

Engine audio modding guide for Wreckfest

by Arto Koivisto

Applies to: Wwise version 2019.2.9

Introduction

Wreckfest is a banger / folk racing video game developed by **Bugbear Entertainment** and published by **THQ Nordic**. This tutorial describes how to create and modify simple loop-based engine audio designs, to be used with customized vehicles in the Windows (PC) version of Wreckfest. The tools provided do not allow modifying engine audio of base vehicles that ship with the game.

The tutorial content targets anyone with next to no prior experience about audio modding or the tools, and is looking to pick up the workflow needed. But some of the topics can also serve as a starting point to gain basic understanding of how engine audio designs are often made for video games.

Even if only the basic workflow is presented there still is quite a lot to take in. So preferably set aside a number of days to fully understand the details and work your way through the tutorial. Don't expect to breeze through in one evening especially if you're unfamiliar with the tools used.

To help you get started by taking small steps, some of the details are split into paragraphs starting with **"TIP:"** or **"NOTE:"**. These exist to slightly expand the topic, but aren't critical to study key basics. You may want to skip these paragraphs on your first read, then come back later to deepen your knowledge.

To fully take advantage of this tutorial, you will also need a copy of a package titled **"wreckfest_vehicle_audio_tools_wwise.zip"**, found under the Wreckfest install directory in the **Tools** subfolder. It contains an example project for all the 'hands on' parts of the tutorial and is also needed to build game compatible assets. This project will be referred to as "The Project" throughout the tutorial.

Should you run into issues following these instructions, then please head over to the [official Wreckfest Discord server](#) or the [official community message board](#). Both have dedicated sections for modding, to discuss and share designs, and help you figure out any specific problem with your audio mod.

License Information

Engine audio assets provided with the The Project have been recorded, edited and kindly donated for this specific use by **Pole Position Production**. The assets are provided for the sole purpose of learning vehicle audio design and creating audio mods for Wreckfest.

Everything included in the The Project is offered for personal use under **Creative Commons** license **By-NC-SA**, Attribution-NonCommercial-ShareAlike. This grants you the rights to create and share your Wreckfest audio mods with other players, but not sell them or include any of the content with other commercial products.

For full license details please visit the following page: <https://creativecommons.org/licenses/by-nc-sa/4.0/>

For more professional grade vehicle recordings please visit Pole Position at <http://pole.se>

Acknowledgments

Big thanks go to:

Niklas, Max and **Robin** at Pole Position for all their work with the example loops and clearing their use for the bundle!

Heddly, Matcha, Sam223, STRMods and **Xi** over at the official Wreckfest Discord server for all their valuable input to help to shape this tutorial!

Initial Tool Setup

The audio in Wreckfest is implemented using **Wwise** by [Audiokinetic](#). This tool consists of a **Software Developer Kit** (SDK) that acts as middleware between game engine and audio designs, and **Authoring Tool**, a graphical editor used to build and manage content like audio objects, mixdown and final assets that the SDK will reference. A run-time version of the SDK is integrated with the game engine and for modding purposes you will only need the Authoring Tool.

The audio control data that gets passed between Wreckfest game engine, **ROMU**, and Wwise is configured to an object database. These objects can be edited using a custom tool called **BagEdit**.

To set up these tools do as follows:

- [Download and install Wwise Launcher](#).
- Run Wwise Launcher and register an account with Audiokinetic.
- After logging in to your account, select 'Wwise' tab, then under 'Install New Version' select "**All**" and locate the correct version of Wwise (same to what's printed below title on first page).
- Start the install process. You will only need the Authoring Tool for Windows platform, and none of the 3rd Party plugins so uncheck any.
- Once Wwise Authoring is installed it should show up in the 'Wwise' tab of the Launcher. Click the wrench icon and create a desktop shortcut.
- Locate **BagEditCommunity.exe** under the Wreckfest installation folder and then, for quick access, create a desktop shortcut for it as well.
- If you did not yet locate The Project under the Tools\ subfolder of Wreckfest main installation folder, do it now and extract to a suitable location. Keep the zip archive under Tools\ for an unmodified copy.
- Open **wreckfest_audio_example.wproj** in Wwise.

It's very important to keep in mind that the Wwise Authoring Tool version needs to match the SDK version that is used in Wreckfest. If there's a mismatch between the two, any audio mods created using **an incorrect version of Wwise will silently fail to load**. If you've previously made audio mods for Wreckfest and wish to update them to work in the game, then please look up Appendix 1 towards the end of this tutorial.

In-Game Vehicle Template

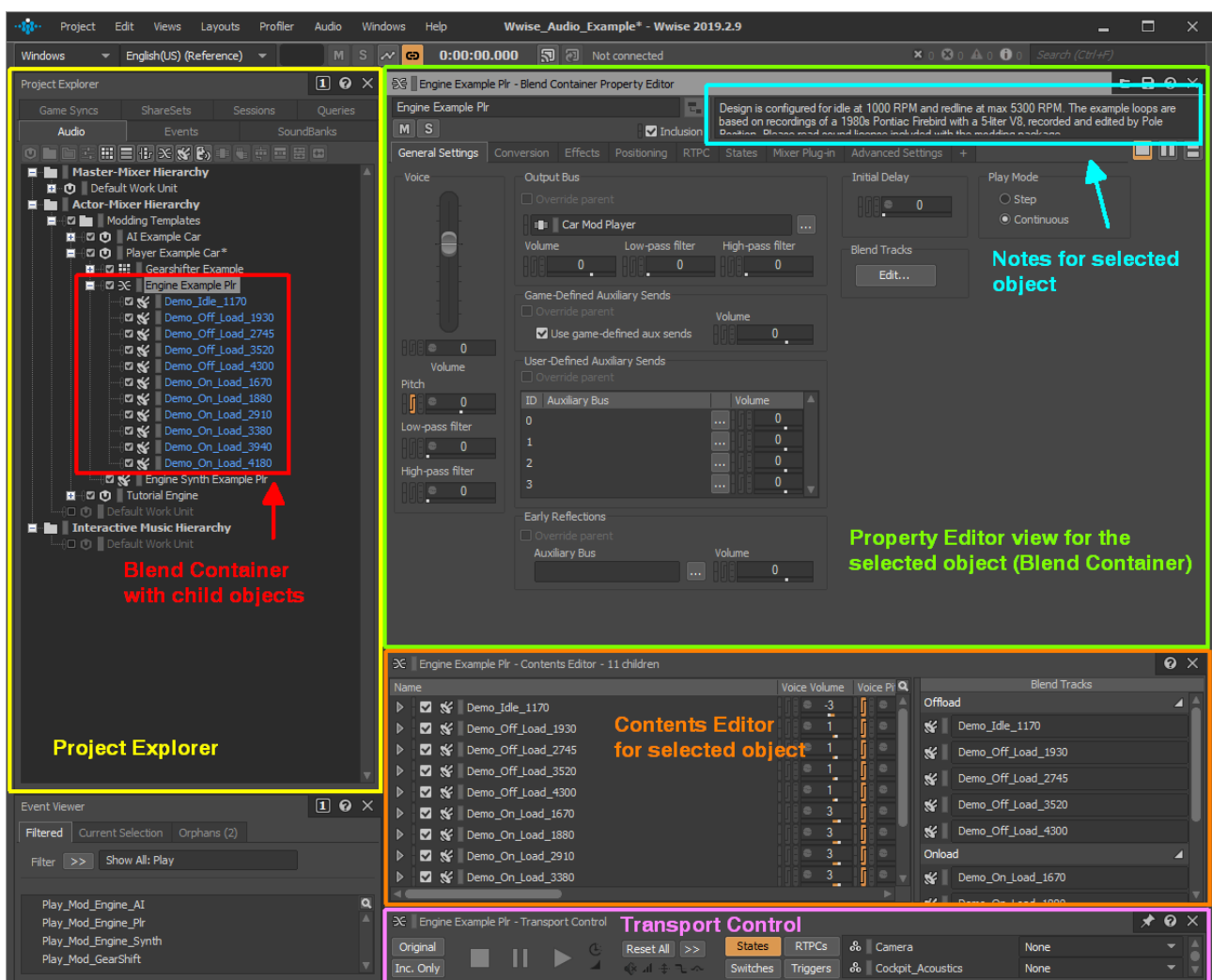
Besides just the above tools, Wreckfest also ships with a pre-configured car template, called '**Example Car**', that will fast-track you into the *wonderful world* of vehicle modding. To enable this template, load up the game, navigate to the '**Misc**' tab of the main menu, select '**Mods**' and enable '**Example Car**'. After restarting the game, the car should then show up in your collection of vehicles.

By default, the Example Car uses sounds made using The Project. As all the necessary objects and configuration is provided, you can at first get started with as little as tweaking these settings or editing assets, and have your first audio mod ready for a test-drive in no time!

Wwise Project Overview

If you are new to Wwise, please keep the [official help documentation](#) at hand, and in particular, maybe study the basics of Wwise graphical user interface with the help of 'Getting to Know.' chapters under [Working with Wwise](#) and [Wwise Project Hierarchy](#). You might also want to have a look at some of the [official video tutorials](#). Additional documentation links to specific topics are also provided throughout this tutorial.

When making or editing engine sounds, you will be primarily working in Designer screen layout and on objects placed under the **Actor-Mixer Hierarchy**. This is a tree / folder structure of parent-child objects, to manage specific settings, signal flow and behaviour of a single object or a grouped set of objects. If Designer layout isn't activated by default, switch to it by pressing F5. This layout consists of **Project Explorer**, **Property Editor**, **Contents Editor** and **Transport Control** views. You will find Actor-Mixer Hierarchy under the 'Audio' tab of the Project Explorer.



