: 100

2. In the Soundcaster, set the states to “InGame” and “a_Stealth”.
3. Start the music playback by playing the “Play_Music” Event.
4. Change the different States; launch the other Events such as Mute_Drums or Set_LPF (low pass filter).
5. With the State set to Fight, click each trigger button (>) to hear the “Backup_Team_Arrives” and “Bonus” stingers.

Example: MIDI Music

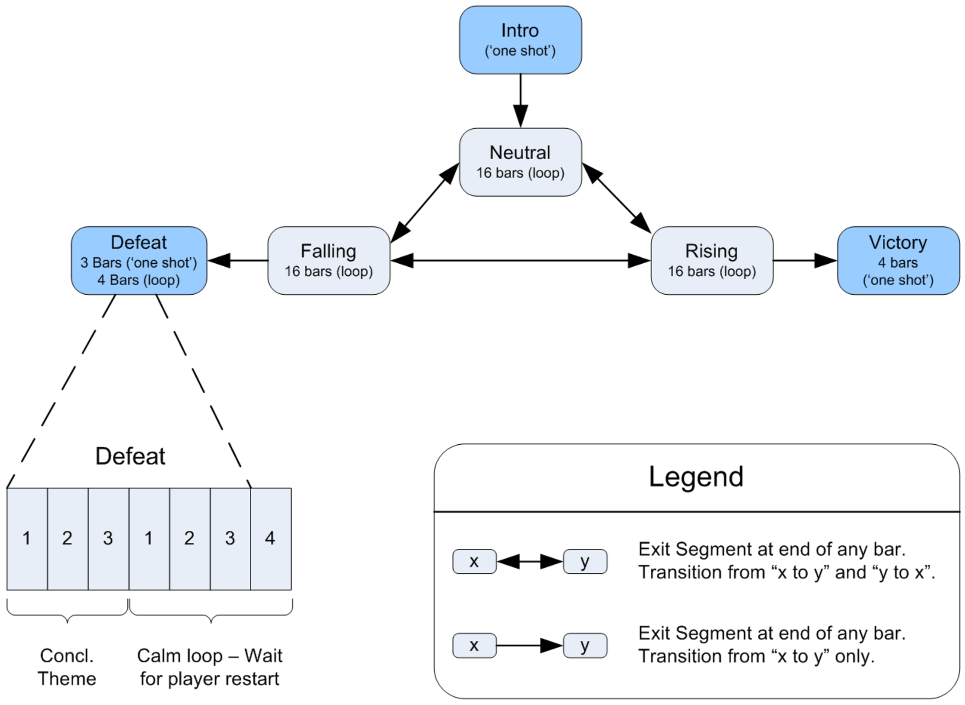
MIDI music provides an example of how a hybrid approach can be taken to build music. This example mixes and matches pre-rendered music and MIDI Sequences, the MIDI Sequences target sampled instruments and a soft synthesizer.

Music Structure

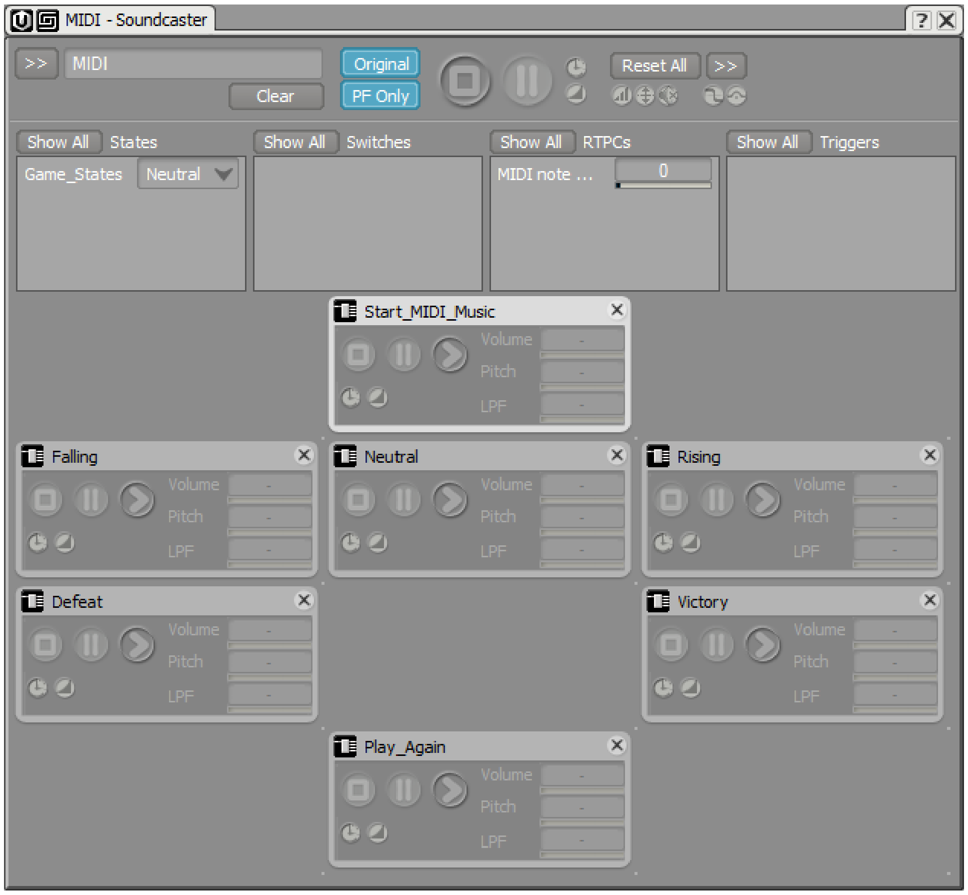
The music has been composed with a fictional puzzle game in mind comprised of five musical states:

