AUDIO

View online: https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/audio

The **Audio** object plays back audio files that have been imported to the project. The Audio object must be added to each project that needs to use audio playback. Audio files can be imported to a project by right-clicking the **Sounds** or **Music** folders in the Project Bar and selecting Import sounds or Import Music, which brings up the Import Audio dialog. See the relevant manual sections for more information on the steps involved to import audio files.

Construct comes with several examples of the Audio plugin; search for *Audio* in the Example Browser to find them.

Scripting

When using JavaScript or TypeScript coding, this object can be accessed via the IAudioObjectType script interface. This mainly provides access to the underlying Web Audio API context.

Categorise audio files correctly

It is important to organise audio files appropriately, because audio files in the **Sounds** folder are downloaded completely before playing, but files in the **Music** folder are streamed. This means if a Music track is accidentally put in the Sounds folder, it would have to download completely before it started playing, which could take a while. However, audio in the Music folder can start playing immediately since it is streamed from the server.

Preloading sounds

The **Preload sounds** project property determines whether sounds (excluding music) are loaded while the loading bar is showing. Preloading sounds means there is more to load before the project can start, and memory use is higher due to having all sounds loaded, but all sound effects can play immediately once it starts. If sounds are not preloaded, the project starts sooner since it did not need to load any sounds, but sounds will be loaded on-demand during the project. In other words, nothing is loaded until a *Play* action starts playing an audio file. Then it starts loading and will play when ready. This also helps minimise memory usage since unused audio is never loaded. However, it can introduce a delay before audio plays for the first time. The delay on first play is a one-off, because after the first play the sound is already loaded and can be played immediately if played a second time.

If *Preload sounds* is disabled, the *Preload* action can be used to start loading an audio file without actually playing it. This can be done on *Start of layout* to start downloading a few important sound effects so there is no delay when they are played for the first time. The *Unload* actions can also be used to remove a loaded sound from memory. This allows you to manually

manage which sounds are loaded in to memory, which is important if you have a large library of sound effects which would use a lot of memory if they were all preloaded.

Music is never preloaded, since music tracks often involve a large download size, and it is not usually important to have music play with as little latency as possible. Music will still stream while playing, but if the latency is important, the *Preload* action can be used to load it in advance of playing.

Audio tags

Some actions affect audio parameters such as the volume for sounds which are already playing. However there can often be many sounds playing at once in a project. In order to identify which sounds you want to affect, sounds are played with an associated *tag*. This is any string that identifies the sound. For example, the player's weapon sound effect could be played with the tag *"PlayerWeapon"* and an enemy's weapon with the tag *"EnemyWeapon"*. Then, the tag can be used in the *Set Volume* action to specify which sound to set the volume for. Tags are case insensitive.

Multiple sounds can also play at once using the same tag. In this case actions like *Set Volume* affect all the sounds playing with that tag.

A tag which is an empty string ("") has a special meaning: it refers only to the last sound played with one of the *Play* actions. This is convenient for playing a sound and immediately setting its volume and other parameters, without having to assign it a unique tag.

Multiple tags

Providing the Audio object property *Enable multiple tags* is enabled (it is on by default), then the tag string can include multiple space-separated tags, such as "player weapon" and "enemy weapon". Then when changing sounds such as to set the volume, all sounds with all the provided tags will be updated. For example setting the volume for tag "weapon" will update sounds played with tags "player weapon" and "enemy weapon", and setting the volume for tags "player", "player weapon" or "weapon player" will update sounds played with the tags "player weapon".

There is one exception to using multiple tags, which is when using effects. Adding effects builds up an effect chain which audio playback can be routed through. However a sound can only be routed through one effect chain. Therefore when adding effects with multiple tags, the effect will be added to multiple effect chains - once for each provided tag. Then when playing sounds with multiple tags, the first tag determines which effect chain the sound is routed to.

If *Enable multiple tags* is disabled, then a tag with spaces is still treated as a single tag. This only exists for backwards compatibility with projects made before the multiple tags feature was introduced.

Autoplay restrictions

Most modern browsers have a limitation in starting audio playback. To avoid annoying users generally browsing the web, audio playback cannot start until the user interacts with the page, such as touching or clicking it. This is a limitation in the browsers themselves and cannot be

worked around. As a result, if you play audio on the start of layout, you may find it does not actually start until the first user interaction.

