

Advanced Scene Manager

Advanced Scene Manager

Welcome to the Advanced Scene Manager wiki!

This wiki is available both [online](#) and [offline](#)

Note: the offline version packaged with asset store releases may not be up-to-date, since documentation may be updated after release.

Last updated at: 31/08/2021

Contact

Want to get into contact with us? Report issues?

[Github issues](#)

support@lazy.solutions

discord.gg/pnRn6zeFEJ

>Note: We accept bug reports and offer help on all three channels. Which one you wish to use, is up to your preference!

Models

[Scene](#)

[SceneCollection](#)

[Profile](#)

Frontend

[SceneManagerWindow](#)

[SceneOverviewWindow](#)

Backend

[SceneManager](#)

[SceneOperation](#)

[SceneAction](#)

[AssetManagement](#)

[OpenSceneInfo](#)

[PreloadedSceneHelper](#)

Utilities

[SceneUtility](#)

[SceneCollectionUtility](#)

[SceneHelper](#)

[CoroutineUtility](#)

[LoadingScreenUtility](#)

[PersistentUtility](#)

[CanvasSortOrderUtility](#)

[AssetRefreshUtility](#)

[Scene merge, split](#)

[HierarchyGUIUtility](#)

[SceneDataUtility](#)

[GuidReferenceUtility](#)

[LockUtility](#)

[ISceneOpened, ISceneClosed, ICollectionOpen, ICollectionClose](#)

[CallbackUtility, Callback analyzer](#)

Misc

[Addressables support](#)

[SceneOverviewUI](#)

[DefaultPauseScreen](#)

[ISceneObject](#)

[Loading screens](#)

[Splash screen](#)

[Support packages](#)

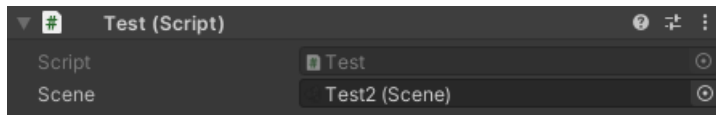
[Package Manager](#)
[Cross-scene references](#)
[Guides](#)
[Quick start](#)
[Loading screen](#)
[Splash screen](#)

Scene

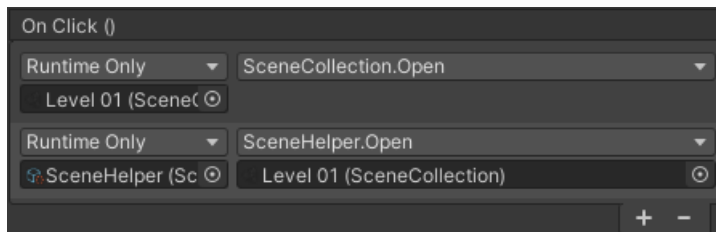
What is a Scene in Advanced Scene Manager?

An ASM scene is our representation of a SceneAsset Unity scene. All SceneAssets in the project will have a corresponding Scene ScriptableObject generated for it. Aside from the slight confusion that might arise at first, this approach has numerous advantages:

- Drag and drop references in inspector:



- UnityEvent support (it might be easier to use SceneHelper as target though, since that will help filter the object picker to only display scenes):



- More robust:

Consider this:

```
using UnityEngine.SceneManagement;
using AdvancedSceneManager.Models;

//-----Unity, Out-of-the-box-----

//Path: Loses reference when scene is moved, renamed, deleted...
//Name: Loses reference when scene renamed, deleted, also
//      conflicts occur when duplicate names exist...
public string scene;

//Loses reference when build index changes...
public int sceneIndex;

public void OpenSceneUnity()
{
    SceneManager.LoadScene(scene);
    SceneManager.LoadScene(sceneIndex);
}
```

Which could be transformed into this:

```
//-----Advanced Scene Manager-----

//Is automatically updated when its associated SceneAsset is moved,
//renamed and deleted.
//Does not rely on build index.
public Scene scene;

public void OpenSceneASM()
{
    scene.Open();
}
```

Properties

isIncluded { get; }

Returns whatever the scene is currently included in build. This also takes into account whatever a scene is forced to be included, which would be any scenes that have been added to a collection, or is set as splash screen.

path { get; }

The path to the Unity scene in the project.

assetID { get; }

The AssetDatabase id of the SceneAsset.

tag { get; }

The Tag of this scene. Tags can be defined on either profiles or collections, where collections take priority over profile.

isActive { get; }

Returns whatever this scene is currently the active scene. Use Activate() to activate a scene.

Methods

static OpenSceneInfo Find(string name, SceneCollection inCollection = null, Profile inProfile = null)

Finds the scene with the specified name.

static OpenSceneInfo[] FindAll(string name, SceneCollection inCollection = null, Profile inProfile = null)

Finds the scenes with the specified name.

OpenSceneInfo GetOpenSceneInfo()

Finds last opened instance of the specified scene, returns null if no instances exist.

Open()

Opens the scene as standalone.

OpenSingle()

Closes all existing scenes and collection, and opens this scene as standalone.

Close()

Closes the first instance of this scene that is open

Reopen()

Closes the first instance of the scene and then opens it again

Toggle()

Toggles the scene on or off, depending on whatever it is open or not.

Toggle(bool enabled)

Toggles the scene on or off, depending on enabled.

SceneAsyncOperation Preload()

Preloads the scene, which loads it into memory but does not display it.

Activate()

Sets the scene as the active scene in the hierarchy.

