

# **Wwise Sample Project Guide**

**Audiokinetic Inc.**

© 2006 Audiokinetic Inc. All rights reserved.  
Patents pending

Wwise is a product of Audiokinetic Inc.  
Wwise Sample Project Guide  
Document No.: 001\_000035\_113\_A

This document is supplied as a guide for the Wwise® product. This guide and the software that it describes is furnished under license and may not be duplicated, reproduced, modified, stored or transmitted, in whole or in part, in any form or by any means, other than as expressly permitted by the terms of such license or with the prior written permission of Audiokinetic Inc.

The content of this guide is furnished for information purposes only, and its content and all features and specifications referred to therein are subject to change without notice. Reasonable care has been taken in preparing the information contained in this document, however, Audiokinetic Inc. disclaims all representations, warranties and conditions, whether express, implied or arising out of usage of trade or course of dealing, concerning this guide and assumes 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 Audiokinetic Inc. has been advised of the possibility of such loss or damage.

This guide is protected by Canadian copyright law and in other jurisdictions by virtue of international copyright treaties.

Wwise® is a registered trade-mark of Audiokinetic Inc. Actor-Mixer, Master-Mixer, SoundFrame, Sound-caster, and Randomizer are all trade-marks of Audiokinetic Inc. All other trade-marks, trade names or company names referenced herein are the property of their respective owners.

#### **Legal Notices:**

Car engine audio content provided by Tinitus, Inc.

With the help of the Conservatory of Recording Arts and Sciences.

© 2007 Tinitus, Inc.

[www.tinitus.net](http://www.tinitus.net)

The Audio Content contained in this file is the property of Tinitus, Inc. Your sole right with regard to this Audio Content is to listen to the Audio Content as part of this Sample Project. This Audio Content cannot be used or modified for a commercial purpose nor for public demonstration.

Footsteps, 24 hour ambiance, Minigun audio content provided by Soundelux Design Music Group

© 2007 Soundelux Design Music Group

[www.soundeluxdmg.com](http://www.soundeluxdmg.com)

The Audio Content contained in this digitally watermarked file is the property of Soundelux Design Music Group. Your sole right with regard to this Audio Content is to listen to the Audio Content as part of this Sample Project. This Audio Content cannot be used or modified for a commercial purpose nor for public demonstration.

Interactive Music audio content provided by Michael McCann and Tim Rideout

© 2007 Michael McCann and Tim Rideout

[www.behaviormusic.com](http://www.behaviormusic.com)

[www.timrideout.com](http://www.timrideout.com)

The Audio Content contained in this file is the property of Michael McCann and Tim Rideout. Your sole right with regard to this Audio Content is to listen to the Audio Content as part of this Sample Project. This Audio Content cannot be used or modified for a commercial purpose nor for public demonstration.

Dialogue audio content provided by Technicolor.

© 2007 Technicolor.

[www.technicolor.com](http://www.technicolor.com)

The Audio Content contained in this file is the property of Technicolor. Your sole right with regard to this Audio Content is to listen to the Audio Content as part of this Sample Project. This Audio Content cannot be used or modified for a commercial purpose nor for public demo.

SoundSeed Impact audio content provided by Wave Generation.

© 2008 Wave Generation.

[www.wavegeneration.ca](http://www.wavegeneration.ca)

The Audio Content contained in this file is the property of Wave Generation. Your sole right with regard to this Audio Content is to listen to the Audio Content as part of this Sample Project. This Audio Content cannot be used or modified for a commercial purpose nor for public demo.



## Wwise Sample Project

For the current release of Wwise, Audiokinetic has prepared a sample project demonstrating real world sound design examples within a fictional game. 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, knowledge base, and video tutorials.

### About the Project

This sample project represents an urban driving and combat game. It includes five examples of different Wwise capabilities. 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.
- **New York City Ambience**—a demonstration of how Wwise allows you to create a complex ambience which evolves over a day and is influenced by weather.
- **Game Music**—these examples demonstrate different techniques for arranging and implementing interactive game music.
- **Dialogue**—a demonstration of how to create dialogue that changes dynamically according to actions in game.
- **SS\_Impact**—a demonstration of many uses of SoundSeed Impact.