Usually you do not need to handle this in your events. If audio cannot be played for any reason, the Audio object will automatically queue it up for playback at the soonest opportunity (usually when the user next clicks or touches). However you should be aware of this when designing your project. For example if the first touch changes layout or stops the music, then the music may never be heard. You may want to start playback then encourage the user to touch the screen with a 'Play' icon or something similar.

The limitation is specific to web browsers. It does not apply if you publish a mobile app. Further, browsers may lift the limitation in some circumstances, such as if you use the *Install* or *Add to home screen* options, or if they identify over time that your web page is regularly used for audio playback that the user wants.

Audio properties

Timescale audio

The project timescale can be used to speed up or slow down playback of the project, for effects like slow-motion. See Delta-time and framerate independence for more information. This property controls whether or not audio is affected by the project's timescale.

- **Off** will play back audio the same regardless of the timescale.
- **On (sounds only)** will play back audio from the Sounds project folder at a different rate depending on the timescale, but will always play back audio from the Music project folder at the same rate.
- **On (sounds and music)** will play back all audio at a different rate depending on the timescale.
- Some browsers may not support audio timescaling at all; test on multiple browsers to establish support.

Save/load

When using savegames, what audio state should be saved and restored.

- **All** will save and restore all audio, so that music and sound effects rewind to the same point they were saved at when loading.
- **Sounds only** will only restore the sound effects playing at the time of the save, and allow music to keep playing through unaffected when loading a game.

- **Music only** conversely only restores the music playing at the time of the save, and allows sound effects to keep playing through unaffected when loading a game.

- **None** does not save or load any audio state at all. Audio will be completely unaffected when loading a game, and any playing music and sound effects will continue to play out to their end.

Play in background

If disabled, then switching browser tab, minimising the browser window, switching to a different mobile app, or otherwise hiding the window will pause all audio and resume it when switching back. This is intended to avoid annoying the user with continued music playback when deciding to do something else, and also helps save battery on mobile. However for some types of app such as music players it may be desirable to keep music playing in these cases, in which case enabling the property allows continued audio playback even when in the background.

Latency hint

Provide a hint to the audio engine about the preferred latency vs. power usage tradeoff. Typically interactive content like games will prefer a low latency mode, but other uses like music playback where latency is not important may prefer to use a more battery-efficient mode. The options are:

- **Interactive** (default): provides the lowest playback latency, but uses more battery power.

- **Balanced**: a middle-ground providing medium playback latency with medium power use.

- **Playback**: provides the highest playback latency, with the lowest power use, suitable for purposes like music playback where the latency is not important.

Enable multiple tags

If checked, multiple tags can be specified for sounds by separating them with spaces, e.g. "player weapon" would count as two separate tags "player" and "weapon". If disabled then sounds can only have one tag, i.e. "player weapon" would be counted as a single tag name including the space. This setting is enabled by default and only exists for backwards-compatibility with projects made before the introduction of the multiple tags feature. For more information see the section *Multiple tags* above.

Panning model

How positioned sounds are panned. *HRTF* uses a realistic model of human hearing, whereas *equal power* is a simple method that preserves the overall power in a stereo channel.

Distance model

The formula to determine volume reduction of positioned sounds relative to the distance to the listener. The options are:

- **Linear**, using the equation *1 - rolloffFactor * (distance - refDistance) / (maxDistance - refDistance)*
- **Inverse**, using the equation *refDistance / (refDistance + rolloffFactor * (distance - refDistance))*
- **Exponential**, using the equation *pow(distance / refDistance, -rolloffFactor)*

Listener Z height

The height of the listener above the layout, in layout pixels, used to determine relative volume and panning of positioned sounds. A low Z height will have intense changes over small distances, whereas a high Z height will have smaller changes over larger distances.

Reference distance

The distance at which the volume of positioned sounds begins to reduce. For best results this should be at least as much as the *Listener Z height*.

Maximum distance

The maximum distance in pixels beyond which positioned sounds no longer reduce their volume.

Roll-off factor

How quickly the volume reduces as positioned sounds move away from the listener. A high roll-off factor means sounds get quieter quickly, whereas a low roll-off factor means sounds will not lose much volume.