Picture 1: Wwise Authoring Tool with Designer layout (F5) active, and Blend Container object selected in Actor-Mixer hierarchy.


Notes have been included with almost every object in The Project, to help you understand a specific use or related details of a given object. Usually these notes show up near the top-right corner of any window or view, or to the right end of any object row such as those in the Contents Editor for a Blend Track. Click on the small Notes box to expand and show all of its contents.

To play or preview the sound of any Actor-Mixer object, first select the object to activate it to the Transport Control, then press **Spacebar** to play. If the object has any automation configured, chances are you may not hear anything until these parameters are set to values that would produce sound in the game. In this case, pressing **Shift + Spacebar** will play the “raw” sound asset instead, bypassing all automation and processing that may have been configured.

In Picture 1, did you notice the “**Demo_***” objects highlighted in blue? For any object in the Actor-Mixer Hierarchy, this means that it hasn’t yet been converted (generated). Any object that is in this state *will not play a sound preview until converted*. To do so first select the object or its parent (to convert multiple objects at once), then press **Shift + C** followed by **Enter**. You will need to do this conversion every time when an object is or turns blue.



Actor-Mixer Object Basics

Majority of work needed to build your engine mod is handled under Actor-Mixer Hierarchy. Take a look at folder **'Modding Templates'**, and you'll find Work Units (icon ) for Example Car (AI and Player separately) and this Tutorial. Poke around the Example Car objects and various tabs of Property Editor to get some idea about the per-object settings.

TIP: Right above Actor-Mixer Hierarchy is the **Master-Mixer Hierarchy**. This is the audio mixing stage of a project. Output of Actor-Mixer objects is always routed to a Master-Mixer object to create the final audio mix. The Master-Mixer Hierarchy is pre-configured in the game, and any changes you make at this level of the project will not carry over to the game. Say, to have audio effects like Distortion or EQ affect the sound of your vehicle, you have to add such plugins into Actor-Mixer objects.

Besides the example assets provided, you can also import some of your own. The assets are called **'Media Files'** in Wwise lingo and, once imported, get converted to **'Sound' objects** (aka **SFX Objects**). These can be freely moved or copied around and nested into Container objects anywhere within the Actor-Mixer Hierarchy. But make note that the *free version of Wwise has a limit of 200 objects per project!* This also includes events and such needed to trigger and control playback of sound objects. If you exceed this limit, soundbanks can no longer be generated using the free version, and you then need to delete some redundant objects or start afresh using a newly extracted copy of The Project.

How do you turn Sound objects into an engine object? In Wwise, loop-based engine designs are made by assigning Sound objects onto **Blend Tracks** (or layers) that belong to a single **Blend Container** object (Picture 1). A Blend Container can have as many Blend Tracks as your design calls for. To fully configure both of these, you will also need to have the **Blend Track Editor** view open along the default Designer view. If you don't see this Editor, press **Control + Shift + B** to open it now. A dual-head monitor setup comes in handy here - just float the windowed Blend Track Editor view over to the second monitor.

So how does a Blend Container work for engine sounds? With its Blend Tracks, it defines the order and length of where each of the Sound objects will play, based on the current vehicle physics simulation parameters like RPM and amount of throttle. Think of Blend Track as a playlist that has its play order defined by the Blend Container, and playback controlled by the game simulation parameters. Each Blend Track can loop indefinitely over a specific point (set by RPM) and the entire playlist can be smoothly seeked through forwards and backwards (engine accelerating or decelerating).

TIP: The playback of a Blend Container object is different compared to other Wwise sound object containers, such as Random and Sequence Containers. What makes Blend Container special is that it can play layers of sounds on top of one another, and specifically do so even for its child Sound objects which are not included on any Blend Track layer. This is useful for sounds that need to play constantly along with the Blend Container, but not be affected by the playlist position within any Blend Track. In other words, you don't need to configure a Blend Track for a child object if it should play at all times.

The common standard for an engine Blend Container is to have at least two Blend Tracks - one for **on-load** and another for **off-load**. Together with RPM these describe *the type and amount of work done by the engine*. There's a separate chapter on Page 14 with technical details of how Load works in Wreckfest, but don't skip ahead just yet.

Over the next few chapters we will look at how to configure a Blend Container using the objects included with the **'Tutorial Engine'** Work Unit. We will set up a replica of Example Car only for tutorial purposes, but the workflow is the same if you have a set of custom engine loop assets.

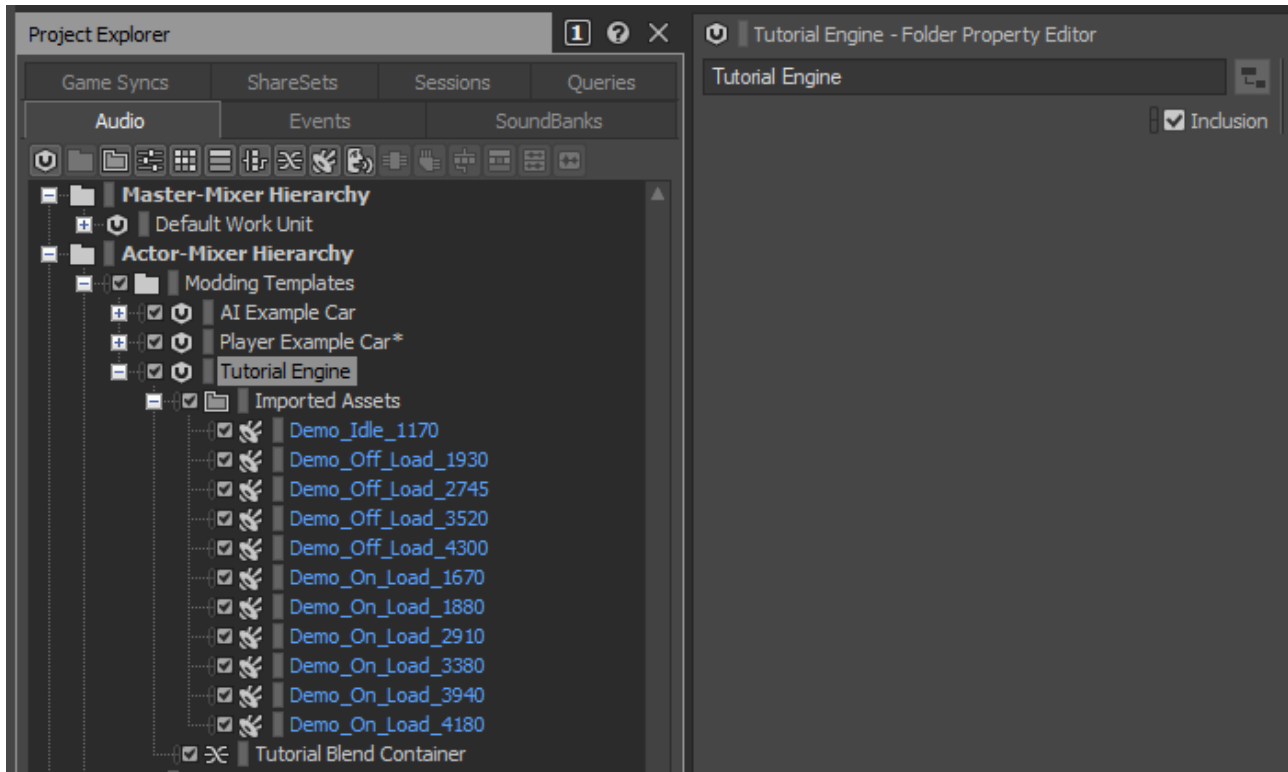
If you're looking to only modify the sound of the Example Car, it still is recommended to read through these chapters. To modify example loops in an external audio editor (such as [Wavosaur](#), [Audacity](#) or [Soundop](#)) you will find them under the folder **'Originals\SFX'** in the project directory. This folder is also where Wwise will place all new Media Files that you import into The Project.

TIP: When starting out with a design, it's a nice practice to add some sort of an 'Import' folder to an Actor-Mixer Work Unit. This will serve as a quick reference for the Sound objects you've imported, to have these objects at hand if you wish to start over with a clean / unconfigured set of assets. Comes in handy for making alternate versions of the engine too.



Initial Sound Object Configuration

Okay let's get started! Locate '**Tutorial Engine**' Work Unit under the Actor-Mixer Hierarchy and expand its contents. You should see an object tree similar to Picture 2. With the exception of some Sound object settings, the 'Imported Assets' folder recreates a situation where you would've first added a folder, naming it 'Imported Assets' and then drag-dropped a set of assets from Windows into the Wwise folder for import.

