pygame.mixer

pygame module for loading and playing sounds

- pygame.mixer.init — initialize the mixer module
- pygame.mixer.pre_init — preset the mixer init arguments
- pygame.mixer.quit — uninitialize the mixer
- pygame.mixer.get_init— test if the mixer is initialized
- pygame.mixer.stop — stop playback of all sound channels
- pygame.mixer.pause — temporarily stop playback of all sound channels
- pygame.mixer.unpause — resume paused playback of sound channels
- pygame.mixer.fadeout— fade out the volume on all sounds before stopping
- pygame.mixer.set_num_channels — set the total number of playback channels
- pygame.mixer.get_num_channels — get the total number of playback channels
- pygame.mixer.set_reserved— reserve channels from being automatically used
- pygame.mixer.find_channel — find an unused channel
- pygame.mixer.get_busy — test if any sound is being mixed
- pygame.mixer.Sound — Create a new Sound object from a file or buffer object
- pygame.mixer.Channel — Create a Channel object for controlling playback

This module contains classes for loading Sound objects and controlling playback. The mixer module is optional and depends on SDL_mixer. Your program should test that pygame.mixerpygame module for loading and playing sounds is available and intialized before using it.

The mixer module has a limited number of channels for playback of sounds. Usually programs tell pygame to start playing audio and it selects an available channel automatically. The default is 8 simultaneous channels, but complex programs can get more precise control over the number of channels and their use.

All sound playback is mixed in background threads. When you begin to play a Sound object, it will return immediately while the sound continues to play. A single Sound object can also be actively played back multiple times.

The mixer also has a special streaming channel. This is for music playback and is accessed through the pygame.mixer.musicpygame module for controlling streamed audio module.

The mixer module must be initialized like other pygame modules, but it has some extra conditions. The pygame.mixer.init() function takes several optional arguments to control the playback rate and sample size. Pygame will default to reasonable values, but pygame cannot perform Sound resampling, so the mixer should be initialized to match the values of your audio resources.

NOTE: Not to get less laggy sound, use a smaller buffer size. The default is set to reduce the chance of scratchy sounds on some computers. You can change the default buffer by calling pygame.mixer.pre_init()preset the mixer init arguments before pygame.mixer.init()initialize the mixer module or pygame.init()initialize all imported pygame modules is called. For example: pygame.mixer.pre_init(44100,-16,2, 1024) The default size was changed from 1024 to 3072 in pygame 1.8.

pygame.mixer.init()

initialize the mixer module

init(frequency=22050, size=-16, channels=2, buffer=4096) -> None

Initialize the mixer module for Sound loading and playback. The default arguments can be overridden to provide specific audio mixing. Keyword arguments are accepted. For backward compatibility where an argument is set zero the default value is used (possible changed by a pre_init call).

The size argument represents how many bits are used for each audio sample. If the value is negative then signed sample values will be used. Positive values mean unsigned audio samples will be used. An invalid value raises an exception.

The channels argument is used to specify whether to use mono or stereo. 1 for mono and 2 for stereo. No other values are supported (negative values are treated as 1, values greater than 2 as 2).

The buffer argument controls the number of internal samples used in the sound mixer. The default value should work for most cases. It can be lowered to reduce latency, but sound dropout may occur. It can be raised to larger values to ensure playback never skips, but it will impose latency on sound playback. The buffer size must be a power of two (if not it is rounded up to the next nearest power of 2).

Some platforms require the pygame.mixerpygame module for loading and playing sounds module to be initialized after the display modules have initialized. The top level pygame.init() takes care of this automatically, but cannot pass any arguments to the mixer init. To solve this, mixer has a function pygame.mixer.pre_init() to set the proper defaults before the toplevel init is used.

