Table of Contents

- Manual
  - Getting Started
    - Introduction
    - Quickstart
  - Key Concepts
    - Music Bank
    - Music Player
    - Playlist
    - Sound
    - Sound Bank
    - Sound Bus
  - Programming
    - Bank Content
    - Bank Management
    - ID Attributes
    - Music Callbacks
    - Music Manager
    - Sound Instance
    - Sound Manager
- API Reference
  - Stem
    - AttenuationMode
    - AudioClipImportMode
    - AudioClipManagementMode
    - IAudioClipContainer
    - IBank
    - ID
    - MusicBank
    - MusicManager
    - MusicPlayer
    - MusicPlayerAddedDelegate
    - MusicPlayerIDAttribute
    - MusicPlayerPlaybackMode

MusicPlayerRemovedDelegate
MusicPlayerRenamedDelegate
PlaybackChangedDelegate
Playlist
PlaylistAddedDelegate
PlaylistIDAttribute
PlaylistRemovedDelegate
PlaylistRenamedDelegate
PlaylistTrack
RetriggerMode
Sound
SoundAddedDelegate
SoundBank
SoundBus
SoundBusAddedDelegate
SoundBusIDAttribute
SoundBusRemovedDelegate
SoundBusRenamedDelegate
SoundBusRuntime
SoundIDAttribute
SoundInstance
SoundManager
SoundRemovedDelegate
SoundRenamedDelegate
SoundVariation
TrackChangedDelegate

Introduction

Overview

Stem is a robust and versatile audio manager designed from scratch to be extremely fast and easy to use.

With **Stem** you get a great solution for your audio needs: SFX with variations, layered music playback, automatic crossfades, music events, pooled sounds and more! And all of that without scene modification: just create a single asset, fill it with audio clips and you're ready to go!

There is also a **Stem Pro** package with additional features such as audio events, workflow improvements, advanced memory management and more.

Basic Features

- **No scene setup required:** start working easily from a single asset;
- **Optimized for runtime:** excellent performance, zero memory allocations per frame;
- **Permanent:** keep playing music and sounds while transitioning between scenes;
- **Lifelike sound:** easily add variations, randomize volume/pitch/delay;
- **Adaptive music:** mix and crossfade music layers the way you want;
- **Supports Unity 5 Audio:** easily connect to mixer groups and use built-in audio effects;
- **Full control:** tune the most important settings right in the asset, react to music events with callbacks.

Pro Features

- **Save time:** use batch import to quickly add new content;
- **Iterate quickly:** tweak and play sounds in Edit Mode;
- **React to gameplay changes:** create audio events without writing code;
- **Keep music in sync:** create complex soundscapes using sync groups;the code;
- **Control audio usage:** use memory management modes to optimize consumption;

Package Contents

The package consists of four folders:

1. *Gizmos* — contains icons for sound and music bank assets;
2. *Stem/Documentation* — contains pdf file with manual and API reference, fully matches the content on the site;
3. *Stem/Samples* — simple scenes that covers basic **Stem** features;
4. *Stem/Scripts* — source code of the package.

Stem source code is divided into five folders:

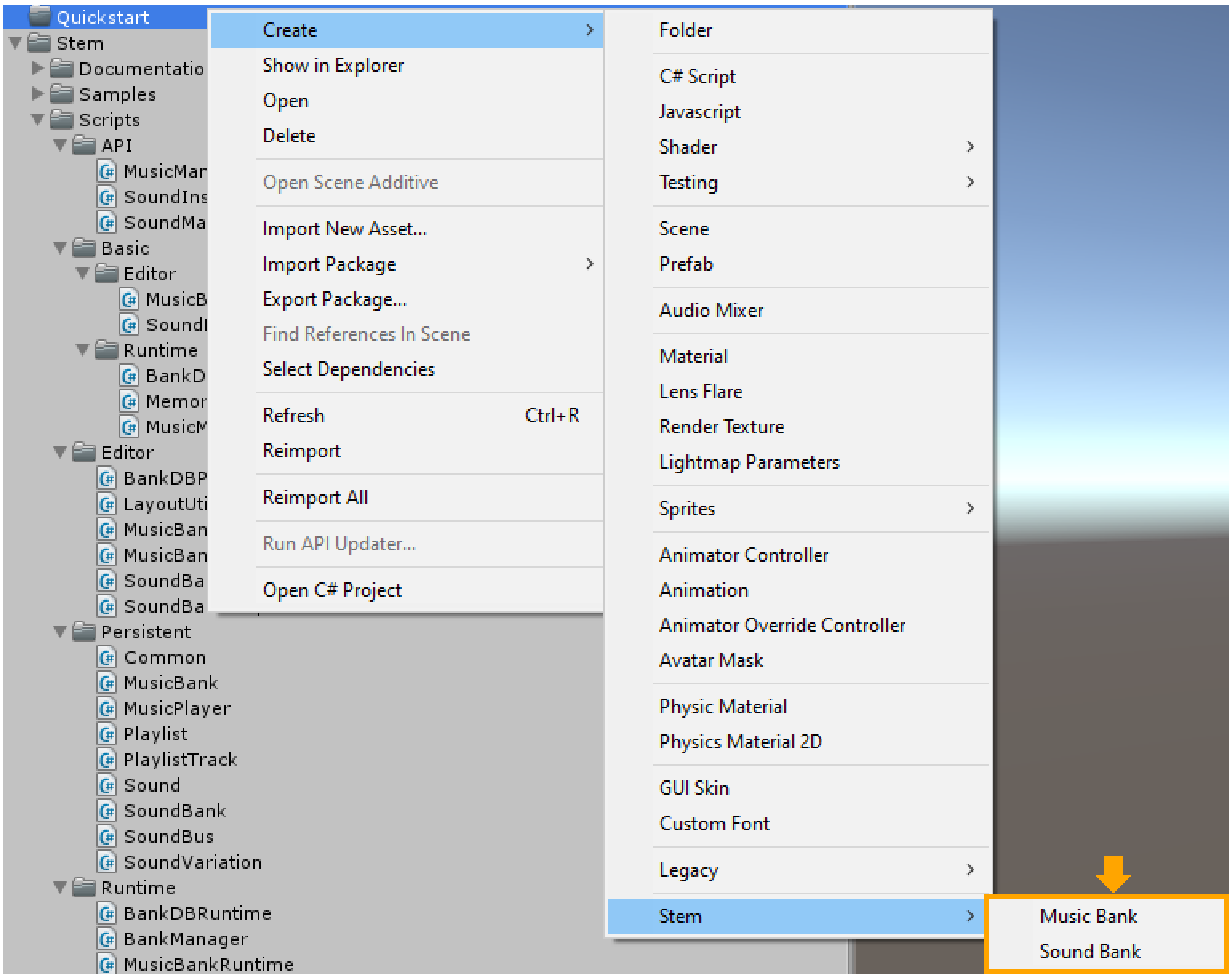
1. *API* — public API classes which you will be working with most of the time;
2. *Persistent* — public data classes, that are stored inside sound and music bank assets;
3. *Editor* — internal editor-only classes (inspectors for sound and music banks, scene postprocessor), will be excluded during the build;
4. *Runtime* — internal logic classes which **Stem** automatically creates on demand;
5. *Basic* or *Pro* — contains specific features depending on **Stem** version.

Quickstart

Create assets

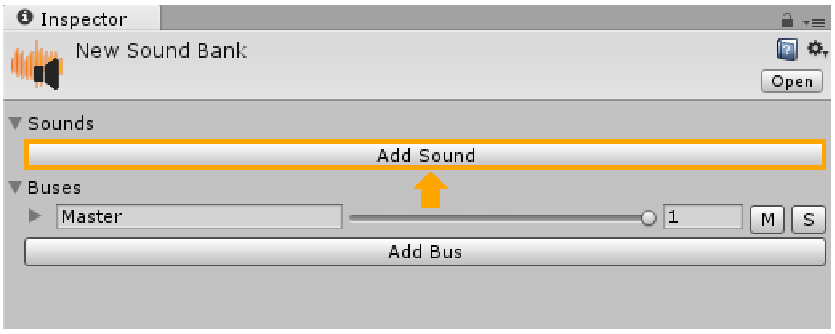
Press the right mouse button on any folder to open the Assets menu.

1. Choose "Create -> Stem -> Music Bank" to create new music bank.
2. Choose "Create -> Stem -> Sound Bank" to create new sound bank.

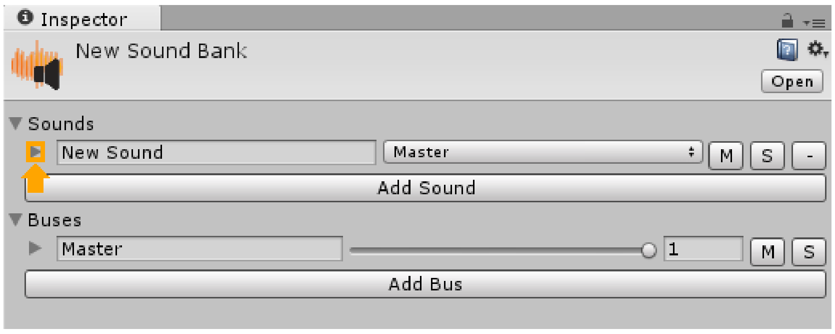


Setup sound bank

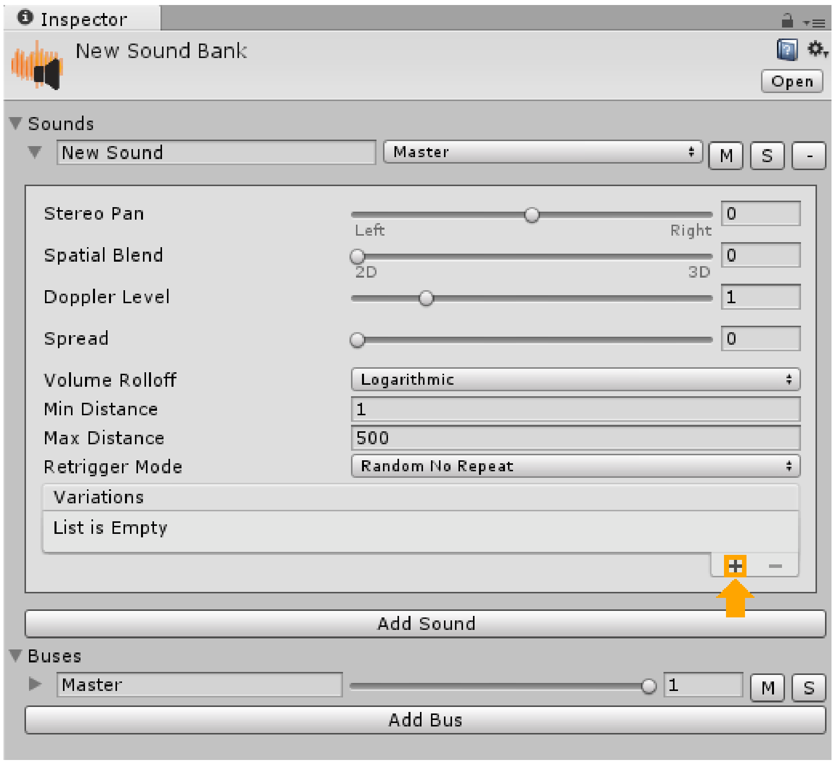
1. Select a sound bank asset and add a new sound by pressing the "Add Sound" button in the inspector window.



2. Expand sound settings by clicking the arrow near the sound name.

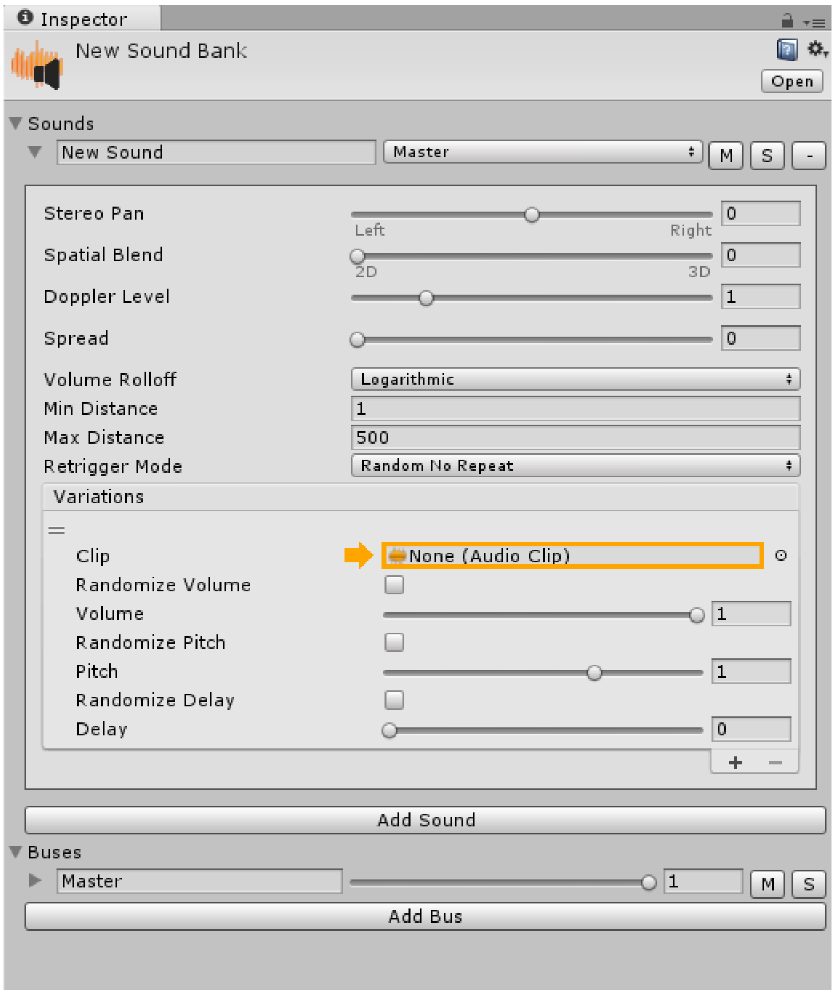


3. Add the first variation to the sound by pressing the "+" button in the "Variations" list.



Stereo Pan 0
Spatial Blend 0
Doppler Level 1
Spread 0
Volume Rolloff Logarithmic
Min Distance 1
Max Distance 500
Retrigger Mode Random No Repeat
Variations
List is Empty

4. Assign an audio clip to the variation by dragging it into the "Clip" property.



Inspector

New Sound Bank

Open

Sounds

New Sound — Master — M S -

Stereo Pan 0 (Left / Right)
Spatial Blend 0 (2D / 3D)
Doppler Level 1
Spread 0
Volume Rolloff Logarithmic
Min Distance 1
Max Distance 500
Retrigger Mode Random No Repeat

Variations

Clip None (Audio Clip)
Randomize Volume ☐
Volume 1
Randomize Pitch ☐
Pitch 1
Randomize Delay ☐
Delay 0

+ −

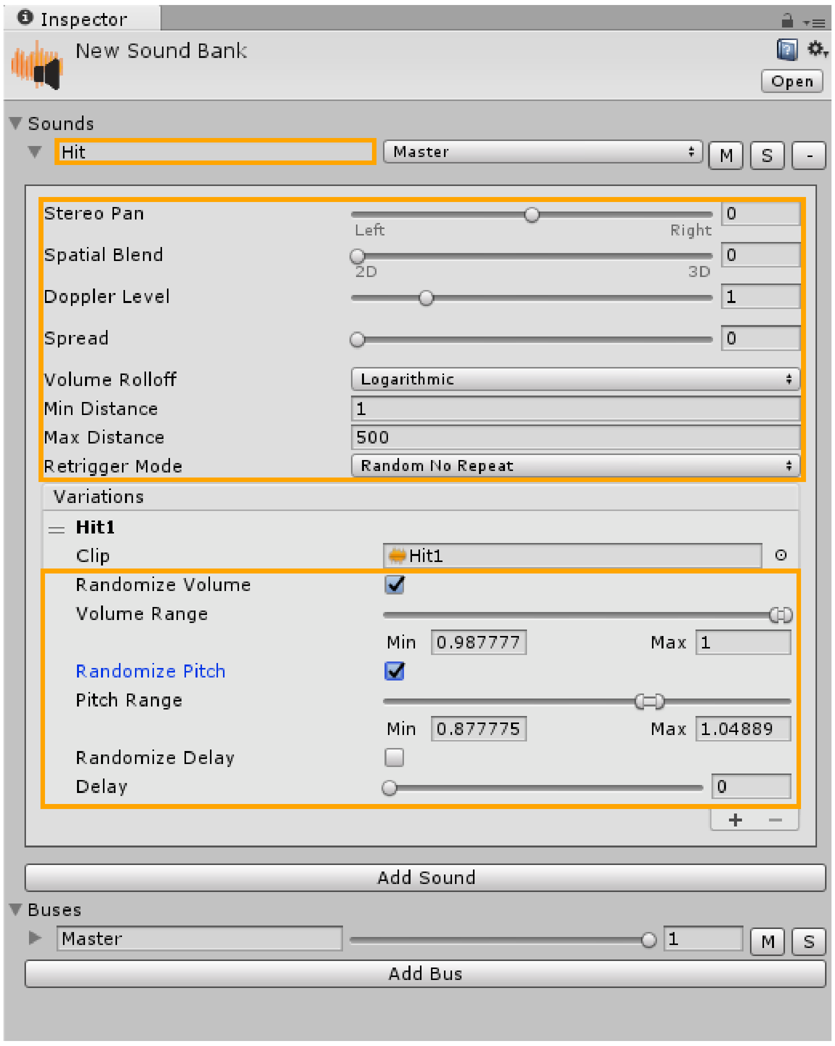
Add Sound

Buses

Master 1 M S

Add Bus

5. Set sound name, adjust parameters and randomization settings.


Inspector

New Sound Bank

Open

Sounds

Hit | Master | M | S | -

Stereo Pan 0 (Left – Right)

Spatial Blend 0 (2D – 3D)

Doppler Level 1

Spread 0

Volume Rolloff: Logarithmic

Min Distance: 1

Max Distance: 500

Retrigger Mode: Random No Repeat

Variations

Hit1

Clip: Hit1

Randomize Volume: ✓

Volume Range: Min 0.987777 Max 1

Randomize Pitch: ✓

Pitch Range: Min 0.877775 Max 1.04889

Randomize Delay: ☐

Delay: 0

Add Sound

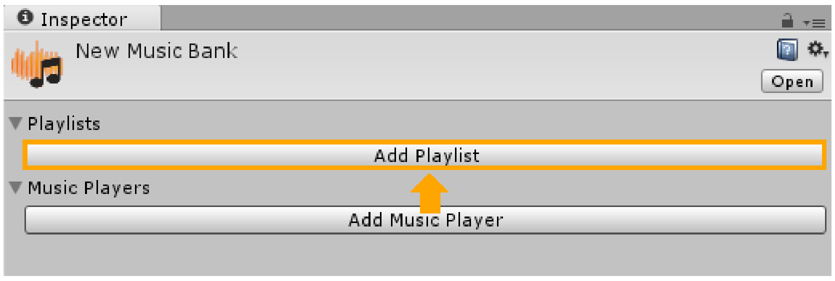
Buses

Master 1 | M | S

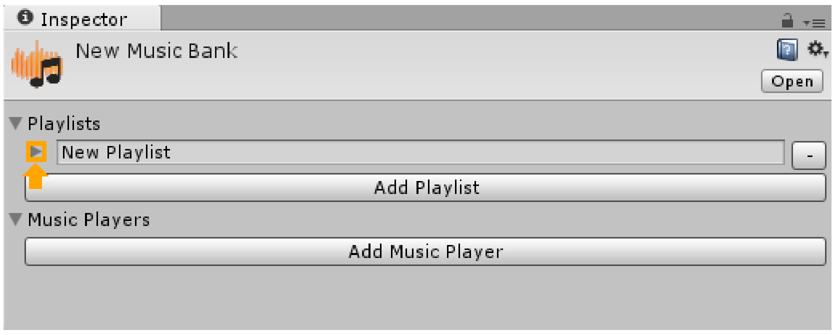
Add Bus

Setup music bank

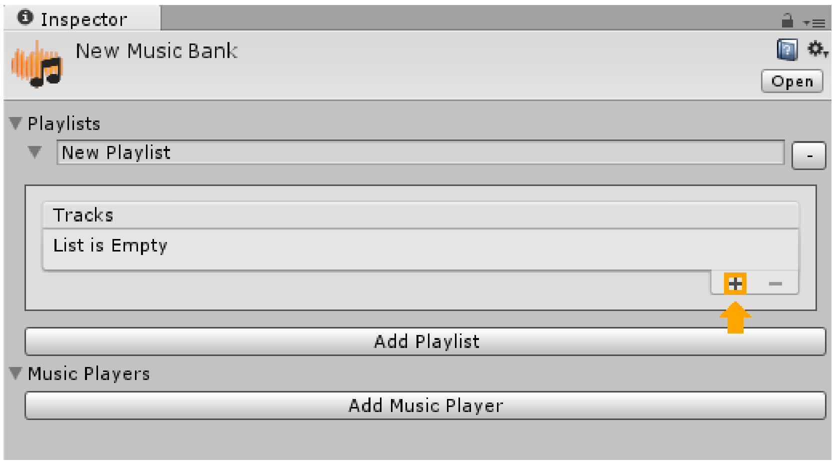
1. Select a music bank asset and add a new playlist by pressing the "Add Playlist" button in the inspector window.



