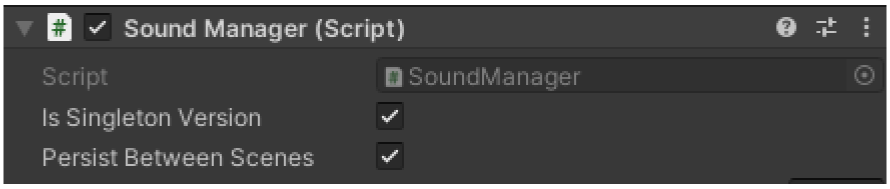
How to use this stuff

Sound Manager

Is Singleton Version makes this object occupy the *SoundManager.current* publilc static property. This will also prevent any other SoundManager objects with this property set to true from being loaded/created while this one still exists.

Persist between scenes will simply prevent the object being destroyed upon scene changes. This would allow you to carry music between scenes without it stopping, for example.


Sound Manager (Script)
Script: SoundManager
Is Singleton Version: ✓
Persist Between Scenes: ✓

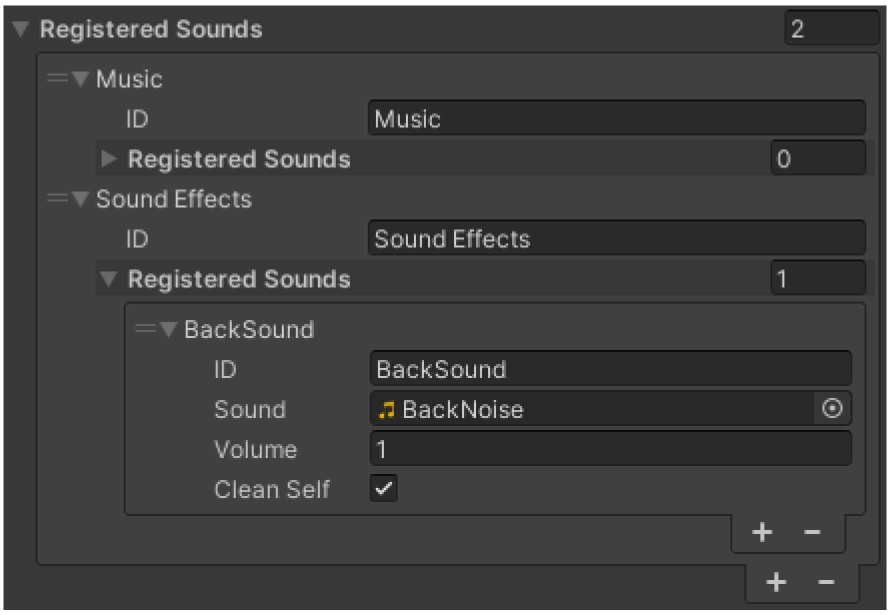
Registered Sounds should contain all the sounds you want to use through this audio manager. These are split into multiple lists so you can organise them, should you want to use a large number of sounds with one manager (such as a music jukebox if you have a lot of tracks, or have many types of sound effects used by one object).
In each of the sub-lists you create, you can add sounds and change default properties of the created audio sources, such as the volume of sounds, which may be useful if your source file is abnormally loud.

ID of an object is the string value used in code to reference this sound.

Sound is the audio clip of the sound itself.

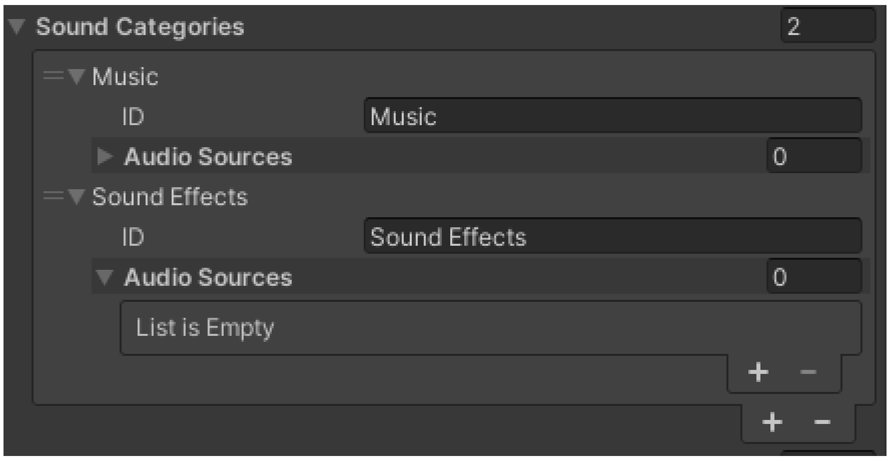
Volume is a value **between 0 and 1** which affects the volume of the sound when it is called.