IsOpenReturnValue IsOpen()

Returns whatever this scene is open.

(SceneCollectioncollection, bool asLoadingScreen)[] FindCollections()

Finds the collections that this scene is associated with.

SceneTag FindTag(SceneCollection collection = null)

Finds the tag for this scene on either the specified collection, or on the active profile. If collection is null, then SceneManager.collection.current will be used instead.

SceneTag GetTagFromCollection(SceneCollection collection = null)

Finds the tag for this scene on the specified collection. If collection is null, then SceneManager.collection.current will be used instead.

SceneTag GetTagFromProfile()

Finds the tag for this scene on the active profile.

The following methods can take a sceneIndex parameter which can be used to distinguish scenes that have been opened multiple times, (i.e. if first instance of the scenes is at the top of the hierarchy and the second is at the bottom, use sceneIndex: 1 for the second instance).

T FindObject(int sceneIndex = 0)

Finds the component of type T in the scene object hierarchy.

IEnumerable<T> FindObjects(sceneIndex = 0)

Finds the components of type T in the scene object hierarchy.

GameObject[] GetRootGameObjects(int sceneIndex = 0)

Gets the root game objects of this scene.

Scene Collection

What is a SceneCollection?

A SceneCollection, more commonly called collection, is a collection of scenes, that are all manipulated at the same time.

To put it simply:

When a collection is opened, then all its contained scenes will be opened.

When a collection is closed, then all its contained scenes will be closed.

Beyond this there are ways to keep scenes open or closed, using either Tags or PersistentUtility.

Loading screens are also supported out of the box for collections.

Properties

string title { get; set; }

The title of this collection.

ScriptableObject extraData { get; set; }

The extra data that is associated with this collection. Use ExtraData() to cast it to the desired type.

Scene[] scenes { get; set; }

Gets the scenes in this collection, note that some might be null if no reference is added in scene manager window.

Scene loadingScreen { get; set; }

The loading screen that is associated with this collection.

LoadingScreenUsage loadingScreenUsage { get; set; }

Specifies what loading screen to use.

Scene activeScene { get; set; }

Specifies the scene that should be activated after collection is opened.

bool openAtStart { get; set; }

Specifies whatever this collection should open on startup.

CollectionThreadPriority loadingPriority

The thread priority to use when opening this collection.

Auto: Automatically decide ThreadPriority based on if loading screen is open.

Low: ThreadPriority.Low,

BelowNormal: ThreadPriority.BelowNormal,

Normal: ThreadPriority.Normal,

High: ThreadPriority.High,

Methods

static Find(string name, bool onlyActiveProfile = true)

Finds the collection with the specified name.

Scene[] AllScenes()

Gets all scenes contained in this collection, including overridden loading screen, if set.

Profile FindProfile()

Find the Profile that this collection is associated with.

SceneTag Tag(scene scene, SceneTag setTo = null)

Returns or sets the tag of a scene in this collection.

T ExtraData() where T : ScriptableObject

Casts and returns extraData as the specified type. Returns null if invalid type.

Open()

Opens the collection.

Toggle()

Toggles the collection.

Toggle(bool enabled)

Toggles the collection. Pass a value to ensure that collection either open or closed.

Reopen()

Reopens the collection, if open.

Close()

Closes the collection, if open.

IsOpen()

Gets whatever the collection is currently open.

Profile

What is a profile?

A profile stores the settings, collections and standalone scenes (that should be included in build) in Advanced Scene Manager. Profiles are easily swappable in SceneManagerWindow and active profile is saved locally, which means that different members of a team can use their own profiles and then the project might use a profile specifically for build or release, if desired.

Events

static Action onProfileChanged

Occurs when profile changes. Profile cannot be changed in builds, and this event will as such also not be called in build.

Properties

static Profile Default { get; }

Gets the default profile. This is used as a fallback when current is null. Note that this profile is created on demand, which means that the asset id will change. If you wish to determine whatever a profile is the default, use isDefault or check if asset name is empty.

static Profile current { get; }

Gets the currently active profile. Setter is available only in editor.

bool isDefault { get; }

Gets if the profile is the default one.

IEnumerable<Scene> scenes

Gets the scenes managed by this profile. Includes both collection and standalone scenes.

Scene loadingScreen

The default loading screen for collections in this profile.

Scene splashScreen

The splash screen to display during startup.

bool createCameraForSplashScreen

Automatically create camera if no main camera found during splash screen.

bool useDefaultPauseScreen

The default pause screen, opens on esc, if enabled.

bool useFadeDuringStartup

Specifies whatever to use fade during startup, might cause flickering if turned off.

Scene[] standaloneScenes

Gets or sets standalone scenes that are set to be included to build.

SceneTag[] tagDefinitions

The layers defined in the tags tab in the scene manager window.

ThreadPriority backgroundLoadingPriority

Sets Application.backgroundLoadingPriority automatically during startup.

bool enableChangingBackgroundLoadingPriority

Enables or disables ASM automatically changing Application.backgroundLoadingPriority, this means

Profile.backgroundLoadingPriority,

SceneCollection.loadingPriority,

SceneOperation.WithLoadingPriority,

will have no effect if disabled.

Methods

`static Profile[] FindAll()`

Finds all profiles in the project.

`static Profile Find(Func<Profile, bool> predicate)`

Finds a profile that matches the predicate.

`int Order(SceneCollection collection)`

Returns the order of this collection.