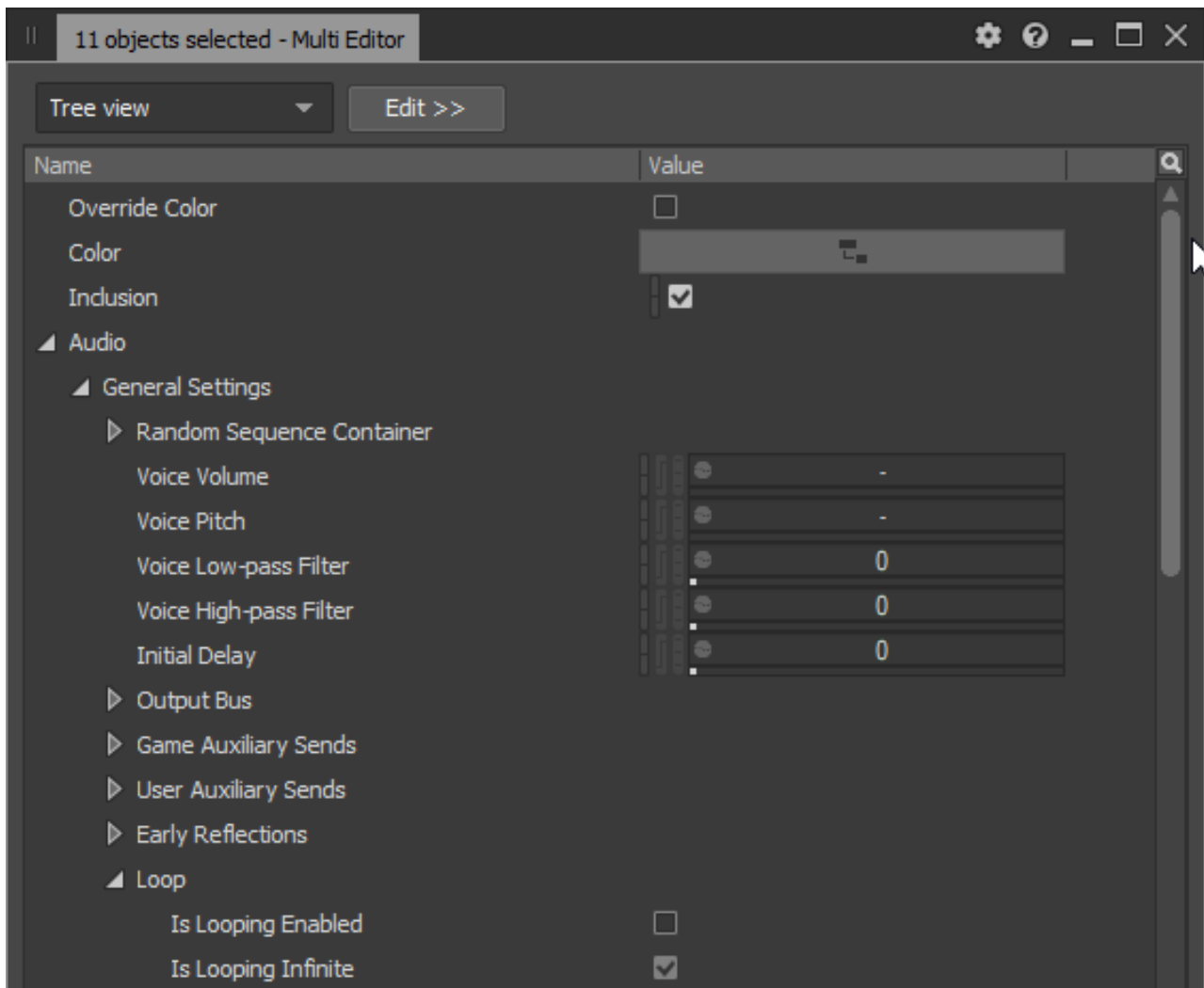


Picture 2: Default objects under the Tutorial Engine Work Unit

We will first need to enable looping for these “newly” imported assets. Looping is needed to keep Sound objects playing indefinitely, say, if you leave the car engine running on idle to grab a cup of coffee. To enable this, first open the Multi Editor view by pressing **Ctrl + M**, then select the 'Imported Assets' folder. In Multi Editor, click **“Edit >> All Children - 11 Objects (Sound)”** and if not visible, then unfold **“Audio > General Settings > Loop”** in the tree view (Picture 3). Enable options “Is Looping Enabled” and “Is Looping Infinite”. This will make all the 11 Sound objects play indefinitely until stopped. Close the Multi Editor view.

Remember that thing about blue Sound objects? Once looping is enabled we still need to convert the files to hear anything. **Shift-click** to select all Sounds objects under 'Imported Assets' and do the **Shift + C** routine. After conversion completes, you can also select one of the Sound objects and play it by pressing **Spacebar**. As soon as you've had enough, press it again to stop.

NOTE: If you import your own assets, you will need to also set other properties, like the Output Bus. For these refer to Sound objects of the Example Car. For any new assets, you need to match the settings to that of a similar object (Sound, Blend Container) either one-by-one in Property Editor or by bulk in Multi Editor.

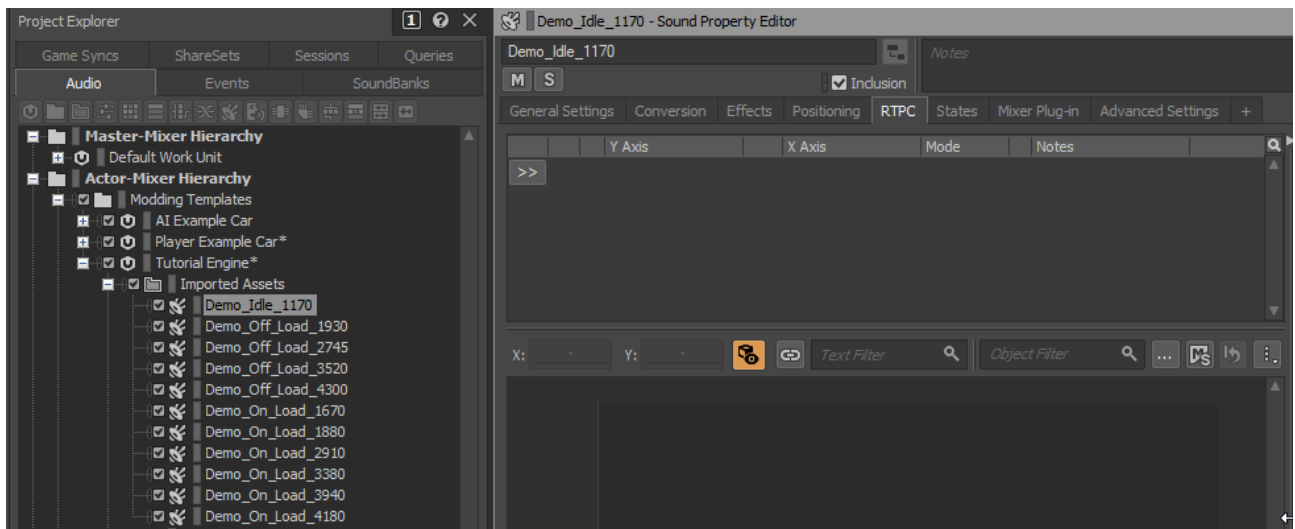


[Picture 3: Where to find 'Is Looping Enabled' in the Multi Editor](#)

To make our Sound objects react to in-game simulation in any kind of way we need to establish a connection between the game and the Sound objects. In Wwise this is handled via RTPC objects which is short for '**Real Time Parameter Control**'. They are our link between the game engine and Wwise SDK specifically for any constantly changing simulation parameter, and in Authoring, are further mapped to control settings of Actor-Mixer objects. You can add as many of these mappings as you like. Let's add one for Sound pitch now.

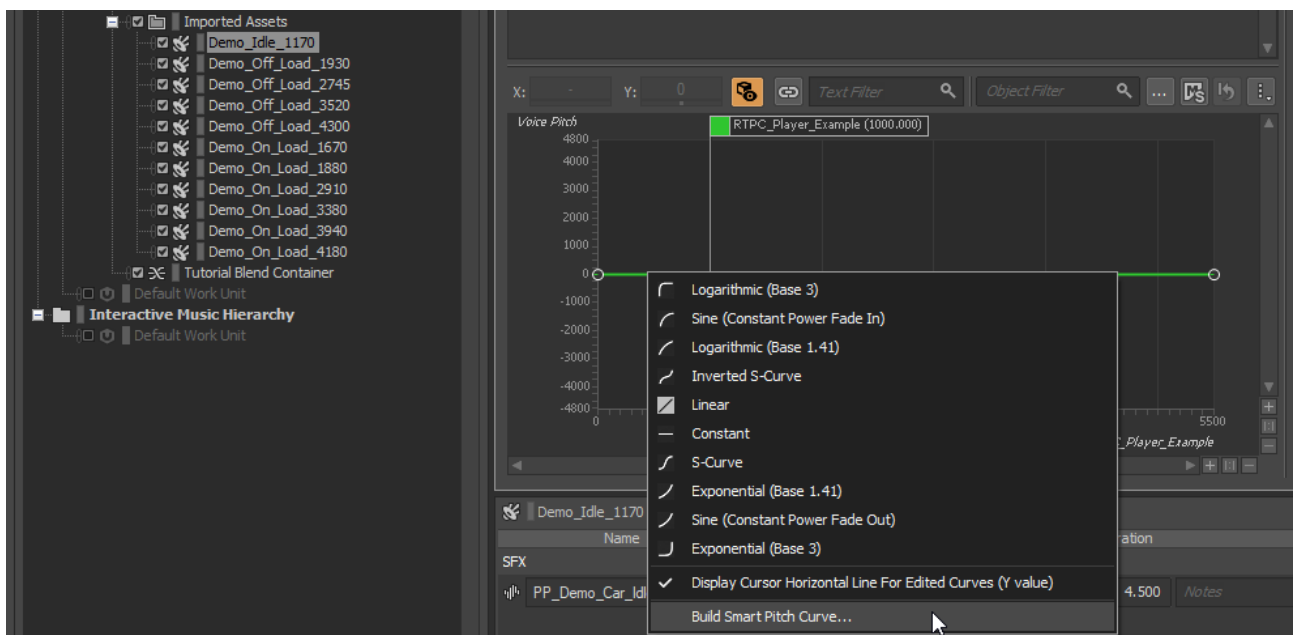
Select any of the Sound objects, then switch to the '**RTPC**' tab in Property Editor (Picture 4). First click the '>>' button to the left of '**Y Axis**' column and from the context menu, select 'Voice Pitch'. A similar '>>' button will now show up next to '**X Axis**' column. Click it and assign '**Game Parameters > Modding Example > RTPC_Player_Example**'. A colored flat line and a flag should now appear in the box below. Try moving the white circles at both ends of the line. Play the sound and drag the flag that is above the X-Y grid, and you should now hear the pitch change depending on the location of the flag. Any such flag is our simulated value for the in-game RTPC, and the particular RTPC which we linked to Voice Pitch, is the RPM parameter for the default Example Car. So now we have established pitch tracking based on RPM for one Sound, but we're not done yet by far!