Cleans Self reflects whether the sound's object automatically destroys itself after the sound has played.



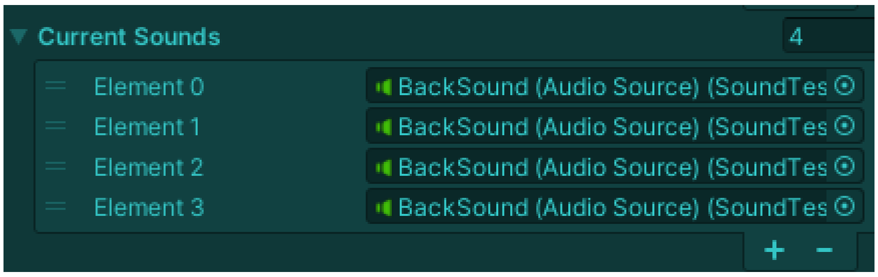
Registered Sounds 2
Music
ID Music
Registered Sounds 0
Sound Effects
ID Sound Effects
Registered Sounds 1
BackSound
ID BackSound
Sound BackNoise
Volume 1
Clean Self ✓

Sound Categories are groups which your sounds are organised under. They contain an encapsulated volume variable and a list of sounds which fit under them. Their **ID** is assigned to be called later when creating sounds or managing multiple at a time. The Audio Sources list under it simply displays during runtime the audio sources currently active under this category.



Sound Categories 2
Music
ID Music
Audio Sources 0
Sound Effects
ID Sound Effects
Audio Sources 0
List is Empty

Current Sounds displays during runtime all audio sources managed by this object, including those not assigned a specific category.



Current Sounds 4

Element 0	BackSound (Audio Source) (SoundTes
Element 1	BackSound (Audio Source) (SoundTes
Element 2	BackSound (Audio Source) (SoundTes
Element 3	BackSound (Audio Source) (SoundTes

To create sounds with the sound manager class, you can do variations of the PlaySound() public method (see documentation). It's also useful to note that this method returns the created AudioSource component, so any additional tweaking you may want to do to the created object is easy.

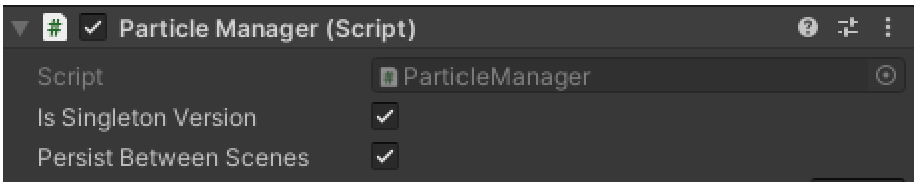
You should not need to hesitate to manually destroy created audio source objects and instance-based managers either, as they clean up after themselves in their OnDestroy() methods before they go.

Particle Manager

Note: This is more or less identical to the SoundManager class, as they work in basically the same way

Is Singleton Version makes this object occupy the *ParticleManager.current* publilc static property. This will also prevent any other ParticleManager objects with this property set to true from being loaded/created while this one still exists.

Persist between scenes will simply prevent the object being destroyed upon scene changes.