2. Expand playlist settings by clicking the arrow near the playlist name.



3. Add the first track to the playlist by pressing the "+" button in the "Tracks" list.


Inspector

New Music Bank

Open

Playlists

New Playlist

Tracks

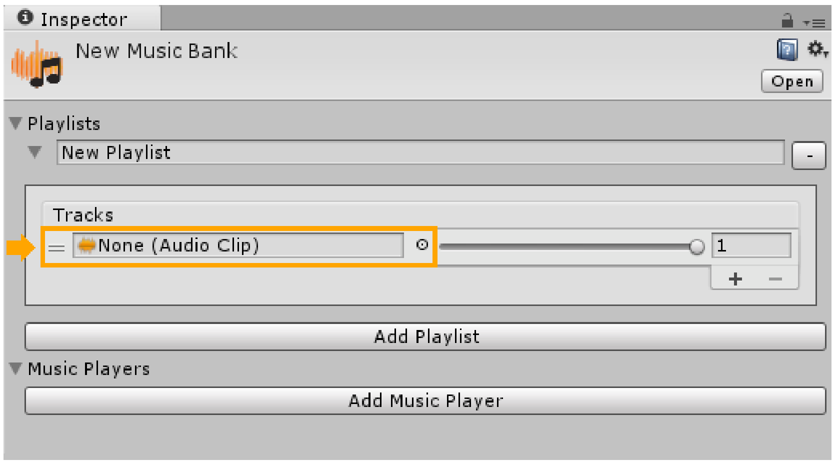
List is Empty

Add Playlist

Music Players

Add Music Player

4. Assign an audio clip to the track.


Inspector

New Music Bank

Open

Playlists

New Playlist

Tracks

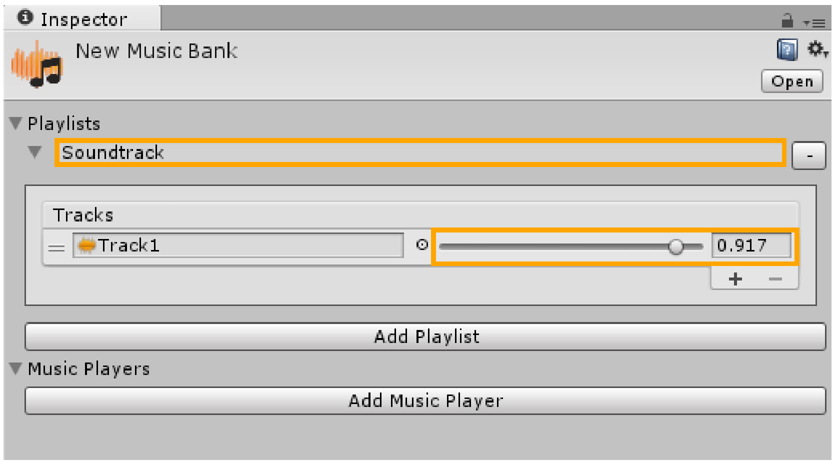
None (Audio Clip) 1

Add Playlist

Music Players

Add Music Player

5. Set playlist name, adjust track volume.


Inspector

New Music Bank

Open

Playlists

Soundtrack

Tracks

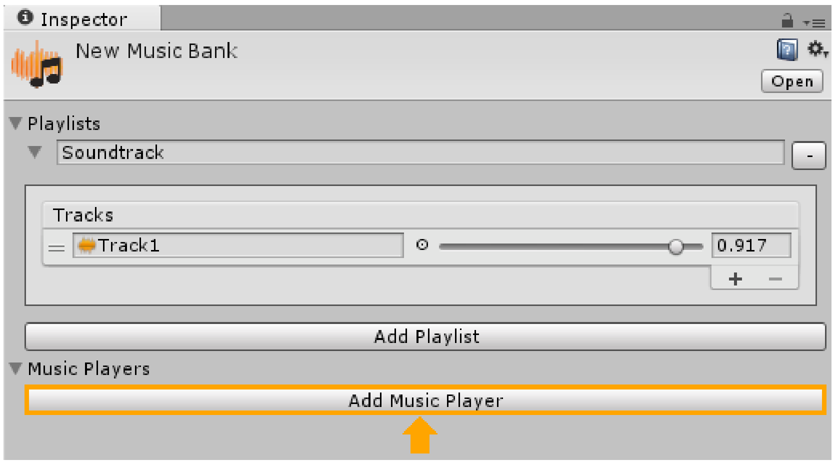
Track1 0.917

Add Playlist

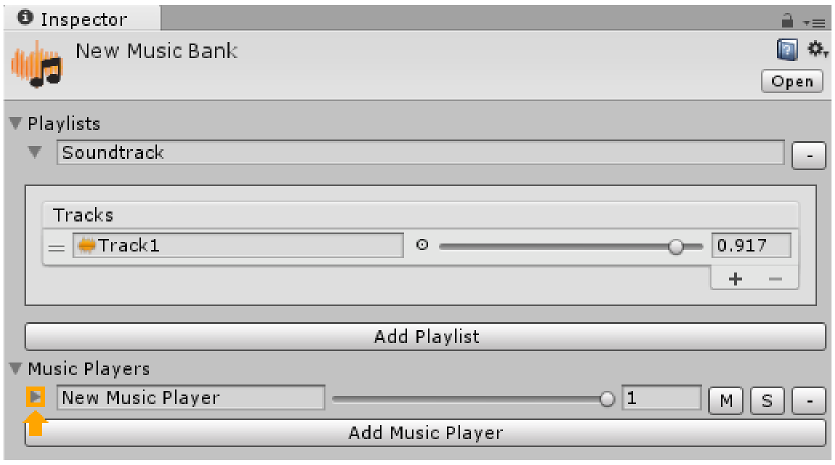
Music Players

Add Music Player

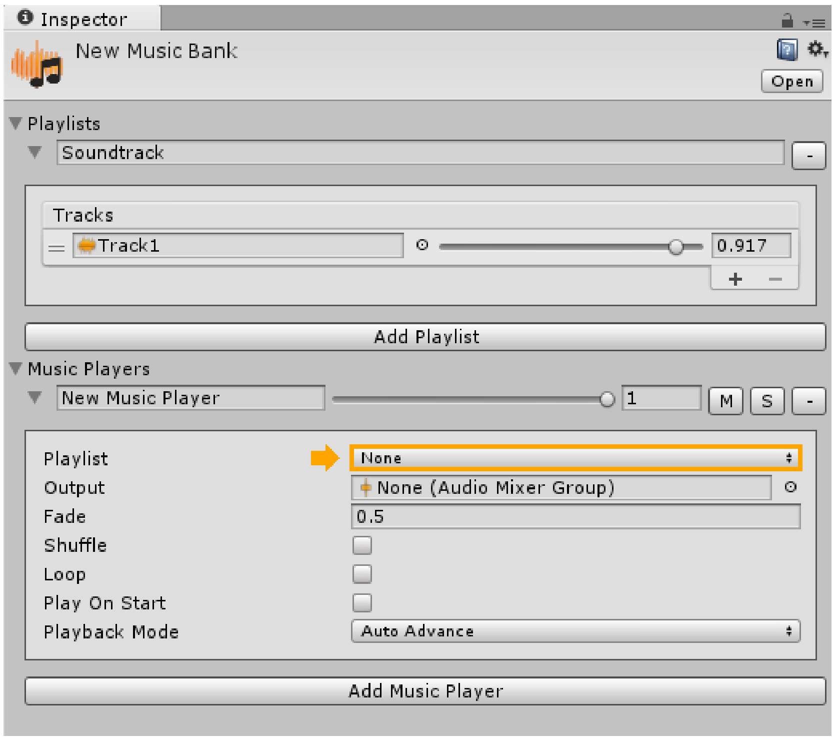
6. Add a new music player by pressing the "Add Player" button in the inspector window.



7. Expand music player settings by clicking the arrow near the music player name.

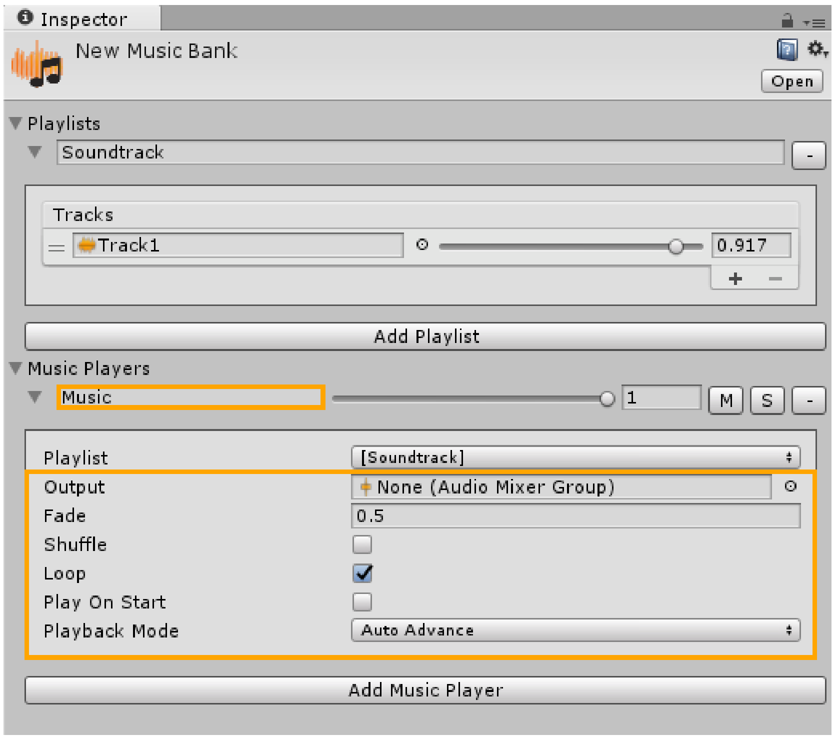


8. Choose the playlist from the "Playlist" drop-down menu.



Playlist	None
Output	None (Audio Mixer Group)
Fade	0.5
Shuffle	☐
Loop	☐
Play On Start	☐
Playback Mode	Auto Advance

9. Set music player name, adjust parameters.



Playlist: [Soundtrack]
Output: None (Audio Mixer Group)
Fade: 0.5
Shuffle: ☐
Loop: ✓
Play On Start: ☐
Playback Mode: Auto Advance

Play one-shot sounds

By sound ID in the script

Use Stem.SoundManager.Play to play one-shot sounds.

```
using UnityEngine;

public class PlayerHit : MonoBehaviour
{
    [Stem.SoundID]
    public Stem.ID soundID = Stem.ID.None;

    private void OnTriggerEnter(Collider collider)
    {
        if (collider.tag == "Enemy")
            Stem.SoundManager.Play3D(soundID, transform.position);
    }
}
```

By sound name in the script

Use Stem.SoundManager.Play to play one-shot sounds.

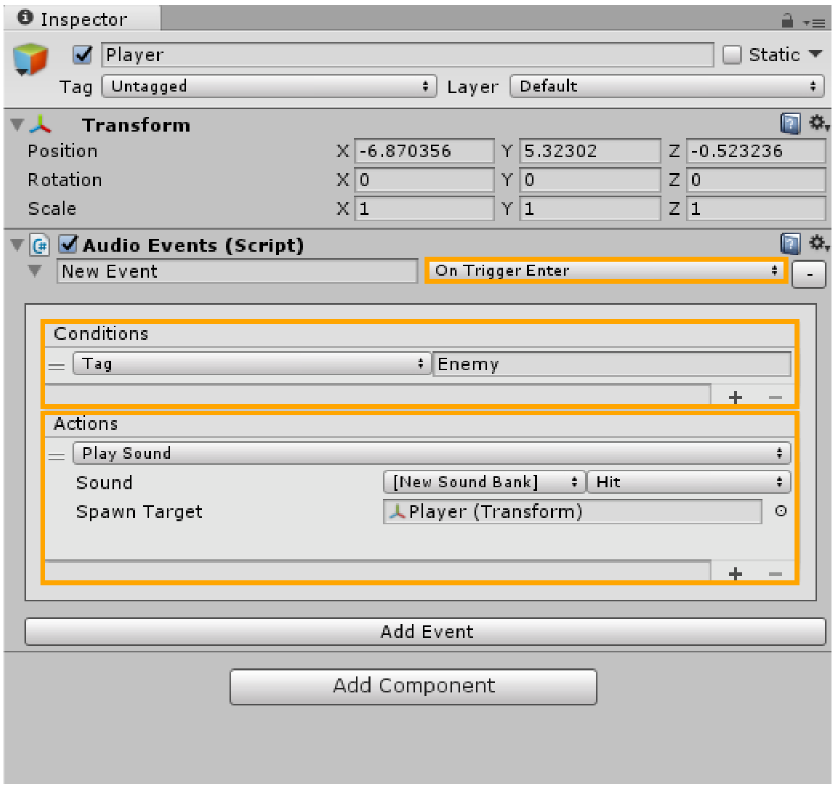
```
using UnityEngine;

public class PlayerHit : MonoBehaviour
{
    public string soundName = "Hit";

    private void OnTriggerEnter(Collider collider)
    {
        if (collider.tag == "Enemy")
            Stem.SoundManager.Play3D(soundName, transform.position);
    }
}
```

(Stem Pro) With AudioEvents component

Add new audio event, set event type to "On Trigger Enter", add a single condition and "Play Sound" action.


Inspector

Player | Static
Tag: Untagged | Layer: Default

Transform
Position X -6.870356 Y 5.32302 Z -0.523236
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Audio Events (Script)
New Event | On Trigger Enter

Conditions
Tag | Enemy

Actions
Play Sound
Sound: [New Sound Bank] | Hit
Spawn Target: Player (Transform)

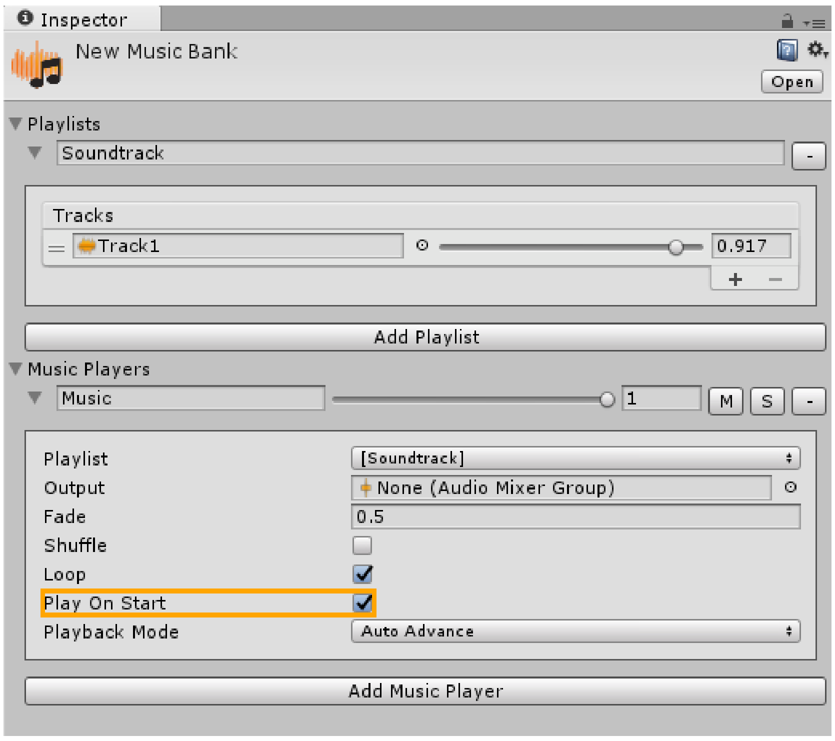
Add Event

Add Component

Start music playback

Via "Play On Start" option in the music bank

Check "Play On Start" option to automatically play current music player when the game starts.


Inspector

New Music Bank | Open

Playlists
Soundtrack

Tracks
Track1 | 0.917

Add Playlist

Music Players
Music | 1 | M | S

Playlist: [Soundtrack]
Output: None (Audio Mixer Group)
Fade: 0.5
Shuffle: ☐
Loop: ✓
Play On Start: ✓
Playback Mode: Auto Advance

Add Music Player

By music player ID in the script

Use Stem.MusicManager.Play to start music playback during level startup.

```
using UnityEngine;

public class LevelStartup : MonoBehaviour
{
    [Stem.MusicPlayerID]
    public Stem.ID musicPlayerID = Stem.ID.None;

    private void Start()
    {
        Stem.MusicManager.Play(musicPlayerID);
    }
}
```

By music player name in the script

Use Stem.MusicManager.Play to start music playback during level startup.

```
using UnityEngine;

public class LevelStartup : MonoBehaviour
{
    public string musicPlayerName = "Music";

    private void Start()
    {
        Stem.MusicManager.Play(musicPlayerName);
    }
}
```

Music Bank

API Reference

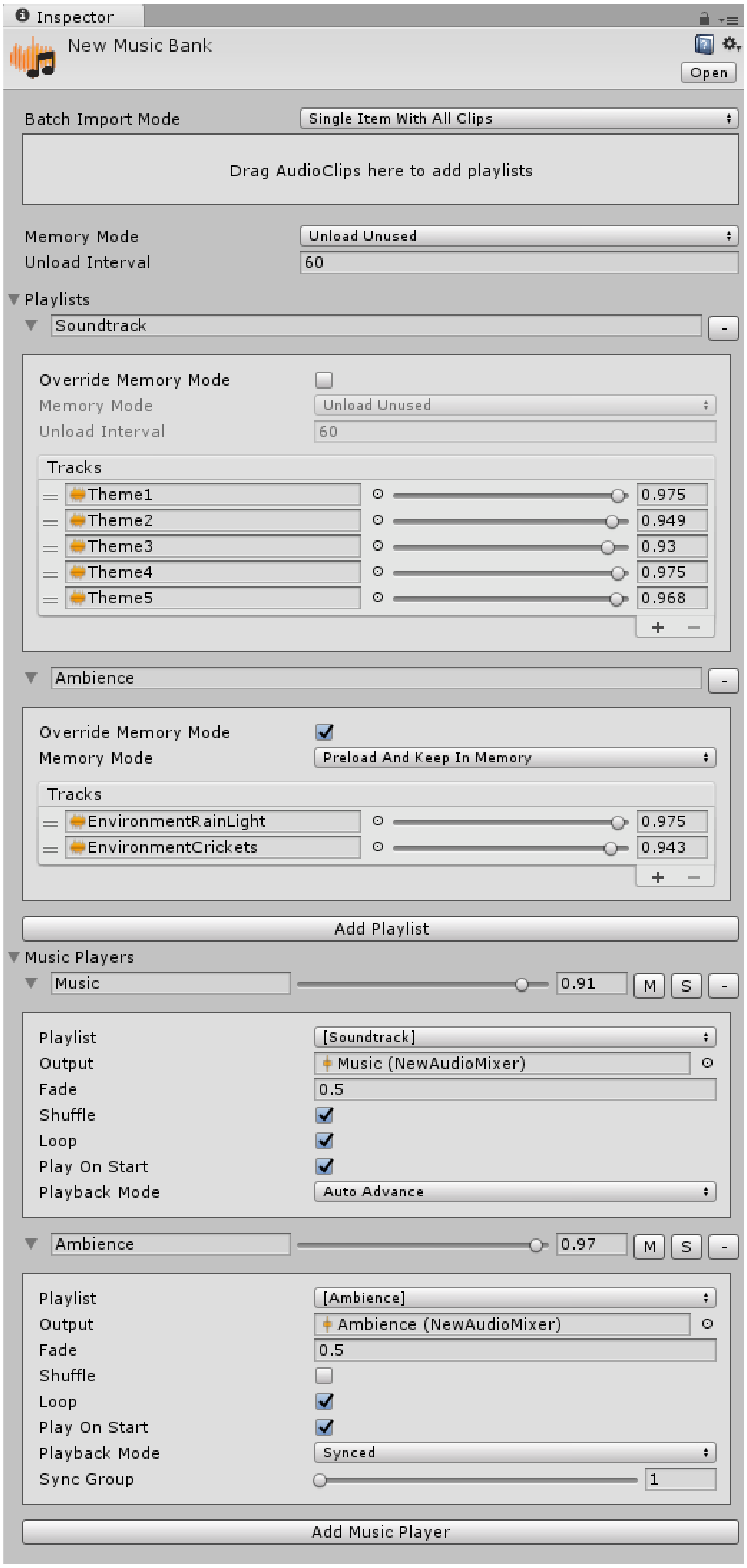
Description

Music bank is a collection of playlists and music players. It's stored inside the project *Assets* folder. Once filled with the content, it's ready to use in the code.

It's possible to have several music banks. Stem will automatically search for the corresponding music bank during the playlist or music player lookup. In the case of name collisions, if multiple banks have music players or playlists with the same name, the primary music bank (which you can only set in the code) will be checked first. Within a bank, the first occurrence of music player or playlist will be used.

All music tracks must be organized into playlists. It could be thematic playlists, i.e. gameplay music, menu music, outdoor ambience, etc. or big single playlist containing all the tracks.