- "Neutral", "Rising" and "Falling" are the main themes that follows the player's progression during a puzzle.
- "Victory" is only heard after "Rising" when the player successfully complete the puzzle.
- "Defeat" is only heard after "Falling" when the player failed the puzzle.

This graph shows the game states and gives an overview of the music structure and transitions between states.



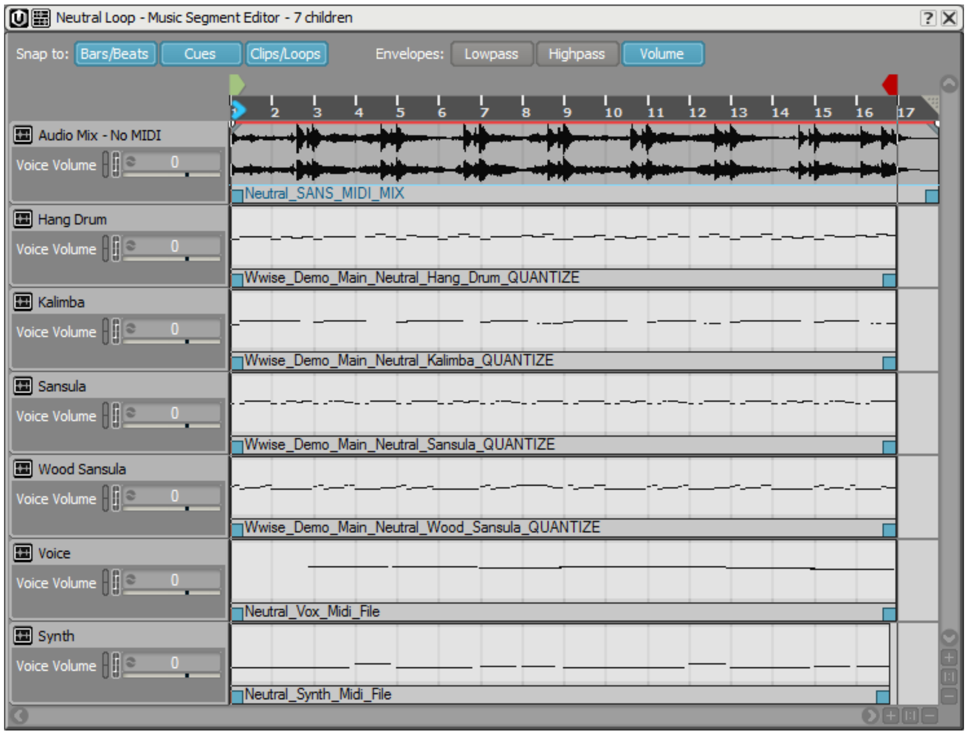
You can listen to the various states by using the Soundcaster session "MIDI". Start by playing the "Start_MIDI_Music" Event in the Soundcaster session, and then trigger the other Events in this session to change from state to state.



