Integrating Audio in your Game: The Wwise Approach

Before jumping into the code and using the Wwise SDK, you should understand the unique approach Wwise uses for building and integrating audio into your game. There are also a few concepts that you should be familiar with in order to work efficiently and get the most out of Wwise.

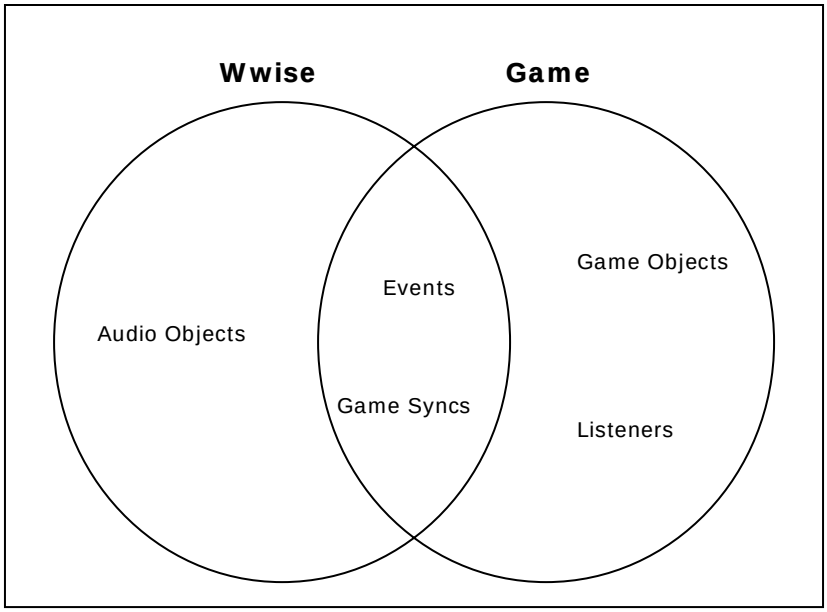
The Wwise approach to building and integrating audio in a game includes five main components:

- Audio Objects
- Events
- Game Syncs
- Game Objects
- Listeners

Each of the components will be discussed in further detail in the following sections, but before moving on, you should understand where each of these components fits in and how they relate to one another.

One of the goals of Wwise was to create a clear distinction between the tasks of the programmer and those of the designer. For example, the audio objects, which represent the individual sounds in your game are created and managed exclusively within the Wwise application by the sound designer. Game objects and listeners, on the other hand, which represent specific game elements that emit or receive audio, are created and managed within the game by the programmer. The final two components, events and game syncs, are used to drive the audio in your game. These two components create the bridge between the audio assets and the game components and are therefore integral to both Wwise and the game.

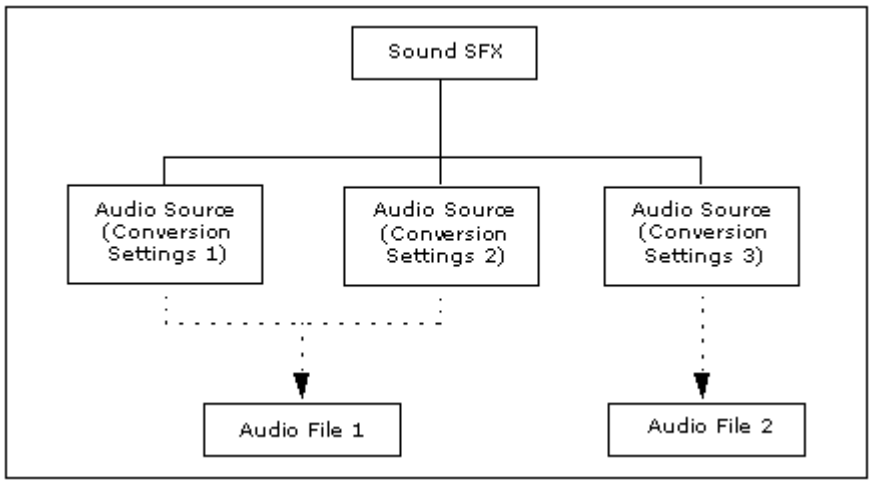
The following illustration demonstrates where each of these components are created and managed.