Audio conditions

Is any playing

True if any audio is currently playing.

Is silent

True if the object has been set in to silent mode using the *Set silent* action.

Is tag playing

Is tag playing

True if any audio with a given tag is currently playing.

On ended

Triggered when a sound with a given tag finishes playing. This is not triggered for looping sounds.

On fade ended

Triggered when a fade started by the *Fade volume* action finishes, for a given tag.

Preloads complete

True when all audio preloaded with one of the preload actions has finished loading.

Audio general actions

Play

Play (by name)

Start playing an audio file with a given tag. The latter action gives you the opportunity to use an expression for the audio file name. The sound can optionally be set to looping when it starts playing. A volume can also be set, given in decibels (dB). A volume of 0 dB is original volume, and below 0 dB attenuates the sound. Note amplification is not supported. For example, entering a value of -10 plays the audio back 10 dB quieter (about half as loud). A stereo pan can also be provided, ranging from -100 (fully left) to 100 (fully right), with the default 0 being middle. Prefer setting the intended volume and pan in the *Play* action; even if followed immediately by a *Set volume* or *Set stereo pan* action, some platforms will momentarily play the audio at the volume and pan given in the *Play* action.

Preload

Preload (by name)

Start loading an audio file so it has no delay before playing. See the section *Preloading sounds* above for more information. Audio does not have to be preloaded before playing - it is optional and only serves to possibly reduce the delay before audio plays for the first time. Once all audio preloaded with this action finishes loading, the *On preloads complete* trigger fires. Note if the project *Preload sounds* property is enabled, there is no point preloading any sounds, since they will always be loaded before the project starts - in this case it only makes sense to preload music.

Seek to

Seek a currently playing sound to a different location in the audio file. The time to seek to is given in seconds.

Set looping

Set a sound either looping (repeating when it finishes) or not looping (stopping when it finishes).

Set master volume

Set the overall volume that is applied to all audio playback.

Set muted

Set a sound either muted (silent) or unmuted (audible).

Set paused

Pause or resume some audio by its tag.

Set playback rate

Change the rate a sound plays back at. If the *Timescale audio* property is used, it combines with the playback rate set by this action.

Set silent

Enable, disable or toggle *Silent mode*. In silent mode all currently playing sounds are muted and no new sounds will play. This is useful for quickly creating an audio toggle on a title screen.

Set volume

Change the volume of a sound. The volume is given in decibels (dB). A volume of 0 dB is original volume, and below 0 dB attenuates the sound. Note amplification is not supported. For example, entering a value of -10 plays the audio back 10 dB quieter (about half as loud). Note it is best to set the initial volume in the *Play* action instead of setting it with this action immediately after playing, since that can sometimes cause a moment of playback at the wrong volume.

Fade volume

Change the volume of a sound over time. This is typically used for fade-in and fade-out effects. The sound will fade from its current volume to the given level in decibels (dB), over the given time period in seconds. When the fade finishes, the sound can either automatically be stopped, or keep playing. For a fade-in, typically the sound will be initially played at a low volume (e.g. -100 dB) and faded in to a high volume (e.g. 0 dB), and keep playing when the fade finishes. For a fade-out, typically the sound will be playing at a high volume (e.g. 0 dB) and faded to a low volume (e.g. -100 dB), and then stopped when the fade finishes, so there

isn't an inaudible sound still playing. Note if *Set volume* is used during a fade, the fade is cancelled and the sound will be left at the volume specified by *Set volume*.

Stop

Stop a sound playing immediately.

Stop all

Stop all currently playing sounds.

Audio effects actions

A selection of well-known audio effects can be added using the *Add effect* actions. Each tag has its own effect chain, and multiple effects can be added to a tag. All audio played with the given tag is then processed by the effect chain. This can be used to create environmental effects and other creative audio features. Audio signal processing is a complex topic and somewhat out of the scope of this manual, so it will not be detailed exhaustively here. Anyone with light experience in audio recording or production should already be familiar with all the effects available. For interactive examples, search for *Audio* in the Example Browser. A brief summary of each effect is provided below.