It is safe to call this more than once, but after the mixer is initialized you cannot change the playback arguments without first calling pygame.mixer.quit().

pygame.mixer.pre_init()

preset the mixer init arguments

pre_init(frequency=22050, size=-16, channels=2, buffersize=4096) -> None

Call pre_init to change the defaults used when the real pygame.mixer.init() is called. Keyword arguments are accepted. The best way to set custom mixer playback values is to call pygame.mixer.pre_init() before calling the top level pygame.init(). For backward compatibility argument values of zero is replaced with the startup defaults.

pygame.mixer.quit()

uninitialize the mixer

quit() -> None

This will uninitialize pygame.mixerpygame module for loading and playing sounds. All playback will stop and any loaded Sound objects may not be compatible with the mixer if it is reinitialized later.

pygame.mixer.get_init()

test if the mixer is initialized

get_init() -> (frequency, format, channels)

If the mixer is initialized, this returns the playback arguments it is using. If the mixer has not been initialized this returns None

pygame.mixer.stop()

stop playback of all sound channels

stop() -> None

This will stop all playback of all active mixer channels.

pygame.mixer.pause()

temporarily stop playback of all sound channels

pause() -> None

This will temporarily stop all playback on the active mixer channels. The playback can later be resumed with pygame.mixer.unpause()

pygame.mixer.unpause()

resume paused playback of sound channels

unpause() -> None

This will resume all active sound channels after they have been paused.

pygame.mixer.fadeout()

fade out the volume on all sounds before stopping

fadeout(time) -> None

This will fade out the volume on all active channels over the time argument in milliseconds. After the sound is muted the playback will stop.

pygame.mixer.set_num_channels()

set the total number of playback channels

set_num_channels(count) -> None

Sets the number of available channels for the mixer. The default value is 8. The value can be increased or decreased. If the value is decreased, sounds playing on the truncated channels are stopped.

pygame.mixer.get_num_channels()

get the total number of playback channels

get_num_channels() -> count

Returns the number of currently active playback channels.

pygame.mixer.set_reserved()

reserve channels from being automatically used

set_reserved(count) -> None

The mixer can reserve any number of channels that will not be automatically selected for playback by Sounds. If sounds are currently playing on the reserved channels they will not be stopped.

This allows the application to reserve a specific number of channels for important sounds that must not be dropped or have a guaranteed channel to play on.

pygame.mixer.find_channel()

find an unused channel

find_channel(force=False) -> Channel

This will find and return an inactive Channel object. If there are no inactive Channels this function will return None. If there are no inactive channels and the force argument is True, this will find the Channel with the longest running Sound and return it.

If the mixer has reserved channels from pygame.mixer.set_reserved() then those channels will not be returned here.

pygame.mixer.get_busy()

test if any sound is being mixed

get_busy() -> bool

Returns True if the mixer is busy mixing any channels. If the mixer is idle then this return False.

pygame.mixer.Sound

Create a new Sound object from a file or buffer object

Sound(filename) -> Sound

Sound(file=filename) -> Sound

Sound(buffer) -> Sound

Sound(buffer=buffer) -> Sound

Sound(object) -> Sound

Sound(file=object) -> Sound

Sound(array=object) -> Sound

- pygame.mixer.Sound.play — begin sound playback
- pygame.mixer.Sound.stop — stop sound playback
- pygame.mixer.Sound.fadeout — stop sound playback after fading out
- pygame.mixer.Sound.set_volume — set the playback volume for this Sound

- pygame.mixer.Sound.get_volume — get the playback volume
- pygame.mixer.Sound.get_num_channels — count how many times this Sound is playing
- pygame.mixer.Sound.get_length— get the length of the Sound
- pygame.mixer.Sound.get_raw — return a bytestring copy of the Sound samples.

Load a new sound buffer from a filename, a python file object or a readable buffer object. Limited resampling will be performed to help the sample match the initialize arguments for the mixer. A Unicode string can only be a file pathname. A Python 2.x string or a Python 3.x bytes object can be either a pathname or a buffer object. Use the 'file' or 'buffer' keywords to avoid ambiguity; otherwise Sound may guess wrong. If the array keyword is used, the object is expected to export a version 3, C level array interface or, for Python 2.6 or later, a new buffer interface (The object is checked for a buffer interface first.)