Each music player can only play a single playlist. Music bank allows doing a layered music playback by creating multiple music players and filling them with different playlists.


Inspector
New Music Bank
Open

Batch Import Mode: Single Item With All Clips

Drag AudioClips here to add playlists

Memory Mode: Unload Unused
Unload Interval: 60

Playlists

Soundtrack

Override Memory Mode: ☐
Memory Mode: Unload Unused
Unload Interval: 60

Tracks

Theme1	0.975
Theme2	0.949
Theme3	0.93
Theme4	0.975
Theme5	0.968

Ambience

Override Memory Mode: ✓
Memory Mode: Preload And Keep In Memory

Tracks

EnvironmentRainLight	0.975
EnvironmentCrickets	0.943

Add Playlist

Music Players

Music 0.91 M S -

Playlist: [Soundtrack]
Output: Music (NewAudioMixer)
Fade: 0.5
Shuffle: ✓
Loop: ✓
Play On Start: ✓
Playback Mode: Auto Advance

Ambience 0.97 M S -

Playlist: [Ambience]
Output: Ambience (NewAudioMixer)
Fade: 0.5
Shuffle: ☐
Loop: ✓
Play On Start: ✓
Playback Mode: Synced
Sync Group: 1

Add Music Player

Properties

PROPERTY	DESCRIPTION
(Stem Pro) Memory Mode	The mode defining how audio clips will be managed in memory.
(Stem Pro) Unload Interval	The interval after which audio clips will be unloaded from memory.

PROPERTY	DESCRIPTION
Playlists	The collection of playlists.
Players	The collection of music players.

(Stem Pro) Batch Import

Stem Pro offers batch import via drag-and-drop. This greatly reduces the time spent on content creation. There are two import modes allowing to create either a new playlist per audio clip or a single playlist with all provided audio clips.

BATCH IMPORT MODE	DESCRIPTION
Single Item With All Clips	A single playlist with all provided audio clips will be created.
Multiple Items With Single Clip	A multiple playlists will be created, one for each provided audio clip.

(Stem Pro) Advanced Memory Management

Stem Pro offers three different management modes which help to reduce overall audio memory usage. Feel free to combine all of them for your needs.

For example, you can create a music bank which will hold all common tracks (e.g. ambience or stingers) and keep it in memory while also having separate music banks for each level with "Unload Unused" memory mode. It's also possible to override memory management mode for individual playlists within the bank.

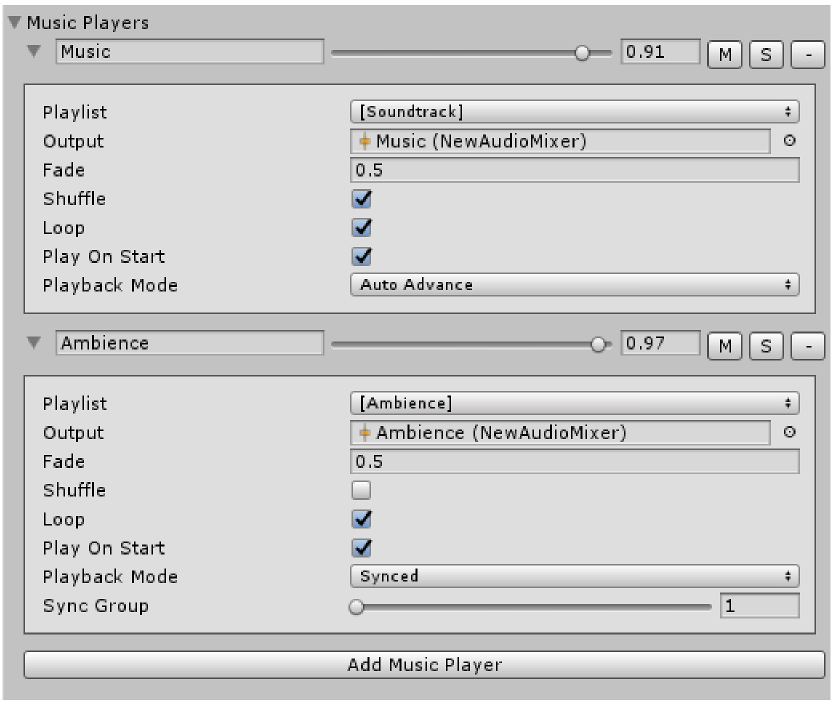
MEMORY MODE	DESCRIPTION
Preload And Keep In Memory	Preload audio clip data during startup and keep it always in memory.
Unload Unused	Unload audio clip data if it was not used for some time.
Manual	Do not manage audio clip data and instead allow the developer to take control.

Music Player

API Reference

Description

A music player is a part of the music bank. It represents an entry point for music playback and defines how it will play the playlist tracks. It also integrates into a Unity Audio Mixer.



Music Players

Music 0.91 M S -

Playlist	[Soundtrack]
Output	Music (NewAudioMixer)
Fade	0.5
Shuffle	✓
Loop	✓
Play On Start	✓
Playback Mode	Auto Advance

Ambience 0.97 M S -

Playlist	[Ambience]
Output	Ambience (NewAudioMixer)
Fade	0.5
Shuffle	☐
Loop	✓
Play On Start	✓
Playback Mode	Synced
Sync Group	1

Add Music Player

Properties

PROPERTY	DESCRIPTION
Name	The name of the music player.
Volume	The master volume of the music player.
Playlist	The drop-down menu allowing to select a playlist. *None* option will require to set the playlist in order to play.
Output	The reference to AudioMixerGroup. Please refer to Unity Manual for details.
Fade	The crossfade duration used by the music player during transitions between tracks or playback state changes.
Shuffle	The flag indicating whether the music player should play tracks in random order.
Loop	The flag indicating whether the music player should repeat playlist tracks after they finish.
Play On Start	The flag indicating whether the music player should start playing once the game started.
Playback Mode	The playback mode defining how music player should play its tracks. *Synced* option is available only in **Stem Pro**.
(Stem Pro) Sync Group	The sync group of the music player. Music players with the same sync group will share playback time.

Playback Mode

Playback Mode and *Loop* properties work in pair. There are six possible combinations:

PLAYBACK MODE	LOOP	BEHAVIOUR
Default	False	Play the current track once and then stop.
	True	Play the current track in a loop and never stop.
Auto Advance	False	Play all playlist tracks once and then stop.
	True	Play all playlist tracks in a loop and never stop.
(Stem Pro) Synced	False	Play the current track once and then stop. Track time is synchronized between other music players with the same *Sync Group* value.
	True	Play the current track in a loop and never stop. Track time is synchronized between other music players with the same *Sync Group* value.

(Stem Pro) Synchronized Music Players

Stem Pro offers synchronization mechanism for music players. This allows switching between different tracks at runtime without worrying they will get out of sync.

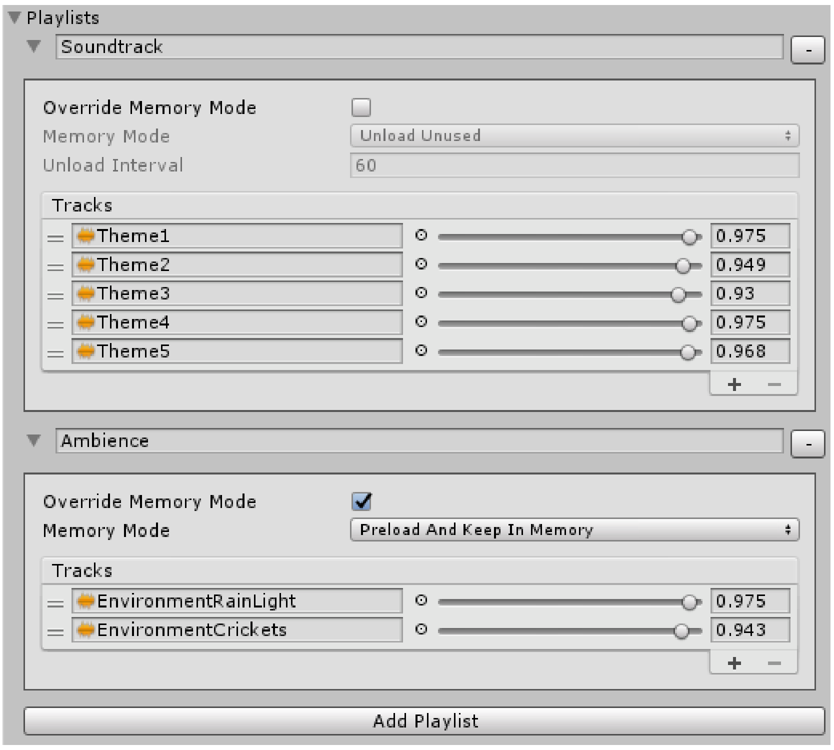
To use this feature set *Playback Mode* property to *Synced* and also assign a *Sync Group* property. *Synced* mode behaves the same way as *Default* except all music players with the same *Sync Group* value will share playback time.

Playlist

API Reference

Description

A playlist is a part of the music bank. It represents a collection of music tracks.



Playlists

Soundtrack

Override Memory Mode: ☐
Memory Mode: Unload Unused
Unload Interval: 60

Tracks

Theme1	0.975
Theme2	0.949
Theme3	0.93
Theme4	0.975
Theme5	0.968

Ambience

Override Memory Mode: ✓
Memory Mode: Preload And Keep In Memory

Tracks

EnvironmentRainLight	0.975
EnvironmentCrickets	0.943

Properties

PROPERTY	DESCRIPTION
Name	The name of the playlist.
(Stem Pro) Override Memory Mode	The flag indicating whether the playlist should use its own memory management mode and unload interval.
(Stem Pro) Memory Mode	The mode defining how audio clips will be managed in memory.
(Stem Pro) Unload Interval	The interval after which audio clips will be unloaded from memory.

Track Properties

API Reference

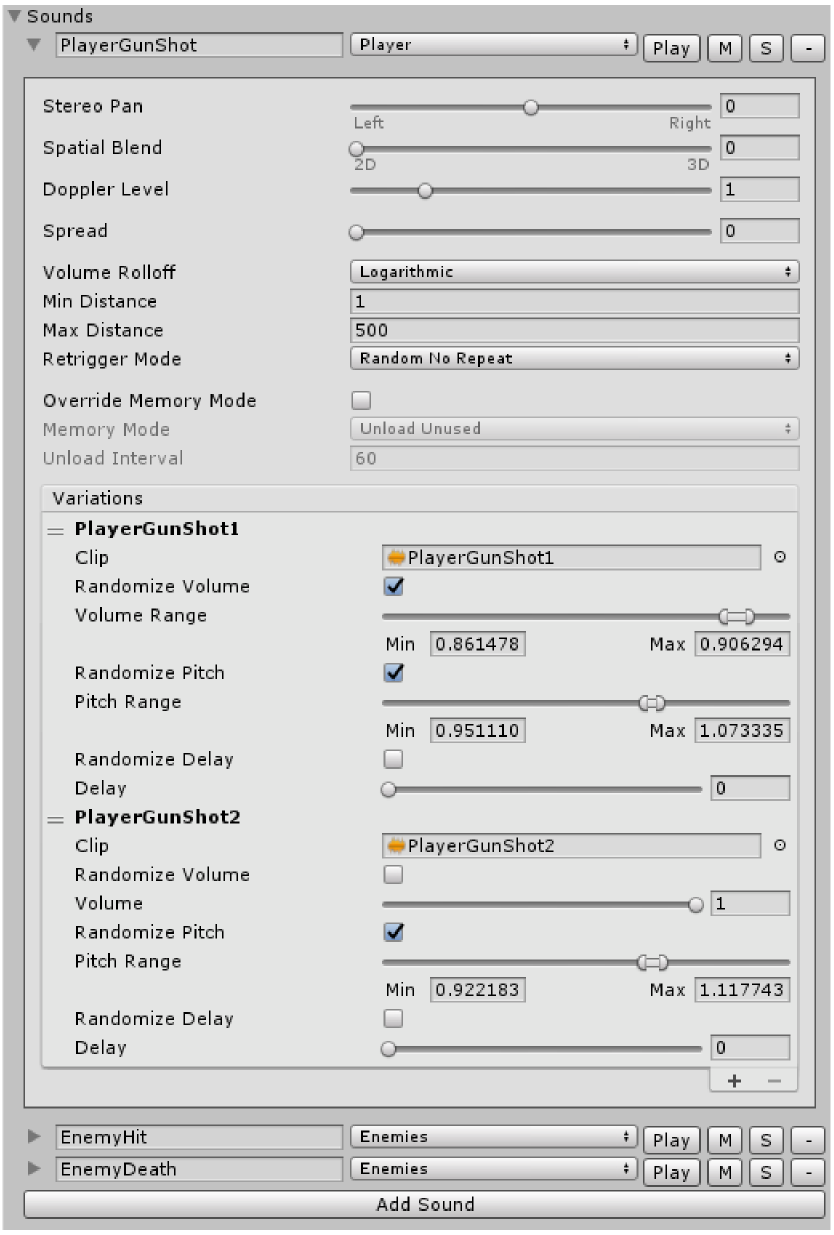
PROPERTY	DESCRIPTION
Clip	The reference to *Audio Clip*. Please refer to Unity Manual for details.
Volume	The master volume of the track.

Sound

API Reference

Description

A sound is a part of the sound bank. It represents a set of parameters defining how the sound will be mixed (most of them are duplicates from *Audio Source*). It could also have multiple audio clips serving as variations. Each sound variation can be randomized in volume, pitch and delay.



Sounds

PlayerGunShot	Player	Play	M	S	-

Stereo Pan	0 (Left – Right)
Spatial Blend	0 (2D – 3D)
Doppler Level	1
Spread	0
Volume Rolloff	Logarithmic
Min Distance	1
Max Distance	500
Retrigger Mode	Random No Repeat
Override Memory Mode	☐
Memory Mode	Unload Unused
Unload Interval	60

Variations

PlayerGunShot1

Clip	PlayerGunShot1
Randomize Volume	✓
Volume Range	Min 0.861478 Max 0.906294
Randomize Pitch	✓
Pitch Range	Min 0.951110 Max 1.073335
Randomize Delay	☐
Delay	0

PlayerGunShot2

Clip	PlayerGunShot2
Randomize Volume	☐
Volume	1
Randomize Pitch	✓
Pitch Range	Min 0.922183 Max 1.117743
Randomize Delay	☐
Delay	0

EnemyHit	Enemies	Play	M	S	-
EnemyDeath	Enemies	Play	M	S	-

Add Sound

Properties

PROPERTY	DESCRIPTION
Name	The name of the sound.
Stereo Pan	The stereo panning parameter defining sound position in a stereo way (left or right).
Spatial Blend	The spatial blend parameter defining how much the sound is affected by 3d spatialisation calculations (attenuation, doppler etc).

PROPERTY	DESCRIPTION
Doppler Level	The scale of doppler effect that will be applied to the sound (if is set to 0, then no effect is applied).
Spread	The spread angle (in degrees) of a 3d stereo or multichannel sound in speaker space.
Volume Rolloff	The attenuation mode defining how sound volume will be lowered over the distance.
Min Distance	The parameter defining the boundary within which the sound won't get any louder. Outside *Min Distance* it will begin to attenuate.
Max Distance	The parameter defining the boundary outside which the sound will be inaudible or stop attenuating depending on **Volume Rolloff** value.
Retrigger Mode	The rule defining how the sound will play variations.
(Stem Pro) Override Memory Mode	The flag indicating whether the sound should use its own memory management mode and unload interval.
(Stem Pro) Memory Mode	The mode defining how audio clips will be managed in memory.
(Stem Pro) Unload Interval	The interval after which audio clips will be unloaded from memory.

> *Stereo Pan*, *Spatial Blend*, *Doppler Level*, *Spread*, *Min Distance*, *Max Distance* parameters duplicate corresponding parameters from *Audio Source*. Please refer to Unity Manual for details.

Variation Properties

API Reference

PROPERTY	DESCRIPTION
Clip	The reference to *AudioClip*. Please refer to Unity Manual for details.
Randomize Volume	The flag indicating whether the sound variation should be randomized in volume.
Volume / Volume Range	The volume of the sound variation. Depending on *Randomize Volume* flag it's either a set value or a random value from the range.
Randomize Pitch	The flag indicating whether the sound variation should be randomized in pitch.
Pitch / Pitch Range	Amount of change in pitch due to slowdown/speed up of the *AudioClip*. Depending on *Randomize Pitch* flag it's either a set value or a random value from the range.
Randomize Delay	The flag indicating whether the sound variation should be randomized in delay.
Delay / Delay Range	The playback delay in seconds. Depending on *Randomize Delay* flag it's either a set value or a random value from the range.

Sound Bank

API Reference

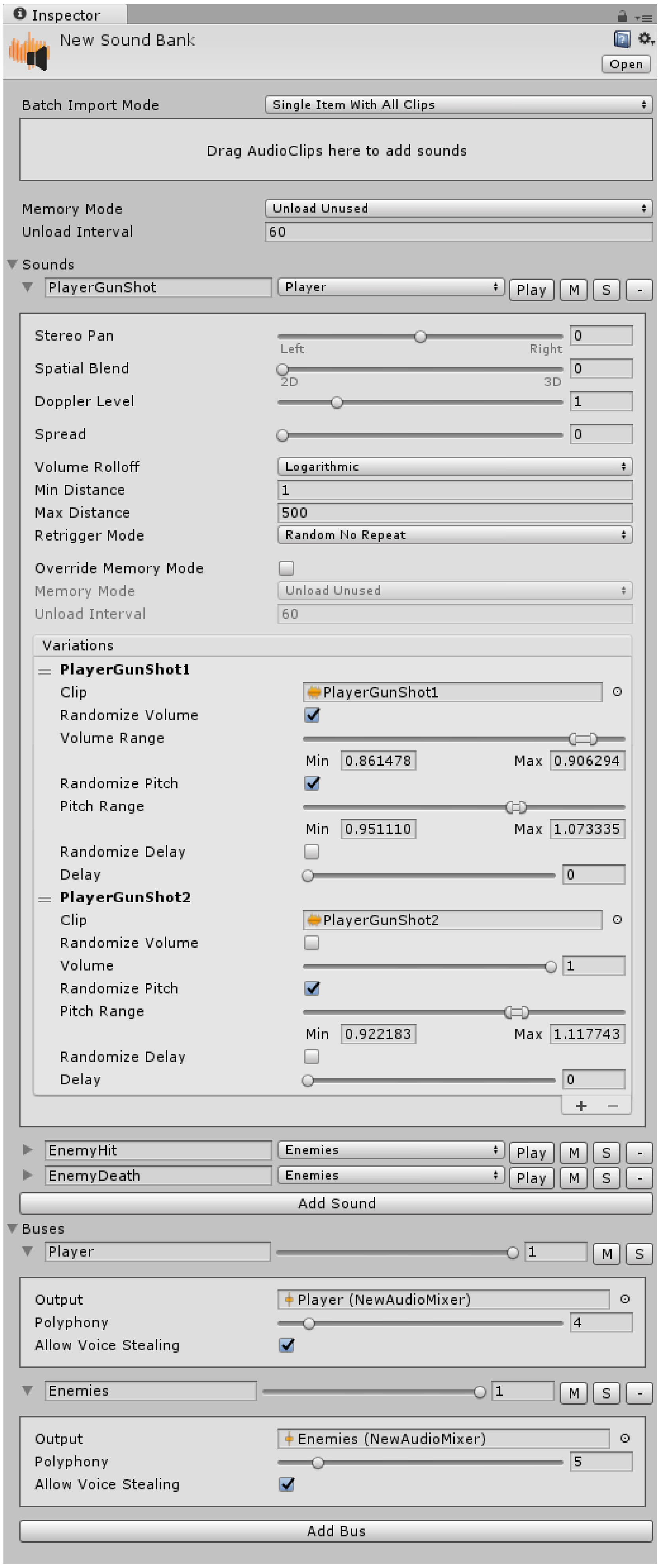
Description

Sound bank is a collection of sounds and sound buses. It's stored inside the project Assets folder. Once filled with the content, it's ready to use in the code.

It's possible to have several sound banks. Stem will automatically search for the corresponding sound bank during the sound or sound bus lookup. In the case of name collisions, if multiple banks have sounds or sound buses with the same name, the primary sound bank (which you can only set in the code) will be checked first. Within a bank, the first occurrence of sound or sound bus will be used.

All sound effects must be organized into sounds with multiple variations. Each sound must have a sound bus assigned. It could be thematic sound buses, i.e. character, enemies, gunshots, etc. or just a single Master sound bus.

Each sound bus controls how many sounds can be played simultaneously.