---

**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.

---

## 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.

### Overall Routing

The Master-Mixer hierarchy for this project is organized in two sections:

- The “Environmental” bus has the Wwise Environmental FX applied to it so that all objects routed to this bus or any of its child busses will be affected by the environmental reverb.  
Currently, this bus contains child busses for SFX only. If this project were to evolve in a typical manner, you could add new child busses for discreet ambient sounds and voices that would be processed with the environment effect. Remember that only one bus in a project can have Environmental effects added to it.
- The “Non-Environmental” bus groups all busses that are not affected by the environmental reverb. In general, looping ambiences and some voices such as commentators or narrators are the types of objects that are not affected by environmental reverb. Music should never be associated with Environmental effects because reverb and other time-based effects can ruin the precise timing required for listenable music.



---

**Note:** The “Music” bus is flagged as “Use for Background Music” on the Xbox 360 and PLAYSTATION3 platforms to be compliant with their technical certification requirements (TRCs).

---

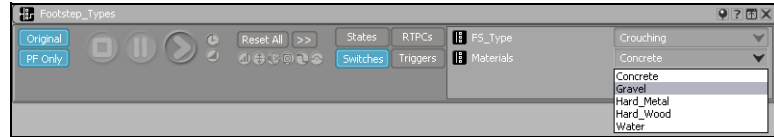
### Example 1: Footsteps

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.



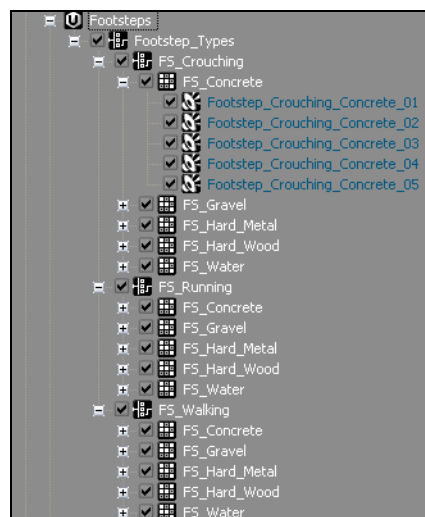
In this example, the parent switch container includes three footstep types:

- Crouching
- Walking
- Running

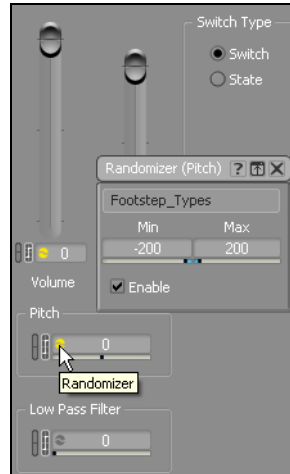
Each of these three footstep types contains another switch container that determines the surfaces on which the characters will walk:

- 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:



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”.



## Example 2: Minigun

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



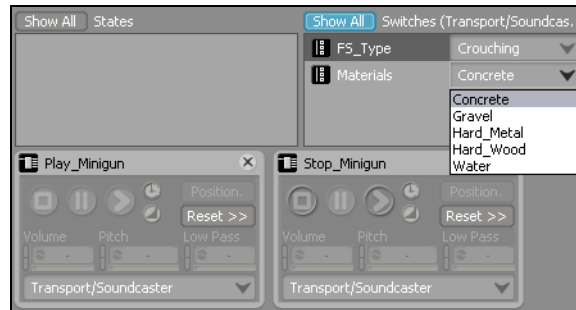
**To listen to the Minigun example:**

1. In the Sessions tab of the Project Explorer, double-click the Minigun\_01 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.

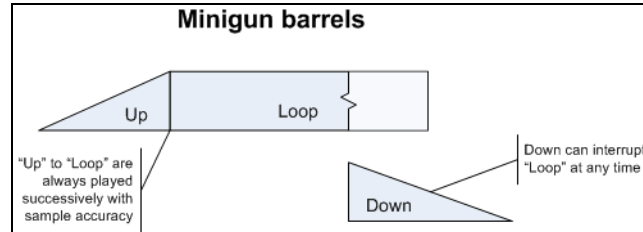


**Note:** The events in the Minigun example have been simplified for this sample project. In a real game, refinements would be made to produce a more realistic simulation. For example, the Play\_Minigun event would be split into two separate events for the barrel and fire sounds, so that each could be controlled more closely.

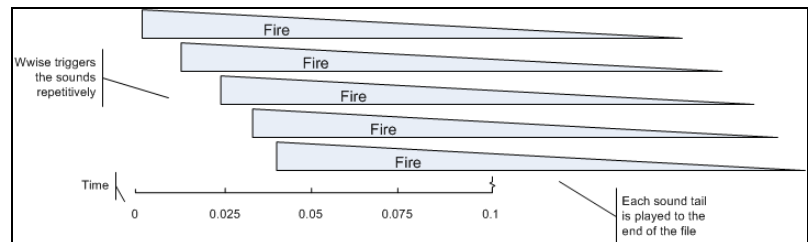
## Minigun Details: Sounds

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.