The Sound object represents actual sound sample data. Methods that change the state of the Sound object will the all instances of the Sound playback. A Sound object also exports an array interface, and, for Python 2.6 or later, a new buffer interface.

The Sound can be loaded from an OGG audio file or from an uncompressed WAV.

Note: The buffer will be copied internally, no data will be shared between it and the Sound object.

For now buffer and array support is consistent with sndarray.make_sound for Numeric arrays, in that sample sign and byte order are ignored. This will change, either by correctly handling sign and byte order, or by raising an exception when different. Also, source samples are truncated to fit the audio sample size. This will not change.

pygame.mixer.Sound(buffer) is new in pygame 1.8 pygame.mixer.SoundCreate a new Sound object from a file or buffer object keyword arguments and array interface support new in pygame 1.9.2

play()

begin sound playback

play(loops=0, maxtime=0, fade_ms=0) -> Channel

Begin playback of the Sound (i.e., on the computer's speakers) on an available Channel. This will forcibly select a Channel, so playback may cut off a currently playing sound if necessary.

The loops argument controls how many times the sample will be repeated after being played the first time. A value of 5 means that the sound will be played once, then repeated five times, and so is played a total of six times. The default value (zero) means the Sound is not repeated, and so is only played once. If loops is set to -1 the Sound will loop indefinitely (though you can still call stop() to stop it).

The maxtime argument can be used to stop playback after a given number of milliseconds.

The fade_ms argument will make the sound start playing at 0 volume and fade up to full volume over the time given. The sample may end before the fade-in is complete.

This returns the Channel object for the channel that was selected.

stop()

stop sound playback

stop() -> None

This will stop the playback of this Sound on any active Channels.

fadeout()

stop sound playback after fading out

fadeout(time) -> None

This will stop playback of the sound after fading it out over the time argument in milliseconds. The Sound will fade and stop on all actively playing channels.

set_volume()

set the playback volume for this Sound

set_volume(value) -> None

This will set the playback volume (loudness) for this Sound. This will immediately affect the Sound if it is playing. It will also affect any future playback of this Sound. The argument is a value from 0.0 to 1.0.

get_volume()

get the playback volume

get_volume() -> value

Return a value from 0.0 to 1.0 representing the volume for this Sound.

get_num_channels()

count how many times this Sound is playing

get_num_channels() -> count

Return the number of active channels this sound is playing on.

get_length()

get the length of the Sound

get_length() -> seconds

Return the length of this Sound in seconds.

get_raw()

return a bytestring copy of the Sound samples.

get_raw() -> bytes

Return a copy of the Sound object buffer as a bytes (for Python 3.x) or str (for Python 2.x) object.

pygame.mixer.Channel

Create a Channel object for controlling playback

Channel(id) -> Channel

- pygame.mixer.Channel.play — play a Sound on a specific Channel
- pygame.mixer.Channel.stop — stop playback on a Channel
- pygame.mixer.Channel.pause — temporarily stop playback of a channel
- pygame.mixer.Channel.unpause — resume pause playback of a channel
- pygame.mixer.Channel.fadeout — stop playback after fading channel out
- pygame.mixer.Channel.set_volume — set the volume of a playing channel
- pygame.mixer.Channel.get_volume — get the volume of the playing channel
- pygame.mixer.Channel.get_busy — check if the channel is active
- pygame.mixer.Channel.get_sound — get the currently playing Sound
- pygame.mixer.Channel.queue — queue a Sound object to follow the current
- pygame.mixer.Channel.get_queue — return any Sound that is queued
- pygame.mixer.Channel.set_endevent — have the channel send an event when playback stops
- pygame.mixer.Channel.get_endevent — get the event a channel sends when playback stops

Return a Channel object for one of the current channels. The id must be a value from 0 to the value of pygame.mixer.get_num_channels()

The Channel object can be used to get fine control over the playback of Sounds. A channel can only playback a single Sound at time. Using channels is entirely optional since pygame can manage them by default.

play()

play a Sound on a specific Channel

play(Sound, loops=0, maxtime=0, fade_ms=0) -> None

This will begin playback of a Sound on a specific Channel. If the Channel is currently playing any other Sound it will be stopped.