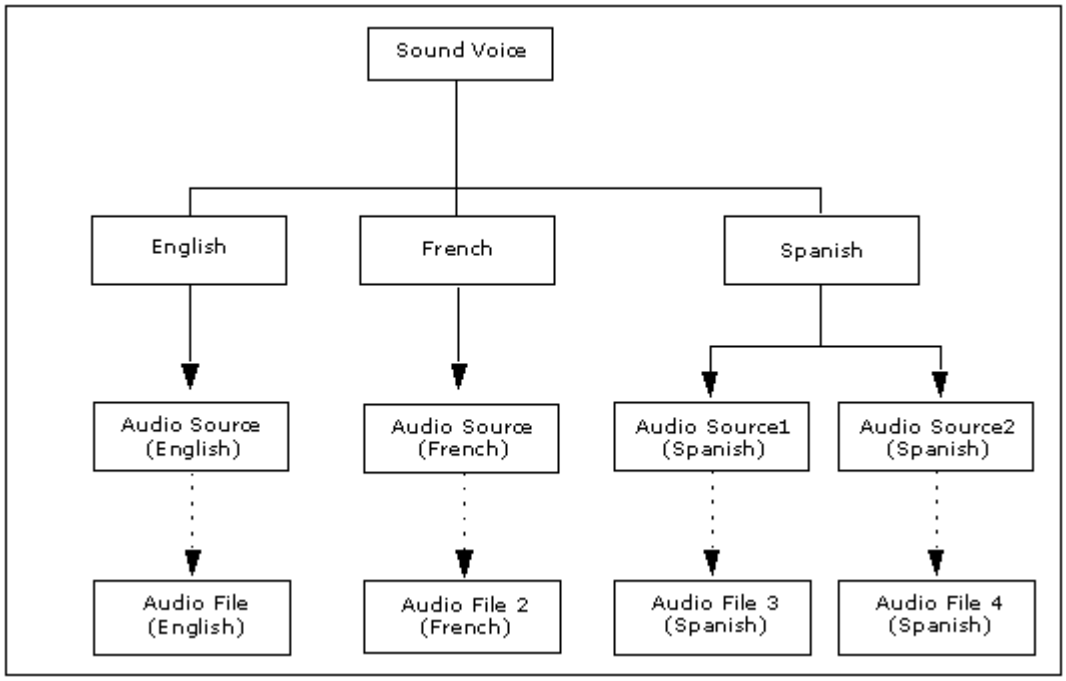
Audio Objects

The various voice and sfx assets in your game are represented in Wwise by special audio objects called sound objects. These sound objects contain sources that are linked to the original audio file.

The audio source is a separate layer between the imported audio file and the sound object. By adding an abstraction layer, you can have multiple sources and audio files all contained within the same sound object.



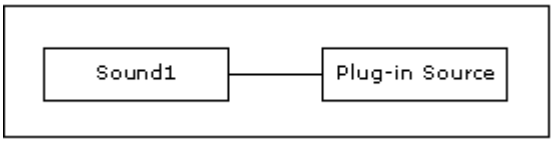
This not only makes it easy to test different conversion settings but also allows you to efficiently manage multi-language development.



Note: Wwise uses a similar method to manage the music and motion assets in your project.

Source Plug-ins

Sound objects not only support audio sources, but they also support plug-in sources.

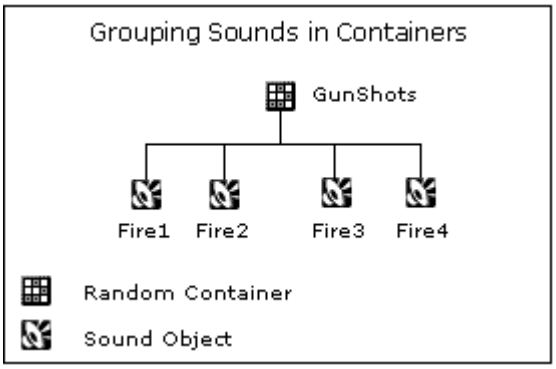


Wwise ships with a variety of source plug-ins, including a tone generator, silence, and audio input plug-in. Although these source plug-ins can be very useful, their main purpose is to show programmers how to build their own. With the creation and management of audio objects being done in Wwise by the sound designer, programmers are now free to develop a variety of source plug-ins, pushing the envelope in audio design and enhancing the overall experience of the game.

Building a Hierarchy of Audio Objects

These sound objects can be grouped together to create a hierarchical project structure. Audio properties and behaviors can be applied at different levels in the hierarchy to give you the control and flexibility you need to build a realistic and immersive game experience.

Containers are used to group the sound objects within your project. They are mainly used to play a group of objects according to a certain behavior, such as random, sequence, switch, and so on. For example, you can group all the gun shot firing sounds into a random container so that a different sound will be played each time the gun is fired in game.



All these audio objects are routed through a hierarchy of busses where additional properties and effects can be applied at a global level.