- **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 10 instances per game object has been set in the Fire sounds Advanced Settings to restrict CPU usage.



**Note:** The trigger rate is not rhythmically perfect when the “Fire” container is played back within Wwise. This happens because all sounds played back from Wwise are streamed from the hard drive. This is a playback limitation within Wwise only; when played back from a game or the Game Simulator, the timing of each fire sounds is sample accurately perfect and constant.

- **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 10 instances per game object has been set in the Shells sounds Advanced Settings to restrict CPU usage.

### Example 3: Car Engine

The Car Engine example demonstrates how you can produce a rich and vivid car engine sound by using a 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.



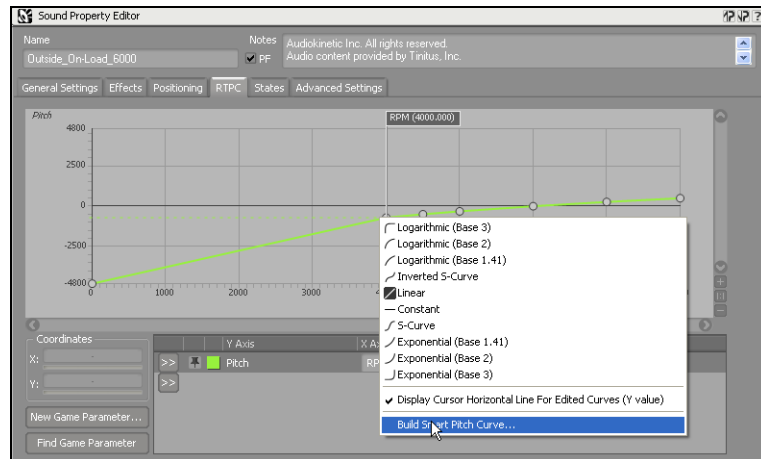
---

**Note:** To hear even better audio results, you can use the Car Simulator application. Refer to the Wwise Knowledge Base for more information.

---

## Car Engine Details: Matching Pitch to RPM Using RTPCs

Each of the engine sounds (except idle) has a pitch curve attached to RPM. 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 processing power.

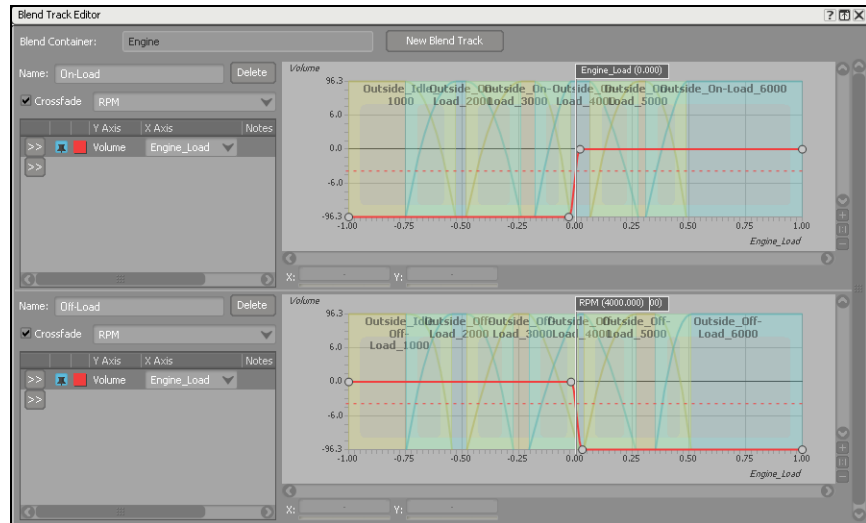
## Car Engine Details: Using Blend Containers

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.



## Example 4: New York City Ambience

The New York City Ambience example shows how you can easily and elegantly create a complex and evolving soundscape. This example uses a blend container to alter game sounds based on two parameters: Time 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 5: Game Music

This project 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**.