The loops argument has the same meaning as in Sound.play(): it is the number of times to repeat the sound after the first time. If it is 3, the sound will be played 4 times (the first time, then three more). If loops is -1 then the playback will repeat indefinitely.

As in Sound.play(), the maxtime argument can be used to stop playback of the Sound after a given number of milliseconds.

As in Sound.play(), the fade_ms argument can be used fade in the sound.

stop()

stop playback on a Channel

stop() -> None

Stop sound playback on a channel. After playback is stopped the channel becomes available for new Sounds to play on it.

pause()

temporarily stop playback of a channel

pause() -> None

Temporarily stop the playback of sound on a channel. It can be resumed at a later time with Channel.unpause()

unpause()

resume pause playback of a channel

unpause() -> None

Resume the playback on a paused channel.

fadeout()

stop playback after fading channel out

fadeout(time) -> None

Stop playback of a channel after fading out the sound over the given time argument in milliseconds.

set_volume()

set the volume of a playing channel

set_volume(value) -> None

set_volume(left, right) -> None

Set the volume (loudness) of a playing sound. When a channel starts to play its volume value is reset. This only affects the current sound. The value argument is between 0.0 and 1.0.

If one argument is passed, it will be the volume of both speakers. If two arguments are passed and the mixer is in stereo mode, the first argument will be the volume of the left speaker and the second will be the volume of the right speaker. (If the second argument is None, the first argument will be the volume of both speakers.)

If the channel is playing a Sound on which set_volume() has also been called, both calls are taken into account. For example:

```
sound = pygame.mixer.Sound("s.wav")
channel = s.play()          # Sound plays at full volume by default
sound.set_volume(0.9)       # Now plays at 90% of full volume.
sound.set_volume(0.6)       # Now plays at 60% (previous value replaced).
channel.set_volume(0.5) # Now plays at 30% (0.6 * 0.5).
```

get_volume()

get the volume of the playing channel

get_volume() -> value

Return the volume of the channel for the current playing sound. This does not take into account stereo separation used by Channel.set_volume(). The Sound object also has its own volume which is mixed with the channel.

get_busy()

check if the channel is active

get_busy() -> bool

Returns true if the channel is actively mixing sound. If the channel is idle this returns False.

get_sound()

get the currently playing Sound

get_sound() -> Sound

Return the actual Sound object currently playing on this channel. If the channel is idle None is returned.

queue()

queue a Sound object to follow the current

queue(Sound) -> None

When a Sound is queued on a Channel, it will begin playing immediately after the current Sound is finished. Each channel can only have a single Sound queued at a time. The queued Sound will only play if the current playback finished automatically. It is cleared on any other call to Channel.stop() or Channel.play().

If there is no sound actively playing on the Channel then the Sound will begin playing immediately.

get_queue()

return any Sound that is queued

get_queue() -> Sound

If a Sound is already queued on this channel it will be returned. Once the queued sound begins playback it will no longer be on the queue.

set_endevent()

have the channel send an event when playback stops

set_endevent() -> None

set_endevent(type) -> None

When an endevent is set for a channel, it will send an event to the pygame queue every time a sound finishes playing on that

channel (not just the first time). Use pygame.event.get() to retrieve the endevent once it's sent.

Note that if you called Sound.play(n) or Channel.play(sound,n), the end event is sent only once: after the sound has been played "n+1" times (see the documentation of Sound.play).

If Channel.stop() or Channel.play() is called while the sound was still playing, the event will be posted immediately.

The type argument will be the event id sent to the queue. This can be any valid event type, but a good choice would be a value between pygame.locals.USEREVENT and pygame.locals.NUMEVENTS. If no type argument is given then the Channel will stop sending endevents.

get_endevent()

get the event a channel sends when playback stops

get_endevent() -> type

Returns the event type to be sent every time the Channel finishes playback of a Sound. If there is no endevent the function returns pygame.NOEVENT.