To easily set up consistent pitch tracking across a set of objects, like our engine loops, Wwise comes with a tool to do this automatically based on the **natural pitch** (or RPM) of the Sound object. We will use this for now, even if the thought of creating our own wacky pitch behavior may seem the more exciting choice.



Picture 4: Sound object and its Property Editor tab for RTPC mappings list (none configured yet)

Look at the Sound object you selected. Its name includes a numeric value which represents the natural RPM of the object ("1170" for object selected in Picture 4). Right-click anywhere on the colored line and select **'Build Smart Pitch Curve'** from the context menu to show a settings dialog (Picture 5). In the upper box, input the numeric value of the Sound object and click **'OK'**. If you move the flag over the line shape that was generated, you should see that the zero crossing of the Y Axis lines up with the numeric value of the Sound object. If not, run the **'Build..'** tool again to check for typos.



Picture 5: Context menu for the RTPC automation mapping with 'Build Smart Pitch Curve' selected

Now that we have one Sound object configured for root pitch tracking, we still need to do the same for the remaining 10 objects. Select the row for our newly created parameter in the RTPC mapping list (above the automation curve), copy it by pressing **Ctrl + C** and paste this mapping to other Sound objects. Once pasted and as you go, run the **'Build..'** tool and input the correct natural RPM value of the object.

NOTE: Root pitch tracking needs to be configured **only** if you want the engine pitch to follow in-game RPM and/or to stay consistent over a set of assets. If you don't care about pitch, or are designing a highly experimental / imaginary engine that doesn't need to follow any real-world RPM convention, you can choose to skip this part of the process.

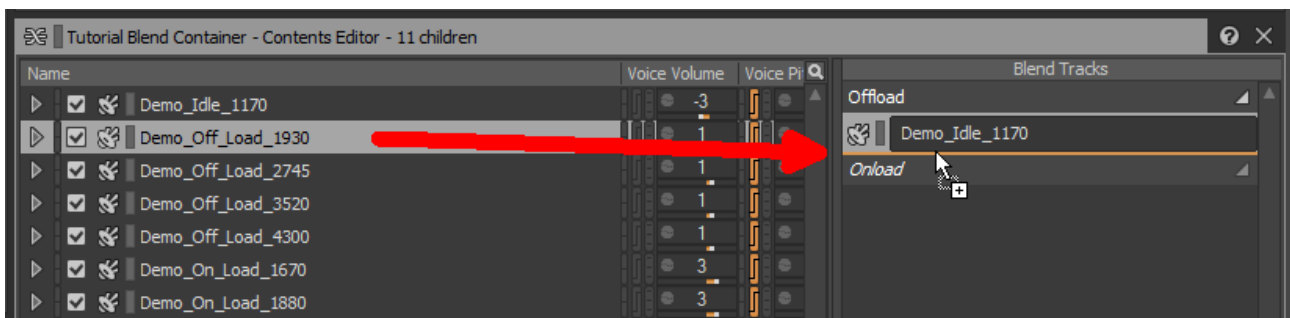
Blend Container Configuration

For details on how to monitor a Blend Track, please refer to chapter.. .. wait for it(!).. [Auditioning the Contents Of a Blend Container](#) of Wwise documentation.

With looping and root pitch tracking configured, we now move on to setting up the **'Tutorial Blend Container'** object. You can find it under the **'Tutorial Engine'** Work Unit. If you do not yet have the Blend Track Editor view open, press **Ctrl + Shift + B** to open it now, and click the said Container object in Project Explorer to show its contents in the Editor view.

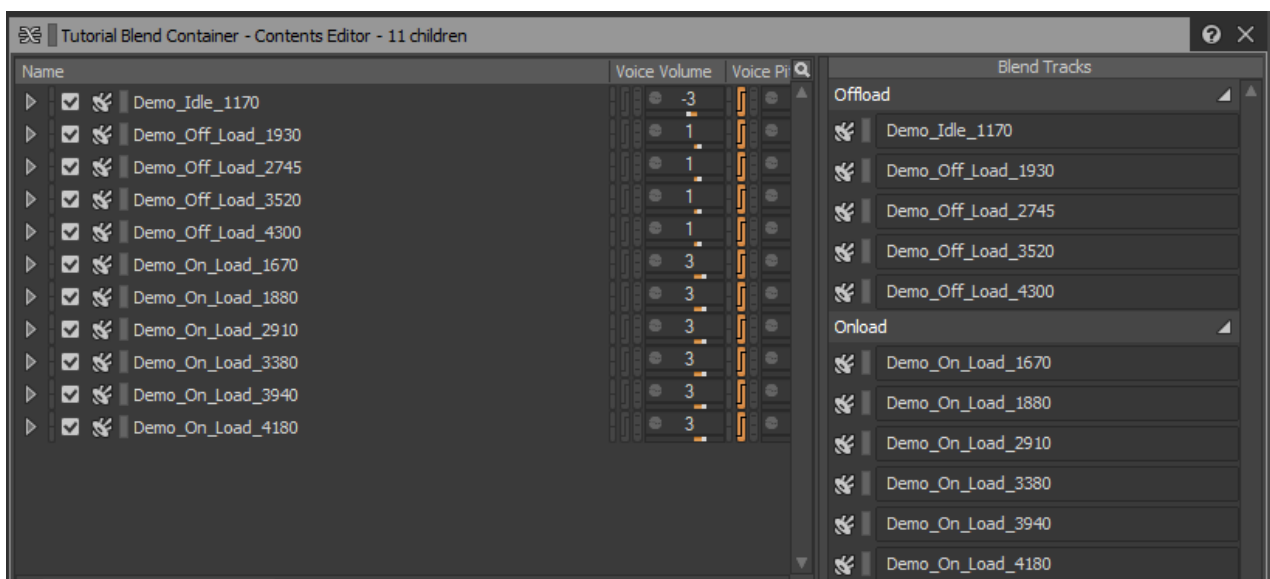
For brevity of this tutorial, the Container is pre-defined with Blend Tracks for on- and off-load and RTPC automation mappings for both. If you were to create an entirely new Container, you would then add Blend Tracks, then assign and calibrate similar mappings.. or copy/paste them from another Blend Container.

First, the Container needs child objects. **Shift + click** to select all the Sound objects under **'Imported Assets'** and **Ctrl + click-drag** them over the Container. This will create copies of the Sound objects. Now look at the Contents Editor for the Container to assign the objects we just added, into the appropriate Blend Tracks.



Picture 6: Adding assets by drag & drop on a Blend Track

You probably noticed that the objects have either **"Off"** or **"On"** in their title - this is our clue for the correct Blend Track! Drag each of the files over to the appropriate Blend Track, sorting them in ascending order based on the natural RPM (idle to highest). The list order defines the order of objects in Blend Track Editor, and you can drag/drop here to reorder. Before moving on make sure that the order matches that shown in Picture 7 before - any changes to list order can only be handled here!

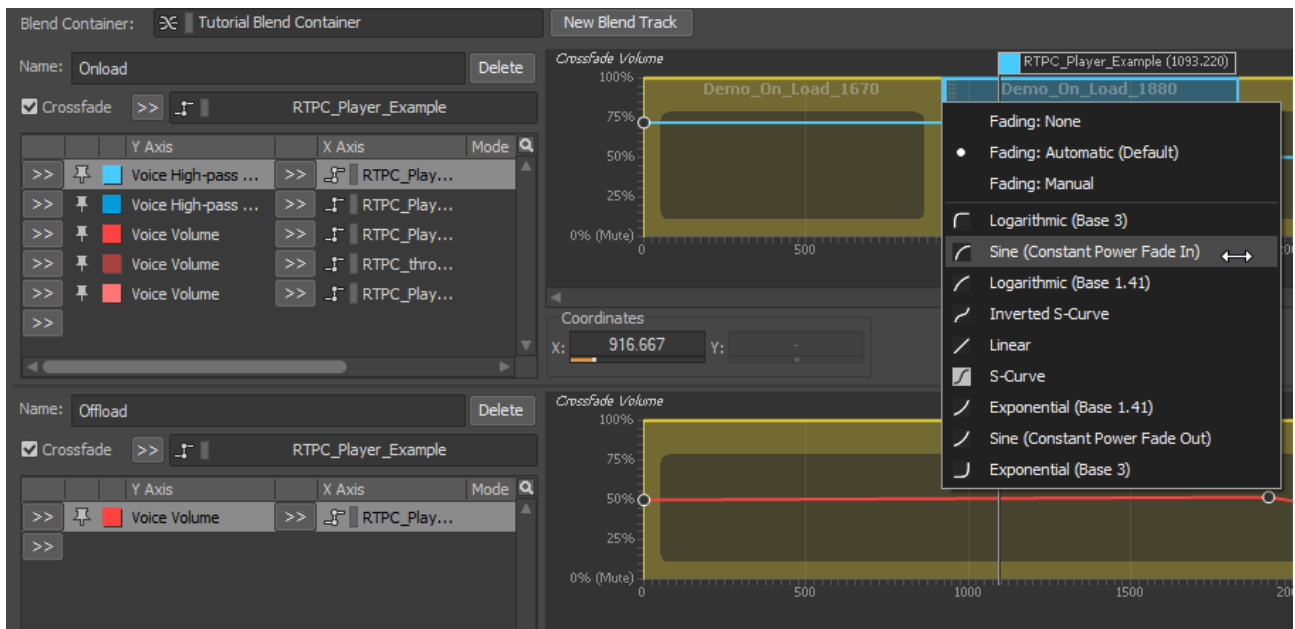


Picture 7: Contents Editor view after adding child objects to Off- and Onload Blend Tracks

Going over to Blend Track Editor, you should see the off- and on-load Blend Tracks now filled with a bunch of colourful boxes. These boxes represent the child objects that we just added to Blend Tracks in the Contents Editor. Note that **'Crossfade'** is enabled and that it's been assigned to track **'RTPC_Player_Example'** (RPM of Example Car); this tells the Blend Container *where to start and stop playback of each Sound object, as the simulation parameter crosses any box boundary*. If there is a gap between boxes, no sound will play while the RTPC value is within the gap.