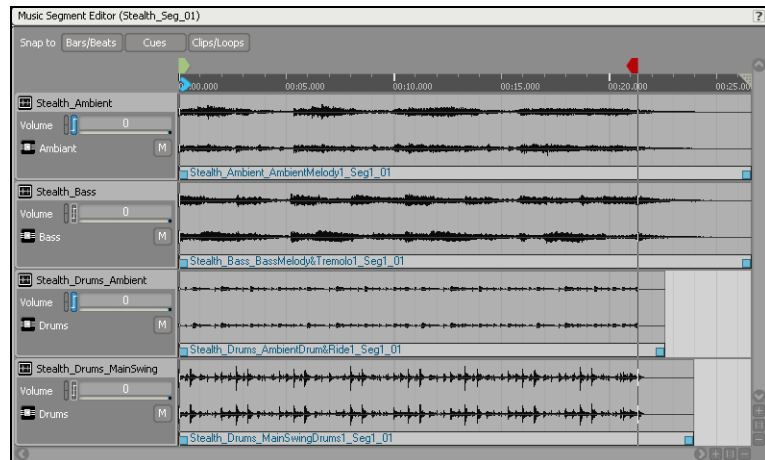
---

**To examine the Stealth music:**

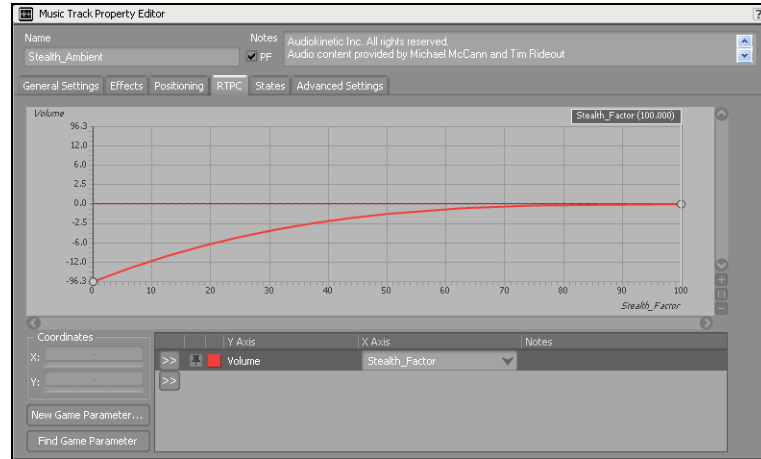
1. Double-click on the music segment named “Stealth\_Seg\_01” to reveal its properties and to see the four tracks in the Music Segment Editor.



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.



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.



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 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 dropdown menu 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\_Music\_Complete” reuses the music from “01\_In-Game Music” and adds more transitions segments, stingers, and a second music switch container level. The following are advanced features included in this example:

- **Two Levels of Music Switches:** A new parent music switch container has been added to manage interactive music transitions between the In-Game music and the Menu music. Notes that the Menu section contains multiple states but only the “Home” state has music written for it. This has been done intentionally, as there are many options you could implement for interactive music in menus.



---

**Note:** Using multiple levels of music switch containers should only be used when the game variables and their music transitions are mostly independent. For example, an in-game switch versus a menu-to-game switch, where the latter generally just fades out whatever is playing to go to or from the menu. If the game variables and their music transitions are inter-related, then you should only create one switch group in Wwise, and have the game perform the logic to map its various variables to this one and only Wwise variable.

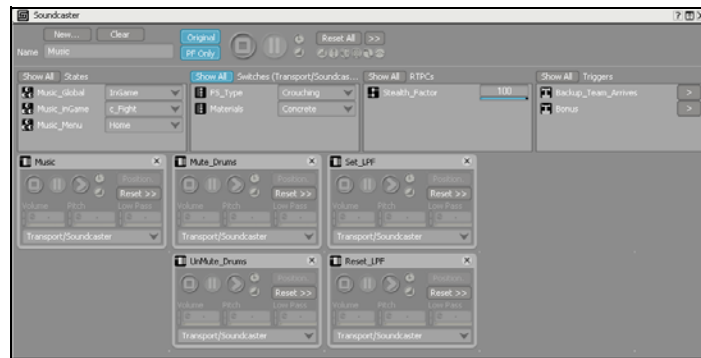
---

- **Stingers:** Both stingers have been composed to play over the Fight music.
  - **Bonus Stinger:** The “Bonus” stinger uses the segment “StingerBonus” for all objects except for the Fight music container, which uses the segment “StingerBonus\_Fight”. “StingerBonus\_Fight” has three sub-tracks 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 “Interlude”, 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\_Music\_Complete” 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 a dedicated transition segment.

- **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”.



2. In the Soundcaster, set the states to “InGame”, “a\_Stealth”, and “Home”.
3. Start the music playback by playing the “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 6: 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.



