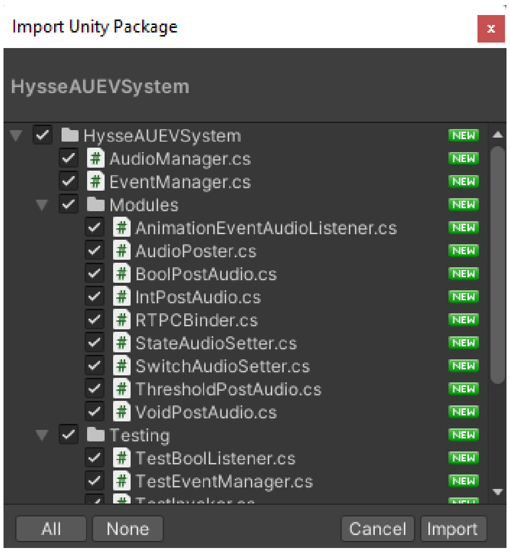
Hysse Audio/Event System

IMPORTANT DISCLAIMER: *The system is dependent on the Wwise framework, so before being able to use it correctly one must integrate Wwise into the Unity project.*

This is the documentation for the HysseAESystem illustrating how to set it up and use it in a Unity project. The system is meant for use in Wwise-integrated projects and its goal is to empower the audio designers by giving them additional tools to hook up the Wwise framework and its components to the custom logic of a Unity game.

Setup

To start using the system one must first integrate Wwise in the Unity project by following the official guide.
https://www.audiokinetic.com/library/edge/?source=InstallGuide&id=integrating_wwise_into_an_unreal_or_unity_project
Afterwards the only step left is to import the HysseAESystem Unity package by double clicking on the corresponding package file while the Unity project is open and confirming by clicking "Import".



```
Import Unity Package
HysseAUEVSystem
▼ ✓ HysseAUEVSystem                      NEW
    ✓ AudioManager.cs                    NEW
    ✓ EventManager.cs                    NEW
  ▼ ✓ Modules                            NEW
      ✓ AnimationEventAudioListener.cs   NEW
      ✓ AudioPoster.cs                   NEW
      ✓ BoolPostAudio.cs                 NEW
      ✓ IntPostAudio.cs                  NEW
      ✓ RTPCBinder.cs                    NEW
      ✓ StateAudioSetter.cs              NEW
      ✓ SwitchAudioSetter.cs             NEW
      ✓ ThresholdPostAudio.cs            NEW
      ✓ VoidPostAudio.cs                 NEW
  ▼ ✓ Testing                            NEW
      ✓ TestBoolListener.cs              NEW
      ✓ TestEventManager.cs              NEW
```

Structure of the package

The package is mostly divided in 2 folders (exception made for the two Managers, Audio and Event, in the root folder): the Modules folder contains all of the system's modules that can be used for managing sound in the game; the Testing folder, on the other hand, contains mocks of the same modules in the first folder exclusively made for testing purposes. The latter use a separate EventManager from the one that you should use in the game, which has a list of mock-up events used by the TestInvoker to test the modules.

The Testing folder

If your purpose is a testing one, or if you want to get familiar with the system's modules, only use the ones in the Testing folder. To do that, first create an Empty Object in the scene and add the TestInvoker component onto it. In here you can set up how to manually fire specific events that can then be listened to by the test modules that you slot on other GameObjects. You're strongly recommended not to modify the TestEventManager since it has already been given the necessary testing events needed to test every possible module.

The Modules folder

If your purpose is to start using the actual modules from the Modules folder in your personal game, you should pay attention to the fact that those modules use the EventManager instead of the TestEventManager; this means that in order to make the modules do something, you need to create the events that they should listen to in the EventManager and fire them from somewhere in your game.

Introduction to the system

To both understand how to use the system and how the system works underneath, one must first have a general overview of what is Wwise and why the system is useful.

What is Wwise

Wwise is a software made by AudioKinetic for interactive media and video games that features an audio authoring tool and a cross-platform sound engine. Most importantly, on top of letting audio designers create and perfect sounds and ambient effects, it also enables them to hook these to in-game logic by providing a list of components that can be used in game editors to send data to Wwise based on game events. The main element to be understood from the Wwise framework is Wwise Events.