`int Order(SceneCollection collection, int? newIndex = null)`

Returns and/or sets the order of this collection in the scene manager window. Only available in editor.

`SceneTag Tag(Scene scene, SceneTag setTo = null)`

Gets or sets a tag defined for a scene on this profile.

`SceneCollection CreateCollection(string name, Action initializeBeforeSave = null)`

Create a collection and add it to this profile. Only available in editor.

`Add(SceneCollection collection)`

Adds a collection to this profile. Only available in editor.

`Remove(SceneCollection collection)`

Remove a collection from this collection. Only available in editor.

`SetStandalone(Scene scene, bool enabled)`

Sets the scene as standalone in this profile. Only available in editor.

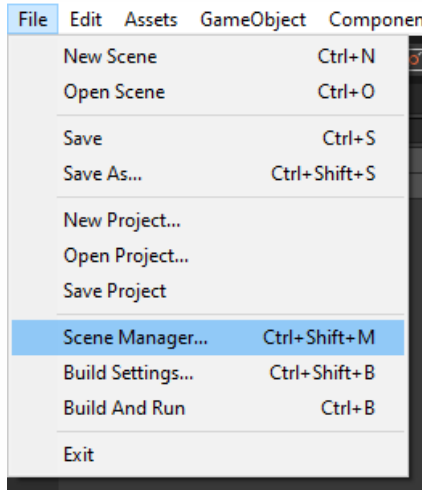
`Delete()`

Deletes this profile, and all associated collections. Prompts user for confirmation. Only available in editor.

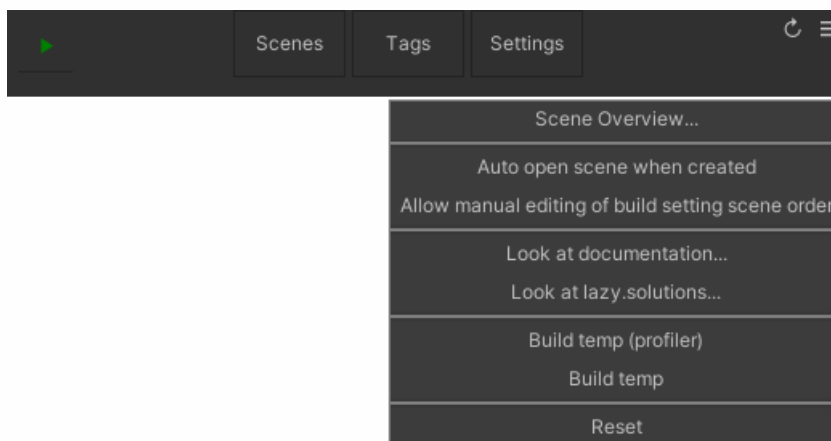
Scene Manager Window


The Scene Manager Window can be used to manage the SceneCollections (commonly referred to as collection) and scenes in a project, and also the behavior of Advanced Scene Manager.

The scene manager window can be opened through:



Header



 button enters play mode and starts startup sequence as if a build was started.

button

Scene Overview...

Opens Scene Overview Window.

Auto open scene when created

When a scene is created from a scene row in a collection, should we open it afterwards?

Allow manual editing of build settings scene order

Build order is managed by ASM and might cause issues if manual editing is turned on, but there might be certain circumstances where it is needed.

Look at documentation...

Open github wiki (this is where you are right now!)

Look at lazy.solutions...

Open our company webpage.

Build temp (profiler)

Builds and runs the project, also attaches profiler.

Build temp

Builds and runs the project.

Reset

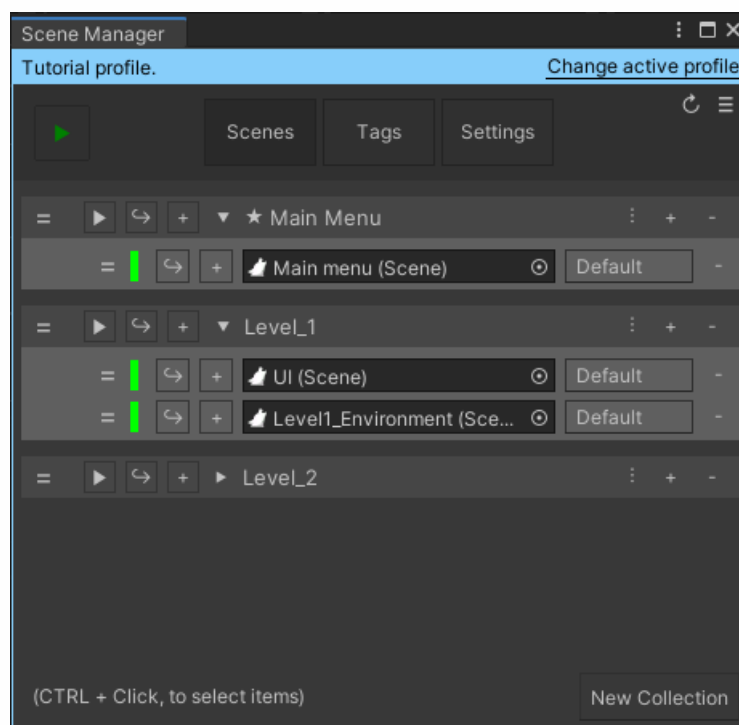
Resets Advanced Scene Manager by deleting profiles, collections, and ASM Scenes (which will be immediately generated again after).