MIDI Music - A Hybrid Approach

The MIDI music uses pre-rendered music and MIDI tracks targeting sampled instruments and a soft synthesizer. Taking "Neutral Loop" music segment as an example, you'll see the following:

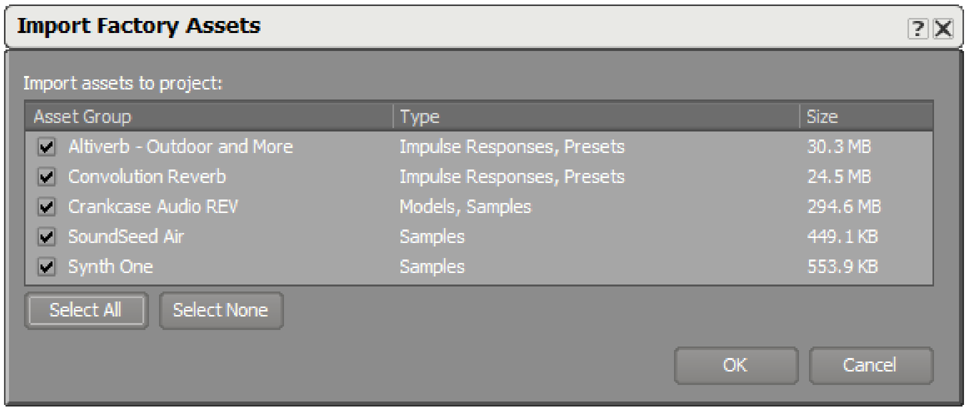
- **Track 1** "Audio Mix - No MIDI" uses a pre-rendered audio files.
- **Tracks 2 to 6** are using MIDI files that each target a different sampled instrument located in the Actor-Mixer Hierarchy.
- **Track 7** "Synth" is also using a MIDI file, but targets a Sound that uses the Wwise Synth One source synthesizer instead of sampled sounds.


Neutral Loop - Music Segment Editor - 7 children
Snap to: Bars/Beats Cues Clips/Loops
Envelopes: Lowpass Highpass Volume
Audio Mix - No MIDI
Neutral_SANS_MIDI_MIX
Hang Drum
Wwise_Demo_Main_Neutral_Hang_Drum_QUANTIZE
Kalimba
Wwise_Demo_Main_Neutral_Kalimba_QUANTIZE
Sansula
Wwise_Demo_Main_Neutral_Sansula_QUANTIZE
Wood Sansula
Wwise_Demo_Main_Neutral_Wood_Sansula_QUANTIZE
Voice
Neutral_Vox_Midi_File
Synth
Neutral_Synth_Midi_File
Voice Volume

Take a look at the Wwise Help for more details on how to create sampled instruments and MIDI tracks in Wwise.

Factory Assets

It is possible to import additional "factory" content in Wwise using the "Import Factory Assets..." option found in the "Project" menu in Wwise. Altiverb and Convolution Reverb are Effect assets while Crankcase REV, SoundSeed Air and Synth One are source plug-in assets.


Import Factory Assets

Import assets to project:

Asset Group	Type	Size
✓ Altiverb - Outdoor and More	Impulse Responses, Presets	30.3 MB
✓ Convolution Reverb	Impulse Responses, Presets	24.5 MB
✓ Crankcase Audio REV	Models, Samples	294.6 MB
✓ SoundSeed Air	Samples	449.1 KB
✓ Synth One	Samples	553.9 KB

Chapter 3. Need Help?

Using Help

Wwise Help contains detailed information on each interface element in Wwise.

To open Help from within Wwise, do one of the following:

- Click the Help icon (?) in the title bar of any of the views or dialog boxes.
- From the menu bar, click Help > Wwise Help.
- Press F1.

Further Resources

This sample project gives an overview of how typical audio structure for game can be built in Wwise. If you want to learn more about Wwise, we suggest you take a look at the following resources

- **Cube Project** - Also available from the Audiokinetic Launcher, Cube is a first person shooter game. Although the game is old and not so great looking, the interest comes from the fact that it contains a few sound Events that can be easily customized. Real-time mixing and profiling is also easily achieved which makes it pretty close to the workflow of working on a game production.
- **Video Tutorials** - Video tutorial are available for all users in the community section of our website. They are also available on AudiokineticWwise's YouTube channel.
- **Wwise Certification** - The Wwise Certification online courses offer several programs for learning Wwise fundamentals and advanced specialized topics.

Note

Materials sent by content providers were all originally high quality (48 kHz, 16 or 24 bits). However, to save download time and disk space, we have reduced the size of some of the sound assets.