Wwise Events

Wwise Events are basically the building blocks of the whole audio design process; they represent a specific audio feature that the audio designer wants to add to the game and they can have multiple Actions for accomplishing their goals.
https://www.audiokinetic.com/library/edge/?source=WwiseFundamentalApproach&id=understanding_events_understanding_events
In the game editor an event can be "Posted" (called/started) and "Stopped" (self-explanatory) by calling the right Wwise API function that interfaces with the AudioKinetic SoundEngine.
Each Event has its own ID and there can be multiple Events relative to the same sound but with different IDs active at the same time. The ID is returned once it has been Posted in the game editor.

The problem with Wwise

Everything about sounds exciting and flawless but unfortunately it comes with its limitations. The biggest of them is that Wwise Components rely on a set of predefined events in order to be notified and this means that attaching sound effects to specific custom logic (which differs from from that predefined set) for one's game is almost always resulting in a programmer having to write repetitive code where that logic is run in the game. Other limitations are the lack of a reusable module for Animation sound events, the lack of event-grouping functionalities, the inability to extend the Wwise framework in an intuitive way, the limited amount of custom triggers directly configurable in Wwise (You can only create up to 32 custom triggers in Wwise because they use a 32 bit mask for selecting them in the Inspector, sorry, nerdy stuff).

The proposed solution

The HAES aims to solve most of these projects and also adds new features that are not present in the Wwise framework at all. This does not mean that the HAES modules are a superset of the Wwise ones as this would be totally out of scope. It is just a collection of modules that tries to fix most of the frustrating elements of the Wwise-Unity integration by also adding some quality of life improvements.

Structure of the HAES

This chapter is going to present the HAES by both giving a how-to-use description and an in-depth explanation of how it works underneath so that both designers and programmers can make good use of the documentation for usage and extension of the HAES.

The EventManager

The EventManager is one of the two cores of the HAES and it is responsible for managing all of the game-specific events to be fired during gameplay. Since it will never be directly used by designers, the following will mostly be a programming-oriented description.
The EventManager is a static class that can be accessed by anywhere in the code containing the definition of all the events that will be fireable in it. The events are instances of the UnityAction<T> class with T being the primitive type that they pass onwards when they are fired (void is an option of course). When one wants to define a new event, one must declare the new UnityAction instance and the static method that will be used to fire it (so that one doesn't need to remember the syntax); on top of that, one must add the name of the newly created event to the correct enum (i.e.: if the event is a UnityAction<float> its name must be added to the FloatEventType enum). This step is needed in order to be able to show the event list in all of the HAES Modules, so that designers can Post Wwise Events based on in-game

events. The final step, closely connected to the previous one, is to add the correct switch-case to the right switch in both the Subscribe and UnSubscribe functions (i.e.: if the event is UnityAction<float> its case must be added to the SubscribeFloat and UnSubscribeFloat functions).

The AudioManager

The AudioManager is the second core of the HAES and it is responsible for interfacing with the Wwise API and being a middle point between Wwise and the custom game logic. Again, this is a component that no designer should use or modify directly and the following description is going to be a programming-oriented one.
The AudioManager, like the EventManager, is a static class reachable from anywhere in the code containing methods for Posting Wwise Events, stopping already playing Events and methods to manage the list of the currently playing Events. It is also where decorative behaviours (such as the grouping and super-grouping or the replayability of sounds) are implemented and made accessible from the outside. It is strongly suggested not to modify this class unless there is an understood bug with it (and an understood solution as well!).

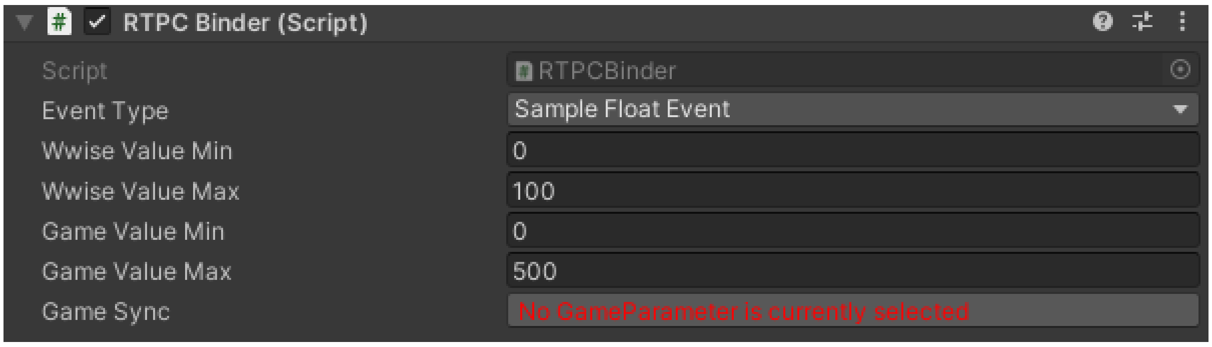
HAES Modules

This will now be a runthrough of all the HAES modules and how to work with them.