If you don't see a grid for RPM under the on-load Blend Track then, to the left, click one of the X-Y mappings that has RTPC_Player_Example defined for the X Axis. Selecting any of the mappings will change the grid to show appropriate scale / range for the X-Y pair.

TIP: Any X-Y mappings defined for a Blend Track are applied to all its objects. If you wish to configure automation for each child object individually, you need to do so in the RTPC tab of the Property Editor view, one child object at a time. Here, copy-pasting automation curves comes in handy.



Picture 8: [Selecting the crossfade type for the left edge of a Blend Track child object.](#)

Starting with the top RPM object, move and resize each object on both Blend Tracks so that they're somewhat centered about their natural RPM range. As you move the boxes, you may notice that adjacent boxes can also overlap. Next, right-click on both left and right edges of each box and, from the context menu, select **'Sine (Constant Power Fade In)'** or **'Sine (Constant Power Fade Out)'** for left and right box edges respectively.

Why this type of edge crossfade? We set it specifically because *it will hold a constant volume level near the midpoint of the crossfade*. If you wish, you can try and listen to this by f.ex. configuring a single crossfade pair to use **'Linear'** fade type instead. We don't want the volume for our engine to dip over a crossfade as it spoils the feeling of continuity, as RPM sweeps over a crossfade.

TIP: If you hear the volume level drop over the crossfade even if setting the 'Sine' type, you then need to experiment with other shape combinations. As a starting point, set one edge fade of a pair to **'Base 3'** type and then try out other shapes for the other.

With objects placed about their natural RPM and edge fades configured, we now need to adjust the amount of overlap (width) for each crossfade. Set Blend Container to play, move **'RTPC_Player_Load'** flag to value 1 (full on-load) and in on-load Blend Track, work your way down from max RPM to idle, adjusting crossfade widths to your liking. You can also type exact RPM values into the text box, but remember to first click the correct crossfade edge that you wish to type in the value for. Once done, set **'RTPC_Player_Load'** to 0 (full off-load) and again work your way down from max RPM.

When adjusting for this width you want to end up with as narrow an area you can get away with. That is, avoid pushing any Sound object too far beyond its natural RPM, while keeping tonal transition between adjacent objects smooth and without a perceivable dip in volume. In general, pitching objects below their natural RPM has more “reach” than going above does. This means that often you can stretch the left edge of a box down more than the right edge up. But it also depends on the pair of objects to crossfade against one another.

NOTE: *Keep in mind that while dragging RTPC flags left and right over the full Blend Track range will simulate how the assigned parameter will affect the sound in-game, what you hear isn't an accurate representation of final in-game sound, where **all parameters change together** according to player input and physics. Nonetheless you can still set static combinations of RTPC flags that will give you some estimate of how parameters (f.ex. Load and RPM) will most likely work together in the game.*

This wraps up the basic Sound object setup for our Tutorial Blend Container! You can still continue modifying the objects by calibrating the loudness levels. Beyond this though we have very little reason to generate and set up game assets for the Tutorial Blend Container since it's a copy of Example Car. For any changes you may want to try out in-game, you are simply faster off with editing the Example Car assets.

If you're adding custom loop assets, don't forget to check out the Appendix about pitch fine-tuning!



How Engine Load Works

Now that we have some grasp of what RTPC automation is, and what it can look like in the context of our Blend Container object, we should side-track briefly and have a look at parameter implementation of engine Load in Wreckfest. Understanding this will help you better simulate and calibrate engine designs as you create them.

The default Load RTPC object for the Player engine is called RTPC_Player_Load. Setting up a custom RTPC object in BagEdit is also possible, and allows you to configure custom RTPC value interpolation in Wwise etc.

As mentioned previously, **Load indicates the amount and type of work being done by the engine** and, coupled with RPM, it's the other key parameter used to represent the engine operation. In racing games, Load is commonly split to **off- and on-load** for audio simulation, and this is what we use for our designs too.

In Wreckfest, Load parameter has a range of 0 to 1: Here neutral load sits in the middle (0.5), with off- and on-load to its left and right respectively. **Neutral load is the real-world equivalent of idling a car with the clutch disengaged.**

When applying throttle, the Load parameter moves above 0.5, going more towards 1 the harder the engine is working to bring the mass of the vehicle up to speed (or to maintain it) that is set with the throttle pedal. Starting from a stationary position on partial throttle and clutch engaged, the value will first jump near 1 and, as the vehicle gains speed, will then drop down to move between 0.5 and 1 depending on current physics simulation values (like road surface, sloping etc.), and as long as moderate throttle is held. With very low amounts of throttle and clutch engaged, the Load value can instead alternate between 0.4 and 0.6. Flooring the throttle will hold Load at 1.

Fully releasing the throttle (to commence engine braking) or briefly disengaging the clutch (to shift gears), will drop the Load value below 0.5 and more towards 0 the harder the engine is working against the momentum of the drivetrain. With the clutch engaged and no throttle applied, engine braking will continue until the vehicle slows to a stop. This will result in slow change of Load roughly between 0.3 to 0.5 after the initial sharp drop below 0.3. Downshifting gears in motion will briefly spike Load up to between 0.6 and 0.9.

Now compare all of the above against how the Blend Container of Example Car is configured. The Voice Volume automation it has for RTPC_Player_Load represents a generic approach for how key points / shapes should be set. But for any new designs you create, don't just blindly copy value mappings from Example Car! Often you need to tweak the X-Y values based on loops used (style of engine), or how you wish to design the car to sound like. Say f.ex. in the case of idle RPM (where both on- and off-load will play), you can adjust balance between the two by moving the on-load automation points at value 0.5, to change characteristics of how idle sounds.

TIP: *It's recommended to try out also Voice High-Pass filter automation for engine states where on- and off-load might overlap. Depending on how the loop assets are edited, mixing together both of the layers at near-equal volume can sometimes cause bass frequencies present in both layers to "fight" against one another. Weird as it may sound, this can cause some of the frequencies to cancel out instead of boosting one another, and you may actually end up with less bass! If you notice that the volume of bass frequencies fluctuates as you hold a steady RPM, then configuring this type of automation should fix it!*

AI Engine Configuration

The recommended practice for Wreckfest engine audio designs is to first complete the loop setup for the Player version, then make a new copy of it and change RTPC mappings to point to appropriate RTPC objects. Compare the Player and AI versions of Example Car for how their settings differ for the Blend Container, Blend Tracks and RTPC mappings.

If your design for Player car has a lot of plugins that are static (no RTPC automation applied to effects parameters), you then want to enable the “Render” option for such plugins. This will bake any such plugin directly into Sound objects, when the Sounds objects are converted for playback (**Shift + C**). This can help optimize your design for performance.

TIP: Talking about performance and default Example Car, each in-game instance of it (one per car) can have 2 to 4 audio loops playing at any given time. Since in Wreckfest there can be up to 23 AI in a given race, very complex designs can then ramp up audio CPU load in no time. This is especially so if you use a lot of effects plugins, or encode audio assets using Vorbis format (.ogg). Effects placed in any top-level Container objects will get instanced (replicated) for all of its child objects at runtime, resulting in x number of copies of each plugin.



Calibrating Overall Volume

Final step to configure any engine audio design is to adjust its volume settings, or even RTPC automation for Voice Volume, to achieve an overall loudness suitable for the game.

Running this calibration will ensure that your engine design will not excessively overload the audio signal chain or unnecessarily lose dynamic variation. In general the more 'raspy' sounding engines, like a big-block V8 with straight exhaust, will cut through the mix better even at lower volumes. In comparison the softer sounding engines (typically stock consumer type) may even require some boosting and tweaks to RTPC automation on low RPM.