- **Analyser:** doesn't change the audio, but can report back frequency domain data
- **Compressor:** automatically boost or reduce volume to even out the overall volume level
- **Convolution:** an advanced effect using another sound as an impulse response to process the audio. This allows for real-world locations to be recorded and the environmental reverb applied
- **Delay:** a feedback loop with a delay, making a sort of simple echo effect.
- **Distortion:** a guitar-amplifier style signal distortion
- **Filter:** boost or reduce certain frequencies, such as a low-pass filter (which cuts out high frequencies). Useful for simple atmospherics, treble/bass adjustment, etc.
- **Flanger:** delays the sound by a few milliseconds then mixes it back in with itself. By oscillating the delay time a sweeping effect is created
- **Gain:** a volume control, which might be useful in longer effect chains. The **Mute** effect is also simply a zero gain effect, which can be useful to add after analysers (so the audio is analysed, but not heard).
- **Phaser:** phase-shifts the sound then mixes it back in with itself. By oscillating the phase shift another sweeping effect is created
- **Stereo pan:** a stereo pan control, allowing an entire group of sounds to be panned left or right together.
- **Tremolo:** automatically oscillates the volume up and down, also known as amplitude modulation. Some interesting amplitude modulation effects can be created by moving the

modulation frequency in to the audible range (above 20 Hz).

- **Ring modulator:** like tremolo, but oscillates all the way through to a full phase inversion

The **Remove all effects** action clears a tag's effects chain, allowing you to add a different selection of effects. The **Set effect parameter** action also allows effect parameters to by dynamically set or faded during playback. Each effect also has a wet/dry mix which can be used to fade in and fade out effects.

Audio memory actions

Unload all audio

Release all loaded audio files from memory. Any subsequently played audio will need to be loaded again.

Unload audio

Unload audio (by name)

Release a specific loaded audio file from memory. If the audio file is not loaded, this has no effect. Any subsequent playback of the audio file will need to load it again. This allows manual control of which audio files are in memory.

Audio panning/positioning actions

Play at object

Play at object (by name)

As per the ordinary *Play* actions, but the sound is positioned at an object. If the object moves (including changing Z elevation) or rotates during playback, the sound follows with it. A cone can be specified to create directional sounds, which follows the object's angle.

Play at position

Play at position (by name)

As per the *Play at object* action, but the sound does not move. It is just played at a fixed position and angle in the layout. A Z co-ordinate can also be specified to play a sound at a given Z elevation, or for full 3D audio support when used with *Set listener orientation*.

Set listener object

Set the object that positioned sounds are calculated relative to. Typically this is set to the object representing the player on *Start of Layout*.

Set listener orientation

By default the listener is oriented for a 2D game. However when using 3D Camera to move the view in 3D, it is also useful to change the listener orientation to match that of the camera for positioned sounds to play back correctly in 3D space. The listener orientation is specified using two vectors: a forwards vector (the direction the listener is facing in), and an up vector (which determines the orientation along the forwards vector). These can usually be set to the *LookVectorX/Y/Z* and *UpX/Y/Z* expressions of the 3D Camera object.

Set listener Z

Set the *Listener Z* property of the audio object, which affects the calculation of positioned sounds.

Set stereo pan

Change the stereo pan of a sound. The pan is a number ranging from -100 (fully left) to 100 (fully right), with 0 being middle. Note it is best to set the initial pan in the *Play* action instead of setting it with this action immediately after playing, since that can sometimes cause a moment of playback with the wrong pan.

> *Setting the stereo pan of a positioned sound will turn off positioning in order to apply the stereo pan.*

Audio scheduling actions

Schedule next play

This action causes the next *Play* action (all variants) to be delayed until the specified time. The delayed playback is sample-accurate. Typically events are only run every 16ms making it difficult to schedule sounds more accurately than that, but this action allows for perfectly scheduled playback, even in between ticks. If the specified time is in the past, it will play immediately. The time given must be relative to the audio hardware clock, which is returned by the *CurrentTime* expression, so typically sounds will be scheduled a short time ahead using an expression of the form *Audio.CurrentTime + N*. See the *Audio scheduling* example in the Start Page for a demonstration.