The RTPC Binder

An RTPC (Real-time Parameter Controls) is a Wwise functionality that lets the audio designer bind an in-game float value to an in-Wwise sound curve or behaviour. The way it was usually used in game projects was to have either the programmer or the audio programmer code the function to bind a specific in-game value with the correct Wwise element. The RTPC Binder fixes this by presenting the designer with a simple and intuitive Inspector where the Wwise RTPC can be picked and binded with an in-game value from a list of all the Events in the game that currently return float values.

How to use it


RTPC Binder (Script)

Script	RTPCBinder
Event Type	Sample Float Event
Wwise Value Min	0
Wwise Value Max	100
Game Value Min	0
Game Value Max	500
Game Sync	No GameParameter is currently selected

- Event Type: The EventManager Event to be binded with the Wwise RTPC.
- Wwise Value Min: The lower bound of Wwise's RTPC value domain (usually 0).
- Wwise Value Max: The upper bound of Wwise's RTPC value domain (usually 100).
- Game Value Min: The lower bound of the in-game variable's value domain (dependent on the variable's nature).
- Game Value Max: The upper bound of the in-game variable's value domain (dependent on the variable's nature).
- Game Sync: The Wwise RTPC value to be binded with the EventManager Event's value.

NOTE: The Game Value Min and Max can both be set based on the actual domain of the variable or to gain a strategic advantage in development (i.e.: a DistanceFromPoint event that sends the current distance between the player and a specific point could be binded with a Wwise RTPC but how would you define when to make the RTPC value reach its max value? the distance could be 0, 100, 1000….So in this case a decision could be made based on the average max distance that the player reaches, etc.)

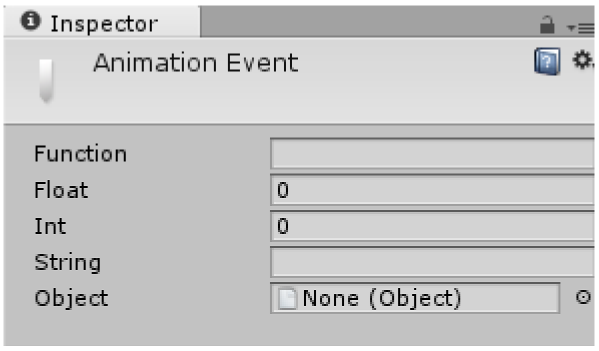
How it works

The component just interpolates the given value from the chosen float event so that it equals Wwise Value Min when the initial event value equals Game Value Min, and it equals Wwise Value Max when the initial event value equals Game Value Max. This value is then set as the RTPC value through an API call to the AKSoundEngine.

The Animation Event Post Audio

Unity allows developers to fire events in specific points of an Animation Clip's timeline. To do this you first need to create a function to be called when the event fires and then add an event to the timeline while also specifying the function to call (the newly created one).
https://docs.unity3d.com/Manual/script-AnimationWindowEvent.html
This process is repetitive and the HAES offers a component to help eliminate the programming side of it.