---

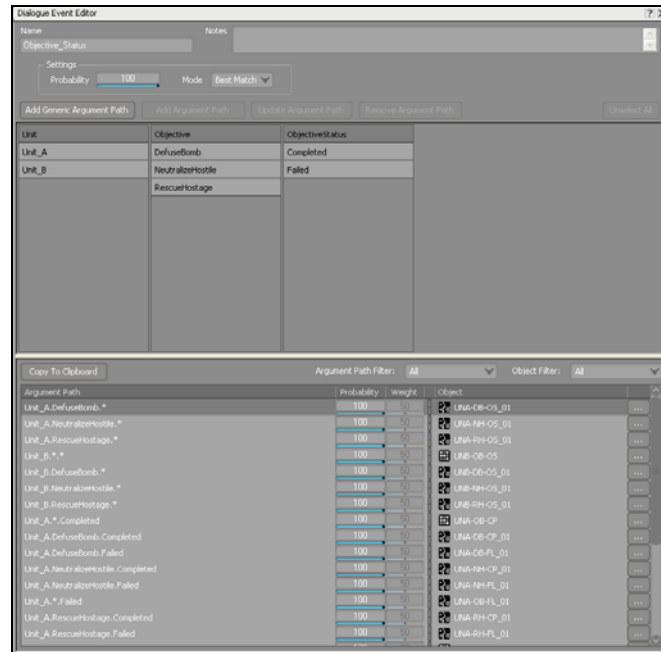
**Note:** Before dynamic dialogue was developed for Wwise, dialogue was sometimes handled with a series of switch containers. However, the dynamic dialogue functionality offers a far more lightweight memory structure as well as increased flexibility.

---

### To listen to the Dialogue example:

1. In Wwise, press F11 to switch to the Dynamic Dialogue layout.
2. In the Events tab of the Project Explorer, locate the Dialogue Work Unit and double-click the Objective\_Status dialogue event.

The Objective\_Status dialogue event is loaded into the Dialogue Event Editor.



3. 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
Unit_A	DefuseBomb	Completed
Unit_B	NeutralizeHostile	Failed
	RescueHostage	

Each argument you choose is highlighted

4. In the Argument Path Filter list, select **Current Selection**.

Copy To Clipboard

Argument Path Filter: Current Selection

Object Filter: All

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

The selected argument path and its associated object are displayed.



5. In the Transport Control, click **Play** to hear the object.

The object associated with the selected argument path is played.

6. 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 flags to trigger the particular sentences. This can be particularly useful in real-time simulations, adventure games, and first-person shooters. You can also use dynamic dialogue to create play-by-play for sports games, with events that are sequenced dynamically by the game engine based on the action.

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
Unit_A	DefuseBomb	Completed
Unit_B	NeutralizeHostile	Failed
	RescueHostage	

Based on these argument paths, dialogue is written and recorded. Each line of dialogue is then imported into Wwise as either a solitary sound voice object or as part of a container.

Copy To Clipboard				Argument Path Filter: All	Object Filter: All
Argument Path	Probability	Weight	Object		
Unit_A.DefuseBomb.*	100	50	UNA-D6-OS_01		...
Unit_A.NeutralizeHostile.*	100	50	UNA-NH-OS_01		...
Unit_A.RescueHostage.*	100	50	UNA-RH-OS_01		...
Unit_B.*	100	50	UNB-OB-OS		...
Unit_B.DefuseBomb.*	100	50	UNB-D6-OS_01		...
Unit_B.NeutralizeHostile.*	100	50	UNB-NH-OS_01		...
Unit_B.RescueHostage.*	100	50	UNB-RH-OS_01		...
Unit_A.*.Completed	100	50	UNA-OB-CP		...
Unit_A.DefuseBomb.Completed	100	50	UNA-D6-CP_01		...
Unit_A.DefuseBomb.Failed	100	50	UNA-D6-FL_01		...
Unit_A.NeutralizeHostile.Completed	100	50	UNA-NH-CP_01		...
Unit_A.NeutralizeHostile.Failed	100	50	UNA-NH-FL_01		...
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_B.*.Completed	100	50	UNB-OB-CP		...
Unit_B.DefuseBomb.Completed	100	50	UNB-D6-CP_01		...
Unit_B.DefuseBomb.Failed	100	50	UNB-D6-FL_01		...
Unit_B.NeutralizeHostile.Completed	100	50	UNB-NH-CP_01		...
Unit_B.NeutralizeHostile.Failed	100	50	UNB-NH-FL_01		...
Unit_B.*.Failed	100	50	UNB-OB-FL_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 7: 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.