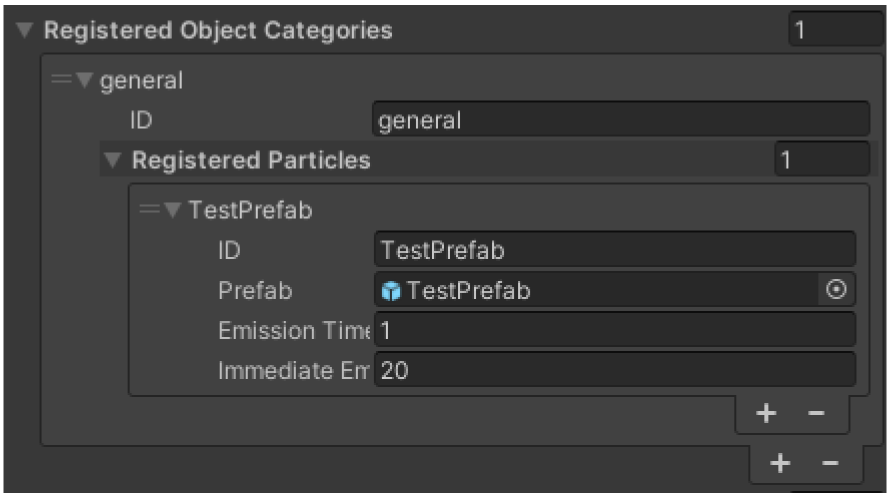
Particle Manager (Script)

Script	ParticleManager
Is Singleton Version	✓
Persist Between Scenes	✓

Registered Particles *inside of* ***Registered Object Categories*** should contain all the prefabs you want to use through this manager class. These are split into multiple lists so you can organise them,

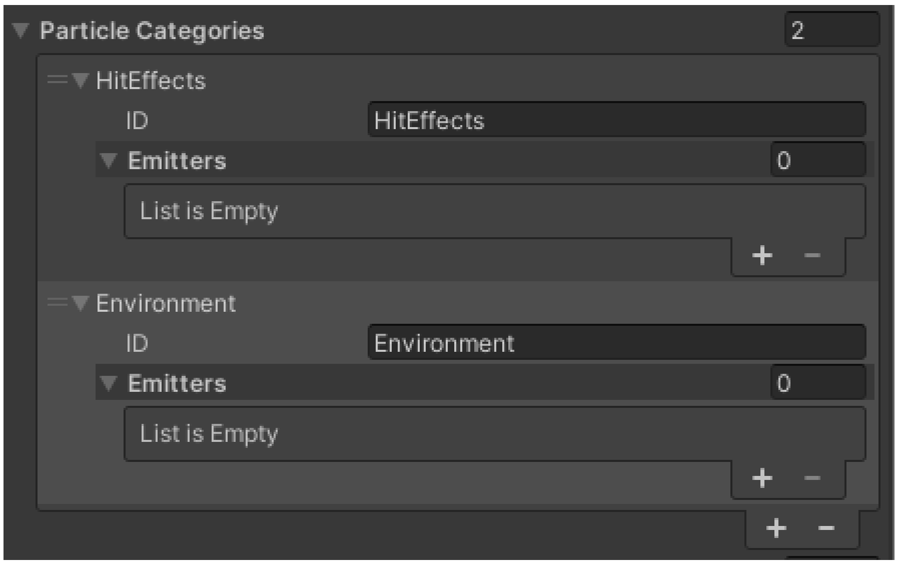
should you want to use a large number of prefabs with one manager.
Much like the sound manager, you can change some properties of spawned objects here instead of having to do it directly in code every time.

> **ID** of an object is the string value used in code to reference this prefab.
>
> **Prefab** is a reference to the GameObject you are spawning.
>
> **Emission Time** is how long a particle system will emit particles before stopping. When the last of the particles disappears, the object will be destroyed.
>
> **Immediate Emission Count** is how many particles are immediately emitted after the object is created.

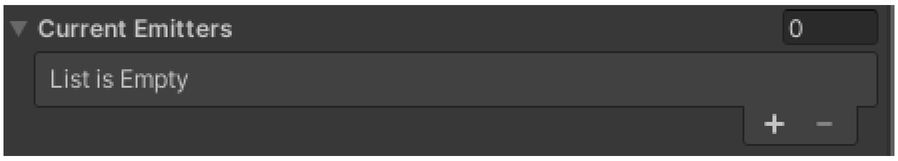


Registered Object Categories 1
general
ID general
Registered Particles 1
TestPrefab
ID TestPrefab
Prefab TestPrefab
Emission Time 1
Immediate Em 20

Particle Categories are groups which your prefabs are organised under. They contain a list of objects under them. Their **ID** is assigned to be called later when creating objects, or managing multiple at a time. You can see any objects spawned under these categories appear (and disappear when removed) during runtime.