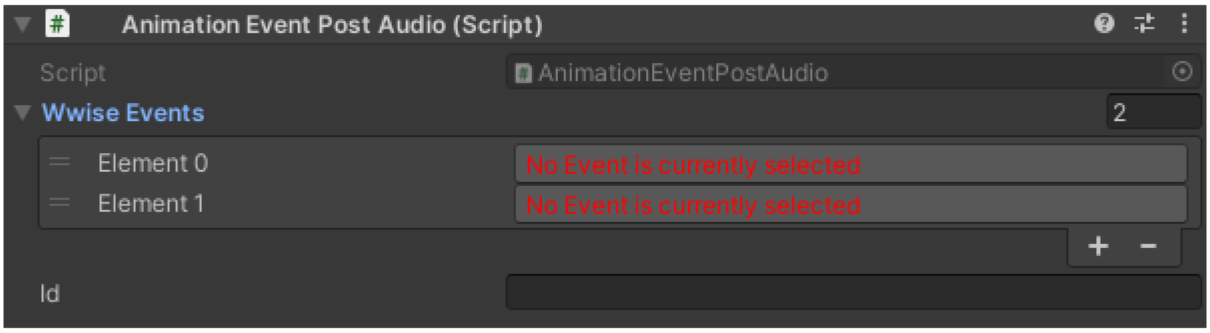
How to use it



Function	
Float	0
Int	0
String	
Object	None (Object)

First add an event by following the previously linked guide. In the event inspector input "PlayEventSound" in the "Function" field and a unique id for the audio event in the "String" field.

Then attach an AnimationEventPostAudio component on the GameObject with the Animator for each animation event you defined (or for each of the ones you want to listen to).



Animation Event Post Audio (Script)

Script	AnimationEventPostAudio
Wwise Events	2
Element 0	No Event is currently selected
Element 1	No Event is currently selected
Id	

- Wwise Events: The list of Wwise events to be Posted when the animation event gets fired.
- Id: The string Id also present on the animation event to listen to.

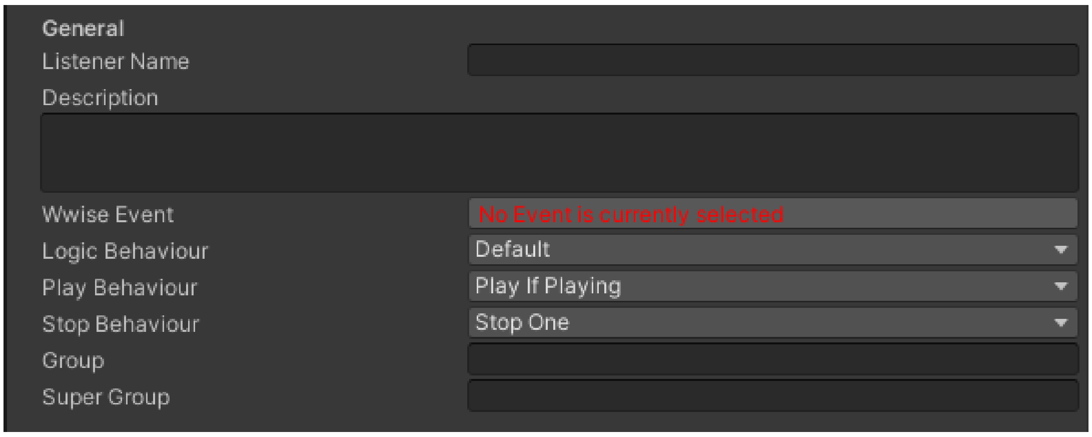
How it works

The component has a function called PlayEventSound that uses the string argument to check if the id set in the animation event is the same set on the AnimationEventPostAudio component. If that's the case the component posts the Wwise Event but without adding its ID to the currently-playing events list in the AudioManager.

The AudioPoster Class

Most of the HAES Modules' goal is to Post Wwise Events based on a varied set of logic behaviours, but they usually share some functionality. This is why all of the Posting HAES Modules inherit from the same parent class (AudioPoster) that includes shared fields and functions.

How to use it



General

Listener Name	
Description	
Wwise Event	No Event is currently selected
Logic Behaviour	Default
Play Behaviour	Play If Playing
Stop Behaviour	Stop One
Group	
Super Group	

- Listener Name: The name of the specific instance of the HAES module. It serves as an easy way to manage them in the inspector.
- Description: General notes that can be written by an audio designer to remind themselves of the purpose of the specific HAES module.
- Wwise Event: The Wwise Event to post.
- Logic Behaviour: Whether or not the logic of the Component should be left untouched or inverted. (i.e.: if a component Posts when a bool variable is “true”, when the logic is “Inverted” it will Post when the variable is “false”).
- Play Behaviour: Whether or not to Post an Event if the same event is already playing in the game.
- Stop Behaviour: The behaviour the component will have when stopping an Event:
    - Stop One: Stops the first instance of the selected Event in chronological order (the first one that was played among the currently playing ones)
    - Stop All: Stops all of the instancesof the selected Event that are currently playing in the game.
    - Stop Group: If the Group field contains a string, it will stop all of the instances of the selected Event that are currently playing that have the same Group.
    - Stop Super Group: If the Super Group field contains a string, it will stop all of the instances of all the Events that are currently playing that have the same Super Group.
- Group: This field is used both by the Play Behaviour and the Stop Behaviour. If it contains a string it adds the instance of the Posted Event to a Group with the string as its name.
- Super Group: This field is used both by the Play Behaviour and the Stop Behaviour. If it contains a string it adds the instance of the Posted Event to a Super Group with the string as its name.