The Scene Manager Window has three tabs:

Scenes | Tags | Settings

Scenes

The Scenes tab is where the collections of scenes are set up.

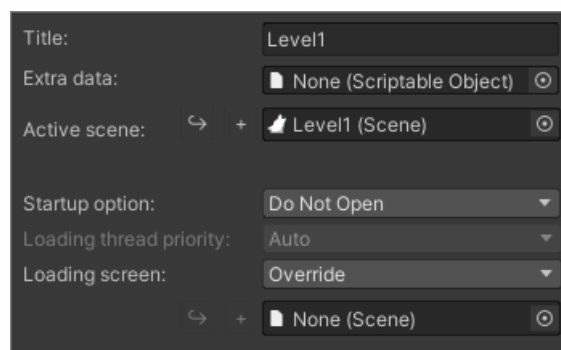


Create new collection

Press **New Collection** button to create a new collection.

Collection properties

To edit properties of a collection, press the **⋮** button on the header to open the collection menu. In the collection menu the following properties can be changed:



Title

The title of this collection.

Extra data:

A scriptable object that is to be associated with this collection, retrievable in code by `SceneCollection.ExtraData()` or `SceneCollection.extraData`.

Active scene

The scene that should be activated after collection has been opened.

Startup option

Can be one of the following values:

Do not open: Don't open this collection during startup. (default)

Open: Open this collection during startup.

Open as persistent: Open this collection during startup, and flag each scene within as persistent (aka `SceneCloseBehavior.KeepOpenAlways`).

Loading thread priority:


Automatically set `Application.LoadingThreadPriority` to specified value when this collection is opened (and is reset when done).

This field will be disabled if 'Background Loading Priority' is disabled in settings tab.

Loading Screen

The loading screen to use when opening or closing this collection.

Add or remove scenes / collection

Scenes can be added to a collection by pressing the  button on its header, Scenes and `SceneAsset` can then be dragged onto the scene fields.

The  button on a collection header or scene row can be used to remove a collection or scene.

Reorder



Hold and drag on  button and drag up and down to reorder collections or scenes.

Open buttons

Tip: Holding shift when using any of the following buttons, will force open any scenes tagged with `SceneOpenBehavior.DoNotOpenWithCollection`.

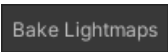
 button can be used to open a collection in play mode.

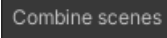
 button can be used to open a collection or scene, closing all other scenes.

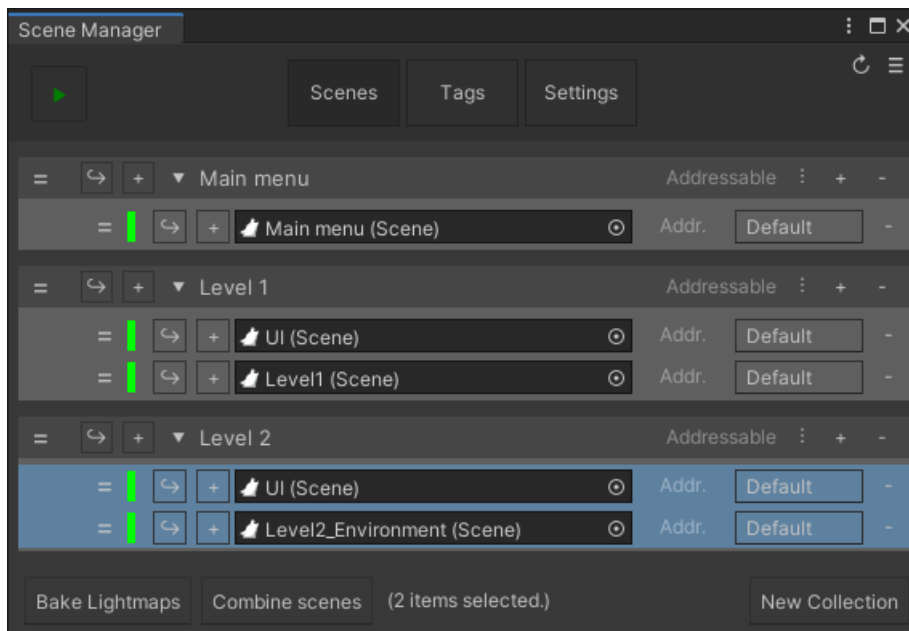
 and  buttons can be used to open or close collections or scenes additively. Collections can only be opened additively in editor, outside of play mode.

Selection, bake lightmaps, combine scenes

Scenes and collections can be selected by holding ctrl and clicking on it in the scenes tab.

When more than one scene are selected (collections count for all its containing scenes), the  button will appear, this will allow you to bake lightmaps for all the selected scenes, and when a collection is selected, all scenes within will be included.