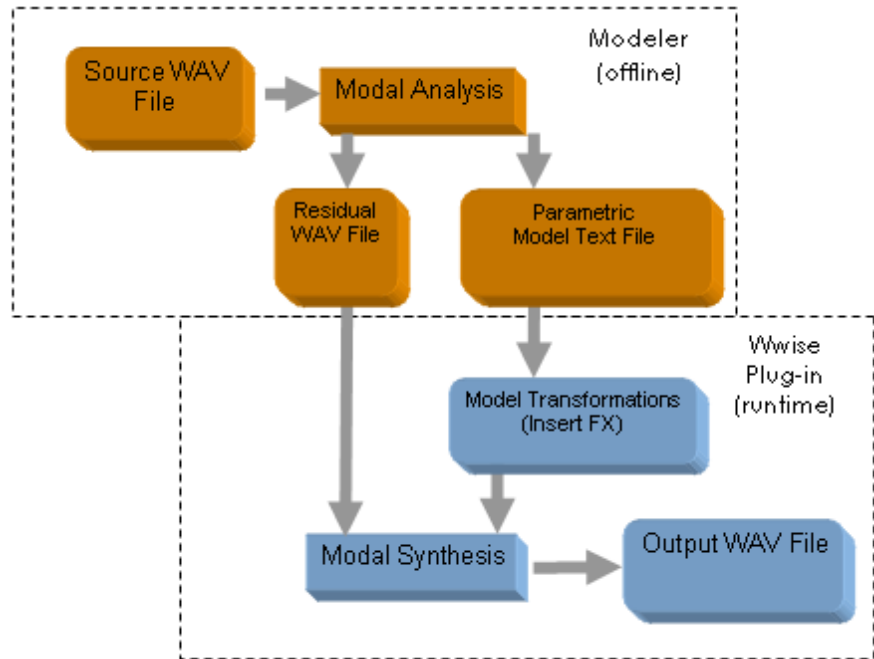
SoundSeed Impact accurately models a wide range of resonant impact type sounds, which means you can use it to create many different sounds in your game. Furthermore, the compromise between audio quality and performance can be controlled at runtime along with other model parameter transformations that 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 WAV files for each variation.
- Increase interactivity by controlling synthesis parameters based on context-specific (possibly physics related) run-time parameters.
- Unleash creativity by intuitively transforming real sounds without real-world constraints.

## 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 wav 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.

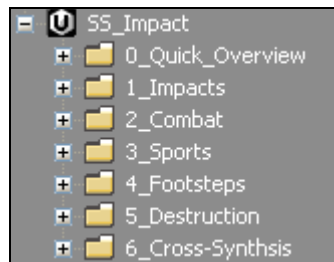
Runtime processing is performed in Wwise using the SoundSeed Impact real-time audio effect plug-in. Using both the residual file and the .SSM model file, the Wwise SoundSeed Impact plug-in allows you to do the following:

- 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

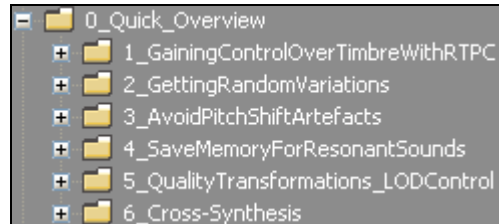
The Sample Project features some practical examples on how SoundSeed Impact can be used. The SS\_Impact work unit of the Sample Project is broken down into the following folders:



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

## 0\_Quick\_Overview

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

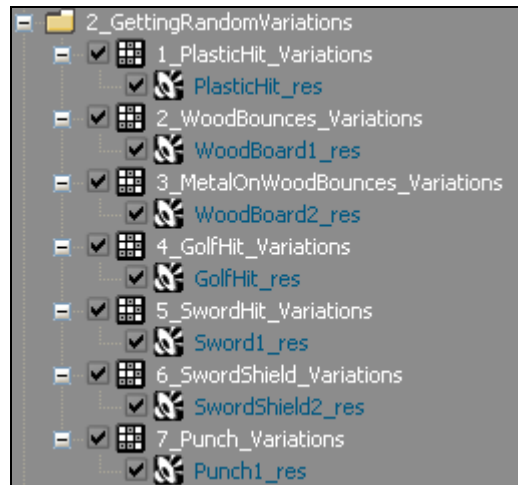


- **1\_GainingControlOverTimbreWithRTPC**—this folder contains examples that show how to control the SoundSeed Impact Synthesis parameters in real time using RTPCs.