NOTE: From now on, all of the AudioPoster’s fields are not going to be explained again in modules that inherit from it.

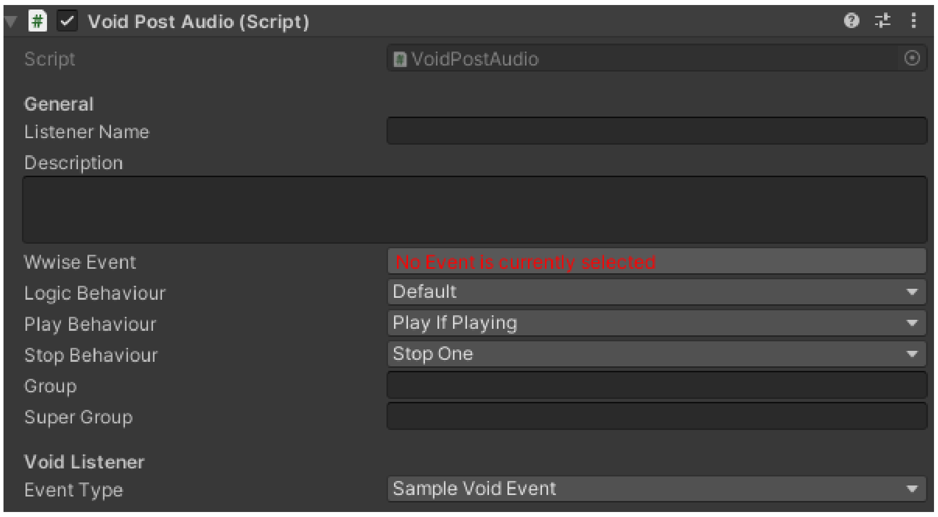
How it works

The AudioPoster basically provides a CallFunction that in turn calls the Play/Stop function for the specified Event based on the Logic Behaviour.
The PlayFunction is essentially calling the AudioManager API variants based on whether one wants to also attach a Group or a Super Group (or both) to an Event, or whether one wants to replay the event if it’s already playing or not.
The StopFunction is calling different AudioManager stopping APIs based on whether one wants to stop a single instance, all of them, the ones in a Group or the ones in a Super Group.

The Void Post Audio

The Void Post Audio is a HAES module used for posting a Wwise event whenever a specific event with no parameters (void) is fired. It is a really straight-forward module and it is one of the most commonly used ones.

How to use it



Void Post Audio (Script)

Script	VoidPostAudio
General	
Listener Name	
Description	
Wwise Event	No Event is currently selected
Logic Behaviour	Default
Play Behaviour	Play If Playing
Stop Behaviour	Stop One
Group	
Super Group	
Void Listener	
Event Type	Sample Void Event

- Event Type: The EventManager's void event to listen to.

NOTE: The default Logic Behaviour for the Void Post Audio is to Post the Wwise Event when the event gets fired. The inverted Logic Behaviour for the Void Post Audio is to Stop the Wwise Event when the event gets fired.

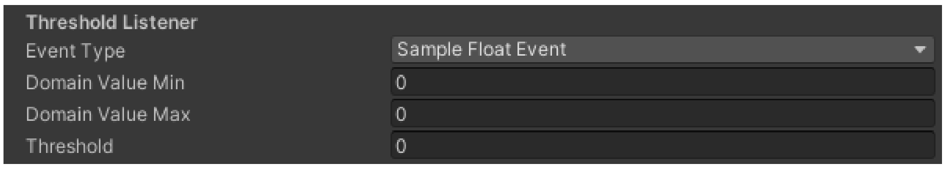
How it works

Come on…

The Threshold Post Audio

The Threshold Post Audio is a HAES module used for posting a Wwise event when the value returned by the selected EventManager float event goes over a defined threshold.