Inspector

New Sound Bank

Open

Batch Import Mode: Single Item With All Clips

Drag AudioClips here to add sounds

Memory Mode: Unload Unused
Unload Interval: 60

Sounds

PlayerGunShot | Player | Play | M | S | -

Stereo Pan: 0 (Left – Right)
Spatial Blend: 0 (2D – 3D)
Doppler Level: 1
Spread: 0
Volume Rolloff: Logarithmic
Min Distance: 1
Max Distance: 500
Retrigger Mode: Random No Repeat

Override Memory Mode: ☐
Memory Mode: Unload Unused
Unload Interval: 60

Variations

PlayerGunShot1
Clip: PlayerGunShot1
Randomize Volume: ✓
Volume Range: Min 0.861478 Max 0.906294
Randomize Pitch: ✓
Pitch Range: Min 0.951110 Max 1.073335
Randomize Delay: ☐
Delay: 0

PlayerGunShot2
Clip: PlayerGunShot2
Randomize Volume: ☐
Volume: 1
Randomize Pitch: ✓
Pitch Range: Min 0.922183 Max 1.117743
Randomize Delay: ☐
Delay: 0

+ −

EnemyHit | Enemies | Play | M | S | -
EnemyDeath | Enemies | Play | M | S | -

Add Sound

Buses

Player | 1 | M | S

Output: Player (NewAudioMixer)
Polyphony: 4
Allow Voice Stealing: ✓

Enemies | 1 | M | S | -

Output: Enemies (NewAudioMixer)
Polyphony: 5
Allow Voice Stealing: ✓

Add Bus

Properties

PROPERTY	DESCRIPTION
(Stem Pro) Memory Mode	The mode defining how audio clips will be managed in memory.
(Stem Pro) Unload Interval	The interval after which audio clips will be unloaded from memory.
Sounds	The collection of sounds.
Buses	The collection of sound buses.

(Stem Pro) Fast Iterations

Stem Pro offers the ability to tweak sounds and hear changes in Edit Mode. You don't have to run the game, just press the "Play" button near desired sound to hear it.

(Stem Pro) Batch Import

Stem Pro offers batch import via drag-and-drop. This greatly reduces the time spent on content creation. There are two import modes allowing to create either a new sound per audio clip or a single sound with all provided audio clips.

BATCH IMPORT MODE	DESCRIPTION
Single Item With All Clips	A single sound with all provided audio clips will be created.
Multiple Items With Single Clip	A multiple sounds will be created, one for each provided audio clip.

(Stem Pro) Advanced Memory Management

Stem Pro offers three different management modes which help to reduce overall audio memory usage. Feel free to combine all of them for your needs.

For example, you can create a sound bank which will hold all common sounds (e.g. UI or player) and keep it in memory while also having separate sound banks for each level with "Unload Unused" memory mode. It's also possible to override memory management mode for individual sounds within the bank.

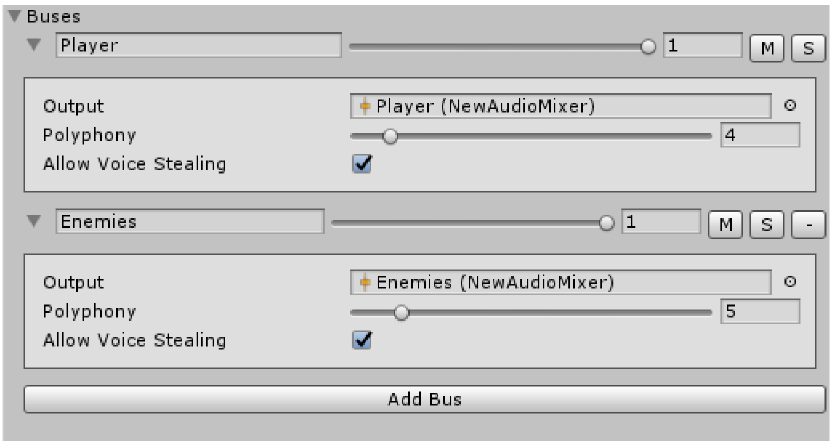
MEMORY MODE	DESCRIPTION
Preload And Keep In Memory	Preload audio clip data during startup and keep it always in memory.
Unload Unused	Unload audio clip data if it was not used for some time.
Manual	Do not manage audio clip data and instead allow the developer to take control.

Sound Bus

API Reference

Description

A sound bus is a part of the sound bank. It groups sound and define the limit of simultaneously playing sounds. It also integrates into a Unity Audio Mixer.



Properties

PROPERTY	DESCRIPTION
Name	The name of the sound bus.
Output	The reference to *Audio Mixer Group*. Please refer to Unity Manual for details.
Polyphony	The number of maximum allowed simultaneously playing sounds in the sound bus.
Allow Voice Stealing	The flag indicating whether the sound bus can stop the oldest playing sound and play the new one if *Polyphony* limit is exceeded.

Bank Content

Description

Both sound and music banks are filled with the content. While most of them have been created and modified in the editor, it's also possible to do that in the code.

Adding content

Call these methods to add new content to the bank:

- Stem.SoundBank.AddSound
- Stem.SoundBank.AddSoundBus
- Stem.MusicBank.AddPlaylist
- Stem.MusicBank.AddMusicPlayer

```
AudioClip clip;
AudioClip[] clips;

Stem.Sound soundEmpty = bank.AddSound("Sound name");
Stem.Sound soundSingle = bank.AddSound("Sound name", clip);
Stem.Sound soundWithVariations = bank.AddSound("Sound name", clips);
```

```
Stem.SoundBus newBus = bank.AddSoundBus("Sound bus name");
```

```
AudioClip track;
AudioClip[] tracks;

Stem.Playlist playlistEmpty = bank.AddPlaylist("Playlist name");
Stem.Playlist playlistSingleTrack = bank.AddPlaylist("Playlist name", track);
Stem.Playlist playlistMultipleTracks = bank.AddPlaylist("Playlist name", tracks);
```

```
Stem.MusicPlayer newPlayer = bank.AddMusicPlayer("Music player name");
```

Global search in all banks

It's recommended to use IDs instead of string references by default. However, string references might be useful in case there're multiple banks with same content names but different audio data. In this case, referencing by string allows using multiple banks as audio skins by changing PrimaryBank property in the corresponding manager.

Primary bank

Use Stem.SoundManager.PrimaryBank or Stem.MusicManager.PrimaryBank property to set a primary bank to the corresponding manager. Primary banks are used to resolve name collisions during the search.

This might be useful in case there're multiple banks with similar content but different audio data. Think of it as audio skin.

```
Stem.SoundManager.PrimaryBank = bank;
```

```
Stem.MusicManager.PrimaryBank = bank;
```

By name

Call these methods to search for existing content in all banks:

- Stem.SoundManager.GetSound

- Stem.SoundManager.GetSoundBus
- Stem.MusicManager.GetPlaylist
- Stem.MusicManager.GetMusicPlayer

In the case of name collisions, if multiple banks have content with the matching name, the PrimaryBank property will be checked first. Within a bank, the first occurrence of found content will be used.

```
Stem.Sound sound = SoundManager.GetSound("Sound name");
Stem.SoundBus soundBus = SoundManager.GetSoundBus("Sound bus name");
```

```
Stem.Playlist playlist = MusicManager.GetPlaylist("Playlist name");
Stem.MusicPlayer musicPlayer = MusicManager.GetMusicPlayer("Music player name");
```

By ID

Call these methods to search for existing content in all banks:

- Stem.SoundManager.GetSound
- Stem.SoundManager.GetSoundBus
- Stem.MusicManager.GetPlaylist
- Stem.MusicManager.GetMusicPlayer

As all IDs are unique, the PrimaryBank property has no effect on the search results. Use ID attributes to get the ID from the Inspector.

```
Stem.ID soundId; // Assume it's filled earlier in the code or was set by the Stem.SoundID attribute
Stem.ID soundBusId; // Assume it's filled earlier in the code or was set by the Stem.SoundBusID attribute

Stem.Sound sound = SoundManager.GetSound(soundId);
Stem.SoundBus soundBus = SoundManager.GetSoundBus(soundBusId);
```

```
Stem.ID playlistId; // Assume it's filled earlier in the code or was set by the Stem.PlaylistID attribute
Stem.ID musicPlayerId; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute

Stem.Playlist playlist = MusicManager.GetPlaylist(playlistId);
Stem.MusicPlayer musicPlayer = MusicManager.GetMusicPlayer(musicPlayerId);
```

Local search in specific bank

By ID

Call these methods to search for existing content in the bank:

- Stem.SoundBank.GetSound
- Stem.SoundBank.GetSoundBus
- Stem.MusicBank.GetPlaylist
- Stem.MusicBank.GetMusicPlayer

Use ID attributes to get the ID from the Inspector.

```
Stem.ID soundId; // Assume it's filled earlier in the code or was set by the Stem.SoundID attribute
Stem.ID soundBusId; // Assume it's filled earlier in the code or was set by the Stem.SoundBusID attribute

Stem.Sound sound = bank.GetSound(soundId);
Stem.SoundBus soundBus = bank.GetSoundBus(soundBusId);
Stem.ID playlistId; // Assume it's filled earlier in the code or was set by the Stem.PlaylistID attribute
Stem.ID musicPlayerId; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute

Stem.Playlist playlist = bank.GetPlaylist(playlistId);
Stem.MusicPlayer musicPlayer = bank.GetMusicPlayer(musicPlayerId);
```

By name

Call these methods to search for existing content in the bank:

- Stem.SoundBank.GetSound
- Stem.SoundBank.GetSoundBus
- Stem.MusicBank.GetPlaylist
- Stem.MusicBank.GetMusicPlayer

In the case of name collisions, the first occurrence of found content will be used.

```
Stem.Sound sound = bank.GetSound("Sound name");
Stem.SoundBus soundBus = bank.GetSoundBus("Sound bus name");
```

```
Stem.Playlist playlist = bank.GetPlaylist("Playlist name");
Stem.MusicPlayer musicPlayer = bank.GetMusicPlayer("Music player name");
```

Removing content

Call these methods to remove existing content from the bank:

- Stem.SoundBank.RemoveSound
- Stem.SoundBank.RemoveSoundBus
- Stem.MusicBank.RemovePlaylist
- Stem.MusicBank.RemoveMusicPlayer

Note that it's not possible to remove all sound buses from a sound bank.

```
Stem.Sound sound; // Assume it's filled earlier in the code
Stem.SoundBus soundBus; // Assume it's filled earlier in the code

bank.RemoveSound(sound);
bank.RemoveSoundBus(soundBus);
```

```
Stem.Playlist playlist; // Assume it's filled earlier in the code
Stem.MusicPlayer musicPlayer; // Assume it's filled earlier in the code

bank.RemovePlaylist(playlist);
bank.RemoveMusicPlayer(musicPlayer);
```

Bank Management

Description

By default, Stem automatically manages all banks assets. In case of a manually created bank, it's required to register it in the corresponding manager for proper use.

Creating banks at runtime

As banks are inherited from ScriptableObject, use ScriptableObject.CreateInstance to create a bank instance. Note that there'll be no .asset file created in the project.

```
Stem.SoundBank bank = ScriptableObject.CreateInstance<Stem.SoundBank>();
// add bank content here
```

```
Stem.MusicBank bank = ScriptableObject.CreateInstance<Stem.MusicBank>();
// add bank content here
```

Registration

Call Stem.SoundManager.RegisterBank or Stem.MusicManager.RegisterBank to register a bank in the corresponding manager. If the return value is true, the bank was successfully registered and is ready to use.

Stem will automatically create all required runtime data like game objects, audio sources, etc.

```
if (Stem.SoundManager.RegisterBank(bank))
{
    // it's ready to use
}
```

```
if (Stem.MusicManager.RegisterBank(bank))
{
    // it's ready to use
}
```

Bank collection

Use Stem.SoundManager.Banks or Stem.MusicManager.Banks property to get a read-only collection of registered banks.

```
int totalSounds = 0;
int totalSoundBuses = 0;
foreach (Stem.SoundBank bank in Stem.SoundManager.Banks)
{
    totalSounds += bank.Sounds.Count;
    totalSoundBuses += bank.Buses.Count;
}
```

```
int totalPlaylists = 0;
int totalMusicPlayers = 0;
foreach (Stem.MusicBank bank in Stem.MusicManager.Banks)
{
    totalPlaylists += bank.Playlists.Count;
    totalMusicPlayers += bank.Players.Count;
}
```

Deregistration

Call Stem.SoundManager.DeregisterBank or Stem.MusicManager.DeregisterBank to deregister a bank in the corresponding manager so it won't be in the search.

Stem will automatically destroy all bank's runtime data like game objects, audio sources, etc.

```
Stem.SoundManager.DeregisterBank(bank);
```

```
Stem.MusicManager.DeregisterBank(bank);
```

ID Attributes

Description

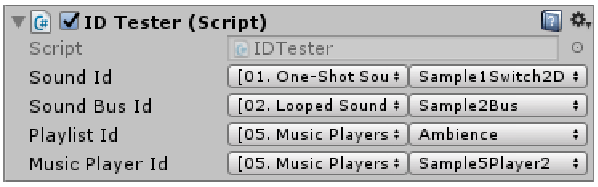
All content in Stem has unique and persistent identifiers. That means once the content is created, it's ID is fixed, thus allowing to reference it. There are four attributes for each type of content (sound, sound bus, playlist, music player) which help to easily assign content ID in the Inspector:

- SoundID
- SoundBusID
- PlaylistID
- MusicPlayerID

It's recommended to use IDs instead of string references by default. However, string references might be useful in case there're multiple banks with same content names but different audio data. In this case, referencing by string allows using multiple banks as audio skins by changing PrimaryBank property in the corresponding manager.

Usage

Inspector



ID Tester (Script)

Script	IDTester	
Sound Id	[01. One-Shot Sou	Sample1Switch2D
Sound Bus Id	[02. Looped Sound	Sample2Bus
Playlist Id	[05. Music Players	Ambience
Music Player Id	[05. Music Players	Sample5Player2

Code

```csharp
using UnityEngine;

public class IDTester : MonoBehaviour
{
    [Stem.SoundID]
    public Stem.ID soundId = Stem.ID.None;

    [Stem.SoundBusID]
    public Stem.ID soundBusId = Stem.ID.None;

    [Stem.PlaylistID]
    public Stem.ID playlistId = Stem.ID.None;

    [Stem.MusicPlayerID]
    public Stem.ID musicPlayerId = Stem.ID.None;

    private void Start()
    {
        Stem.Sound sound = Stem.SoundManager.GetSound(soundId);
        if (sound != null)
            Debug.LogFormat("Found sound: {0}", sound.Name);

        Stem.SoundBus soundBus = Stem.SoundManager.GetSoundBus(soundBusId);
        if (soundBus != null)
            Debug.LogFormat("Found sound bus: {0}", soundBus.Name);

        Stem.Playlist playlist = Stem.MusicManager.GetPlaylist(playlistId);
        if (playlist != null)
            Debug.LogFormat("Found playlist: {0}", playlist.Name);

        Stem.MusicPlayer musicPlayer = Stem.MusicManager.GetMusicPlayer(musicPlayerId);
        if (musicPlayer != null)
            Debug.LogFormat("Found music player: {0}", musicPlayer.Name);
    }
}
```

Music Callbacks

Description

Music callbacks allow handling music events such as track or playback state changes. It's possible to subscribe to a specific music player as well as handle all music events in a global callback.

Global callbacks

Subscribe to these events to set a global music callback:

- Stem.MusicManager.OnPlaybackStarted
- Stem.MusicManager.OnPlaybackStopped
- Stem.MusicManager.OnPlaybackPaused
- Stem.MusicManager.OnTrackChanged

```
using UnityEngine;

public class MusicManagerCallbacks : MonoBehaviour
{
    private void Start()
    {
        Stem.MusicManager.OnPlaybackStarted += OnPlaybackStarted;
        Stem.MusicManager.OnPlaybackStopped += OnPlaybackStopped;
        Stem.MusicManager.OnPlaybackPaused += OnPlaybackPaused;
        Stem.MusicManager.OnTrackChanged += OnTrackChanged;
    }

    private void OnDestroy()
    {
        Stem.MusicManager.OnPlaybackStarted -= OnPlaybackStarted;
        Stem.MusicManager.OnPlaybackStopped -= OnPlaybackStopped;
        Stem.MusicManager.OnPlaybackPaused -= OnPlaybackPaused;
        Stem.MusicManager.OnTrackChanged -= OnTrackChanged;
    }

    private void OnPlaybackStarted(Stem.MusicPlayer player)
    {
        Debug.LogFormat("[Global Callback] {0}: playback started", player.Name);
    }

    private void OnPlaybackStopped(Stem.MusicPlayer player)
    {
        Debug.LogFormat("[Global Callback] {0}: playback stopped", player.Name);
    }

    private void OnPlaybackPaused(Stem.MusicPlayer player)
    {
        Debug.LogFormat("[Global Callback] {0}: playback paused", player.Name);
    }

    private void OnTrackChanged(Stem.MusicPlayer player, Stem.PlaylistTrack track)
    {
        Debug.LogFormat("[Global Callback] {0}: track changed to {1}", player.Name, (track != null) ?
track.Name : "none");
    }
}
```

Local callbacks

Subscribe to these events to set a local music callback:

- Stem.MusicPlayer.OnPlaybackStarted
- Stem.MusicPlayer.OnPlaybackStopped
- Stem.MusicPlayer.OnPlaybackPaused
- Stem.MusicPlayer.OnTrackChanged

Note that you need to get a Stem.MusicPlayer instance to use those events.

```
using UnityEngine;

public class MusicPlayerCallbacks : MonoBehaviour
{
    [Stem.MusicPlayerID]
    public Stem.ID id = Stem.ID.None;

    private Stem.MusicPlayer cachedPlayer;

    private void Start()
    {
        cachedPlayer = Stem.MusicManager.GetMusicPlayer(id);
        if (cachedPlayer != null)
        {
            cachedPlayer.OnPlaybackStarted += OnPlaybackStarted;
            cachedPlayer.OnPlaybackStopped += OnPlaybackStopped;
            cachedPlayer.OnPlaybackPaused += OnPlaybackPaused;
            cachedPlayer.OnTrackChanged += OnTrackChanged;
        }
    }

    private void OnDestroy()
    {
        if (cachedPlayer != null)
        {
            cachedPlayer.OnPlaybackStarted -= OnPlaybackStarted;
            cachedPlayer.OnPlaybackStopped -= OnPlaybackStopped;
            cachedPlayer.OnPlaybackPaused -= OnPlaybackPaused;
            cachedPlayer.OnTrackChanged -= OnTrackChanged;
        }
    }

    private void OnPlaybackStarted(Stem.MusicPlayer player)
    {
        Debug.LogFormat("[Local Callback] {0}: playback started", player.Name);
    }

    private void OnPlaybackStopped(Stem.MusicPlayer player)
    {
        Debug.LogFormat("[Local Callback] {0}: playback stopped", player.Name);
    }

    private void OnPlaybackPaused(Stem.MusicPlayer player)
    {
        Debug.LogFormat("[Local Callback] {0}: playback paused", player.Name);
    }

    private void OnTrackChanged(Stem.MusicPlayer player, Stem.PlaylistTrack track)
    {
        Debug.LogFormat("[Local Callback] {0}: track changed to {1}", player.Name, (track != null) ?
track.Name : "none");
    }
}
```

Music Manager

API Reference

Description

Music Manager is the entry point for music playback and music bank management. It does not require to create any additional game objects in order to use it.

Playlist controls

Call Stem.MusicManager.SetPlaylist to assign a playlist to a music player. Once assigned, it'll automatically start playing.

Use default crossfade duration from the music player:

```
Stem.ID musicPlayerID; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute
Stem.ID playlistID; // Assume it's filled earlier in the code or was set by the Stem.PlaylistID attribute

Stem.MusicManager.SetPlaylist(musicPlayerID, playlistID);
```

Or override it:

```
Stem.ID musicPlayerID; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute
Stem.ID playlistID; // Assume it's filled earlier in the code or was set by the Stem.PlaylistID attribute
float newCrossfade = 5.0f;

Stem.MusicManager.SetPlaylist(musicPlayerID, playlistID, newCrossfade);
```

Track controls

Call these methods to set a desired track to a music player:

- Stem.MusicManager.Next
- Stem.MusicManager.Prev
- Stem.MusicManager.Seek
- Stem.MusicManager.Seek

Use default crossfade duration from the music player:

```
Stem.ID musicPlayerID; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute

// Will advance music player to next track
Stem.MusicManager.Next(musicPlayerID);

// Will advance music player to previous track
Stem.MusicManager.Prev(musicPlayerID);

// Will advance music player to a target track by name
Stem.MusicManager.Seek(musicPlayerID, "Level1 Theme");

// Will advance music player to a target track by playlist index
Stem.MusicManager.Seek(musicPlayerID, 0);
```

Or override it:

```
Stem.ID musicPlayerID; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute
float newCrossfade = 5.0f;

Stem.MusicManager.Next(musicPlayerID, newCrossfade);
Stem.MusicManager.Prev(musicPlayerID, newCrossfade);
Stem.MusicManager.Seek(musicPlayerID, "Level1 Theme", newCrossfade);
Stem.MusicManager.Seek(musicPlayerID, 0, newCrossfade);
```

Playback controls

Call these methods to change playback state of a music player:

- Stem.MusicManager.Play
- Stem.MusicManager.Stop
- Stem.MusicManager.Pause

Use default crossfade duration from the music player:

```
Stem.ID musicPlayerID; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute

// Will start playing current track of the music player.
Stem.MusicManager.Play(musicPlayerID);

// Will pause playing current track of the music player.
Stem.MusicManager.Pause(musicPlayerID);

// Will resume playing current track of the music player.
Stem.MusicManager.Play(musicPlayerID);

// Will stop playing current track of the music player.
Stem.MusicManager.Stop(musicPlayerID);
```

Or override it:

```
Stem.ID musicPlayerID; // Assume it's filled earlier in the code or was set by the Stem.MusicPlayerID
attribute
float newCrossfade = 5.0f;

Stem.MusicManager.Play(musicPlayerID, newCrossfade);
Stem.MusicManager.Pause(musicPlayerID, newCrossfade);
Stem.MusicManager.Play(musicPlayerID, newCrossfade);
Stem.MusicManager.Stop(musicPlayerID, newCrossfade);
```

Sound Instance

API Reference

Description

Sound Instance is a single playing audio source. Most sound instances are managed by the Sound Manager and should not be used directly.