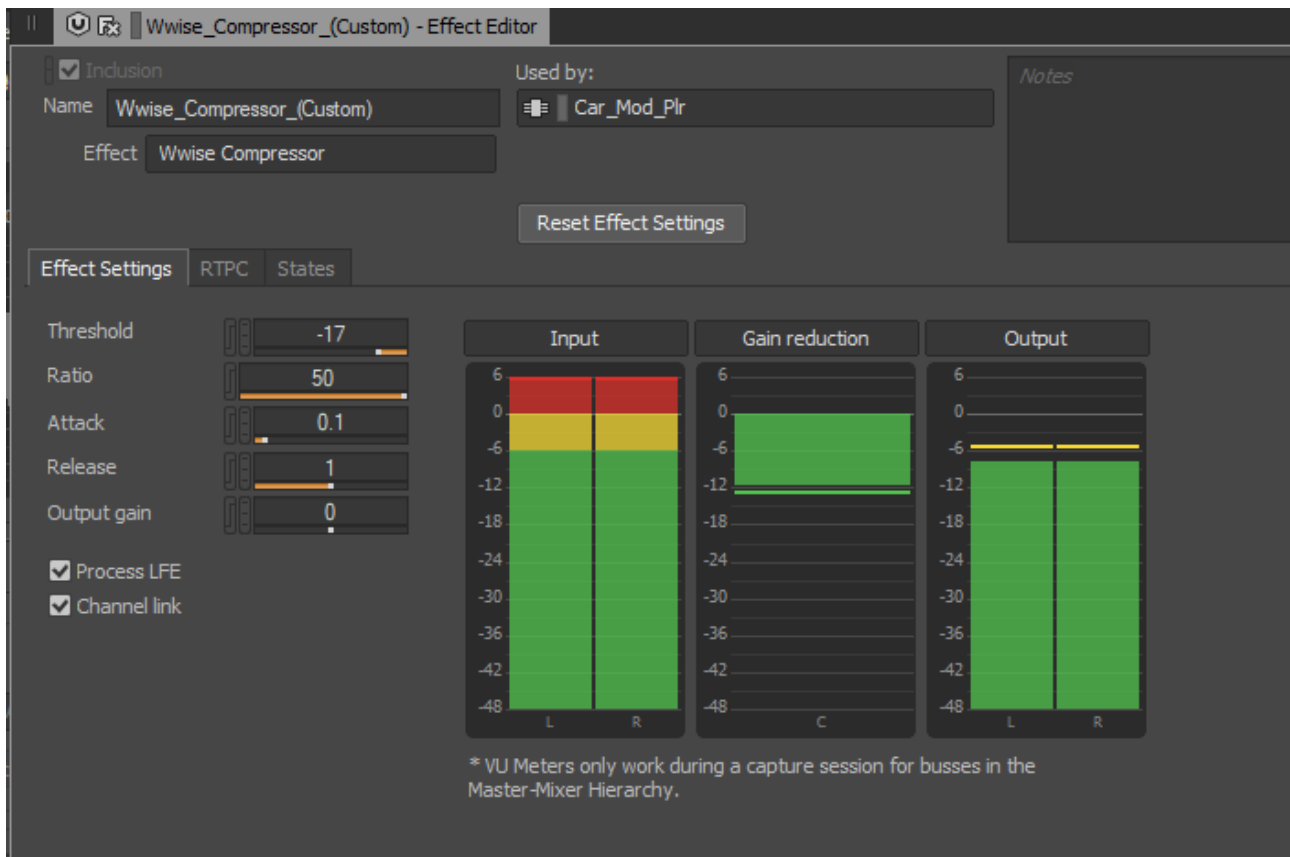
We will only look at precise configuration for the Player vehicle because AI is difficult to simulate in Wwise Authoring alone. Key difference between Player and AI mix buses is that Player bus will only ever need to accommodate *one vehicle at a time*, whereas the AI bus will accommodate *all instances of AI* that use your audio mod.

The Project comes with separate mix buses for Player and AI vehicles, '**Car Mod Player**' and '**Car Mod AI**' respectively, located under Master-Mixer Hierarchy. The game objects under Actor-Mixer Hierarchy have been pre-configured to send their audio to these buses, but if you added any new objects please make sure that they are routed to these buses. Both the Player and AI buses come with a compressor effect plugin to replicate settings used in the game. For the context of The Project this compressor allows you to check (and should you wish to use it) that the loudness of your engine design is about good, and nothing more. As mentioned before, *any changes you make to Master-Mixer Hierarchy will not carry over to the game*.

So how to check for sound levels with your design? The fastest way is to simply A/B compare against the Blend Container object for Example Car! Take turns at playing both objects configured to similar RTPC settings (RPM and Load) and, if the two sound about equally loud, your design is good to go! If not, adjust the Voice Volume slider of your Blend Container object.

For a slightly more engineered approach, you can monitor how the compressor effect is reacting to your audio mod. To do this proceed as follows:

- Locate 'Car Mod Player' bus object under Master-Mixer structure, and with the object selected, then navigate to the '**Effects**' tab in Property Editor view and double-click the compressor to open **Effect Editor**.
- Press **Alt + C** to start capture, select the Blend Track object for your engine and press **Spacebar** to play. The meters in Effect Editor should now update.
- Check that engine simulation is running at full on-load and near redline by adjusting Load and RPM values either with the individual RTPC flags under Blend Track Editor or in RTPC list under Transport Control. Leave Throttle RTPC at zero.
- In Transport Control, select the States tab, change Camera angle to '**Chase**' and Game to '**Race**'.
- Check Input meter in Effect Editor. If the peak value reads close to -6dB with no or very little gain reduction, you're about set. If input shows red or there's a lot of gain reduction, you need to decrease levels (see Picture 9).
- If the engine is too quiet or too loud, adjust the Voice Volume slider of your engine object to match. The slider can only go as loud as +12, but you can key in larger values to its numeric value box.
- Once level calibration is good, press alt-C again to stop the capture and proceed to generate soundbank(s).



Picture 9: [Ouch! If the compressor metering looks like this for your engine mod, you need to turn it down.](#)

If you want to create the AI version using the same Player engine design, then make a copy of it and adjust the Voice Volume slider of this Actor-Mixer object around -3 to -6 dB lower. Exact value is something that you need to test for in-game, but in general an engine with soft / smooth sound needs to be lowered less than that raspy V8.

Test the loudness of your AI vehicle at least in races with single and multiple opponents, the latter preferably outside the starting grid after vehicles have had some laps to spread out over the length of the track. If the AI levels sound about equal to other standard vehicles in the game (and at about equal distance to the camera), you're all set! If the AI car sounds too quiet on low RPM but ok on high RPM, then you rather need to adjust Voice Volume automation of the Blend Container for less dynamic variation. One good way to check and compare is the race replay; here you can switch between camera angles and vehicles.

In the context of full mixdown, the AI needs to be set to a lower volume so that by default the Player car will have priority in the mix. This is needed f.ex. when driving manual stick by ear, which requires you to be able to hear the engine RPM to time shifts. Rebalancing parts of the mixdown to their taste, like to favor AI cars instead of the Player, should be left to the player to decide, and this is accomplished via the in-game settings. Exclusive to Trackside and Free camera angles (Spectate / Replay game modes), there is additional mix automation that will more closely match the loudness between Player and AI vehicles.

TIP: All the volume values configured in the Actor-Mixer Hierarchy parent-child chain are summed, so it's best to try to steer clear of structures f.ex. where the child object is configured to +2 and it's parent to -2. If there is no clear use for this configuration, say, to pre-mix multiple child objects and adjust overall gain with the parent object, rather set both to 0.

Creating Soundbanks

Soundbank objects are containers for Wwise Authoring objects to selectively load only what's needed at runtime to both conserve memory and speed up loading times. In the context of a vehicle, a Soundbank contains all sounds belonging to a specific make or model of a vehicle, loaded only if the vehicle is actually included in a race. In order to create a Soundbank object for your audio mod, we first need to define what to include.

If you're only modifying objects under the example Player or AI Actor-Mixer Work Units, these are readily included in similarly titled Soundbanks.. But you still need to generate and copy the files over to the correct folder after making any changes! Any new objects added to the Work Unit of either Example Car get automatically added to appropriate soundbanks too, unless explicitly unchecked under the **'Edit'** tab of **SoundBank Editor** view.

To build and generate Soundbank(s) for your vehicle:

- Switch Wwise to **SoundBank** layout by pressing F7.
- If you wish to create an entirely new Soundbank, switch to the **'SoundBanks'** tab in Project Explorer. Select a Work Unit where you want to place the Soundbank, click the appropriate button in the Project Explorer toolbar and name your Soundbank. This will be used for the filename and which you also need to configure in BagEdit data. The new bank will now appear in the SoundBank Manager view above.
- If you created a new Soundbank, select it in Project Explorer, then switch over to the Audio tab, locate Actor-Mixer objects you wish to add, and drag & drop each of them to the **'Add'** tab of SoundBank Editor view. Any Event and Game Sync objects that the added Actor-Mixer objects reference will be automatically included.
- Ensure that the bank(s) you wish to generate for is selected, along with Windows platform and English language.
- Click **'Generate Selected'** and wait for the process to complete.

If you get an error for invalid audio event with your design in the game, then some object is likely missing from the Soundbank. Check the error for details and adjust the contents of your soundbank appropriately. If needed, the **'Edit'** tab allows you to exclude specific items, say, when adding an entire Work Unit to a Soundbank. Say, if you set up a folder for imported assets, there is no need to include it in the Soundbank.

For additional info about soundbanks, please read chapters [Generating a Soundbank](#) and [Building Soundbanks](#) of Wwise documentation. There's no need to copy over any other generated files than those named after the engine soundbank, as none of the others will be used!

Integrating Audio Mod To Wreckfest

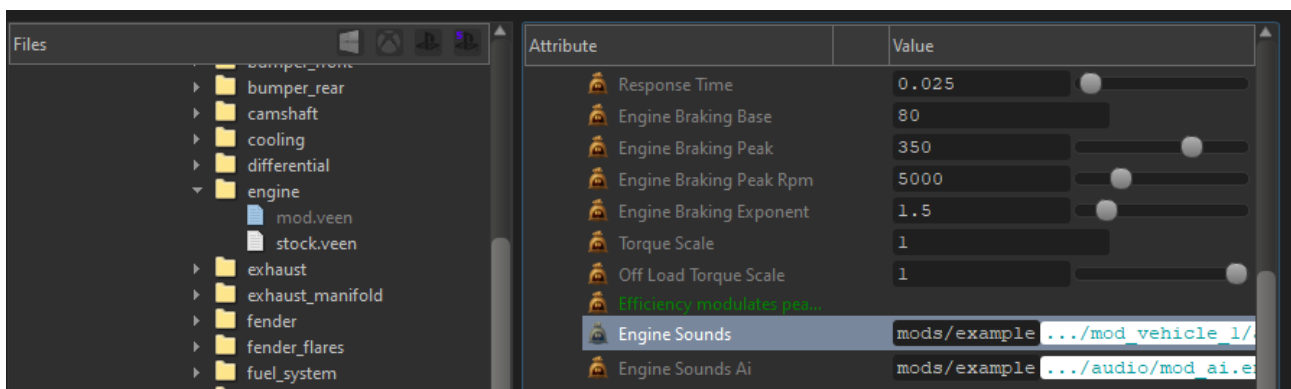
Last step in getting your audio mod to the game is to at least copy over the soundbanks. These are files with .bnk file extension, and you'll find them under the GeneratedSoundbanks subfolder of the Wwise project. These soundbank files need to go into the correct folder for your vehicle. Refer to the folder paths of Example Car below.