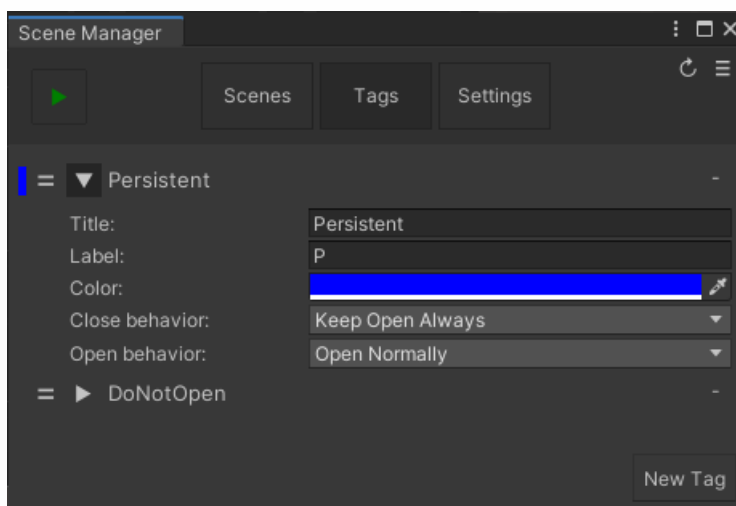
When more than one scene are selected (collections do not count), the  button will appear, this will allow you to quickly merge scenes together, note that this cannot be undone, except for using the split scene utility.



Tags

Tags can be used to specify custom open or close behavior, when opened or closed through a collection.

Press **New Tag** button to create a new tag.



Title

The title of the tag.

Label

Can be used to show a label to quickly identify scenes with this tag in the scenes tab.

Color

Can be used to show a strip of color to quickly identify scenes with this tag in the scenes tab.

Close behavior: - Close - (default) Closes scene like normal. - Keep open if next collection also contains scene - Keeps the scene open when opening a collection that also contains the scene, otherwise close. - Keep open always - Persistent, ASM will never close automatically.

Open behavior: - (default) Open normally - Open like normal - Do not open in collection - Do not open this scene automatically when the associated collection is opened

Settings

The settings tab can be used change behavior of advanced scene manager.



Profile:

The active Profile in ASM.

Options

Splash screen:

The custom splash screen to display during startup, plays after Unity's splash screens, if enabled. Provides more flexibility than Unity's splash screen since ours are merely implemented as scenes with a script using IEnumerator callback.

Startup loading screen:

The loading screen that will be used during startup.

Loading screen:

The loading screen that will be used for collections that are set to use the default loading screen.

Background loading priority:

If enabled, automatically set `Application.backgroundLoadingPriority` during startup.

Note that collections can also specify loading priority during open, but that is only temporary, and value will be reset back to this when done.

Disable this if you are using another asset which relies on managing this setting by itself.

Create camera for splash screens:

Automatically create camera for splash screens, if no main camera is found.

Use default pause screen:

ASM has a default pause screen for those times when you build and then forget that no one has created a pause screen yet, forcing you to alt-f4, then restart again. This toggles default pause screen on or off.

Enable cross-scene references:

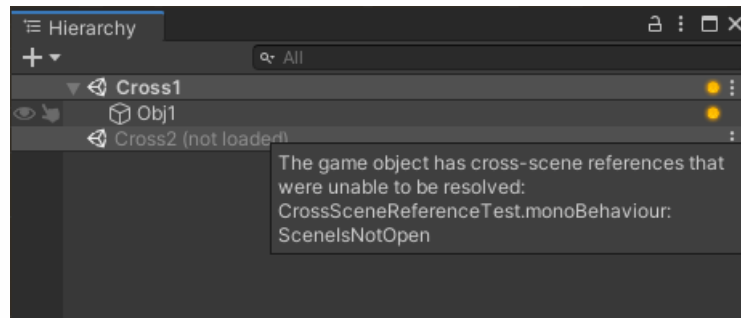
Enables or disables cross-scene references.

Appearance

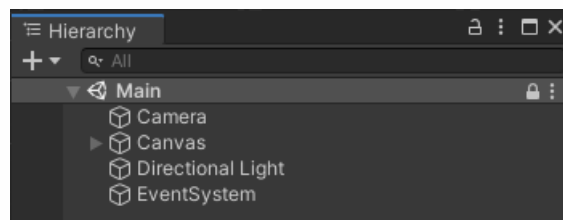
The following properties are only visible if the corresponding package is installed from the package manager, if none are installed then this entire section will be hidden.

These properties are also local to the computer / user, so feel free to modify these to your personal preference.

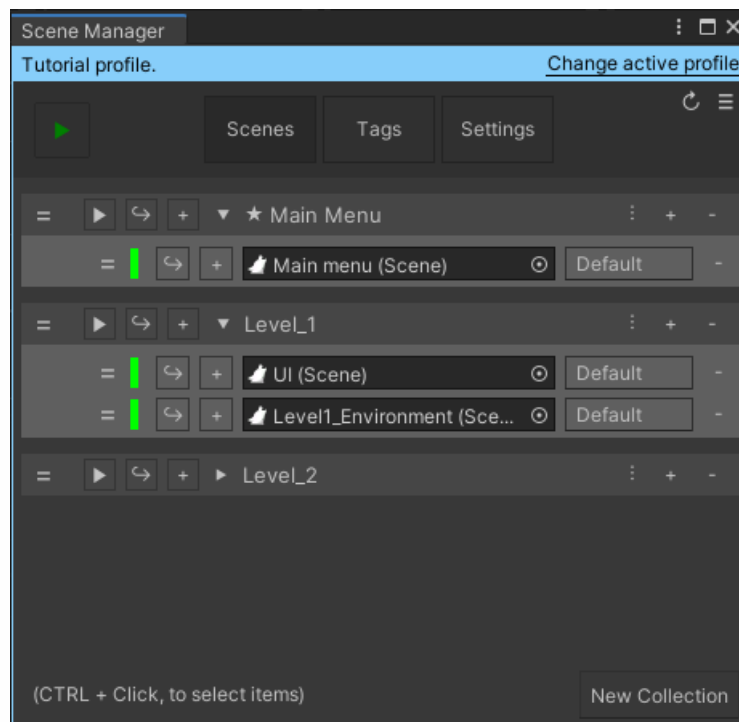
Display unresolved cross scene reference icon:



Display lock buttons:



Display addressable buttons:



Options for toggling visibility for the following also available:

- Collection play button
- Collection open button
- Collection open additive button
- Persistent scene indicator

Log

Pointless opening of collections during startup:

Enables or disables warnings during startup that a collection was opened, then closed because another collection was also set to open during startup, and no scene within was persistent.