However, this class is useful for custom mixing logic and manual playback. There are four main use cases:

1. Playing looped sounds
2. Changing sounds at runtime
3. Changing volume and pitch at runtime
4. Attaching to another game object

Playing looped sounds

Use the Stem.SoundInstance.Looped flag to play looped sound.

```
using UnityEngine;

public class TVNoise : MonoBehaviour
{
    [Stem.SoundID]
    public Stem.ID soundID = Stem.ID.None;

    private Stem.SoundInstance soundInstance = null;

    private void OnEnable()
    {
        if (soundInstance != null)
            return;

        // Will do a lookup for sound with the mathcing ID in all sound banks
        // and return a reference to a sound instance from the sound pool.
        soundInstance = Stem.SoundManager.GrabSound(soundID);

        if (soundInstance != null)
        {
            // Play looped
            soundInstance.Looped = true;
            soundInstance.Play();
        }
    }

    private void OnDisable()
    {
        // Return the instance to the sound pool
        Stem.SoundManager.ReleaseSound(soundInstance);
        soundInstance = null;
    }
}
```

Changing sounds at runtime

Use the Stem.SoundInstance.Sound property to play different sounds.

```
using UnityEngine;
```

```
public class Draggable : MonoBehaviour
{
    [Stem.SoundID]
    public Stem.ID dragStartSoundID = Stem.ID.None;

    [Stem.SoundID]
    public Stem.ID dragEndSoundID = Stem.ID.None;

    [Stem.SoundID]
    public Stem.ID dragLoopSoundID = Stem.ID.None;

    private Stem.SoundInstance soundInstance = null;
    private Stem.Sound dragStart = null;
    private Stem.Sound dragEnd = null;
    private Stem.Sound dragLoop = null;
    private bool dragging = false;

    private void Awake()
    {
        // Grab an empty sound instance from the sound pool.
        soundInstance = Stem.SoundManager.GrabSound();

        dragStart = Stem.SoundManager.GetSound(dragStartSoundID);
        dragEnd = Stem.SoundManager.GetSound(dragEndSoundID);
        dragLoop = Stem.SoundManager.GetSound(dragLoopSoundID);
    }

    private void OnDestroy()
    {
        // Return the instance to the sound pool
        Stem.SoundManager.ReleaseSound(soundInstance);
        soundInstance = null;
    }

    private void Update()
    {
        if (dragging && soundInstance.Sound == dragStart && !soundInstance.Playing)
        {
            // Play looped
            soundInstance.Sound = dragLoop;
            soundInstance.Looped = true;
            soundInstance.Play();
        }
    }

    public void StartDrag()
    {
        // Play one-shot
        soundInstance.Sound = dragStart;
        soundInstance.Looped = false;
        soundInstance.Play();

        dragging = true;
    }

    public void EndDrag()
    {
        // Play one-shot
        soundInstance.Sound = dragEnd;
        soundInstance.Looped = false;
        soundInstance.Play();

        dragging = false;
    }
}
```

Changing volume and pitch at runtime

Use Stem.SoundInstance.Volume and Stem.SoundInstance.Pitch properties to change parameters during playback.

```
using UnityEngine;

public class Siren : MonoBehaviour
{
    [Stem.SoundID]
    public Stem.ID sirenSoundID = Stem.ID.None;

    public float frequency = 440.0f;

    public float baseVolume = 0.8f;
    public float additionalVolume = 0.2f;

    public float basePitch = 1.0f;
    public float additionalPitch = 0.2f;

    private Stem.SoundInstance soundInstance = null;

    private void Awake()
    {
        // Will do a lookup for sound with the matching ID in all sound banks
        // and return a reference to a sound instance from the sound pool.
        soundInstance = Stem.SoundManager.GrabSound(sirenSoundID);

        if (soundInstance != null)
        {
            // Play looped
            soundInstance.Looped = true;
            soundInstance.Play();
        }
    }

    private void OnDestroy()
    {
        // Return the instance to the sound pool
        Stem.SoundManager.ReleaseSound(soundInstance);
        soundInstance = null;
    }

    private void Update()
    {
        // Calculate current sine wave sample
        float timeNow = Time.realtimeSinceStartup;
        float sample = Mathf.Sin(timeNow * frequency);

        // Modulate pitch and volume
        soundInstance.Volume = baseVolume + additionalVolume * sample;
        soundInstance.Pitch = basePitch + additionalPitch * sample;
    }
}
```

Attaching to another game object

Use the Stem.SoundInstance.Target property to attach sound instance to another game object.

```
using UnityEngine;

public class Rocket : MonoBehaviour
{
    [Stem.SoundID]
    public Stem.ID engineSoundID = Stem.ID.None;

    private Stem.SoundInstance soundInstance = null;

    private void Awake()
    {
        // Will do a lookup for sound with the matching ID in all sound banks
        // and return a reference to a sound instance from the sound pool.
        soundInstance = Stem.SoundManager.GrabSound(engineSoundID);

        if (soundInstance != null)
        {
            // Play looped
            soundInstance.Looped = true;
            soundInstance.Play();

            // Attach to the game object
            soundInstance.Target = transform;
        }
    }

    private void OnDestroy()
    {
        // Return the instance to the sound pool
        Stem.SoundManager.ReleaseSound(soundInstance);
        soundInstance = null;
    }
}
```

Sound Manager

API Reference

Description

Sound Manager is the entry point for sound playback and sound instance management. It does not require to create any additional game objects in order to use it.

All currently playing sounds are represented by the Sound Instance class. There are two types of them:

1. One-shot sound instances — they are played and automatically managed by the sound manager. When playing a sound, Sound Manager will look up for available sound instance and use it for playback.
2. Manual sound instances — the sound manager allows getting a reference to a sound instance from the sound pool for manual playback.

One-shot sound instances

Call these methods to play a one-shot sound:

- Stem.SoundManager.Play
- Stem.SoundManager.Play3D

Use default volume, pitch and delay values from the sound:

```
Stem.ID soundId; // Assume it's filled earlier in the code or was set by the Stem.SoundID attribute
Vector3 soundPosition = new Vector3(100.0f, 0.0f, 0.0f);

Stem.SoundManager.Play(soundID);
Stem.SoundManager.Play3D(soundID, soundPosition);
```

Or override them:

```
Stem.ID soundId; // Assume it's filled earlier in the code or was set by the Stem.SoundID attribute
Vector3 soundPosition = new Vector3(100.0f, 0.0f, 0.0f);
float newVolume = 0.5f;
float newPitch = 0.9f;
float newDelay = 0.1f;

Stem.SoundManager.Play(soundID, newVolume, newPitch, newDelay);
Stem.SoundManager.Play3D(soundID, soundPosition, newVolume, newPitch, newDelay);
```

Manual sound instances

Call Stem.SoundManager.GrabSound to get a reference to a sound instance from the sound pool:

```
Stem.ID soundId; // Assume it's filled earlier in the code or was set by the Stem.SoundID attribute

Stem.SoundInstance soundInstance = Stem.SoundManager.GrabSound(soundID);
```

Call Stem.SoundInstance.Play or Stem.SoundInstance.Play3D to manuall play a sound. Use default volume, pitch and delay values from the sound:

```
Vector3 soundPosition = new Vector3(100.0f, 0.0f, 0.0f);
soundInstance.Play();
soundInstance.Play3D(soundPosition);
```

Or override them:

```
Vector3 soundPosition = new Vector3(100.0f, 0.0f, 0.0f);
float newVolume = 0.5f;
float newPitch = 0.9f;
float newDelay = 0.1f;

soundInstance.Play(newVolume, newPitch, newDelay);
soundInstance.Play3D(soundPosition, newVolume, newPitch, newDelay);
```

Once the sound instance is not needed anymore, call Stem.SoundManager.ReleaseSound to return it back to the sound pool:

```
Stem.SoundManager.ReleaseSound(soundInstance);
```

Playback controls

Call these methods to change playback state of all sound instances:

- Stem.SoundManager.Pause
- Stem.SoundManager.UnPause
- Stem.SoundManager.Stop

```
// Stop all playing sound instances (including manual sound instances)
Stem.SoundManager.Pause();

// Resume all playing sound instances (including manual sound instances)
Stem.SoundManager.UnPause();

// Stop all playing sound instances (including manual sound instances)
Stem.SoundManager.Stop();
```

Namespace Stem

Classes

MusicBank

The persistent storage for playlists and music players.

MusicManager

The main class for music playback and bank management.

MusicPlayer

The persistent storage for playback rules of a particular playlist.

MusicPlayerIDAttribute

The attribute class used to make an int variable in a script be restricted to a music player id.

Playlist

The persistent collection of playlist tracks.

PlaylistIDAttribute

The attribute class used to make an int variable in a script be restricted to a playlist id.

PlaylistTrack

The persistent storage for music audio data.

Sound

The persistent storage for sound variations and the most important audio source settings.

SoundBank

The persistent storage for sounds and sound buses.

SoundBus

The persistent storage for playback rules of a group of sounds.

SoundBusIDAttribute

The attribute class used to make an int variable in a script be restricted to a sound bus id.

SoundBusRuntime

SoundIDAttribute

The attribute class used to make an int variable in a script be restricted to a sound id.

SoundInstance

The game object with audio source component. Used for manual playback and custom mixing logic.

SoundManager

The main class for sound playback and bank management.

SoundVariation

The persistent storage for sound effect audio data.

Structs

ID

The universal unique identifier used to reference bank content.

Interfaces

IAudioClipContainer

The container interface used by memory manager for audio clip management.

IBank

The common bank interface for runtime state management.

Enums

AttenuationMode

Defines how sound volume will be lowered over the distance.

AudioClipImportMode

Defines how new audio content will be created after the drag-drop event. Provided audio clips will be used as sound variations or playlist tracks.

AudioClipManagementMode

Defines how audio clips will be managed in memory.

MusicPlayerPlaybackMode

Defines how music player plays its tracks.

RetriggerMode

Defines how sound will play its variations.

Delegates

MusicPlayerAddedDelegate

A music bank callback function, called after adding music player to the bank.

MusicPlayerRemovedDelegate

A music bank callback function, called before removing the music player from the bank.

MusicPlayerRenamedDelegate

A music bank callback function, called after changing music player name.

PlaybackChangedDelegate

A music callback function, called when the music player changes playback state (playing, stopped, paused).

PlaylistAddedDelegate

A music bank callback function, called after adding playlist to the bank.

PlaylistRemovedDelegate

A music bank callback function, called before removing the playlist from the bank.

PlaylistRenamedDelegate

A music bank callback function, called after changing playlist name.

SoundAddedDelegate

A sound bank callback function, called after adding sound to the bank.

SoundBusAddedDelegate

A sound bank callback function, called after adding sound bus to the bank.

SoundBusRemovedDelegate

A sound bank callback function, called before removing the sound bus from the bank.

SoundBusRenamedDelegate

A sound bank callback function, called after changing sound bus name.

SoundRemovedDelegate

A sound bank callback function, called before removing the sound from the bank.

SoundRenamedDelegate

A sound bank callback function, called after changing sound name.

TrackChangedDelegate

A music callback function, called when the music player transitions to a new track.

Enum AttenuationMode

Defines how sound volume will be lowered over the distance.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public enum AttenuationMode
```

Fields

NAME	DESCRIPTION
Linear	A linear rolloff.
Logarithmic	A real-world rolloff.

Enum AudioClipImportMode

Defines how new audio content will be created after the drag-drop event. Provided audio clips will be used as sound variations or playlist tracks.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public enum AudioClipImportMode
```

Fields

NAME	DESCRIPTION
MultipleItemsWithSingleClip	Create multiple items with a single audio clips.
SingleItemWithAllClips	Create a single item and put all audio clips to it.

Enum AudioClipManagementMode

Defines how audio clips will be managed in memory.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public enum AudioClipManagementMode
```

Fields

NAME	DESCRIPTION
Manual	Do not manage audio clip data and instead allow the developer to take control.
PreloadAndKeepInMemory	Preload audio clip data during startup and keep it in memory.
UnloadUnused	Unload audio clip data if it was not used for some time.

Interface IAudioClipContainer

The container interface used by memory manager for audio clip management.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public interface IAudioClipContainer
```

Methods

GetAudioClip(Int32)

Gets the audio clip at the specified index.

Declaration

```
AudioClip GetAudioClip(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	The zero-based index of the audio clip to get.

Returns

TYPE	DESCRIPTION
AudioClip	A reference to an audio clip.

GetAudioClipManagementMode()

Gets the audio clip management mode of the container.

Declaration

```
AudioClipManagementMode GetAudioClipManagementMode()
```

Returns

TYPE	DESCRIPTION
AudioClipManagementMode	An enum value.

GetAudioClipUnloadInterval()

Gets the audio clip unload interval of the container.

Declaration

```
float GetAudioClipUnloadInterval()
```

Returns

TYPE	DESCRIPTION
System.Single	The time interval in seconds.

Remarks

This value is only used if GetAudioClipManagementMode() return value is UnloadUnused

GetNumAudioClips()

Gets the number of audio clips in the container.

Declaration

```
int GetNumAudioClips()
```

Returns

TYPE	DESCRIPTION
System.Int32	The number of audio clips.

Interface IBank

The common bank interface for runtime state management.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public interface IBank
```

Methods

GetBankID()

Returns the bank ID.

Declaration

```
ID GetBankID()
```

Returns

TYPE	DESCRIPTION
ID	An ID value.

RegenerateBankID()

Generates a new unique ID for the bank.

Declaration

```
void RegenerateBankID()
```

Remarks

This method is automatically called by Stem during serialization process. Don't call it manually as it may break existing ID references.

Struct ID

The universal unique identifier used to reference bank content.

Implements

System.IEquatable<ID>

Inherited Members

System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetType()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public struct ID : IEquatable<ID>
```

Constructors

ID(Int32, Int32, Int32, Int32, Int32)

Creates a new ID based on bank and sound/sound bus/playlist/music player identifiers.

Declaration

```
public ID(int guidA, int guidB, int guidC, int guidD, int id)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	guidA	The first part of a bank identifier.
System.Int32	guidB	The second part of a bank identifier.
System.Int32	guidC	The third part of a bank identifier.
System.Int32	guidD	The fourth part of a bank identifier.
System.Int32	id	Unique identifier of a sound, sound bus, playlist or music player.

Fields

None

The shorthand for writing ID(0, 0, 0, 0, 0) that does not refer to anything.

Declaration

```
public static readonly ID None
```

Field Value

TYPE	DESCRIPTION
ID	A read-only ID.

Properties

BankGuidA

The first part of a bank this ID refers to.

Declaration

```
public readonly int BankGuidA { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	An integer value.

BankGuidB

The second part of a bank this ID refers to.

Declaration

```
public readonly int BankGuidB { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	An integer value.

BankGuidC

The third part of a bank this ID refers to.

Declaration

```
public readonly int BankGuidC { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	An integer value.

BankGuidD

The fourth part of a bank this ID refers to.

Declaration

```
public readonly int BankGuidD { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	An integer value.

ItemId

The sound, sound bus, playlist or music player this ID refers to.

Declaration

```
public readonly int ItemId { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	An integer value.

Remarks

This value corresponds to Sound, SoundBus, Playlist or MusicPlayer ID.

Methods

BankEquals(ID)

Checks if two IDs are referencing to the same bank.

Declaration

```
public bool BankEquals(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	An ID to compare.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if both IDs reference to the same bank. False otherwise.

Equals(ID)

Checks if two IDs are equal.

Declaration

```
public bool Equals(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	An ID to compare.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if both IDs are equal. False otherwise.

Equals(Object)

Checks if two object instances are equal.

Declaration

```
public override bool Equals(object obj)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Object	obj	A reference to an object.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if `obj` is an ID and both IDs are equal. False otherwise.

Overrides

System.ValueType.Equals(System.Object)

GetHashCode()

Calculates the hash of an ID.

Declaration

```
public override int GetHashCode()
```

Returns

TYPE	DESCRIPTION
System.Int32	A hash code for the ID.

Overrides

System.ValueType.GetHashCode()

ToString()

Returns a string that represents the ID.

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
System.String	A string representation of an ID.

Overrides

System.ValueType.ToString()

Operators

Equality(ID, ID)

Checks if two IDs are equal.

Declaration

```
public static bool operator ==(ID id1, ID id2)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id1	The first ID to compare.
ID	id2	The second ID to compare.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if both IDs are equal. False otherwise.

Inequality(ID, ID)

Checks if two IDs are not equal.

Declaration

```
public static bool operator !=(ID id1, ID id2)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id1	The first ID to compare.
ID	id2	The second ID to compare.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if both IDs are not equal. False otherwise.

Implements

System.IEquatable<T>

Class MusicBank

The persistent storage for playlists and music players.

Inheritance

System.Object
MusicBank

Implements

IBank
ISerializationCallbackReceiver

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class MusicBank : ScriptableObject, IBank, ISerializationCallbackReceiver
```

Properties

Players

The collection of music players.

Declaration

```
public ReadOnlyCollection<MusicPlayer> Players { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<MusicPlayer>	A reference to a read-only collection of music players.

PlaylistBatchImportMode

The batch import mode defining how new playlists will be created after the drag-drop event.

Declaration

```
public AudioClipImportMode PlaylistBatchImportMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClipImportMode	An enum value.

PlaylistManagementMode

The default audio clip management mode for all playlists of the music bank.

Declaration

```
public AudioClipManagementMode PlaylistManagementMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClipManagementMode	An enum value.

Remarks

By default, all playlists will use this value for audio clip management, however, it can be overridden by the OverrideAudioClipManagement flag.

Playlists

The collection of playlists.

Declaration

```
public ReadOnlyCollection<Playlist> Playlists { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<Playlist>	A reference to a read-only collection of playlists.

PlaylistUnloadInterval

The default audio clip unload interval for all playlists of the the music bank.

Declaration

```
public float PlaylistUnloadInterval { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	A time interval in seconds.

Remarks

By default, all playlists will use this value for audio clip unload interval, however, it can be overridden by the OverrideAudioClipManagement flag.

ShowPlayers

The flag indicating whether the music bank inspector should show the 'Players' group.

Declaration

```
public bool ShowPlayers { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the 'Players' group is shown. False otherwise.

Remarks

This property is used only by the music bank inspector and does nothing during runtime.

ShowPlaylists

The flag indicating whether the music bank inspector should show the 'Playlists' group.

Declaration

```
public bool ShowPlaylists { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the 'Playlists' group is shown. False otherwise.

Remarks

This property is used only by the music bank inspector and does nothing during runtime.

Methods

AddMusicPlayer(String)

Adds a new music player to the music bank.

Declaration

```
public MusicPlayer AddMusicPlayer(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the music player.

Returns

TYPE	DESCRIPTION
MusicPlayer	A reference to a newly created music player.

AddPlaylist(String)

Adds an empty playlist to the music bank.

Declaration

```
public Playlist AddPlaylist(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the playlist.

Returns

TYPE	DESCRIPTION
Playlist	A reference to a newly created playlist.

AddPlaylist(String, AudioClip)

Adds a new playlist with a single track to the music bank.

Declaration

```
public Playlist AddPlaylist(string name, AudioClip clip)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the playlist.
AudioClip	clip	A reference to the audio clip with music data.

Returns

TYPE	DESCRIPTION
Playlist	A reference to a newly created playlist.

AddPlaylist(String, AudioClip[])

Adds a new playlist with multiple tracks to the music bank.

Declaration

```
public Playlist AddPlaylist(string name, AudioClip[] clips)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the playlist.
AudioClip[]	clips	An array of audio clips with music data.

Returns

TYPE	DESCRIPTION
Playlist	A reference to a newly created playlist.

GetBankID()

Returns music bank ID.

Declaration

```
public ID GetBankID()
```

Returns

TYPE	DESCRIPTION
ID	An ID value.

GetMusicPlayer(ID)

Searches for the specified music player by ID.

Declaration

```
public MusicPlayer GetMusicPlayer(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID that refers to a music player.

Returns

TYPE	DESCRIPTION
MusicPlayer	A reference to a music player, if found. Null reference otherwise.

GetMusicPlayer(String)

Searches for the specified music player with a matching name.

Declaration

```
public MusicPlayer GetMusicPlayer(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the music player.

Returns

TYPE	DESCRIPTION
MusicPlayer	A reference to a music player, if found. Null reference otherwise.

GetMusicPlayerID(Int32)

Gets an ID to the specific music player.

Declaration

```
public ID GetMusicPlayerID(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	Zero-based index of the music player in the current music bank.

Returns

TYPE	DESCRIPTION
ID	An ID to the specific music player.

GetPlaylist(ID)

Searches for the specified playlist by ID.

Declaration

```
public Playlist GetPlaylist(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID that refers to a playlist.