How to use it


Threshold Listener

Event Type	Sample Float Event
Domain Value Min	0
Domain Value Max	0
Threshold	0

- Event Type: The EventManager’s float event to listen to.
- Domain Value Min: The lower bound of the event’s value domain.
- Domain Value Max: The upper bound of the event’s value domain.
- Threshold: The threshold to check against.

NOTE: The default Logic Behaviour for the Threshold Post Audio is to Post the Wwise Event when the float event’s value is higher then or equal to the threshold value and Stop it otherwise. The inverted Logic Behaviour for the Threshold Post Audio is to Play the Wwise Event when the event’s value is lower than the threshold value and Stop it otherwise.

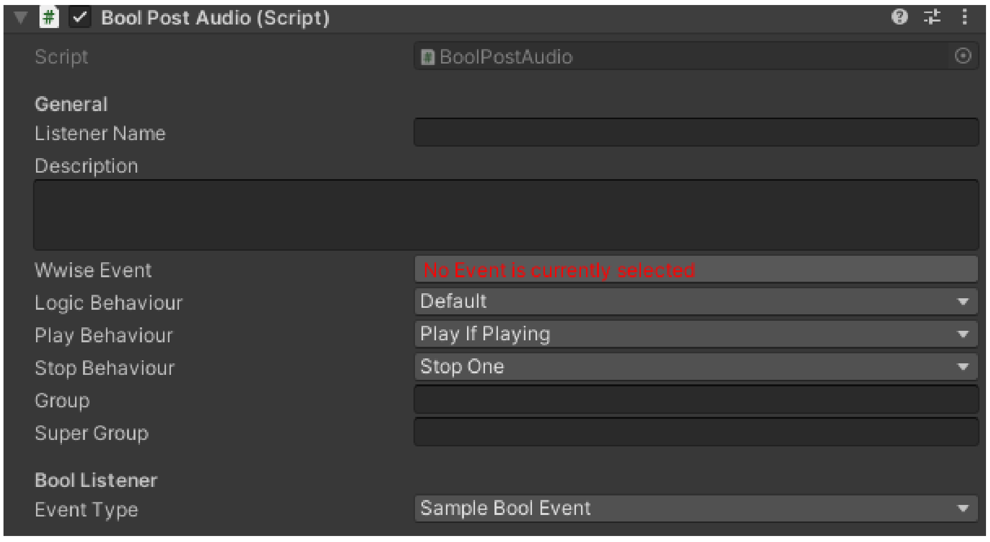
How it works

The component is simply checking if the value is over the threshold or not and based on the Logic Behaviour, Post/Stop the Wwise Event.

The Bool Post Audio

The Bool Post Audio is a HAES module used for posting a Wwise event when the value returned by the selected EventManager’s bool event is “true”.

How to use it


Bool Post Audio (Script)

Script	BoolPostAudio
General	
Listener Name	
Description	
Wwise Event	No Event is currently selected
Logic Behaviour	Default
Play Behaviour	Play If Playing
Stop Behaviour	Stop One
Group	
Super Group	
Bool Listener	
Event Type	Sample Bool Event

- Event Type: The EventManager’s bool event to listen to.

NOTE: The default Logic Behaviour for the Bool Post Audio is to Post the Wwise Event when the bool event's value is true and stop it otherwise. The inverted Logic Behaviour for the Bool Post Audio is to Play the Wwise Event when the bool event's value is false and Stop it otherwise.