Pointless opening of scenes during startup:

Enables or disables warnings during startup that a scene was opened, then closed when a collection was opened because scene was not persistent.

Deleting temp Build:

Enables or disables a message that indicates that the temp build was deleted after process ended, this is mostly just useful for debugging when issues cause folder to not be deleted and is a quick way to know if this feature works as intended or not.

Unresolved cross-scene reference:

Enables or disables warnings when a cross-scene reference could not be resolved when a scene was opened or closed.

Scene Overview Window

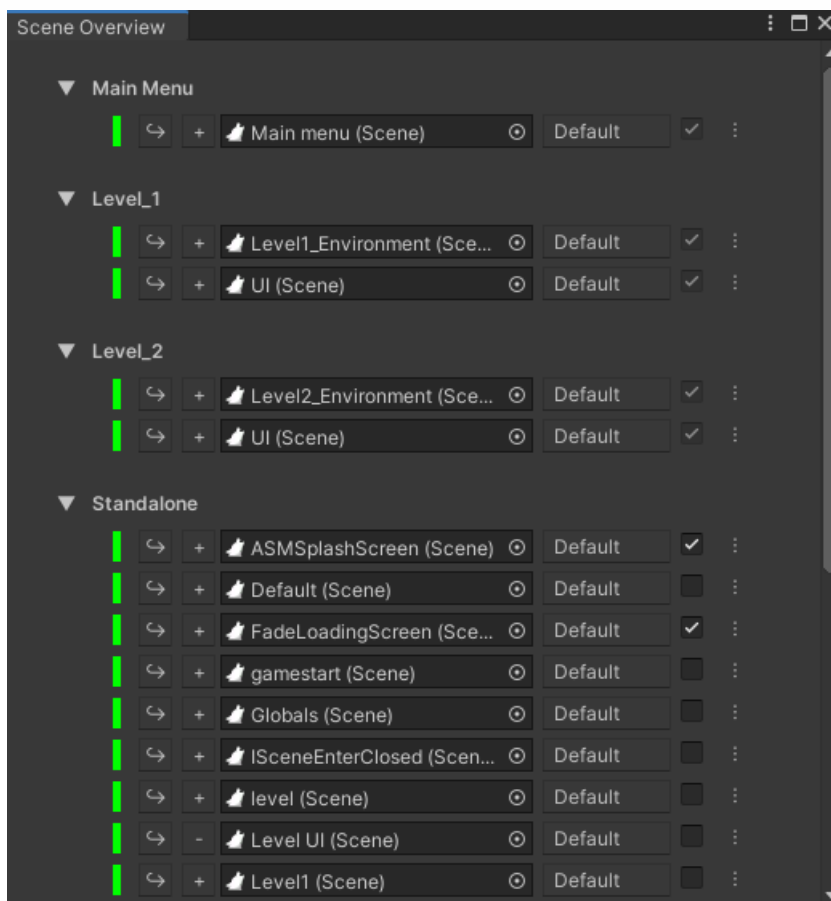
The scene overview window provides an easy overview over all scenes in the project, grouped by what collection(s) they are a part of. All scenes that are not part of a collection appear under the standalone group.

From here you can: > Open scene as either single (close all open scenes beforehand) or as additive.

Change tag for a scene in the current profile.

Set whatever a scene should be included in build. Scenes that are part of a collection will always be included, but standalone scene needs to explicitly be included here or through unity's build settings window.

Set scene as persistent in editor. This means that when a scene is opened in the editor, this scene can be automatically opened as well. For runtime, PersistentUtility can be used.



Scene Manager

In ASM we use two scene managers, `CollectionSceneManager` and `StandaloneSceneManager`, which may seem confusing at first, but it is probably more straightforward than it seems at first glance.

These two scene managers can be accessed with:

```
AdvancedSceneManager.SceneManager.collection  
AdvancedSceneManager.SceneManager.standalone
```

They both inherit from `SceneManagerBase`, which means that most logic, methods and properties are exactly the same across both scene managers, but does of course implement behaviors specific to either standalone scenes or collection scenes.

SceneManagerBase

`SceneManagerBase` is the base class for collection and standalone scene managers, and contains the core logic for scene management.

Events

`Action sceneOpened`
Occurs when a scene is opened in this scene manager.

`Action sceneClosed`
Occurs when a scene is closed in this scene manager.

Properties

`ReadOnlyCollection openScenes { get; }`
The open scenes in this scene manager.

Methods

`OpenSceneInfo Find(Scene scene)`
Finds last open instance of the specified Scene.

`OpenSceneInfo Find(Scene scene)`
Finds the open instance of the specified Scene.

`OpenSceneInfo GetLastScene()`
Gets the last opened Scene.

`bool IsOpen(OpenSceneInfo scene)`
Gets if the Scene is open.

`SceneOperation Open(Scene scene)`
Open the Scene.