Returns

TYPE	DESCRIPTION
Playlist	A reference to a playlist, if found. Null reference otherwise.

GetPlaylist(String)

Searches for the specified playlist with a matching name.

Declaration

```
public Playlist GetPlaylist(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the playlist.

Returns

TYPE	DESCRIPTION
Playlist	A reference to a playlist, if found. Null reference otherwise.

GetPlaylistID(Int32)

Gets an ID to the specific playlist.

Declaration

```
public ID GetPlaylistID(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	Zero-based index of the playlist in the current music bank.

Returns

TYPE	DESCRIPTION
ID	An ID to the specific playlist.

OnAfterDeserialize()

Prepares music bank for runtime use after deserialization.

Declaration

```
public void OnAfterDeserialize()
```

Remarks

This method is automatically called by Unity during deserialization process. Don't call it manually.

OnBeforeSerialize()

Prepares music bank for serialization.

Declaration

```
public void OnBeforeSerialize()
```

Remarks

This method is automatically called by Unity during serialization process. Don't call it manually.

RegenerateBankID()

Generates a new unique ID for the music bank.

Declaration

```
public void RegenerateBankID()
```

Remarks

This method is automatically called by Stem during serialization process. Don't call it manually as it may break existing ID

This method is automatically called by Stem during serialization process. Don't call it manually as it may break existing ID references.

RemoveMusicPlayer(MusicPlayer)

Removes existing music player from the music bank.

Declaration

```
public void RemoveMusicPlayer(MusicPlayer player)
```

Parameters

TYPE	NAME	DESCRIPTION
MusicPlayer	player	A reference to a music player.

Remarks

This method does nothing if the music player was not found in the music bank.

RemovePlaylist(Playlist)

Removes existing playlist from the music bank.

Declaration

```
public void RemovePlaylist(Playlist playlist)
```

Parameters

TYPE	NAME	DESCRIPTION
Playlist	playlist	A reference to a playlist.

Remarks

This method does nothing if the playlist was not found in the music bank.

All existing music players containing removed playlist will set their playlist reference to null.

Events

OnMusicPlayerAdded

The delegate informing about adding music players.

Declaration

```
public event MusicPlayerAddedDelegate OnMusicPlayerAdded
```

Event Type

TYPE	DESCRIPTION
MusicPlayerAddedDelegate	

OnMusicPlayerRemoved

The delegate informing about removing music players.

Declaration

```
public event MusicPlayerRemovedDelegate OnMusicPlayerRemoved
```

Event Type

TYPE	DESCRIPTION
MusicPlayerRemovedDelegate	

OnMusicPlayerRenamed

The delegate informing about the change of music player names.

Declaration

```
public event MusicPlayerRenamedDelegate OnMusicPlayerRenamed
```

Event Type

TYPE	DESCRIPTION
MusicPlayerRenamedDelegate	

OnPlaylistAdded

The delegate informing about adding playlists.

Declaration

```
public event PlaylistAddedDelegate OnPlaylistAdded
```

Event Type

TYPE	DESCRIPTION
PlaylistAddedDelegate	

OnPlaylistRemoved

The delegate informing about removing playlists.

Declaration

```
public event PlaylistRemovedDelegate OnPlaylistRemoved
```

Event Type

TYPE	DESCRIPTION
PlaylistRemovedDelegate	

OnPlaylistRenamed

The delegate informing about the change of playlist names.

Declaration

```
public event PlaylistRenamedDelegate OnPlaylistRenamed
```

Event Type

TYPE	DESCRIPTION
PlaylistRenamedDelegate	

Implements

IBank
ISerializationCallbackReceiver

Class MusicManager

The main class for music playback and bank management.

Inheritance

System.Object
MusicManager

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public static class MusicManager
```

Properties

Banks

The collection of all registered music banks.

Declaration

```
public static ReadOnlyCollection<MusicBank> Banks { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<MusicBank>	A reference to a read-only collection of music banks.

PrimaryBank

The primary music bank that will be searched first in case of name collisions.

Declaration

```
public static MusicBank PrimaryBank { get; set; }
```

Property Value

TYPE	DESCRIPTION
MusicBank	A reference to a primary music bank.

Methods

DeregisterBank(MusicBank)

Deregisters existing music bank.

Declaration

```
public static bool DeregisterBank(MusicBank bank)
```

Parameters

TYPE	NAME	DESCRIPTION
MusicBank	bank	A reference to a music bank to deregister.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if music bank was succesfully deregistered. False otherwise.

GetMusicPlayer(ID)

Searches for the specified music player with a matching ID.

Declaration

```
public static MusicPlayer GetMusicPlayer(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID referring to the music player.

Returns

TYPE	DESCRIPTION
MusicPlayer	A reference to a music player, if found. Null reference otherwise.

GetMusicPlayer(String)

Searches for the specified music player with a matching name.

Declaration

```
public static MusicPlayer GetMusicPlayer(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the music player.

Returns

TYPE	DESCRIPTION
MusicPlayer	A reference to a music player, if found. Null reference otherwise.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

GetPlaylist(ID)

Searches for the specified playlist with a matching ID.

Declaration

```
public static Playlist GetPlaylist(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID referring to the playlist.

Returns

TYPE	DESCRIPTION
Playlist	A reference to a playlist, if found. Null reference otherwise.

GetPlaylist(String)

Searches for the specified playlist with a matching name.

Declaration

```
public static Playlist GetPlaylist(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the playlist.

Returns

TYPE	DESCRIPTION
Playlist	A reference to a playlist, if found. Null reference otherwise.

Remarks

If multiple banks have playlists with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found playlist will be used.

IsPlaying(ID)

Checks whether or not specified music player with a matching ID is playing.

Declaration

```
public static bool IsPlaying(ID playerID)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if the found music player is playing. False otherwise.

IsPlaying(String)

Checks whether or not specified music player with a matching name is playing.

Declaration

```
public static bool IsPlaying(string playerName)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if the found music player is playing. False otherwise.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

Next(ID, Nullable<Single>)

Advances music player to next track.

Declaration

```
public static void Next(ID playerID, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

This method does nothing if no playlist was assigned to the music player. Use SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Next(String, Nullable<Single>)

Advances music player to next track.

Declaration

```
public static void Next(string playerName, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

This method does nothing if no playlist was assigned to the music player. Use SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Pause()

Pauses all music players from all music banks.

Declaration

```
public static void Pause()
```

Remarks

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Pause(ID, Nullable<Single>)

Pauses music player.

Declaration

```
public static void Pause(ID playerID, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Pause(String, Nullable<Single>)

Pauses music player.

Declaration

```
public static void Pause(string playerName, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Play(ID, Nullable<Single>)

Plays music player.

Declaration

```
public static void Play(ID playerID, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Play(Nullable<Single>)

Plays all music players from all music banks.

Declaration

```
public static void Play(float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Play(String, Nullable<Single>)

Plays music player.

Declaration

```
public static void Play(string playerName, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Prev(ID, Nullable<Single>)

Advances music player to previous track.

Declaration

```
public static void Prev(ID playerID, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

This method does nothing if no playlist was assigned to the music player. Use SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Prev(String, Nullable<Single>)

Advances music player to previous track.

Declaration

```
public static void Prev(string playerName, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

This method does nothing if no playlist was assigned to the music player. Use SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

RegisterBank(MusicBank)

Registers new music bank.

Declaration

```
public static bool RegisterBank(MusicBank bank)
```

Parameters

TYPE	NAME	DESCRIPTION
MusicBank	bank	A reference to a music bank to register.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if music bank was succesfully registered. False otherwise.

Seek(ID, Int32, Nullable<Single>)

Advances music player to a track by index.

Declaration

```
public static void Seek(ID playerID, int track, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.Int32	track	Zero-based index of the track in the current playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

Target track must be one of current playlist tracks.

The index value represents track order as they appear in the playlist (e.g. setting index to one will seek to the second playlist track and so on). Shuffle order is ignored.

This method does nothing if no playlist was assigned to the music player. Use SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Seek(ID, String, Nullable<Single>)

Advances music player to a track with a matching name.

Declaration

```
public static void Seek(ID playerID, string track, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.String	track	Name of the track in the current playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

Target track must be one of current playlist tracks.

This method does nothing if no playlist was assigned to the music player. Use SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Seek(String, Int32, Nullable<Single>)

Advances music player to a track by index.

Declaration

```
public static void Seek(string playerName, int track, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.Int32	track	Zero-based index of the track in the current playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

Target track must be one of current playlist tracks.

The index value represents track order as they appear in the playlist (e.g. setting index to one will seek to the second playlist track and so on). Shuffle order is ignored.

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Seek(String, String, Nullable<Single>)

Advances music player to a track with a matching name.

Declaration

```
public static void Seek(string playerName, string track, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.String	track	Name of the track in the current playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

Target track must be one of current playlist tracks.

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

SetPlaylist(ID, ID, Nullable<Single>)

Sets a playlist to a music player.

Declaration

```
public static void SetPlaylist(ID playerID, ID playlistID, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
ID	playlistID	ID referring to the playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If music player was playing another track it'll automatically crossfade to first track of the new playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

SetPlaylist(ID, String, Nullable<Single>)

Sets a playlist to a music player.

Declaration

```
public static void SetPlaylist(ID playerID, string playlistName, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.String	playlistName	Name of the playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have playlists with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found playlist will be used.

If music player was playing another track it'll automatically crossfade to first track of the new playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

SetPlaylist(String, ID, Nullable<Single>)

Sets a playlist to a music player.

Declaration

```
public static void SetPlaylist(string playerName, ID playlistID, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
ID	playlistID	ID referring to the playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

If music player was playing another track it'll automatically crossfade to first track of the new playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

SetPlaylist(String, String, Nullable<Single>)

Sets a playlist to a music player.

Declaration

```
public static void SetPlaylist(string playerName, string playlistName, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.String	playlistName	Name of the playlist.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have music players/playlists with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player/playlist will be used.

If music player was playing another track it'll automatically crossfade to first track of the new playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Stop(ID, Nullable<Single>)

Stops music player.

Declaration

```
public static void Stop(ID playerID, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	playerID	ID referring to the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Stop(Nullable<Single>)

Stops all music players from all music banks.

Declaration

```
public static void Stop(float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Stop(String, Nullable<Single>)

Stops music player.

Declaration

```
public static void Stop(string playerName, float? fade = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	playerName	Name of the music player.
System.Nullable<System.Single>	fade	Crossfade duration in seconds.

Remarks

If multiple banks have music players with a matching name, the primary music bank will be checked first. Within a bank, the first occurrence of found music player will be used.

This method does nothing if no playlist was assigned to the music player. SetPlaylist(String, String, Nullable<Single>) to assign a playlist.

Non-null crossfade parameter value will override MusicPlayer.Fade value.

Events

OnPlaybackPaused

The delegate informing about playback pause in any of the music players.

Declaration

```
public static event PlaybackChangedDelegate OnPlaybackPaused
```

Event Type

TYPE	DESCRIPTION
PlaybackChangedDelegate	

Remarks

This delegate will only be called after the music player fades out.

OnPlaybackStarted

The delegate informing about playback start in any of the music players.

Declaration

```
public static event PlaybackChangedDelegate OnPlaybackStarted
```

Event Type

TYPE	DESCRIPTION
PlaybackChangedDelegate	

OnPlaybackStopped

The delegate informing about playback stop in any of the music players.

Declaration

```
public static event PlaybackChangedDelegate OnPlaybackStopped
```

Event Type

TYPE	DESCRIPTION
PlaybackChangedDelegate	

Remarks

This delegate will only be called after the music player fades out.

OnTrackChanged

The delegate informing about tracks changes in any of the music players.

Declaration

```
public static event TrackChangedDelegate OnTrackChanged
```

Event Type

TYPE	DESCRIPTION
TrackChangedDelegate	

Class MusicPlayer

The persistent storage for playback rules of a particular playlist.

Inheritance

System.Object
MusicPlayer

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public class MusicPlayer : ISerializationCallbackReceiver
```

Properties

Audible

The flag indicating if the music player can be heard.

Declaration

```
public bool Audible { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the music player can be heard. False otherwise.

Bank

The music bank the music player belongs to.

Declaration

```
public MusicBank Bank { get; set; }
```

Property Value

TYPE	DESCRIPTION
MusicBank	A reference to a music bank.

Fade

The crossfade parameter that is used when the music player transitions between tracks or playback states.

Declaration

```
public float Fade { get; set; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Single	Crossfade duration in seconds.

ID

The unique identifier for fast access to the music player.

Declaration

```
public int ID { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	A unique identifier value of the music player.

Loop

The flag indicating whether the music player should repeat playlist tracks after they finish.

Declaration

```
public bool Loop { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the music player is looping playlist tracks. False otherwise.

Remarks

If PlaybackMode value is AutoAdvance then this flag is indicating whether the music player should repeat the whole playlist instead of individual playlist track.

MixerGroup

The reference to an audio mixer group. Please refer to Unity Scripting Reference for details.

Declaration

```
public AudioMixerGroup MixerGroup { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioMixerGroup	A reference to a mixer group.

Muted

The flag indicating if the music player is muted and can't be heard.

Declaration

```
public bool Muted { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the music player is muted. False otherwise.

Remarks

This flag may be overridden by the Soloed flag, i.e. if the music player is simultaneously muted and soloed it'll be audible.

Name

The name of the music player. Used for fast search in corresponding music bank.

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	Name of the music player.

PlaybackMode

The playback mode defining how music player should play its tracks.

Declaration

```
public MusicPlayerPlaybackMode PlaybackMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
MusicPlayerPlaybackMode	An enum value.

Playlist

The reference to a playlist which will be played.

Declaration

```
public Playlist Playlist { get; set; }
```

Property Value

TYPE	DESCRIPTION
Playlist	A reference to a playlist.

PlayOnStart

The flag indicating whether the music player should start playing once the game started.

Declaration

```
public bool PlayOnStart { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the music player should play on start. False otherwise.

Shuffle

The flag indicating whether the music player should play tracks in random order.

Declaration

```
public bool Shuffle { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the music player is playing playlist tracks in random order. False if it's playing sequentially.

Remarks

Note that tracks will be reshuffled again after the player will finish playing all the tracks.

Soloed

The flag indicating if the music player is soloed. If set to true, all other non-solo music players won't be audible.

Declaration

```
public bool Soloed { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the music player is soloed. False otherwise.

Remarks

This flag may override the Muted flag, i.e. if the music player is simultaneously muted and soloed it'll be audible.

SyncGroup

The sync group of the music player. Music players with the same sync group will share playback time.

Declaration

```
public byte SyncGroup { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Byte	Sync group of the music player.

Remarks

Sync groups work only if PlaybackMode value is Synced.

Unfolded

The flag indicating whether the music bank inspector should show advanced settings for the music player.

Declaration

```
public bool Unfolded { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if advanced settings are shown. False otherwise.

Remarks

This property is used only by the music bank inspector and does nothing during runtime.

Volume

The volume of the music player.

Declaration

```
public float Volume { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Volume of the music player. Value must be in [0;1] range.

Methods

OnAfterDeserialize()

Prepares sound for runtime use after deserialization.

Declaration

```
public void OnAfterDeserialize()
```

Remarks

This method is automatically called by Unity during deserialization process. Don't call it manually.

OnBeforeSerialize()

Prepares sound for serialization.

Declaration

```
public void OnBeforeSerialize()
```

Remarks

This method is automatically called by Unity during serialization process. Don't call it manually.

Events

OnPlaybackPaused

The delegate informing about playback pause in the music player.

Declaration

```
public event PlaybackChangedDelegate OnPlaybackPaused
```

Event Type

TYPE	DESCRIPTION
PlaybackChangedDelegate	

Remarks

This delegate will only be called after the music player fades out.

OnPlaybackStarted

The delegate informing about playback start in the music player.

Declaration

```
public event PlaybackChangedDelegate OnPlaybackStarted
```

Event Type

TYPE	DESCRIPTION
PlaybackChangedDelegate	

OnPlaybackStopped

The delegate informing about playback stop in the music player.

Declaration

```
public event PlaybackChangedDelegate OnPlaybackStopped
```

Event Type

TYPE	DESCRIPTION
PlaybackChangedDelegate	

Remarks

This delegate will only be called after the music player fades out.

OnTrackChanged

The delegate informing about tracks changes in the music player.

Declaration

```
public event TrackChangedDelegate OnTrackChanged
```

Event Type

TYPE	DESCRIPTION
TrackChangedDelegate	

Delegate MusicPlayerAddedDelegate

A music bank callback function, called after adding music player to the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void MusicPlayerAddedDelegate(MusicPlayer player);
```

Parameters

TYPE	NAME	DESCRIPTION
MusicPlayer	player	A reference to a newly added music player.

Class MusicPlayerIDAttribute

The attribute class used to make an int variable in a script be restricted to a music player id.

Inheritance

System.Object

MusicPlayerIDAttribute

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class MusicPlayerIDAttribute : PropertyAttribute
```

Remarks

When this attribute is used, the variable will be shown as two dropdown fields in the inspector instead of the default number field.

Enum MusicPlayerPlaybackMode

Defines how music player plays its tracks.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public enum MusicPlayerPlaybackMode
```

Fields

NAME	DESCRIPTION
AutoAdvance	The music player will advance to the next track from the current playlist after the current track is finished. For non-zero Fade property values the music player will automatically crossfade between current and next tracks.
Default	The music player will play a single track from the current playlist.
Synced	The music player will synchronize current track time according to other music players in a sync group. All music players with the same SyncGroup value will be automatically synchronized.

Delegate MusicPlayerRemovedDelegate

A music bank callback function, called before removing the music player from the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void MusicPlayerRemovedDelegate(MusicPlayer player, int index);
```

Parameters

TYPE	NAME	DESCRIPTION
MusicPlayer	player	A reference to a music player to be removed.
System.Int32	index	An index in corresponding Players collection.

Delegate MusicPlayerRenamedDelegate

A music bank callback function, called after changing music player name.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void MusicPlayerRenamedDelegate(MusicPlayer player, int index, string oldName, string
newName);
```

Parameters

TYPE	NAME	DESCRIPTION
MusicPlayer	player	A reference to a music player.
System.Int32	index	An index in corresponding Players collection.
System.String	oldName	An old name of the music player.
System.String	newName	A new name of the music player.

Delegate PlaybackChangedDelegate

A music callback function, called when the music player changes playback state (playing, stopped, paused).

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void PlaybackChangedDelegate(MusicPlayer player);
```

Parameters

TYPE	NAME	DESCRIPTION
MusicPlayer	player	A reference to a music player.

Class Playlist

The persistent collection of playlist tracks.

Inheritance

System.Object
Playlist

Implements

IAudioClipContainer
ISerializationCallbackReceiver

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public class Playlist : IAudioClipContainer, ISerializationCallbackReceiver
```

Properties

AudioClipManagementMode

The audio clip management mode of the playlist.

Declaration

```
public AudioClipManagementMode AudioClipManagementMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClipManagementMode	An enum value.

Remarks

This value is used only if OverrideAudioClipManagement flag is true.

AudioClipUnloadInterval

The audio clip unload interval of the playlist.

Declaration

```
public float AudioClipUnloadInterval { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	A time interval in seconds.

Remarks

This value is used only if OverrideAudioClipManagement flag is true and GetAudioClipManagementMode() return value is UnloadUnused.

Bank

The music bank the playlist belongs to.

Declaration

```
public MusicBank Bank { get; set; }
```

Property Value

TYPE	DESCRIPTION
MusicBank	A reference to a music bank.

ID

The unique identifier for fast access to the playlist.

Declaration

```
public int ID { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	A unique identifier value of the playlist.

Name

The name of the playlist. Used for fast search in corresponding music bank.

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	Name of the playlist.

OverrideAudioClipManagement

The flag indicating whether the playlist should use its own audio clip management mode and unload interval.

Declaration

```
public bool OverrideAudioClipManagement { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the playlist has overridden audio clip management and unload interval. False otherwise.

Remarks

This flag defines the behaviour of GetAudioClipManagementMode() and GetAudioClipUnloadInterval() methods.

If true, the playlist will use its own AudioClipManagementMode and AudioClipUnloadInterval properties. Otherwise, PlaylistManagementMode and PlaylistUnloadInterval properties of the containing bank will be used.

Tracks

The collection of playlist tracks.

Declaration

```
public ReadOnlyCollection<PlaylistTrack> Tracks { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<PlaylistTrack>	A reference to a read-only collection of playlist tracks.

Unfolded

The flag indicating whether the music bank inspector should show advanced settings for the playlist.

Declaration

```
public bool Unfolded { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if advanced settings are shown. False otherwise.

Remarks

This property is used only by the music bank inspector and does nothing during runtime.

Methods

AddTrack(AudioClip)

Adds a single music track to the playlist.

Declaration

```
public PlaylistTrack AddTrack(AudioClip clip)
```

Parameters

TYPE	NAME	DESCRIPTION
AudioClip	clip	A reference to the audio clip with music data.

Returns

TYPE	DESCRIPTION
PlaylistTrack	A reference to a newly created playlist track.

AddTracks(AudioClip[])

Adds multiple music tracks (one per audio clip) to the playlist.

Declaration

```
public void AddTracks(AudioClip[] clips)
```

Parameters

TYPE	NAME	DESCRIPTION
AudioClip[]	clips	An array of audio clips with music data.

GetAudioClip(Int32)

Gets the audio clip at the specified index.

Declaration

```
public AudioClip GetAudioClip(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	The zero-based index of the audio clip to get.