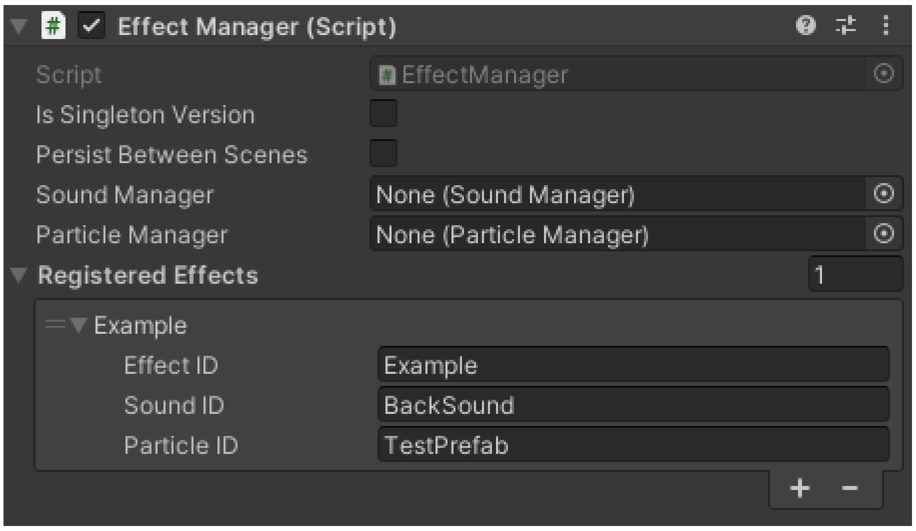

Particle Categories 2
HitEffects
ID HitEffects
Emitters 0
List is Empty
Environment
ID Environment
Emitters 0
List is Empty

Current Emitters displays during runtime all objects managed by this particle manager, including those not assigned a specific category when created.


Current Emitters 0
List is Empty

Effect Manager

Also similar to the prior managers, but much simpler


Effect Manager (Script)
Script EffectManager
Is Singleton Version
Persist Between Scenes
Sound Manager None (Sound Manager)
Particle Manager None (Particle Manager)
Registered Effects 1
Example
Effect ID Example
Sound ID BackSound
Particle ID TestPrefab

The effect manager allows you to pair up sound and effect IDs to execute them together with one line of code. It also has the same singleton-instance dual functionality as the other classes.

Do keep in mind the sound manager and particle manager properties need to be set to whichever instances you are using before you use any effects.

It may be advisable to use the systems independently for a finer control of both individual components, but this is still a viable alternative.

Class Documentation

How to use the systems in code.

Sound Manager

Derives from MonoBehaviour. Class name is SoundManager.

Public Properties

[bool] isSingletonVersion	This object is becomes publicly accessible through *SoundManager.current when loaded.* *This sound manager does not load if another sound manager with this property enabled already exists.*
[Static SoundManager] current	Public access to the current singleton soundmanager. There can not be multiple of these at once.
[Static bool] singletonReady	Reflects if a singleton sound manager currently exists.
[bool] persistBetweenScenes	Enabling this will prevent the audio manager being destroyed upon scene change
[Static float] masterVolume	Master volume multiplier. 1 by default.
[List<SoundCategory>] soundCategories	Categories which sounds can be sorted under when created. Serialized and editable in the inspector. SoundCategory is a custom class defined in the SoundManager script for storage purporses.
[List <AudioSource>] currentSounds	All AudioSources currently in the scene which are managed by this SoundManager.