Audio Objects - Roles and Responsibilities

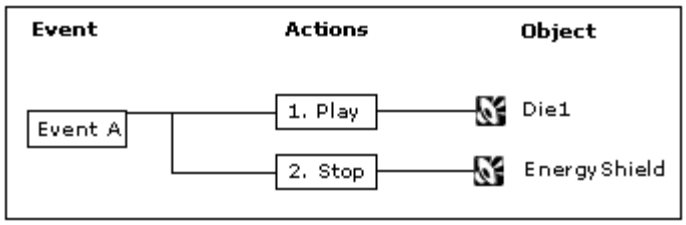
The following table shows you which tasks related to audio objects are the responsibility of the sound designer and which ones are the responsibility of the programmer:

Tasks	Sound Designer	Programmer
Create sound objects for game audio assets	✓	
Group objects and build project hierarchy	✓	
Define sound properties and behaviors	✓	
Route audio objects through busses	✓	
Develop source plug-ins		✓

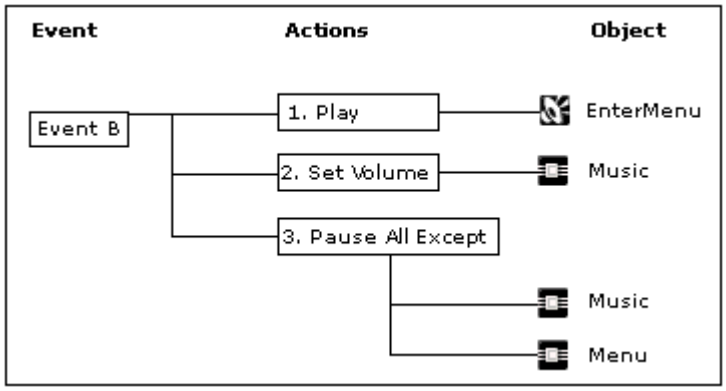
Audio objects are only available in Wwise, which means that they are not exposed in the SDK.

Events

Wwise uses events to drive the audio in your game. These events apply actions to the different sound objects or object groups in your project hierarchy. The actions you select specify whether the Wwise objects will play, stop, pause, and so on. For example, let's say you are creating a first-person shooter game and you want to create an event for when the player dies. This event will play a special "Die" sound and will stop the "EnergyShield" sound that is currently playing. The following illustration demonstrates how this event would look in Wwise.



The sound designer can pick from a long list of action types to drive the audio in game, including Mute, Set Volume, Enable Effect Bypass, and so on. For example, let's say you created a second event for when the player leaves the game to enter the menu. This event will play the "Enter_Menu" sound, decrease the volume of the music bus by -10dB, and pause everything else. The following illustration demonstrates how this event would look in Wwise.



Defining Event Scope

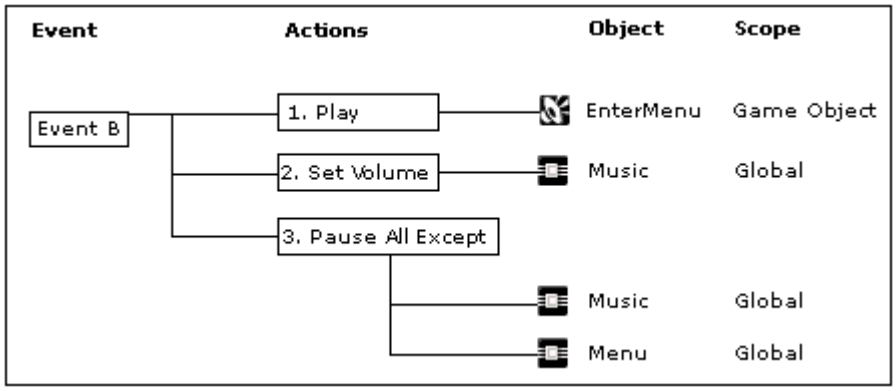
Every action within an event has a corresponding scope setting. The scope determines whether the event action is applied globally to all game objects or to the specific game object that triggered the event. For some actions, the sound designer can choose the scope, and for other actions, the scope is pre-determined.

If we look again at EventB, for example, the scope of each event action would be as follows:

Event Action	Scope	Comments
Play > Menu_Enter	Game Object	The scope is set to Game Object because play events are always triggered by a single game object.
Set Volume > Music	Global	The scope is set to Global because the Set Volume action is applied to a bus, which, by its very nature, is global.
Pause All Except > Music	Global	The scope is automatically set to Global because the Pause All Except action is applied to the music bus, which, by its very nature, is global.