If you only modified the assets and Actor-Mixer objects of the Example Car, say, without adding any new trigger events or RPM RTPC, then you only need to copy SB_Player_Example.bnk (and SB_AI_Example.bnk for AI) to folder **../mods/example/data/vehicle/mod_vehicle_1/audio** under the Wreckfest installation folder and start the game. If you plan to iterate on your design, then creating a .bat script to automate the copying is highly recommended!

On the other hand, if you modified anything outside the Actor-Mixer tree, like changed soundbank name, added new events or RTPC, you then need to run BagEdit to adjust for changes. Otherwise Wreckfest will reference incorrect objects and your audio mod will not work. Here's how to adjust data for Example Car:

1. Start BagEdit and open **...mods/example/data/vehicle/mod_vehicle_1/audio/mod.ENGs** to adjust Player vehicle or **../mod_ai.ENGs** to adjust AI vehicle.
2. Adjust the settings to reflect correct objects in your audio mod soundbank, be it the soundbank file name, RTPCs or events
3. You could also create entirely new .ENGs and .ENGs files, you will also need to see that the engine part of the car references the correct objects. In the left pane, navigate to **...mods/example/data/vehicle/mod_vehicle_1/part/engine** and edit both **mod.VEEN** and **stock.VEEN** to reference correct .ENGs and .ENGs files (Picture 10).

Soundbank and modded car names can be anything but the folder structure has to match. For example **...mods/MYOWNMOD/data/vehicle/MYOWNCAR/...**



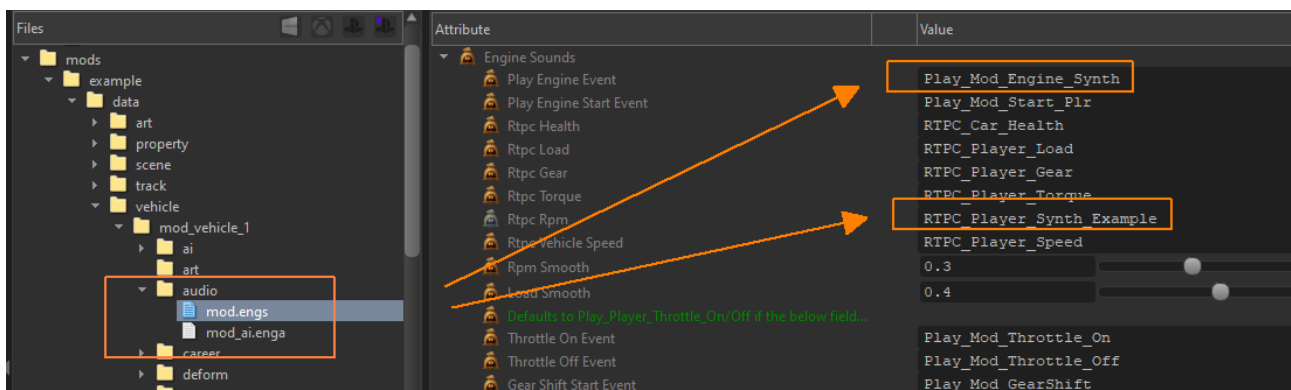
Picture 10: Editing .veen files with BagEdit

Synthetic Engine Example

The Player version of Example Car also includes a basic synthesized “engine” template, implemented using the Wwise Synth One source plugin. This is mostly a fun addition to explore sounds and to study basic sound synthesis with. If you’ve grown up with retro games like the Pole Position arcade, you should definitely enjoy playing around with the template.. But it can also serve as a starting point for some extra textures to go with other sounds: You could duplicate multiple instances of the object, or just dumb down the existing automation for a simple single-oscillator sine wave and to have a for a crazy sub bass go with your loop assets.

To play around with the parameters in Wwise Authoring, first press **Shift + X** to open the **Source Editor** window, then click on the **“Player Engine Synth Example”** under Actor-Mixer Hierarchy tree. Press **Spacebar** to play the sound, and, in the Source Editor view, navigate to the RTPC tab. Here you can see all the RTPC automation that the synth object is configured with. While the sound is playing, drag the **‘RTPC_Player_Synth_Example’** flag around to simulate RPM change.

By default, the Synth Engine object is also included with Example Car (in SB_Player_Example.bnk), and to try it in-game you only need to modify the audio event data in BagEdit. Start it up, locate **mod.enga** under **mod_vehicle_1\audio** directory and modify events according to Picture 11. Restart the game for changes to take effect.



Picture 11: Mod.enga changes for Synthetic Engine in BagEdit

To play both the loop and synthesized engine sounds instead, revert any changes in Bagedit and instead add the ‘Player Engine Synth Example’ object to trigger from event Play_Mod_Engine_Plr. Check for tuning between the loops you have and synth, adjusting the voice pitch (under General Settings tab) of the synth to match. When done, generate and copy over the appropriate .bnk from the Wwise project directory.

The synth object comes pre-configured only with a RPM RTPC to track pitch. What it lacks is variation to simulate on- and off-load states. As a fun challenge, see if you can add some flavor of this, maybe using low-/highpass filters and effects like distortion or EQ placed on the ‘Player Engine Synth Example’ object, then adding RTPC control to some parameters. Similar to loop-based designs, you can simulate the effects that each RTPC will have on a parameter in Wwise Authoring.

TIP: By default the pitch-tracking RTPC for this synth example is configured with ridiculously high RPM, just so that it will work with just about any BagEdit vehicle config you throw at it. You may want to duplicate the object and its RTPC to set a lower max RPM. This makes any fine automation curve tweaks a lot easier!

Tips and Tricks

Here's a small collection of miscellaneous suggestions for your engine designs.

- **Where to source engine sound recordings?** Try looking up some of the free annual GDC (Game Developer Conference) Soundpack bundles. These contain a huge variety of free-to-use sounds like vehicle recordings, all by professional recordists and audio designers.
- You could also try record engine sounds off of games, or any car you have access to. But getting both clean off- and on-load loops off a real car requires considerable effort.
- If you find it difficult to source recordings for both off- and on-load loops, one option is to create the latter by post-processing (or adding elements) to a set of good off-load loops. **Remember:** middle value range of Load RTPC is pretty much equal to that of revving the car on neutral gear, and this is both simple to record and hold a steady RPM on.
- Using Youtube for engine loops is **not recommended**. YT videos are commonly recorded using portable devices with integrated microphones (f.ex. mobile phones) and these are generally unable to handle the sound pressure of a loud car. Videos titled something like "RAW ENGINE SOUND!!1!1" can often only serve as a showcase to what a distorted microphone capsule sounds like.
- That said, don't let it stop or discourage you from experimenting! After all we're doing this for fun and to learn, and, for this purpose (and lack of anything better) one-shot sound recordings like exhaust backfire can still turn out reasonably ok with a bit of editing.
- Re-pitching loops way above their natural RPM often results in very abrasive and tiresome sounds to the ear, especially so with high rpm loops that contain a lot of intake noise (hiss)!
- Talking about intake noise - It changes in loudness and tone according to RPM, but not in pitch. If you include this type of sonic element to your design separately, you then don't need to use the 'Build Smart Pitch Curve' feature. If you still feel that the pitch needs to change, rather experiment with far more gentle automation curves or edit the 'Engine Synth Example' for a noise source, then process it with a mildly resonant band-boost EQ, automating this resonant frequency with RPM.
- If opting to synthesize loop assets using VST Instruments, then following Hertz-to-RPM conversion like **1 RPM = 1/60 Hertz (Hz)**, or **1000 RPM = ~16.67 Hz** will add a sprinkle of realism to your engine mod. Well, if that's what you're after anyway.. ;-)
- Look up one of the many RPM-to-Hertz / Hertz-to-RPM tools online for quick conversion reference.
- If looking to include a turbo or supercharger sound texture you can set up a new Blend Container, or additional Blend Tracks under the main engine Blend Container, to simulate these. For any new Blend Container(s), you will also need to modify the event (triggering the sound) to include a 'Play' event.
- For a turbocharger layer, try adding the blow-off valve (BOV) flutter as a separate Sound object and have the Player Throttle Off event trigger it. If you wish for the sound to only play over a certain RPM range, then add RTPC automation to control the volume.
- Under the 'Effects' tab of Actor-Mixer Property Editor, try adding effects plugins like EQ to address problematic frequencies in your engine recordings or emphasize / cut a specific frequency over certain RPM range.
- To dynamically alter your engine sounds, try controlling effects plugin parameters with RTPCs such as Player_Load, Player_RPM or Player_Throttle.

APPENDIX 1 - Updating A Existing Audio Mod

If you've previously done audio mods for Wreckfest, you need to regenerate soundbanks for these mods when the version of Wwise SDK used in the game changes. A SDK upgrade most often occurs if a critical bug gets fixed, or if support for a new hardware platform is added.

While a newer version of Wwise Authoring can import project files made with any older version, doing so is not recommended because there may have been some changes (like new features added, yay?) to audio routing or processing settings on the Master-Mixer level. To fully make use of these changes and upgrade existing soundbanks to work with the game, the only option is to import appropriate objects to the latest version of The Project version, adjust design(s) and regenerate soundbanks.