Returns

TYPE	DESCRIPTION
AudioClip	A reference to an audio clip.

GetAudioClipManagementMode()

Gets the audio clip management mode of the playlist.

Declaration

```
public AudioClipManagementMode GetAudioClipManagementMode()
```

Returns

TYPE	DESCRIPTION
AudioClipManagementMode	An enum value.

Remarks

If OverrideAudioClipManagement value is true, AudioClipManagementMode will be used. Otherwise, PlaylistManagementMode value of the containing bank will be used.

GetAudioClipUnloadInterval()

Gets the audio clip unload interval of the playlist.

Declaration

```
public float GetAudioClipUnloadInterval()
```

Returns

TYPE	DESCRIPTION
System.Single	The time interval in seconds.

Remarks

This value is only used if GetAudioClipManagementMode() return value is UnloadUnused

If OverrideAudioClipManagement value is true, AudioClipUnloadInterval will be used. Otherwise, PlaylistUnloadInterval value of the containing bank will be used.

GetNumAudioClips()

Gets the number of audio clips in the playlist.

Declaration

```
public int GetNumAudioClips()
```

Returns

TYPE	DESCRIPTION
System.Int32	The number of audio clips.

OnAfterDeserialize()

Prepares playlist for runtime use after deserialization.

Declaration

```
public void OnAfterDeserialize()
```

Remarks

This method is automatically called by Unity during deserialization process. Don't call it manually.

OnBeforeSerialize()

Prepares playlist for serialization.

Declaration

```
public void OnBeforeSerialize()
```

Remarks

This method is automatically called by Unity during serialization process. Don't call it manually.

Implements

IAudioClipContainer
ISerializationCallbackReceiver

Delegate PlaylistAddedDelegate

A music bank callback function, called after adding playlist to the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void PlaylistAddedDelegate(Playlist playlist);
```

Parameters

TYPE	NAME	DESCRIPTION
Playlist	playlist	A reference to a newly added playlist.

Class PlaylistIDAttribute

The attribute class used to make an int variable in a script be restricted to a playlist id.

Inheritance

System.Object

PlaylistIDAttribute

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class PlaylistIDAttribute : PropertyAttribute
```

Remarks

When this attribute is used, the variable will be shown as two dropdown fields in the inspector instead of the default number field.

Delegate PlaylistRemovedDelegate

A music bank callback function, called before removing the playlist from the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void PlaylistRemovedDelegate(Playlist playlist, int index);
```

Parameters

TYPE	NAME	DESCRIPTION
Playlist	playlist	A reference to a playlist to be removed.
System.Int32	index	An index in corresponding Playlists collection.

Delegate PlaylistRenamedDelegate

A music bank callback function, called after changing playlist name.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void PlaylistRenamedDelegate(Playlist playlist, int index, string oldName, string newName);
```

Parameters

TYPE	NAME	DESCRIPTION
Playlist	playlist	A reference to a playlist.
System.Int32	index	An index in corresponding Playlists collection.
System.String	oldName	An old name of the playlist.
System.String	newName	A new name of the playlist.

Class PlaylistTrack

The persistent storage for music audio data.

Inheritance

System.Object
PlaylistTrack

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public class PlaylistTrack
```

Properties

Clip

The audio clip with audio data. Please refer to Unity Scripting Reference for details.

Declaration

```
public AudioClip Clip { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClip	A reference to an audio clip.

Name

The name of the track.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	Name of the current audio clip being used. Null reference otherwise.

Playlist

The playlist the track belongs to.

Declaration

```
public Playlist Playlist { get; set; }
```

Property Value

TYPE	DESCRIPTION
Playlist	A reference to a playlist.

Volume

The volume of the track.

Declaration

```
public float Volume { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Volume of the track. Value must be in [0;1] range.

Enum RetriggerMode

Defines how sound will play its variations.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public enum RetriggerMode
```

Fields

NAME	DESCRIPTION
PingPong	Play variations repeatedly from start to end and vice versa.
Random	Play random variations with possible repetitions.
RandomNoRepeat	Play random variations without repetitions.
Sequential	Play variations in a sequence as they stored in the sound.

Class Sound

The persistent storage for sound variations and the most important audio source settings.

Inheritance

System.Object
Sound

Implements

IAudioClipContainer
ISerializationCallbackReceiver

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public class Sound : IAudioClipContainer, ISerializationCallbackReceiver
```

Properties

AttenuationMode

The attenuation mode defining how sound volume will be lowered over the distance.

Declaration

```
public AttenuationMode AttenuationMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
AttenuationMode	An enum value.

Remarks

Note that attenuation rules applies only for 3D sounds. Please refer to Unity Scripting Reference for details.

Audible

The flag indicating if the sound can be heard.

Declaration

```
public bool Audible { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the sound can be heard. False otherwise.

Remarks

If the Bus is inaudible, the sound will also be inaudible.

AudioClipManagementMode

The audio clip management mode of the sound.

Declaration

```
public AudioClipManagementMode AudioClipManagementMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClipManagementMode	An enum value.

Remarks

This value is used only if OverrideAudioClipManagement flag is true.

AudioClipUnloadInterval

The audio clip unload interval of the sound.

Declaration

```
public float AudioClipUnloadInterval { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	A time interval in seconds.

Remarks

This value is used only if OverrideAudioClipManagement flag is true and GetAudioClipManagementMode() return value is UnloadUnused.

Bank

The sound bank the sound belongs to.

Declaration

```
public SoundBank Bank { get; set; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
SoundBank	A reference to a sound bank.

Bus

The reference to a sound bus which will manage the sound.

Declaration

```
public SoundBus Bus { get; set; }
```

Property Value

TYPE	DESCRIPTION
SoundBus	A reference to a sound bus.

Remarks

If set to null, DefaultBus will be used.

DopplerLevel

The parameter defining the Doppler scale for the sound.

Declaration

```
public float DopplerLevel { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Doppler scale of the sound. The value must be greater or equal to zero.

Remarks

This parameter duplicates corresponding parameter from AudioSource. Please refer to Unity Scripting Reference for details.

ID

The unique identifier for fast access to the sound.

Declaration

```
public int ID { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	A unique identifier value of the sound.

MaxDistance

The parameter defining the boundary outside which the sound will be inaudible or stop attenuating depending on

AttenuationMode value.

Declaration

```
public float MaxDistance { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Distance in units.

Remarks

This parameter duplicates corresponding parameter from AudioSource. Please refer to Unity Scripting Reference for details.

MinDistance

The parameter defining the boundary within which the sound won't get any louder.

Declaration

```
public float MinDistance { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Distance in units.

Remarks

This parameter duplicates corresponding parameter from AudioSource. Please refer to Unity Scripting Reference for details.

Muted

The flag indicating if the sound is muted and can't be heard.

Declaration

```
public bool Muted { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the sound is muted. False otherwise.

Remarks

This flag may be overridden by the Soloed flag, i.e. if the sound is simultaneously muted and soloed it'll be audible.

Name

The name of the sound. Used for fast search in corresponding sound bank.

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	Name of the sound.

OverrideAudioClipManagement

The flag indicating whether the sound should use its own audio clip management mode and unload interval.

Declaration

```
public bool OverrideAudioClipManagement { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the sound has overridden audio clip management and unload interval. False otherwise.

Remarks

This flag defines the behaviour of GetAudioClipManagementMode() and GetAudioClipUnloadInterval() methods.

If true, the sound will use its own AudioClipManagementMode and AudioClipUnloadInterval properties. Otherwise, SoundManagementMode and SoundUnloadInterval properties of the containing bank will be used.

PanStereo

The stereo panning parameter defining sound position in a stereo way (left or right).

Declaration

```
public float PanStereo { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Stereo pan of the sound. The value must be in [-1;1] range.

Remarks

This parameter duplicates corresponding parameter from AudioSource. Please refer to Unity Scripting Reference for details.

Soloed

The flag indicating if the sound is soloed. If set to true, all other non-solo sounds won't be audible.

Declaration

```
public bool Soloed { get; set; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Boolean	True, if the sound is soloed. False otherwise.

Remarks

This flag may override the Muted flag, i.e. if the sound is simultaneously muted and soloed it'll be audible.

SpatialBlend

The spatial blend parameter defining how much the sound is affected by 3d spatialisation calculations (attenuation, doppler etc).

Declaration

```
public float SpatialBlend { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Spatial blend of the sound. The value must be in [0;1] range.

Remarks

This parameter duplicates corresponding parameter from AudioSource. Please refer to Unity Scripting Reference for details.

Spread

The parameter defining the spread angle (in degrees) of a 3d stereo or multichannel sound in speaker space.

Declaration

```
public float Spread { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Spread angle of the sound. The value must be in [0;360] range.

Remarks

This parameter duplicates corresponding parameter from AudioSource. Please refer to Unity Scripting Reference for details.

Unfolded

The flag indicating whether the sound bank inspector should show advanced settings for the sound.

Declaration

```
public bool Unfolded { get; set; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
System.Boolean	True, if advanced settings are shown. False otherwise.

Remarks

This property is used only by the sound bank inspector and does nothing during runtime.

VariationRetriggerMode

The retrigger mode defining how the sound will play variations.

Declaration

```
public RetriggerMode VariationRetriggerMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
RetriggerMode	An enum value.

Variations

The collection of sound variations.

Declaration

```
public ReadOnlyCollection<SoundVariation> Variations { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<SoundVariation>	A reference to a read-only collection of sound variations.

Volume

The master volume of the sound.

Declaration

```
public float Volume { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Volume of the sound. Value must be in [0;1] range.

Methods

AddVariation(AudioClip)

Adds a single sound variation to the sound.

Declaration

```
public SoundVariation AddVariation(AudioClip clip)
```

Parameters

TYPE	NAME	DESCRIPTION
AudioClip	clip	A reference to the audio clip with audio data.

Returns

TYPE	DESCRIPTION
SoundVariation	A reference to a newly created sound variation.

AddVariations(AudioClip[])

Adds multiple sound variations (one per audio clip) to the sound.

Declaration

```
public void AddVariations(AudioClip[] clips)
```

Parameters

TYPE	NAME	DESCRIPTION
AudioClip[]	clips	An array of audio clips with audio data.

GetAudioClip(Int32)

Gets the audio clip at the specified index.

Declaration

```
public AudioClip GetAudioClip(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	The zero-based index of the audio clip to get.

Returns

TYPE	DESCRIPTION
AudioClip	A reference to an audio clip.

GetAudioClipManagementMode()

Gets the audio clip management mode of the sound.

Declaration

```
public AudioClipManagementMode GetAudioClipManagementMode()
```

Returns

TYPE	DESCRIPTION
AudioClipManagementMode	An enum value.

Remarks

If OverrideAudioClipManagement value is true, AudioClipManagementMode will be used. Otherwise, SoundManagementMode value of the containing bank will be used.

GetAudioClipUnloadInterval()

Gets the audio clip unload interval of the sound.

Declaration

```
public float GetAudioClipUnloadInterval()
```

Returns

TYPE	DESCRIPTION
System.Single	The time interval in seconds.

Remarks

This value is only used if GetAudioClipManagementMode() return value is UnloadUnused

If OverrideAudioClipManagement value is true, AudioClipUnloadInterval will be used. Otherwise, SoundUnloadInterval value of the containing bank will be used.

GetNumAudioClips()

Gets the number of audio clips in the sound.

Declaration

```
public int GetNumAudioClips()
```

Returns

TYPE	DESCRIPTION
System.Int32	The number of audio clips.

OnAfterDeserialize()

Prepares sound for runtime use after deserialization.

Declaration

```
public void OnAfterDeserialize()
```

Remarks

This method is automatically called by Unity during deserialization process. Don't call it manually.

OnBeforeSerialize()

Prepares sound for serialization.

Declaration

```
public void OnBeforeSerialize()
```

Remarks

This method is automatically called by Unity during serialization process. Don't call it manually.

Implements

IAudioClipContainer
ISerializationCallbackReceiver

Delegate SoundAddedDelegate

A sound bank callback function, called after adding sound to the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void SoundAddedDelegate(Sound sound);
```

Parameters

TYPE	NAME	DESCRIPTION
Sound	sound	A reference to a newly added sound.

Class SoundBank

The persistent storage for sounds and sound buses.

Inheritance

System.Object
SoundBank

Implements

IBank
ISerializationCallbackReceiver

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class SoundBank : ScriptableObject, IBank, ISerializationCallbackReceiver
```

Properties

Buses

The collection of sound buses.

Declaration

```
public ReadOnlyCollection<SoundBus> Buses { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<SoundBus>	A reference to a read-only collection of sound buses.

DefaultBus

The sound bus which will be used by default on newly created sounds.

Declaration

```
public SoundBus DefaultBus { get; }
```

Property Value

TYPE	DESCRIPTION
SoundBus	A reference to a sound bus.

ShowSoundBuses

The flag indicating whether the sound bank inspector should show the 'Buses' group.

Declaration

```
public bool ShowSoundBuses { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the 'Buses' group is shown. False otherwise.

Remarks

This property is used only by the sound bank inspector and does nothing during runtime.

ShowSounds

The flag indicating whether the sound bank inspector should show the 'Sounds' group.

Declaration

```
public bool ShowSounds { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the 'Sounds' group is shown. False otherwise.

Remarks

This property is used only by the sound bank inspector and does nothing during runtime.

SoundBatchImportMode

The batch import mode defining how new sounds will be created after the drag-drop event.

Declaration

```
public AudioClipImportMode SoundBatchImportMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClipImportMode	An enum value.

SoundManagementMode

The default audio clip management mode for all sounds of the sound bank.

Declaration

```
public AudioClipManagementMode SoundManagementMode { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClipManagementMode	An enum value.

Remarks

By default, all sounds will use this value for audio clip management, however, it can be overridden by the

OverrideAudioClipManagement flag.

Sounds

The collection of sounds.

Declaration

```
public ReadOnlyCollection<Sound> Sounds { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<Sound>	A reference to a read-only collection of sounds.

SoundUnloadInterval

The default audio clip unload interval for all sounds of the the sound bank.

Declaration

```
public float SoundUnloadInterval { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	A time interval in seconds.

Remarks

By default, all sounds will use this value for audio clip unload interval, however, it can be overridden by the OverrideAudioClipManagement flag.

Methods

AddSound(String)

Adds an empty sound to the sound bank.

Declaration

```
public Sound AddSound(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.

Returns

TYPE	DESCRIPTION
Sound	A reference to a newly created sound.

AddSound(String, AudioClip)

Adds a sound with a single variation to the sound bank.

Declaration

```
public Sound AddSound(string name, AudioClip clip)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.
AudioClip	clip	A reference to the audio clip with audio data.

Returns

TYPE	DESCRIPTION
Sound	A reference to a newly created sound.

AddSound(String, AudioClip[])

Adds a sound with multiple variations to the sound bank.

Declaration

```
public Sound AddSound(string name, AudioClip[] clips)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.
AudioClip[]	clips	An array of audio clips with audio data.

Returns

TYPE	DESCRIPTION
Sound	A reference to a newly created sound.

AddSoundBus(String)

Adds a new sound bus to the sound bank.

Declaration

```
public SoundBus AddSoundBus(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound bus.

Returns

TYPE	DESCRIPTION
SoundBus	A reference to a newly created sound bus.

GetBankID()

Returns sound bank ID.

Declaration

```
public ID GetBankID()
```

Returns

TYPE	DESCRIPTION
ID	An ID value.

GetSound(ID)

Searches for the specified sound by ID.

Declaration

```
public Sound GetSound(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID that refers to a sound.

Returns

TYPE	DESCRIPTION
Sound	A reference to a sound, if found. Null reference otherwise.

GetSound(String)

Searches for the specified sound with a matching name.

Declaration

```
public Sound GetSound(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.

Returns

TYPE	DESCRIPTION
Sound	A reference to a sound, if found. Null reference otherwise.

GetSoundBus(ID)

Searches for the specified sound bus by ID.

Declaration

```
public SoundBus GetSoundBus(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID that refers to a sound bus.

Returns

TYPE	DESCRIPTION
SoundBus	A reference to a sound bus, if found. Null reference otherwise.

GetSoundBus(String)

Searches for the specified sound bus with a matching name.

Declaration

```
public SoundBus GetSoundBus(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound bus.

Returns

TYPE	DESCRIPTION
SoundBus	A reference to a sound bus, if found. Null reference otherwise.

GetSoundBusID(Int32)

Gets an ID to the specific sound bus.

Declaration

```
public ID GetSoundBusID(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	Zero-based index of the sound bus in the current sound bank.

Returns

TYPE	DESCRIPTION
ID	An ID to the specific sound bus.

GetSoundID(Int32)

Gets an ID to the specific sound.

Declaration

```
public ID GetSoundID(int index)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	index	Zero-based index of the sound in the current sound bank.

Returns

TYPE	DESCRIPTION
ID	An ID to the specific sound.

OnAfterDeserialize()

Prepares sound bank for runtime use after deserialization.

Declaration

```
public void OnAfterDeserialize()
```

Remarks

This method is automatically called by Unity during deserialization process. Don't call it manually.

OnBeforeSerialize()

Prepares sound bank for serialization.

Declaration

```
public void OnBeforeSerialize()
```

Remarks

This method is automatically called by Unity during serialization process. Don't call it manually.

RegenerateBankID()

Generates a new unique ID for the sound bank.

Declaration

```
public void RegenerateBankID()
```

Remarks

This method is automatically called by Stem during serialization process. Don't call it manually as it may break existing ID references.

RemoveSound(Sound)

Removes existing sound from the sound bank.

Declaration

```
public void RemoveSound(Sound sound)
```

Parameters

TYPE	NAME	DESCRIPTION
Sound	sound	A reference to a sound.

Remarks

This method does nothing if the sound was not found in the sound bank.

RemoveSoundBus(SoundBus)

Removes existing sound bus from the sound bank.

Declaration

```
public void RemoveSoundBus(SoundBus bus)
```

Parameters

TYPE	NAME	DESCRIPTION
SoundBus	bus	A reference to a sound bus.

Remarks

This method does nothing if the sound bus was not found in the sound bank.

Events

OnSoundAdded

The delegate informing about adding sounds.

Declaration

```
public event SoundAddedDelegate OnSoundAdded
```

Event Type

TYPE	DESCRIPTION
SoundAddedDelegate	

OnSoundBusAdded

The delegate informing about adding sound buses.

Declaration

```
public event SoundBusAddedDelegate OnSoundBusAdded
```

Event Type

TYPE	DESCRIPTION
SoundBusAddedDelegate	

OnSoundBusRemoved

The delegate informing about removing sound buses.

Declaration

```
public event SoundBusRemovedDelegate OnSoundBusRemoved
```

Event Type

TYPE	DESCRIPTION
SoundBusRemovedDelegate	

OnSoundBusRenamed

The delegate informing about the change of sound bus names.

Declaration

```
public event SoundBusRenamedDelegate OnSoundBusRenamed
```

Event Type

TYPE	DESCRIPTION
SoundBusRenamedDelegate	

OnSoundRemoved

The delegate informing about removing sounds.

Declaration

```
public event SoundRemovedDelegate OnSoundRemoved
```

Event Type

TYPE	DESCRIPTION
SoundRemovedDelegate	

OnSoundRenamed

The delegate informing about the change of sound names.

Declaration

```
public event SoundRenamedDelegate OnSoundRenamed
```

Event Type

TYPE	DESCRIPTION
SoundRenamedDelegate	

Implements

IBank
ISerializationCallbackReceiver

Class SoundBus

The persistent storage for playback rules of a group of sounds.

Inheritance

System.Object
SoundBus

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public class SoundBus : ISerializationCallbackReceiver
```

Properties

AllowVoiceStealing

The flag indicating whether the sound bus can stop the oldest playing sound and play the new one if Polyphony limit is exceeded.

Declaration

```
public bool AllowVoiceStealing { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if sound bus will be stopping the oldest playing sounds. False if the sound bus won't allow new sounds to play.

Audible

The flag indicating if the sound bus can be heard.

Declaration

```
public bool Audible { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the sound bus can be heard. False otherwise.

Bank

The sound bank the sound bus belongs to.

Declaration

```
public SoundBank Bank { get; set; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
SoundBank	A reference to a sound bank.

ID

The unique identifier for fast access to the sound bus.

Declaration

```
public int ID { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	A unique identifier value of the sound bus.

MixerGroup

The reference to an audio mixer group. Please refer to Unity Scripting Reference for details.

Declaration

```
public AudioMixerGroup MixerGroup { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioMixerGroup	A reference to a mixer group.

Muted

The flag indicating if the sound bus is muted and can't be heard.

Declaration

```
public bool Muted { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the sound bus is muted. False otherwise.

Remarks

This flag may be overridden by the Soloed flag, i.e. if the sound bus is simultaneously muted and soloed it'll be audible.

Name

The name of the sound bus. Used for fast search in corresponding sound bank.

Declaration

```
public string Name { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.String	Name of the sound bus.

PlayLimitInterval

The time interval in seconds limiting consecutive sound bus plays.