`SceneOperation OpenWithoutReturnValue(Scene scene)`
Open the Scene.

`SceneOperation Reopen(OpenSceneInfo scene)`
Reopens the Scene.

`SceneOperation Close(OpenSceneInfo scene)`
Close the Scene.

`bool CanOpen(Scene scene)`
Gets if this scene manager can open the specified Scene.

SceneOperation Toggle(OpenSceneInfo scene, bool? enabled = null)

Toggles the Scene open or closed. If null, the Scene will be toggled on or off depending on whatever the Scene is open or not. Pass a value to ensure that the Scene is either open or closed.

SceneOperation EnsureOpen(Scene scene)

Ensures that the Scene is open.

Collection Scene Manager

The collection scene manager manages the scenes of the active collection. Throws `OpenSceneException` or `CloseSceneException` if a specified scene is not part of the active collection.

```
AdvancedSceneManager.SceneManager.collection
```

Events

Action opened

Called when a collection is opened.

Action closed

Called when a collection is closed.

Properties

SceneCollection current { get; }

The currently open collection.

Methods

bool CanOpen(Scene scene)

Gets whatever the scene can be opened by the current collection.

SceneOperation Open(Scene scene)

Opens a scene. Throws a `OpenSceneException` if the scene cannot be opened by the current collection.

SceneOperation Close(OpenSceneInfo scene)

Closes a scene. Throws a `CloseSceneException` if the scene is not a part of the current collection.

SceneOperation Open(SceneCollection collection, bool ignoreLoadingScreen = false)

Opens the collection.

SceneOperation Reopen()

Reopens the current collection.

SceneOperation Close()

Closes the current collection.

SceneOperation Toggle(SceneCollection collection, bool? enabled = null)

Toggles the collection. If null, collection will be toggled on or off depending on whatever collection is open or not. Pass a value to ensure that collection either open or closed.

bool IsOpen(SceneCollection collection)

Gets whatever the collection is currently open.

Standalone Scene Manager

The standalone scene manager manages scenes that are opened outside of a collection. For example an pause screen or persistent scenes.

```
AdvancedSceneManager.SceneManager.standalone
```

If any scenes are opened manually using `UnityEngine.SceneManagement.SceneManager`, then they will be tracked here.

Methods

`SceneOperation OpenSingle(Scene scene)`

Closes existing scenes and opens the specified one. This will close the current collection.

`SceneOperation Preload(Scene scene)`

Preloads the Scene.

Please note that Unity only supports a single preloaded scene at a time, and that all subsequent scene operations will be halted until this scene is fully loaded and activated, or closed.

Utility Scene Manager

The utility scene manager (not a real scene manager, just named that to be consistent and easy to find) contains convenience functions for working with scenes that might be open in either scene manager.

```
AdvancedSceneManager.SceneManager.utility
```

Events

`Action queueEmpty`

Occurs when an queued scene operation finishes and queue is empty.

`ActiveSceneChangedHandler(OpenSceneInfo previousScene, OpenSceneInfo activeScene)`

`ActiveSceneChanged`

Occurs when the active scene changes.

`Action<OpenSceneInfo, SceneManagerBase> SceneOpened`

Occurs when a scene is opened, in either scene manager.

`Action<OpenSceneInfo, SceneManagerBase> SceneClosed`

Occurs when a scene is closed, in either scene manager.

`Action LoadingScreenOpening`

Occurs when a loading screen is about to be opened. This is after scene is opened, but before `LoadingScreen.OnOpen` is called.

`Action LoadingScreenOpened`

Occurs when a loading screen has opened. This is after `LoadingScreen.OnOpen` has been called.

`Action LoadingScreenClosing`

Occurs when a loading screen is about to close. This is before `LoadingScreen.OnClose` has been called.

`Action LoadingScreenClosed`

Occurs when a loading screen has closed. This is after `LoadingScreen.OnClose` has been called, but before scene is closed.

Properties

`IEnumerable openScenes`

Gets all currently open scenes in both scene managers.

`bool isBusy`

Gets whatever ASM is currently busy because one or more scene operations are in the queue.

`ISceneOperation currentOperation`

The currently executing scene operation

`ReadOnlyCollection operationQueue`

Gets the current scene operation queue.

Methods

RegisterOpenCallback(T scene, Action onOpen = null, Action onClose = null, bool persistent = false)

where T : Object, ISceneObject

Registers a callback for when a scene or collection has opened, or closed, the callback is removed once it has been called, unless persistent is true.

UnregisterCallback(T scene, Action onOpen = null, Action onClose = null)

where T : Object, ISceneObject

Unregisters a callback.

IEnumerator DoSceneCallback(OpenSceneInfo scene, Func<T, IEnumerator> action)

Performs a callback on the scripts in the specified scene.

SceneOperation Close(OpenSceneInfo scene)

Closes a scene regardless of whatever it is associated with a collection, or is was opened as stand-alone.

SceneOperation CloseAll()

Closes all scenes.

SceneOperation Toggle(Scene scene, bool? enabled = null)

Toggles the scene open or closed, if the scene is part of the current collection, then the scene will be toggled within the collection, otherwise, it will be toggled as a stand-alone scene. If null, the scene will be toggled on or off depending on whatever the scene is open or not. Pass a value to ensure that the scene either open or closed.