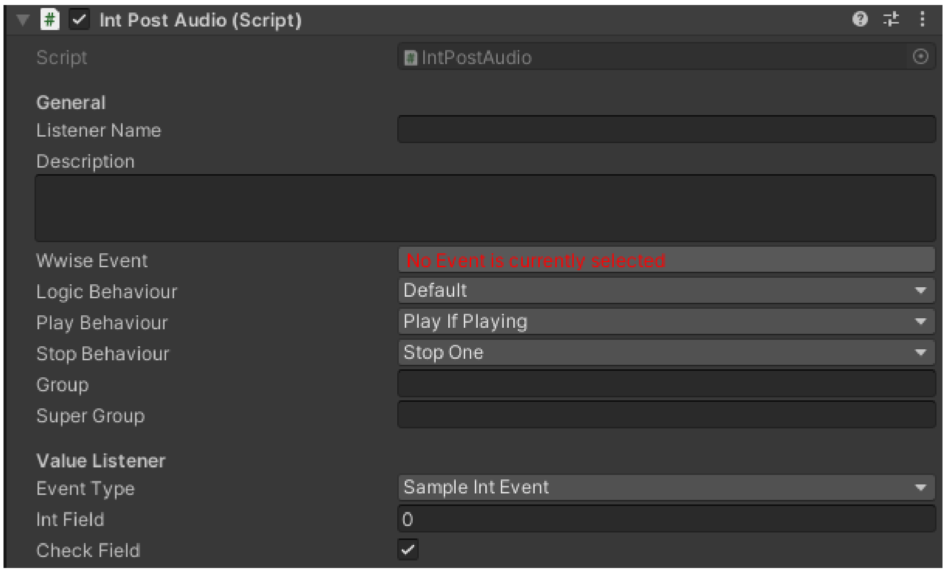
How it works

Come on…

The Int Post Audio

The Int Post Audio is a HAES module used for posting a Wwise event when the value returned by the selected EventManager's int event is equal to the value inserted by the user.

How to use it



Int Post Audio (Script)

Script	IntPostAudio
General	
Listener Name	
Description	
Wwise Event	No Event is currently selected
Logic Behaviour	Default
Play Behaviour	Play If Playing
Stop Behaviour	Stop One
Group	
Super Group	
Value Listener	
Event Type	Sample Int Event
Int Field	0
Check Field	✓

- Event Type: The EventManager's int event to listen to.
- Int Field: The int value to check the event's value against.
- Check Field: Whether or not to care about the Int Field. If this is unchecked the module is going to behave exactly as a Void Audio Post but with int events as callers. (It can be useful to uncheck this in those cases when events are made to pass an int value that is only useful for programmers and not at all connected to audio)

How it works

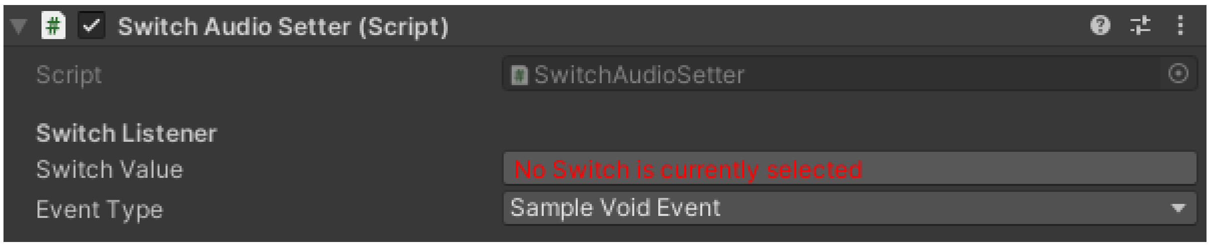
Are you serious?...

The Switch Audio Setter

The Switch Audio Setter is a HAES module used for setting a Wwise Switch Value whenever the EventManager's void event is fired.

https://www.audiokinetic.com/library/edge/?source=Help&id=working_with_switches_working_with_switches

How to use it


Switch Audio Setter (Script)

Script: SwitchAudioSetter

Switch Listener
- Switch Value: No Switch is currently selected
- Event Type: Sample Void Event

- Switch Value: The Wwise Switch Value to set.
- Event Type: The EventManager's void event to listen to.

How it works

The component makes use of the AKSoundEngine API directly in order to call the SetValue on the AK.Wwise.Switch component.

The State Audio Setter

The State Audio Setter is a HAES module used for setting a Wwise State Value whenever the EventManager's void event is fired.

https://www.audiokinetic.com/library/edge/?source=Help&id=states#:~:text=A%20State%20is%20a%20Wwise,to%20changes%20in%20the%20game.

How to use it/How it works

The component is almost identical to the Switch Audio Setter but instead it uses an AK.Wwise.State underneath, so have a look at the latter's description to know how to use it.

Extending the system

If one wants to create and add new modules to the HAES, one could go two routes: if the new module is going to Post a Wwise Event then it should inherit from AudioPoster and use either the CallFunction or the PlayFunction to post the event. If the new module, on the other hand, is going to use some other feature of the Wwise Event then the custom logic to be attached to that feature must first be implemented in the AudioManager and then called in the new module.