Pause All Except > Menu	Global	The scope is automatically set to Global because the Pause All Except action is applied to the menu bus, which, by its very nature, is global.

The following illustration demonstrates how this event would look in Wwise.



Note: Scope is an important concept that applies to many elements in Wwise. Understanding the scope of each element will help you decide when to use each element in different situations.

Integrating Events into your Game

After creating the events for your game, the sound designer can package them into SoundBanks. These SoundBanks are then loaded into your game, where the events can be triggered by your game's code. For example, when the player is killed, you would play the special “Die” sound and stop the “EnergyShield” sound by triggering the corresponding event.

To integrate these events into your game, the programmer must specify onto which game object the event actions will be performed. This is done by posting each event. An event should be posted by your game’s code whenever you want the audio to change. You can post events using strings or ids.

Benefits of Using Wwise Events

One main advantage to this method for triggering sound in your game is that it gives the sound designer additional control and flexibility without requiring any additional programming. All events are created in Wwise by the sound designer and they are then integrated into the game by the programmer. Once events are integrated in the game, the sound designer can continue working on them, changing or modifying the actions they contain, or the objects to which they refer. Since your game is still triggering the same event, the changes made by the sound designer will take effect in the game without requiring extra work from the developer, and without recompiling the code.

Events - Roles and Responsibilities

The following table shows you which tasks related to events are the responsibility of the sound designer and which ones are the responsibility of the programmer:

Tasks	Sound Designer (Wwise)	Programmer (Game Code/Tools)*
Creating events	✓	
Assigning event actions to audio objects	✓	
Defining the scope of event actions	✓	
Posting events in game		✓

* - The roles and responsibilities for events may change depending on your implementation and the level of integration of Wwise within your tools.

Game Objects

Game objects are the central concept in Wwise because every event triggered in the sound engine is associated with a game object. A game object generally represents a particular object or element in your game that can emit a sound, including characters, weapons, ambient objects, such as torches, and so on. In some cases, however, you may want to assign game objects to different parts of an in-game element. For example, you can assign a different game object to different parts of a giant character so that the footstep sounds and the character's voice emanate from different locations within the 3D sound space.

Note: If you are familiar with the Unreal game engine, game objects in Wwise are similar to Actors in Unreal.

For every game object, Wwise stores a variety of information that it will use to determine how each sound will be played back in game. Any of the following types of information may be associated with the game object:

- Property offset values of an audio object associated with the game object, including volume and pitch.
- 3D position and orientation.
- Game syncs information, including states, switches, and RTPCs.
- Environmental effects.
- Obstruction and Occlusion.

Note: Unlike other properties, attenuation is applied on the audio object and not on the game object. This gives the sound designer more flexibility to control the attenuation for each sound individually. The 3D Game Object view in Wwise allows the sound designer to view the game objects to which sounds are associated, the position of the game objects in relation to the listener, along with the attenuation radius for each sound.

Registering Game Objects

Before you can use game objects, the programmer needs to register them in the game code. When you no longer need the game objects, you should un-register them, because the sound engine will continue to store their related information (3DPosition, RTPC, switches, and so on) until the game object associated with these values is unregistered.

Scope – Game Object vs Global

By using game objects, Wwise introduces the concept of scope, which was discussed briefly in the Events section. The scope determines the level at which properties and events are applied to the sounds in your game. You now have the choice to apply these elements at the game object level or globally. The specific situation and/or action that is taking place in game, will determine the scope and ultimately the approach you take in Wwise.

For example, let's say you are creating a first-person shooter game. The main character in your game must navigate the city streets to capture the enemy's flags. As the character walks through the city, you will hear his footsteps. If want to change the properties or sounds associated with these footsteps, you will only want to apply these changes locally at the level of the game objects specifically related to the main character's feet. On the other hand, if your character submerges himself underwater, all the sounds that continue to play within the surrounding environment, such as explosions and vehicles, will need to be modified. In cases like these, you will want the changes to be made on a global scale.

Benefits of Using Game Objects

By using game objects, the management of audio has been simplified because programmers only have to keep track of game objects and not the individual sounds.

Once the game objects are created, programmers only need to post events, set up the game syncs, including switches, states, and RTPCs, and in-game environments. The specific details of which sound is played and how it will play are defined by the sound designer in Wwise. By using this approach, you can save a huge amount of time when dealing with the multitude of sounds associated with the various entities within your game.

