GMWwise Manual

A Guide on how to integrate and use GMWwise

ChrisTofer Klassen

2017

Contents

[Overview 1](#_Toc487974924)

[Supported Wwise Functionalities 1](#_Toc487974925)

[Supported plugins 2](#_Toc487974926)

[Error Checking 2](#_Toc487974927)

[Integration 4](#_Toc487974928)

[Tutorial 4](#_Toc487974929)

[Adding Banks and Streamed FIles 4](#_Toc487974930)

[Creating a Sound Controller 4](#_Toc487974931)

[Defining and Posting Events 5](#_Toc487974932)

[Toggling Wwise Profiling 6](#_Toc487974933)

[Working with other Functionalities 7](#_Toc487974934)

[Closing Thoughts 7](#_Toc487974935)

# Overview

The purpose of this document is to introduce GMWwise, describe its usage, and help you understand how to integrate it with an existing GameMaker: Studio project, as well as how to use it.

GMWwise is a GameMaker: Studio extension that, along with a set of .dll and .dylib files, will allow you to integrate Wwise audio banks with your project. It was originally created by Cédric Liaudet.

GMWwise currently supports version **2016.2.3.6077** of Wwise on Windows and Mac.

# Supported Wwise Functionalities

GMWwise supports many of the core functionalities offered by the Wwise engine. These functionalities are:

* Plugin registration
* Listener position setting (2D and 3D)
* Object position setting (2D and 3D)
* Game object groups
* States and State Groups
* Switches and Switch Groups
* Events
* Triggers
* Object RTPCs
* Global RTPCs
* Profiling integration

# Supported plugins

GMWwise currently supports the following plugins:

* Sine Source
* Tone Source
* Silence Source
* Delay
* Stereo Delay
* Parametric EQ
* Matrix Reverb
* Room Verb
* Convolution Reverb
* Compressor
* Gain
* Expander
* Flanger
* Tremolo
* Harmonizer
* Peak Limiter
* Pitch Shifter
* Meter
* Vorbis Decoder
* Guitar Distortion
* Motion Generator
* Rumble
* Sound Seed Impact
* Sound Seed Wind
* Sound Seed Woosh
* Synth One
* Time Stretch

# Error Checking

Most of the GMWwise functions communicate results in the form of their return type: a **double**. Functions that have a return type will return **0** if no error has occurred, and **-1** if an error has occurred.

In the event that the return value is **-1**, you can call gmwGetError()to retrieve the error code. This error code corresponds to the AKRESULT enum defined on the AudioKinetic website:

<https://www.audiokinetic.com/library/edge/?source=SDK&id=_common_2_ak_types_8h_a64f7d1f79613cc4dcc49a4efba6caa63.html>

# Integration

The integration of GMWwise into your project, existing or new, is a relatively simple process. Please follow the steps listed below.

1. Create or open your GameMaker: Studio project.
2. **For GameMaker: Studio 2 only:**
   1. Open the GameMaker Marketplace within the project
   2. Find and download the “GMWwise” extension
3. **For GameMaker: Studio 1.4 only:**
   1. Right click on ‘Extensions’ in your resource tree and select ‘Import Extension’
   2. Navigate to ‘Files/GMWwise (GMS 1.4).gmez’ and import it
4. Create a group within the **Included Files** section and call it ‘sound’.
5. Create a group within the **Included Files** section and call it ‘banks’.

At this point, you’re ready to start working with GMWwise! Continue on to the next section to learn how all of this works.

# Tutorial

This section of the manual will teach you how to start building code that will allow you to start setting up, playing, and profiling audio. Please note that many of the concepts covered in this tutorial are demonstrated in the Sample Project that came as part of this download. This tutorial will reference specific code segments from within the Sample Project, so you may want to have it open as you follow these steps.

This tutorial will be divided into a number of subsections for your convenience, each one covering a specific aspect of GMWwise.

**Important:** In order to follow this tutorial, you will want to use the banks provided with the Sample Project, or a bank that you personally have created using a supported version of Wwise.

## Adding Banks and Streamed FIles

Using either the provided banks from the Sample Project, or your own banks, perform the following steps for each bank:

1. Navigate to the desired bank file.
2. Click and drag the bank into the **‘banks’** group in your **Included Files**.

Make sure that you have included the Initialization bank when doing this.

Perform the same process for any streamed audio files used by Wwise.

## Creating a Sound Controller

The first thing that we are going to do is create a Controller object that will be responsible for initializing, running, and tearing down the GMWwise engine. These concepts can be found in **objSoundController** within the Sample Project.