To accomplish this, you can either 1) follow import procedures described in [Using Templates in Wwise](#), or 2) move them between alternate versions of The Project inside temporary Work Units. Here's how to go about the latter:

- Load up your existing installation of Wwise and the old project you have.
- Create a temporary Work Unit for each object type you want to move. These are most often the Actor-Mixer type, but if you've made any custom objects, they can also include at least Events and Game Syncs (RTPC). Place copies of these object types into appropriate Work Units. There's no need to copy over any default RTPC and Event objects that ship with the template.
- Save changes and exit Wwise Authoring.
- If you haven't yet extracted a copy of "**wreckfest_vehicle_audio_tools_wwise.zip**", do it now.
- Navigate to the folder of your old project and copy all the temporary Work Units (.wwu) you made in the previous step, into appropriate directories in the new version you just extracted.
- From your old project, also copy any custom audio assets you have under the folder '**\Originals\Sfx**' to the matching directory under the new template project.
- Launch the new Wwise install (matching the version printed on this document) and load up the new project. You will see a warning about upgrading the version for the temporary Work Units. If warnings show a critical error (like something missing) then make note of any, re-open the old project to make (and save) the appropriate changes and again copy updated .wwu (Work Unit) on top of previous ones.
- After you get past the warnings, you should now find the newly imported old Work units in the object hierarchies. You can continue to use these work units, but please compare object settings against new templates and adjust as needed. Commonly you may need to adjust at least the Output Bus and Conversion preset found under the Property Editor tabs. What objects you need to compare and adjust depends on any Wwise object IDs that may have changed between template versions. These should be mentioned with the warnings about upgrading Work Units.
- Continue back from chapter "**Calibrating Overall Volume**".

APPENDIX 2 - Making Your Own Loop Assets

Providing a step-by-step tutorial on how to create engine audio assets is far too big a topic for this tutorial, but here are some pointers to help you get started. First, any custom asset you make needs to..

1. have a constant pitch (= steady RPM),
2. have a known 'natural' RPM value (= eg. at which RPM the asset was recorded), and
3. loop seamlessly.

And add to that, you will need to have two sets of loops; one for on-load and other for off-load. These could be further split even to separate layers for engine and exhaust, then have their mix balance adjusted appropriately for the selected camera angle (via state automation).

TIP: Similar to Example Car, the *majority of vehicles in Wreckfest have a pre-mixed (baked) set of loops based on the Chase camera angle. Loop assets made for this camera angle often have around 60-40% to 70-30% mix of exhaust-to-engine. Looking at the car from this angle, the exhaust is commonly both physically 'closer' to and aimed towards the camera, whereas the engine is obstructed by the chassis.*

Preparing assets from recordings where a vehicle is recorded while being driven, you will often need to do some amount of pitch 'flattening', so as to hold a steady RPM (and thus constant pitch) over a set amount of time. Spectral visualizers, like the '**Spectrograph Spectrogram Meter**' JS plugin included with [Cockos Reaper](#), come in handy to graphically present some of this pitch change. This type of tool can also help you estimate for correct RPM, as fundamental and harmonic frequencies are clearly visualized.

As for the amount of loops, one good rule of thumb is to have natural RPMs at about 1000 RPM apart. But you might need even more depending on how lively your vehicle sounds are, to capture nuances over a specific RPM range. Try to steer clear of having just a few loops for the entire RPM range; this will add very audible imperfections to the sound, because the loops are very likely played far beyond their natural pitch.

For the RPM range, you should have enough loops to cover **+200 RPM above desired redline**. You do not need to record at redline though; the loop near top RPM can be usually pushed a bit higher than loops on the lower RPM range before starting to sound bad. Say, for our default Example Car, the natural RPM for top on-load asset is 4180 RPM whereas its max RPM is at 5500. Thus the engine redline configured in BagEdit can be at most 5300 RPM (5500-200). The +200 RPM is mostly needed when you drive manual and shift down to start engine braking at redline.

TIP: *It's recommended to always include natural RPM and load state with the filename for quick reference later on. Take a look at the example audio assets and you'll notice they are named in similar fashion, f.ex. 'PP_Demo_Car_Off_Load_3520'. This means that the asset is for engine off-load at 3520 RPM.*

APPENDIX 3 - Importing New Assets

When you have a set of loops ready for import, drag & drop them (or the whole folder) over a Work Unit (or any Container object) in the Actor-Mixer Hierarchy to import. Please refer to Wwise documentation about [The Importing Process](#) and [Importing Media Files](#).

It is **not recommended** to copy files directly to anywhere under “**..Originals\SFX**” folder, because Wwise Authoring may lose track of unique IDs that it assigns to each object. IDs are needed to maintain reference within the project, like when making copies of Sound objects. Any copies will reference the same loop assets and we don’t want to break this connection.

With the assets imported, continue from chapter “**Initial Sound Object Configuration**” on Page 8.

TIP: *If your design calls for a higher max RPM than that of Example Car, it’s best to create a new RTPC for it before adding any of the RTPC automation described in chapter “Initial Sound Object Configuration”. Increasing the RTPC max value later on comes with some risk of messing up automation you have defined previously, or cause extra work in adjusting the automation to match new max RPM. So it’s best to have it defined from the get-go before anything else. You can find RTPC objects under the ‘Game Syncs’ tab of Project Explorer.*



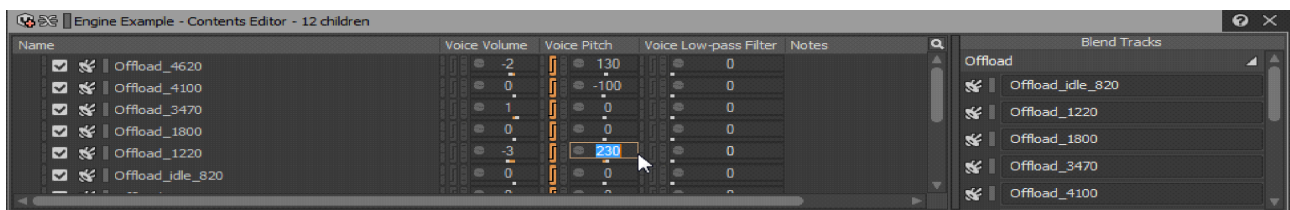
APPENDIX 4 - Fine-Tuning Sound Objects for Pitch

As you monitor and configure custom assets you've imported, you may sometimes find that adjacent objects in a Blend Track are slightly out of tune against one another with just the plain root pitch tracking configured.

Pitch issues commonly originate with Hertz-to-RPM conversion, where RPM is coarsely estimated from fundamental and harmonic frequencies present in the loops. In Wwise these tuning issues (between adjacent objects) should be addressed via tweaks in the Contents Editor of a Blend Container. RTPC automation curves of Blend Tracks should be only used to control the overall characteristics of the engine design, **not** to kludge pitch or volume issues between child objects.

The process to monitor and correct for issues is similar to how we previously went about adjusting the crossfade widths: Audition and scrub through both on- and off-load layers separately (by setting RTPC_Player_Load flag), and work your way down from max to idle RPM. It is always recommended to start from max RPM because the engine note / tone (fundamental harmonic frequencies) are the most clear and thus easiest to correct for.

While auditioning the Blend Container, you can adjust Voice Pitch properties by typing in values to the appropriate box under Contents Editor (Picture 12). For pitch the value is expressed in Cents, where 100 cents equals one semitone on the chromatic scale, 12 distinct pitches in an octave. If you type in a value of 1200, the object will play one octave above its natural pitch (or RPM).



Picture 12: Adjusting child object pitch & volume using Contents Editor

Throughout the tuning process, it's a good practise to flip between full off- and on-load every now and then, checking that the pitch tracking between the two Blend Tracks stays consistent. Sometimes the volume of harmonic tones can change over RPM so that the fundamental frequency becomes overpowered by the harmonic, in turn possibly contributing to misconfiguration of root pitch or fine-tuning. If looking to learn more about harmonics, start by this Wikipedia page on [Fundamental Frequency](#).

For tutorial purposes, to get a feel for how a fine pitch mismatch can sound like, try modifying Blend Container of Example Car by altering Voice Pitch parameters of adjacent objects on either Blend Track and monitor for the change as you audition / scrub over the crossfade. Sometimes even a change less than 10 cents can cause fine but very audible issues between any two objects.

For further reference please read chapters [Managing Crossfades](#) and [Working with Blend Tracks](#) of Wwise documentation.

APPENDIX 5 - List of Supported Plugins

Here are alphabetical lists of Effect and Source plugins which you can use for vehicle audio designs. Click the plugin name for more info. Note that these links point to the latest version (“Edge”) of Wwise SDK and not the version mentioned in this document.

Wwise Effects Plugins

Effect plugins are used to process the output of any Actor-Mixer object. Add plugins under the ‘Effects’ tab of Property Editor by clicking icon ‘>>’.

- [Compressor](#)
- [Delay](#)
- [Expander](#)
- [Flanger](#)
- [Gain](#)
- [Guitar Distortion](#)
- [Matrix Reverb](#)
- [Meter](#)
- [Parametric EQ](#)
- [Peak Limiter](#)
- [Pitch Shifter](#)
- [RoomVerb](#)
- [Time Stretch](#)
- [Tremolo](#)

Wwise Source Plugins

Source plugins output synthesized sound that is generated at run-time. Right-click on any Work Unit and select ‘**New Child > (Source plugin name)**’ from the context menus.

- [Silence](#) - Generates silence of specified length.
- [Sine](#) - Sine wave synthesizer for one-shot sounds, with minimal synthesis features.
- [Synth One](#) - Synthesizer for looping sounds, with traditional sound synthesis features.
- [Tone Generator](#) - Synthesizer for one-shot sounds, with basic synthesis features.