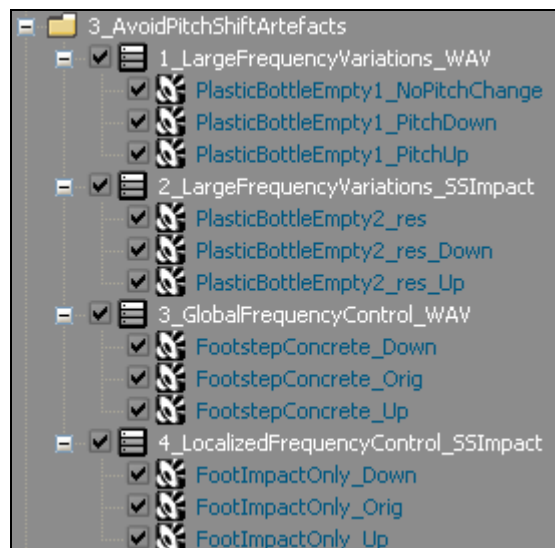


- **1\_EmptyPlasticBottle\_FrequencyControl**—demonstrates the effect of frequency transformation in isolation. Play with the "PlasticBottle\_Frequency" RTPC slider in the Transport Control to affect the perceived size of the bottle without getting pitch shift artifacts or duration changes.
- **2\_MetalHit\_DampingControl**—demonstrates 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.

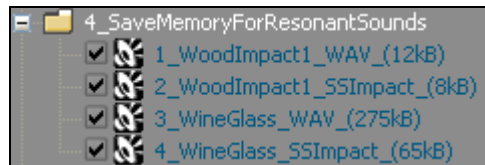
- **2\_GettingRandomVariations**—this folder contains examples that show how virtually an unlimited number of playback variations can be achieved from a single source file. Play the Random Container for each example.



- **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.



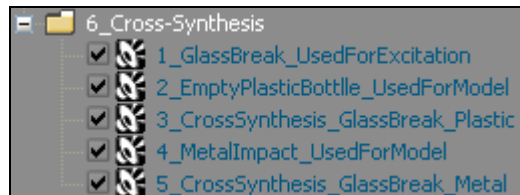
- **4\_SaveMemoryForResonantSounds**—this folder contains examples that demonstrate the possible runtime memory savings that are possible for highly resonant sounds.



- **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.

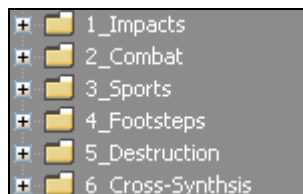


- **6\_Cross-Synthesis**—this folder contains examples that show some creative uses of SoundSeed Impact. In these examples, data model files from different sources are combined with residual files.

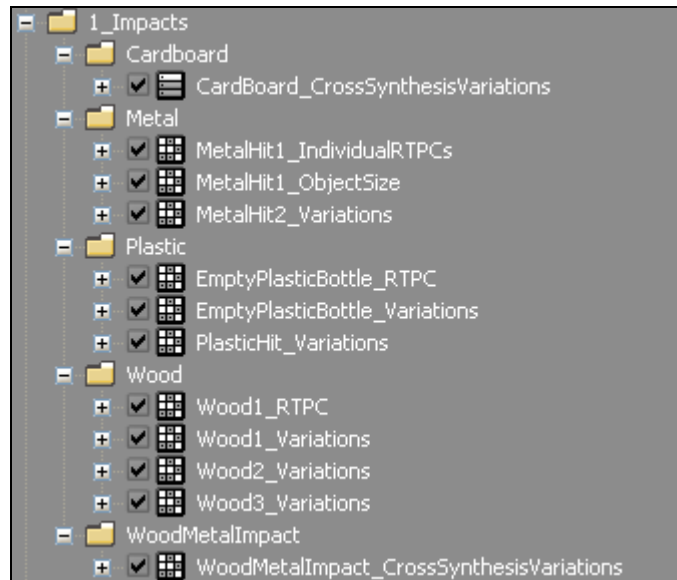


## Detailed Examples

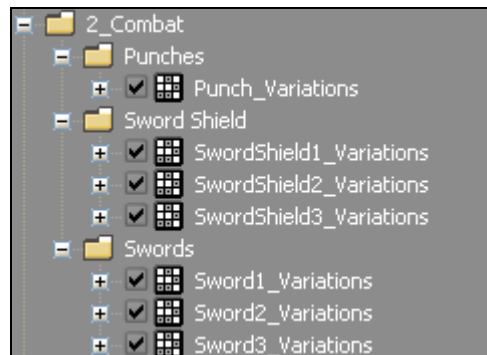
The remaining folders contain more elaborate examples of typical uses of SoundSeed Impact.