> *Note that audio scheduling works best if you set the Use worker project property to No. This is because worker mode does not have direct access to the audio clock, which adds a small amount of latency and variance to playback which may be noticeable when scheduling.*

Audio web actions

Add remote URL

Play audio from a URL by adding it first with this action. Specify the URL to be played, the type (aka the MIME type) of the audio to be played, and pick a name to use for this URL. On its own this action does nothing - no request will be made to the given URL. However you can then pass your chosen name to the other actions that work "by name", such as *Play (by name)* or *Preload (by name)*, it will then play from the URL you associated with that name. If these actions ask for the folder, it is not really used, but chooses whether the audio is fully downloaded and decoded before playback (for *Sounds*), or streamed (for *Music*). As with any other audio file, you can use the *Preload (by name)* action to preload the audio at the given URL, to ensure subsequent playback starts promptly.

> *WebM Opus is the only audio format that works across all browsers and platforms. To ensure your audio works everywhere, make sure the URL serves WebM Opus audio. Otherwise whether or not audio playback works will depend on the audio codec support of the current browser/platform.*

Audio expressions

AnalyserFreqBinAt(Tag, Index, Bin)

Get the magnitude of energy in an analyser's frequency bin. AA analyser effect must already be added to a tag. *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on). *Bin* is the frequency bin number to retrieve from, up to *AnalyserFreqBinCount*.

AnalyserFreqBinCount(Tag, Index)

Get the number of frequency bins returned by an analyser. An analyser effect must already be added to a tag. *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on).

AnalyserPeakLevel(Tag, Index)

Get the peak level of audio in the last FFT window from an analyser. *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on). The value is returned in dBFS (0 dB for peak level, and negative values for lower). If you intend to use this value it is recommended to use an FFT size of 1024, because at a system sample rate of 44.1 KHz the value will update about 43 times a second. Projects usually run at 60 FPS, and smaller FFT sizes may cause FFT windows to be missed since they change faster than the framerate.

AnalyserRMSLevel(Tag, Index)

Get the RMS level of audio in the last FFT window from an analyser (the square root of the average of the squared sample values). *Index* must be the index of the effect (for example, 0 if the analyser is the first added effect for that tag, 1 if the second added effect, and so on).

The value is returned in dBFS (0 dB for peak level, and negative values for lower). If you intend to use this value it is recommended to use an FFT size of 1024, because at a system sample rate of 44.1 KHz the value will update about 43 times a second. Projects usually run at 60 FPS, and smaller FFT sizes may cause FFT windows to be missed since they change faster than the framerate.

EffectCount(Tag)

Get the number of effects in the effect chain for a tag.

CurrentTime

Get the audio clock time in seconds. Where supported, this is returned from the audio hardware, providing the correct time against which to schedule audio playback. It is important to use this value to calculate playback times in the *Schedule next play* action.

Duration(Tag)

Get the duration in seconds of an audio sample with a tag.

> *This expression will only return the correct value once the audio has finished loading and decoding. If you need to access the value immediately after starting playback, consider first preloading the sound.*

NormalizedVolume(Volume, dbThreshold)

This is a helper expression to convert a volume expressed in the range 0-100 (such as with a slider bar in a user interface) to a dB value suitable for actions such as *Set volume*. The *dbThreshold* parameter is the dB that will be set for a volume value of 10 (e.g. -40). The returned dB volume will start at 0 for volume value 100 and linearly drop to *dBThreshold* at volume value 10. Below that, the last 0-10 range is a linear dropoff in linear range, i.e. the dB level will drop off much more quickly until it reaches -Infinity at volume value 0 for silence.

MasterVolume

Return the current master volume set using the *Set master volume* action.

OutputLatency

The time in seconds reported by the system as the estimated output latency, i.e. the time between the system sending an audio buffer to play, and the time at which the first sample in the buffer is actually processed by the audio output device. This can vary between devices and can also change over time. If the value is unavailable or unknown, it will return 0.

> *The output latency will remain unknown until playback actually starts if autoplay*

restrictions are in place.

PlaybackRate(Tag)

Get the current playback rate of a sound with a tag. The default playback rate is 1, and is set with the *Set playback rate* action.

PlaybackTime(Tag)

Get the current playback time in seconds of a sound with a tag. This starts at 0 and counts up to the duration, except for looping sounds which keep counting up past the duration.

SampleRate

Return the audio output sample rate in Hz, typically 44100 or 48000.

Volume(Tag)

Get the volume set for a sound with a tag.