Public Methods

[Returns AudioSource] PlaySound(string soundID) PlaySound(string soundID, SoundCategory category) PlaySound(string soundID, vector3 position, SoundCategory category) PlaySound(string soundID, SoundCategory category) PlaySound(string soundID, GameObject emitter, SoundCategory category) PlaySound(string soundID, GameObject emitter)	Creates an object with an AudioSource and SoundObject script attached. The properties of these will reflect what was defined by the user in the SoundManager's

	sound registration for the given ID. Multiple overloads available, see left. These allow the user to create a sound at the given position and/or under a given category, as well as giving the user the opttion to automatically parent sounds to GameObjects, which may be useful if the source of a sound moves.
[Returns SoundCategory] GetCategoryFromID(string categoryID)	Gets you the category object of the specified ID under the SoundManager object if it exists. Useful for PlaySound() if you're using categories.
[Returns Void] PlayAllSounds() PlayAllSounds(List<AudioSource> sourceList)	Plays all sounds managed by this object. Can also be provided a list of audio sources through an overload instead.
[Returns Void] PauseAllSounds() PauseAllSounds(List<AudioSource> sourceList)	Pauses all sounds managed by this object. Can also be provided a list of audio sources through an overload instead.
[Returns Void] RemoveAllSounds() RemoveAllSounds(List<AudioSource> sourceList)	Removes/Destroys all sounds managed by this object. Can also be provided a list of audio sources through an overload instead.

Particle Manager

Derives from MonoBehaviour. Class name is ParticleManager.

Public Properties

[bool] isSingletonVersion	This object is becomes publicly accessible through *ParticleManager.current when loaded.* *This particle manager does not load if another particle manager with this property enabled already exists.*
[Static ParticleManager] current	Public access to the current singleton ParticleManager. There can not be multiple of these at once.
[Static bool] singletonReady	Reflects if a singleton particle manager currently exists.
[bool] persistBetweenScenes	Enabling this will prevent the object being destroyed upon scene change

[List<SoundCategory>] particleCategories	Categories which objects can be sorted under when created. Serialized and editable in the inspector. ParticleCategory is a custom class defined in the ParticleManager script for storage purporses.
[List <GameObject>] currentEmitters	All GameObjects currently in the scene which are managed by this ParticleManager.

Public Methods

[Returns GameObject] CreateParticle(string particleID) CreateParticle(string soundID, SoundCategory category) CreateParticle (string soundID, vector3 position, SoundCategory category) CreateParticle (string soundID, SoundCategory category) CreateParticle (string soundID, GameObject emitter, SoundCategory category) CreateParticle (string soundID, GameObject emitter)	Instantiates the specified registered prefab, attaching a ParticleObject script attached. The properties of these will reflect what was defined by the user in the ParticleManager's registration for the given ID. Multiple overloads available, see left. These allow the user to create an object at the given position and/or under a given category, as well as giving the user the opttion to automatically parent sounds to GameObjects, which may be useful if the source of a sound moves.
[Returns ParticleCategory] GetCategoryFromID(string categoryID)	Gets you the category object of the specified ID under the ParticleManager object if it exists. Useful for CreateParticle() if you want to use categories.
[Returns Void] RemoveAll) RemoveAll(List<GameObject> sourceList)	Removes/Destroys all GameObjects managed by this object. Can also be provided a list of GameObjects through an overload instead.

Effect Manager

Derives from MonoBehaviour. Class name is EffectManager.

Public Properties

[bool] isSingletonVersion	This object is becomes publicly accessible through *ParticleManager.current when loaded.* *This particle manager does not load if another particle manager with this property enabled already exists.*
[Static ParticleManager] current	Public access to the current singleton ParticleManager. There can not be multiple of these at once.

[Static bool] singletonReady	Reflects if a singleton particle manager currently exists.
[bool] persistBetweenScenes	Enabling this will prevent the object being destroyed upon scene change
[List<SoundCategory>] particleCategories	Categories which objects can be sorted under when created. Serialized and editable in the inspector. ParticleCategory is a custom class defined in the ParticleManager script for storage purporses.
[List <GameObject>] currentEmitters	All GameObjects currently in the scene which are managed by this ParticleManager.

Public Methods

[Returns CreatedEffect] CreateEffect(string effectID)
CreateEffect(string effectID, vector3 position)
CreateEffect(string effectID, GameObject emitter)
	Creates a new instance of a CreatedEffect custom class.
	This class contains public properties which reference the created particle GameObject (CreatedEffect.particle), and the created sound's AudioSource (CreatedEffect.sound).
	Overloads can be used to define and original position/parent for the created objects