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



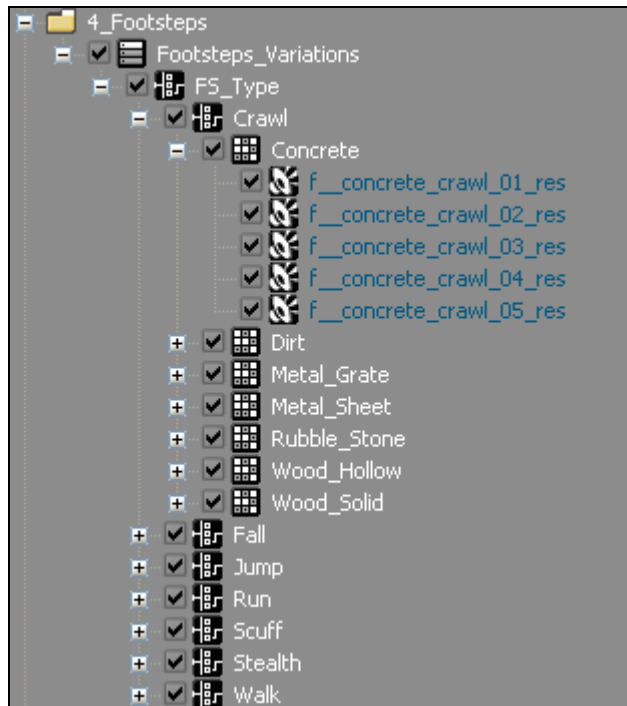
- **2\_Combat**—this folder contains examples of transformations possible for combat class type sounds. Play the random containers to hear the different sound variations.



- **3\_Sports**—this folder contains examples of transformations possible for sports type sounds. Play the random containers to hear the different sound variations.

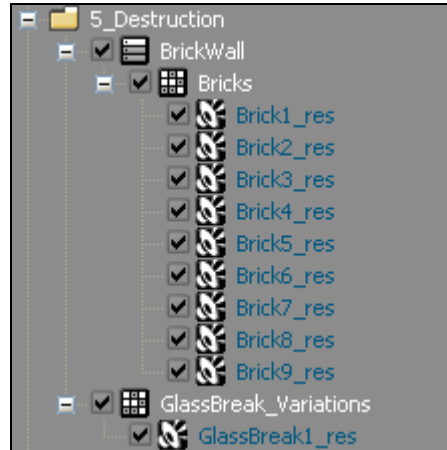


- **4\_Footsteps**—this folder contains a dramatic example of the potential of SoundSeed Impact. Here we can see a series of nested switch containers, random containers and residual sounds/SoundSeed impact plug-ins. Be sure to select the “Switches” button in the Transport Control so that you can switch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

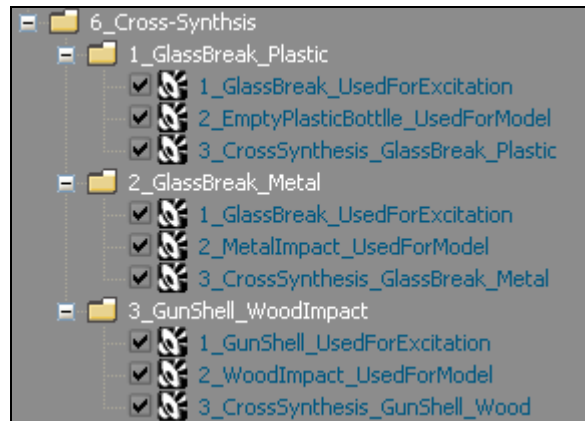




- **5\_Destruction**—this folder contains an example of the destruction of a brick wall. It demonstrates how SoundSeed Impact can be used to obtain more variations on individual grains in a granular synthesis setting. Additionally you can gain control over the sound of the brick wall destruction by changing,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




## 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.

### Contacting Support

Audiokinetic has established a complete [online support center](#) for our maintenance and evaluation customers. The following resources are available:

- A feedback form to submit details about bugs, crashes, and/or to suggest a feature, or make any general inquiries.
- Access to all the latest product downloads.
- The [Wwise Knowledge Base](#) with knowledge base articles, tips and tricks.

Video tutorials are also available for all users in the community section of our web site. You can also contact us directly at: [support@audiokinetic.com](mailto:support@audiokinetic.com).



---

**Note:** E-mail support is only available for maintenance and registered evaluation customers.

---

### Got Comments?

We'd appreciate any comments or suggestions you may have about these release notes or any other piece of our documentation. Just send them to: [documentation@audiokinetic.com](mailto:documentation@audiokinetic.com)