1. Create an object and name it **objSoundController** (Or whatever you like!).
2. Copy the code from the Sample Project’s **Create Event.**

This code initializes the GMWwise engine. When copying the code over, ensure that the paths specified by

gmwSetBasePath(...);

Corresponds to the group that you have created in the root of your Included Files. Do the same for the banks initialized within the Create Event.

1. Copy the code from the Sample Project’s **Destroy Event.**

This code tears down the GMWwise engine and unloads all banks from memory. Ensure that the banks being unloaded correspond to the ones you loaded in Create.

1. Copy the code from the Sample Project’s **Step Event.**

This code process the GMWwise engine for each frame of the game.

1. Ensure that **objSoundController** is a persistent object, and then add it to the starting room of your game.

## Defining and Posting Events

In this tutorial, we will be defining events in order to show you how to work with GMWwise.

**Important:** This tutorial will not cover other Wwise concepts such as Triggers and States. It is assumed that you, or a team member, is familiar enough with Wwise to understand how the other available components work. The process is generally the same as how Events work.

1. In the Create Event of **objSoundController** (or wherever you see fit), define an event in the following way:

global.AA = BB;

In the above line, AA is the name of your event. I tend to name events with a prefixof **WWE\_** in order to remind myself that this is an event, rather than a switch, state, etc. BB refers to the **id** of the event, which you can retrieve via the **Wwise\_IDs.h** or **SoundBank.txt** files generated by Wwise (if you enable Generate header file in your Wwise project settings).

1. Create or open a game object that will be responsible for posting the event.
2. In the Create Event of this object, type the following line of code:

gmwRegisterObject(id, global.DEFAULT\_GROUP, “XX”);

In the above line, XX is the string name of the registered object. This name is used for Wwise engine profiling, and can be whatever you like. This line of code is registering the object with the default audio group within the Wwise engine, and **must** be done before making GMWwise calls from this object.

1. In the Destroy Event of this object, type the following line of code:

gmwUnregisterObject(id, global.DEFAULT\_GROUP);

This line of code unregisters the object from the Wwise engine. It is important to remember to do this, in order to conserve system resources.

1. Within the object, locate the place in code where you would like your event to be posted.
2. At this location, type the following line of code:

gmwPostEvent(global.AA, id);

This function posts an audio event to the Wwise engine from the object. The event posted should be the same as the one that you created above.

1. **If this object is moving**, put the following line of code into the Step Event:

gmwSet2DPosition(id, x, y, 0, 0);

This function sets the position of the object within the Wwise engine to match the object’s position in-game.

1. **If this object is the focus of the camera**, put the following line of code into the Step Event:

gmwSet2DListenerPosition(x, y, 0, 0);

This function sets the listener position of the Wwise engine to the object’s position.

1. Run the game and confirm that the event has correctly been posted.

## Toggling Wwise Profiling

It is important to remember that when building a release version of your game, you should disable profiling within the Wwise engine. This is done through the **Extensions** section of the Resource Tree.

1. Expand the **GMWwise** extension.
2. Double click on the **GMWwise.dll** file.
3. Remove the existing **Proxy Files** from the list.
4. Add the release versions of the .dll and .dylib files as Windows and Mac OS X proxies.
5. Ensure that the extension copies to all Windows and Mac OS X platforms in the list below.

## Working with other Functionalities

While this tutorial does not offer guidance on how to work with other GMWwise functionalities, the process is largely similar to defining and posting an event.

The key principles of working with the other functionalities are:

* Ensure that you register objects with the Wwise engine before using them.
* Ensure that you are defining constants for your events, switches, etc.
* Ensure that you are defining constants for your state groups and switch groups.
* Refer to the function headers of the GMWwise script API when unsure about how a function works.
* Use the Sample Project when you are confused.
  + The Sample Project provides an implementation of parameters, switches, events, and states.

# Closing Thoughts

I did not create GMWwise, but I have worked hard to bring it up to date in the original author’s stead. I am currently using this plugin as part of my own team’s ongoing project, and will be updating it when I can to keep it in stride with the current stable release of Wwise.

If you are having issues with GMWwise, or would like to suggest a new feature, please use the GitHub issue tracker and pull request system. You can also contact me at [magnesiumninja@gmail.com](mailto:magnesiumninja@gmail.com).

Thank you for taking the time to read this guide. I hope that GMWwise helps your next (or current) project sound amazing!