Declaration

```
public float PlayLimitInterval { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Time interval in seconds. Zero means no limit.

Polyphony

The number of maximum allowed simultaneously playing sounds in the sound bus.

Declaration

```
public byte Polyphony { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Byte	Number of allowed sounds.

Soloed

The flag indicating if the sound bus is soloed. If set to true, all other non-solo sound buses won't be audible.

Declaration

```
public bool Soloed { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if the sound bus is soloed. False otherwise.

Remarks

This flag may override the Muted flag, i.e. if the sound bus is simultaneously muted and soloed it'll be audible.

Unfolded

The flag indicating whether the sound bank inspector should show advanced settings for the sound bus.

Declaration

```
public bool Unfolded { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if advanced settings are shown. False otherwise.

Remarks

This property is used only by the sound bank inspector and does nothing during runtime.

Volume

The volume of the sound bus.

Declaration

```
public float Volume { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Volume of the sound bus. Value must be in [0;1] range.

Methods

OnAfterDeserialize()

Prepares sound for runtime use after deserialization.

Declaration

```
public void OnAfterDeserialize()
```

Remarks

This method is automatically called by Unity during deserialization process. Don't call it manually.

OnBeforeSerialize()

Prepares sound for serialization.

Declaration

```
public void OnBeforeSerialize()
```

Remarks

This method is automatically called by Unity during serialization process. Don't call it manually.

Delegate SoundBusAddedDelegate

A sound bank callback function, called after adding sound bus to the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void SoundBusAddedDelegate(SoundBus bus);
```

Parameters

TYPE	NAME	DESCRIPTION
SoundBus	bus	A reference to a newly added sound bus.

Class SoundBusIDAttribute

The attribute class used to make an int variable in a script be restricted to a sound bus id.

Inheritance

System.Object
SoundBusIDAttribute

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class SoundBusIDAttribute : PropertyAttribute
```

Remarks

When this attribute is used, the variable will be shown as two dropdown fields in the inspector instead of the default number field.

Delegate SoundBusRemovedDelegate

A sound bank callback function, called before removing the sound bus from the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void SoundBusRemovedDelegate(SoundBus bus, int index);
```

Parameters

TYPE	NAME	DESCRIPTION
SoundBus	bus	A reference to a sound bus to be removed.
System.Int32	index	An index in corresponding Buses collection.

Delegate SoundBusRenamedDelegate

A sound bank callback function, called after changing sound bus name.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void SoundBusRenamedDelegate(SoundBus bus, int index, string oldName, string newName);
```

Parameters

TYPE	NAME	DESCRIPTION
SoundBus	bus	A reference to a sound bus.
System.Int32	index	An index in corresponding Buses collection.
System.String	oldName	An old name of the sound bus.
System.String	newName	A new name of the sound bus.

Class SoundBusRuntime

Inheritance

System.Object
SoundBusRuntime

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class SoundBusRuntime
```

Constructors

SoundBusRuntime(Transform, SoundBus)

Declaration

```
public SoundBusRuntime(Transform transform, SoundBus bus_)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	transform	
SoundBus	bus_	

Methods

GrabSound(Sound)

Declaration

```
public SoundInstance GrabSound(Sound sound)
```

Parameters

TYPE	NAME	DESCRIPTION
Sound	sound	

Returns

TYPE	DESCRIPTION
SoundInstance	

Pause()

Declaration

```
public void Pause()
```

Stop()

Declaration

```
public void Stop()
```

UnPause()

Declaration

```
public void UnPause()
```

Update(SoundBank)

Declaration

```
public void Update(SoundBank bank)
```

Parameters

TYPE	NAME	DESCRIPTION
SoundBank	bank	

Class SoundIDAttribute

The attribute class used to make an int variable in a script be restricted to a sound id.

Inheritance

System.Object

SoundIDAttribute

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class SoundIDAttribute : PropertyAttribute
```

Remarks

When this attribute is used, the variable will be shown as two dropdown fields in the inspector instead of the default number field.

Class SoundInstance

The game object with audio source component. Used for manual playback and custom mixing logic.

Inheritance

System.Object
SoundInstance

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public class SoundInstance
```

Properties

Looped

The flag indicating that the sound instance is looping. Set whether it should replay the audio clip after it finishes.

Declaration

```
public bool Looped { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if sound instance is looping. False otherwise.

Paused

The flag indicating that the sound instance is paused.

Declaration

```
public bool Paused { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if sound instance is paused. False otherwise.

Pitch

The pitch property allows controlling how high or low the tone of the audio source is.

Declaration

```
public float Pitch { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Pitch value in [3;3] range.

Playing

The flag indicating that the sound instance is playing.

Declaration

```
public bool Playing { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if sound instance is playing. False otherwise.

Position

The position of the sound instance in world space.

Declaration

```
public Vector3 Position { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector3	A world space coordinates of the sound instance.

Remarks

Non-null Target will override this property value.

Sound

The reference to a sound which will be used for playback. Changing this value allows playing different sounds.

Declaration

```
public Sound Sound { get; set; }
```

Property Value

TYPE	DESCRIPTION
Sound	A reference to a sound.

Target

The transform component to which sound instance is attached.

Declaration

```
public Transform Target { get; set; }
```

Property Value

TYPE	DESCRIPTION
Transform	A reference a transform component.

Remarks

Once set, it will override Position property value.

TimeSamples

The playback position in samples.

Declaration

```
public int TimeSamples { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	An offset in samples from the start of an audio clip.

Volume

The volume property allows controlling the overall level of sound coming to the audio source.

Declaration

```
public float Volume { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Volume value in [0;1] range.

Methods

Pause()

Pauses sound.

Declaration

```
public void Pause()
```

Play(Nullable<Int32>, Nullable<Single>, Nullable<Single>, Nullable<Single>)

Plays sound.

Declaration

```
public void Play(int? variationIndex = null, float? volume = null, float? pitch = null, float? delay = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Nullable<System.Int32>	variationIndex	Variation index. Must be within Variations range.
System.Nullable<System.Single>	volume	Volume of the sound. Value must be in [0;1] range.
System.Nullable<System.Single>	pitch	Pitch of the sound. Value must be in [-3;3] range.
System.Nullable<System.Single>	delay	Delay of the sound. Value must be greater or equal to zero.

Remarks

Audio source position will be set to (0, 0, 0). Make sure Sound.SpatialBlend is zero, otherwise it may not be mixed correctly.

Non-null variation index parameter value will override SoundVariation.Volume value.

Non-null volume parameter value will override SoundVariation.Volume value.

Non-null pitch parameter value will override SoundVariation.Pitch value.

Non-null delay parameter value will override SoundVariation.Delay value.

Play3D(Vector3, Nullable<Int32>, Nullable<Single>, Nullable<Single>, Nullable<Single>)

Plays sound in 3D space.

Declaration

```
public void Play3D(Vector3 position, int? variationIndex = null, float? volume = null, float? pitch = null,
float? delay = null)
```

Parameters

TYPE	NAME	DESCRIPTION
Vector3	position	Position of the sound.
System.Nullable<System.Int32>	variationIndex	Variation index. Must be within Variations range.
System.Nullable<System.Single>	volume	Volume of the sound. Value must be in [0;1] range.
System.Nullable<System.Single>	pitch	Pitch of the sound. Value must be in [-3;3] range.

TYPE	NAME	DESCRIPTION
System.Nullable<System.Single>	delay	Delay of the sound. Value must be greater or equal to zero.

Remarks

Non-null variation index parameter value will override SoundVariation.Volume value.

Non-null volume parameter value will override SoundVariation.Volume value.

Non-null pitch parameter value will override SoundVariation.Pitch value.

Non-null delay parameter value will override SoundVariation.Delay value.

Stop()

Stops sound.

Declaration

```
public void Stop()
```

UnPause()

Resumes sound.

Declaration

```
public void UnPause()
```

Class SoundManager

The main class for sound playback and bank management.

Inheritance

System.Object
SoundManager

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public static class SoundManager
```

Properties

Banks

The collection of all registered sound banks.

Declaration

```
public static ReadOnlyCollection<SoundBank> Banks { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.ObjectModel.ReadOnlyCollection<SoundBank>	A reference to a read-only collection of sound banks.

PrimaryBank

The primary sound bank that will be searched first in case of name collisions.

Declaration

```
public static SoundBank PrimaryBank { get; set; }
```

Property Value

TYPE	DESCRIPTION
SoundBank	A reference to a primary sound bank.

Methods

DeregisterBank(SoundBank)

Deregisters existing sound bank.

Declaration

```
public static bool DeregisterBank(SoundBank bank)
```

Parameters

TYPE	NAME	DESCRIPTION
SoundBank	bank	A reference to a sound bank to deregister.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if sound bank was succesfully deregistered. False otherwise.

GetSound(ID)

Searches for the specified sound with a matching unique ID.

Declaration

```
public static Sound GetSound(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID referring to the sound.

Returns

TYPE	DESCRIPTION
Sound	A reference to a sound, if found. Null reference otherwise.

GetSound(String)

Searches for the specified sound with a matching name.

Declaration

```
public static Sound GetSound(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.

Returns

TYPE	DESCRIPTION
Sound	A reference to a sound, if found. Null reference otherwise.

Remarks

If multiple banks have sounds with a matching name, the primary sound bank will be checked first. Within a bank, the first occurrence of found sound will be used.

GetSoundBus(ID)

Searches for the specified sound bus with a matching unique ID.

Declaration

```
public static SoundBus GetSoundBus(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID referring to the sound bus.

Returns

TYPE	DESCRIPTION
SoundBus	A reference to a sound bus, if found. Null reference otherwise.

GetSoundBus(String)

Searches for the specified sound bus with a matching name.

Declaration

```
public static SoundBus GetSoundBus(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound bus.

Returns

TYPE	DESCRIPTION
SoundBus	A reference to a sound bus, if found. Null reference otherwise.

Remarks

If multiple banks have sound buses with a matching name, the primary sound bank will be checked first. Within a bank, the first occurrence of found sound bus will be used.

GrabSound()

Grabs an empty sound instance from the sound pool. Used for manual playback and custom mixing logic.

Declaration

```
public static SoundInstance GrabSound()
```

Returns

TYPE	DESCRIPTION
SoundInstance	A reference to an empty sound instance.

Remarks

This method may increase the size of the sound pool causing additional memory allocations.

When a sound instance is not needed anymore, use ReleaseSound(SoundInstance) to return it back to the sound pool.

GrabSound(ID)

Grabs a sound instance from the sound pool. Used for manual playback and custom mixing logic.

Declaration

```
public static SoundInstance GrabSound(ID id)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID referring to the sound.

Returns

TYPE	DESCRIPTION
SoundInstance	A reference to a sound instance.

Remarks

If multiple banks have sounds with a matching name, the primary sound bank will be checked first. Within a bank, the first occurrence of found sound will be used.

This method may increase the size of the sound pool causing additional memory allocations.

When a sound instance is not needed anymore, use ReleaseSound(SoundInstance) to return it back to the sound pool.

GrabSound(String)

Grabs a sound instance from the sound pool. Used for manual playback and custom mixing logic.

Declaration

```
public static SoundInstance GrabSound(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.

Returns

TYPE	DESCRIPTION
SoundInstance	A reference to a sound instance.

Remarks

If multiple banks have sounds with a matching name, the primary sound bank will be checked first. Within a bank, the first occurrence of found sound will be used.

This method may increase the size of the sound pool causing additional memory allocations.

When a sound instance is not needed anymore, use ReleaseSound(SoundInstance) to return it back to the sound pool.

Pause()

Pauses all playing sounds.

Declaration

```
public static void Pause()
```

Remarks

This method will also stop all sounds instances returned from GrabSound() or GrabSound(String).

Play(ID, Nullable<Int32>, Nullable<Single>, Nullable<Single>, Nullable<Single>)

Plays one-shot sound.

Declaration

```
public static void Play(ID id, int? variationIndex = null, float? volume = null, float? pitch = null, float?
delay = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID referring to the sound.
System.Nullable<System.Int32>	variationIndex	Variation index. Must be within Variations range.
System.Nullable<System.Single>	volume	Volume of the sound. Value must be in [0;1] range.
System.Nullable<System.Single>	pitch	Pitch of the sound. Value must be in [-3;3] range.

TYPE	NAME	DESCRIPTION
System.Nullable<System.Single>	delay	Delay of the sound. Value must be greater or equal to zero.

Remarks

Audio source position will be set to (0, 0, 0). Make sure Sound.SpatialBlend is zero, otherwise it may not be mixed correctly.

Non-null variation index parameter value will override SoundVariation.Volume value.

Non-null volume parameter value will override SoundVariation.Volume value.

Non-null pitch parameter value will override SoundVariation.Pitch value.

Non-null delay parameter value will override SoundVariation.Delay value.

Play(String, Nullable<Int32>, Nullable<Single>, Nullable<Single>, Nullable<Single>)

Plays one-shot sound.

Declaration

```
public static void Play(string name, int? variationIndex = null, float? volume = null, float? pitch = null,
float? delay = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.
System.Nullable<System.Int32>	variationIndex	Variation index. Must be within Variations range.
System.Nullable<System.Single>	volume	Volume of the sound. Value must be in [0;1] range.
System.Nullable<System.Single>	pitch	Pitch of the sound. Value must be in [-3;3] range.
System.Nullable<System.Single>	delay	Delay of the sound. Value must be greater or equal to zero.

Remarks

If multiple banks have sounds with a matching name, the primary sound bank will be checked first. Within a bank, the first occurrence of found sound will be used.

Audio source position will be set to (0, 0, 0). Make sure Sound.SpatialBlend is zero, otherwise it may not be mixed correctly.

Non-null variation index parameter value will override SoundVariation.Volume value.

Non-null volume parameter value will override SoundVariation.Volume value.

Non-null pitch parameter value will override SoundVariation.Pitch value.

Non-null delay parameter value will override SoundVariation.Delay value.

Play3D(ID, Vector3, Nullable<Int32>, Nullable<Single>, Nullable<Single>, Nullable<Single>)

Plays one-shot sound in 3D space.

Declaration

```
public static void Play3D(ID id, Vector3 position, int? variationIndex = null, float? volume = null, float?
pitch = null, float? delay = null)
```

Parameters

TYPE	NAME	DESCRIPTION
ID	id	ID referring to the sound.
Vector3	position	Position of the sound.
System.Nullable<System.Int32>	variationIndex	Variation index. Must be within Variations range.
System.Nullable<System.Single>	volume	Volume of the sound. Value must be in [0;1] range.
System.Nullable<System.Single>	pitch	Pitch of the sound. Value must be in [-3;3] range.
System.Nullable<System.Single>	delay	Delay of the sound. Value must be greater or equal to zero.

Remarks

Non-null variation index parameter value will override SoundVariation.Volume value.

Non-null volume parameter value will override SoundVariation.Volume value.

Non-null pitch parameter value will override SoundVariation.Pitch value.

Non-null delay parameter value will override SoundVariation.Delay value.

Play3D(String, Vector3, Nullable<Int32>, Nullable<Single>, Nullable<Single>, Nullable<Single>)

Plays one-shot sound in 3D space.

Declaration

```
public static void Play3D(string name, Vector3 position, int? variationIndex = null, float? volume = null,
float? pitch = null, float? delay = null)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	Name of the sound.

TYPE	NAME	DESCRIPTION
Vector3	position	Position of the sound.
System.Nullable<System.Int32>	variationIndex	Variation index. Must be within Variations range.
System.Nullable<System.Single>	volume	Volume of the sound. Value must be in [0;1] range.
System.Nullable<System.Single>	pitch	Pitch of the sound. Value must be in [-3;3] range.
System.Nullable<System.Single>	delay	Delay of the sound. Value must be greater or equal to zero.

Remarks

If multiple banks have sounds with a matching name, the primary sound bank will be checked first. Within a bank, the first occurrence of found sound will be used.

Non-null variation index parameter value will override SoundVariation.Volume value.

Non-null volume parameter value will override SoundVariation.Volume value.

Non-null pitch parameter value will override SoundVariation.Pitch value.

Non-null delay parameter value will override SoundVariation.Delay value.

RegisterBank(SoundBank)

Registers new sound bank.

Declaration

```
public static bool RegisterBank(SoundBank bank)
```

Parameters

TYPE	NAME	DESCRIPTION
SoundBank	bank	A reference to a sound bank to register.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if sound bank was succesfully registered. False otherwise.

ReleaseSound(SoundInstance)

Releases sound instance and return it back to the sound pool.

Declaration

```
public static bool ReleaseSound(SoundInstance instance)
```

Parameters

TYPE	NAME	DESCRIPTION
SoundInstance	instance	A reference to a sound instance.

Returns

TYPE	DESCRIPTION
System.Boolean	True, if the sound instance was successfully returned to sound pool. False otherwise.

Remarks

Once the sound instance is returned back to a sound pool, it's possible to reuse it again by calling GrabSound() or GrabSound(String).

Stop()

Stops all playing sounds.

Declaration

```
public static void Stop()
```

Remarks

This method will also stop all sounds instances returned from GrabSound() or GrabSound(String).

UnPause()

Resumes all paused sounds.

Declaration

```
public static void UnPause()
```

Remarks

This method will also resume all sounds instances returned from GrabSound() or GrabSound(String).

Delegate SoundRemovedDelegate

A sound bank callback function, called before removing the sound from the bank.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void SoundRemovedDelegate(Sound sound, int index);
```

Parameters

TYPE	NAME	DESCRIPTION
Sound	sound	A reference to a sound to be removed.
System.Int32	index	An index in corresponding Sounds collection.

Delegate SoundRenamedDelegate

A sound bank callback function, called after changing sound name.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void SoundRenamedDelegate(Sound sound, int index, string oldName, string newName);
```

Parameters

TYPE	NAME	DESCRIPTION
Sound	sound	A reference to a sound.
System.Int32	index	An index in corresponding Sounds collection.
System.String	oldName	An old name of the sound.
System.String	newName	A new name of the sound.

Class SoundVariation

The persistent storage for sound effect audio data.

Inheritance

System.Object
SoundVariation

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: Stem

Assembly: Stem.dll

Syntax

```
[Serializable]
public class SoundVariation
```

Properties

Clip

The audio clip with audio data. Please refer to Unity Scripting Reference for details.

Declaration

```
public AudioClip Clip { get; set; }
```

Property Value

TYPE	DESCRIPTION
AudioClip	A reference to an audio clip.

Delay

The delay of the sound variation.

Declaration

```
public float Delay { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	FixedDelay value if RandomizeDelay is false. Otherwise random delay value from RandomDelay range.

FixedDelay

The fixed delay of the sound variation.

Declaration

```
public float FixedDelay { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Fixed delay in seconds of the sound variation. Value must be greater or equal to zero.

FixedPitch

The fixed pitch of the sound variation.

Declaration

```
public float FixedPitch { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Fixed pitch of the souhnd variation. Value must be in [-3;3] range.

FixedVolume

The fixed volume of the sound variation.

Declaration

```
public float FixedVolume { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Single	Fixed volume of the sound variation. Value must be in [0;1] range.

Name

The name of the sound variation.

Declaration

```
public string Name { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	Name of the current audio clip being used. Null reference otherwise.

Pitch

The pitch of the sound variation.

Declaration

```
public float Pitch { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	FixedPitch value if RandomizePitch is false. Otherwise random pitch value from RandomPitch range.

RandomDelay

The random delay range of the sound variation.

Declaration

```
public Vector2 RandomDelay { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector2	Random delay range of the sound variation. Vector components store range boundaries (x - min, y - max).

RandomizeDelay

The flag indicating which property is used for Delay.

Declaration

```
public bool RandomizeDelay { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if RandomDelay is used. Otherwise FixedDelay is used.

RandomizePitch

The flag indicating which property is used for Pitch.

Declaration

```
public bool RandomizePitch { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if RandomPitch is used. Otherwise FixedPitch is used.

RandomizeVolume

The flag indicating which property is used for Volume.

Declaration

```
public bool RandomizeVolume { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	True, if RandomVolume is used. Otherwise FixedVolume is used.

RandomPitch

The random pitch range of the sound variation.

Declaration

```
public Vector2 RandomPitch { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector2	Random pitch range of the sound variation. Vector components store range boundaries (x - min, y - max).

RandomVolume

The random volume range of the sound variation.

Declaration

```
public Vector2 RandomVolume { get; set; }
```

Property Value

TYPE	DESCRIPTION
Vector2	Random volume range of the sound variation. Vector components store range boundaries (x - min, y - max).

Volume

The volume of the sound variation.

Declaration

```
public float Volume { get; }
```

Property Value

TYPE	DESCRIPTION
System.Single	FixedVolume value if RandomizeVolume is false. Otherwise random volume value from RandomVolume range.

Delegate TrackChangedDelegate

A music callback function, called when the music player transitions to a new track.

Namespace: Stem

Assembly: Stem.dll

Syntax

```
public delegate void TrackChangedDelegate(MusicPlayer player, PlaylistTrack oldTrack, PlaylistTrack newTrack);
```

Parameters

TYPE	NAME	DESCRIPTION
MusicPlayer	player	A reference to a music player.
PlaylistTrack	oldTrack	An old track that the music player was playing.
PlaylistTrack	newTrack	A new track that the music player will play next.