Game Objects - Roles and Responsibilities

The following table shows you which tasks related to game objects are the responsibility of the sound designer and which ones are the responsibility of the programmer:

Tasks	Sound Designer (Wwise)	Programmer (Game Code/Tools)*
Associate game objects to 3D objects in the game	✓	✓
Register/Unregister game objects		✓
Update game object positioning information		✓
Set the attenuation for each audio object	✓	
Defining event scope	✓	

* - The roles and responsibilities for game objects may change depending on your implementation and the level of integration of Wwise within your tools.

Listeners

A listener represents a microphone in the game. A listener has a position and orientation in the game's 3D space. During game play, the coordinates of the listener are compared with the game object's position, so that 3D sounds associated with game objects can be assigned to the appropriate speakers to mimic a real 3D environment. Programmers must ensure that the listener's positional information is kept up to date; otherwise sounds will be rendered through the wrong speakers.

Multiple Listeners

In a single-player game where you always see only one point of view in the game, one listener is enough. However, if multiple players can play on the same system, or if multiple views are displayed at the same time, each view requires its own listener so audio is appropriately rendered for all of these views. The Wwise sound engine supports up to eight listeners.

By default, every registered game object is assigned to the first listener only. However, the programmer has the flexibility to dynamically assign or unassign any game object from any listener.

There are many challenges to implementing audio for multiple listeners because the positioning of sound sources doesn't always make sense in relation to what each player is seeing. This is mostly caused by a game using only one set of speakers to reproduce the 3D environment for several players. Wwise offers a variety of tools and techniques to offset this limitation so that the audio experience is as realistic as possible for all players. For more information about how Wwise handles multiple listeners, refer to the section "Integrating Listeners" in the SDK.

Listeners - Roles and Responsibilities

The following table shows you which tasks related to listeners are the responsibility of the sound designer and which ones are the responsibility of the programmer:

Tasks	Sound Designer (Wwise)	Programmer (Game Code/Tools)*
Setting the listener's positional information		✓
Assigning game objects to listeners		✓
Managing multiple listeners		✓

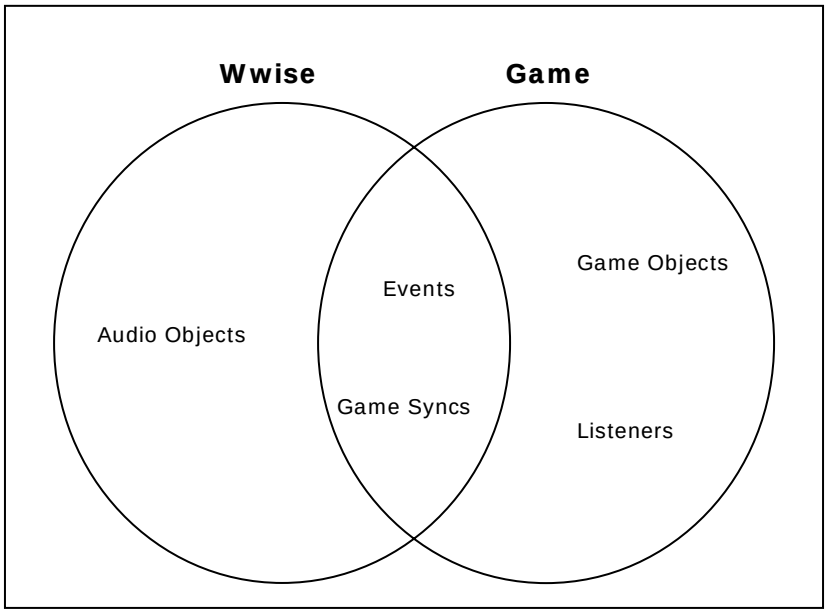
- The roles and responsibilities for listeners may change depending on your implementation and the level of integration of Wwise within your tools.

Conclusion

From its inception, Wwise has attempted to create a clear distinction between the roles of the sound designer and the programmer. Each group has its own core competencies, and should be able to focus on the tasks that push game audio to the next level, enhancing the overall gaming experience. From this separation of tasks, came the five main components that make up the core of Wwise:

- Audio objects
- Events
- Game Syncs
- Game objects
- Listeners

Each component falls under the responsibility of either the sound designer or the programmer, as shown in the following illustration.



There are two components that are integral to both Wwise and the game: Events and Game Syncs. These two components, which drive the audio in your game, create the necessary bridge between the audio assets in Wwise and the components managed in the game.

Wwise represents a paradigm shift in the way audio is developed and integrated in video games. It does require game designers and developers to approach their work in a new way, but it also allows them to work more efficiently and to focus on their areas of expertise.

Now that you have a basic understanding of Wwise's approach to game audio development, you are ready to jump in, and take full advantage of all Wwise has to offer.