SceneOperation Toggle(UnityEngine.SceneManagement.Scene scene, bool? enabled = null)

Toggles the scene open or closed, if the scene is part of the current collection, then the scene will be toggled within the collection, otherwise, it will be toggled as a stand-alone scene. If null, the scene will be toggled on or off depending on whatever the scene is open or not. Pass a value to ensure that the scene either open or closed.

IsOpenReturnValue IsOpen(Scene scene)

Gets whatever the scene is open, either as part of a collection, or as stand-alone.

IsOpenReturnValue IsOpen(UnityEngine.SceneManagement.Scene scene)

Gets whatever the scene is open, either as part of a collection, or as stand-alone.

OpenSceneInfo FindOpenScene(UnityEngine.SceneManagement.Scene scene)

Finds the OpenSceneInfo of this scene.

OpenSceneInfo FindOpenScene(Scene scene)

Finds the first open instance of this Scene, if it is open.

OpenSceneInfo FindPreloadedScene(Scene scene)

Find first preloaded instance this scene.

OpenSceneInfo FindPreloadedScene(UnityEngine.SceneManagement.Scene scene)

Find first preloaded instance this scene.

IEnumerable<(OpenSceneInfo scene, SceneManagerBase sceneManager)>

FindPreloadedScenes()

Finds all current preloaded scenes.

SetActive(Scene scene)

Sets a scene as the activate scene.

SetActive(UnityEngine.SceneManagement.Scene scene)

Sets a scene as the activate scene.

OpenSceneInfo activeScene

Gets the currently open scene.

Runtime

Methods

Runtime manages startup and quit processes.

`Start(bool quickStart = false, SceneCollection collection = null, bool ignoreDoNotOpen = false)`

Starts startup sequence.

Enters playmode if in editor.

`quickStart`: Skips splash screen. `collection`: Specifies a collection to be opened, after all other collection and scenes flagged to open during startup is opened.

`ignoreDoNotOpen`: Opens any scenes tagged with

`SceneOpenBehavior.DoNotOpenWithCollection`, which would normally not be opened.

`Restart(bool showSplashScreen = false)`

Restarts game and plays startup sequence again. Skips splash screens by default. Unsets persistent scenes.

Enters playmode if in editor.

`RegisterQuitCallback(IEnumerator coroutine)`

Register a callback to be called before quit.

`UnregisterQuitCallback(IEnumerator coroutine)`

Unregister a callback that was to be called before quit.

`CancelQuit()`

Cancels a quit in progress.

Only usable during a `RegisterQuitCallback` or while `isQuitting` is true.

`Quit(bool fade = true)`

Calls quitCallbacks and quits the game, optionally with a fade animation.

Properties

`bool isQuitting { get; }`

Gets whatever ASM is currently in the process of quitting.

Exceptions

`OpenSceneException`

Thrown in certain circumstances when a scene could not be opened. Methods that throw this all denote this the xml comments.

`CloseSceneException`

Thrown in certain circumstances when a scene could not be closed. Methods that throw this all denote this the xml comments.

Scene Operation

The scene operation is what executes the logic of Advanced Scene Manager. A scene operation takes lists of scenes to open and close, and a few other properties to modify behavior, and then generates and executes the actions when it is at front of the queue.

- Process:
1. Queue and wait until we're up
 2. Create actions¹
 3. Block input (cannot change properties anymore)
 4. Open loading screen, if one is defined
 5. Call collection closed callbacks, if applicable
 6. Run generated actions, one by one.
 7. Set active scene, if a collection was opened
 8. Call callbacks added through WithCallback methods
 9. Call collection open callbacks
 10. Dequeue (allow next operation to run)
 11. Hide loading screen

¹ Operation will exit early here if no actions were generated (and none added through WithAction(SceneOperation)).

enum Phase

Defines the phases of a scene operation. These are passed to the associated loading screen using LoadingScreen.OnScenePhaseChanged callback.

CloseCallbacks:

The scene operation is currently executing close callbacks on the scenes that are being closed, if any.

UnloadScenes:

The scene operation is currently unloading the scenes, if any.

LoadScenes:

The scene operation is currently loading the scenes, if any.

OpenCallbacks:

The scene operation is currently executing open callbacks on the scenes that are being opened, if any.

FinishLoad:

The scene operation is currently finishing loading / activating the scenes, if any.

CustomActions:

The scene operation is currently executing custom actions, added through SceneOperationBase{TSelf}.WithAction(SceneAction[]) or similar methods, if any.

Events

static Action queueEmpty

Occurs when an queued operation finishes and queue is empty.
SceneManager.utility.queueEmpty is a proxy for this event.

Properties

static bool isBusy

Gets whatever ASM is currently busy because one or more operations are in the queue.

static ReadOnlyCollection queue { get; }

The queue of operations.

bool keepWaiting { get; }

Inherited from CustomYieldInstruction. Tells unity whatever the operation is done or not.

public SceneOperation.Phase phase { get; }

The phase the this scene operation is currently in.

ReadOnlyCollection open { get; }

The scenes to open.

ReadOnlyCollection close { get; }

The scenes to close.

ReadOnlyCollection reopen { get; }

The scenes to reopen.

SceneCollection collection { get; }

The associated collection.

Scene loadingScreen { get; }

The loading screen to use for this operation, overrides loading screen defined on collection, if one is defined.

bool useLoadingScreen { get; }

Prevents loading screen from being used on this operation if set to false.

ThreadPriority? loadingPriority { get; }

Gets the loading priority for the background thread.

Defaults to SceneCollection.loadingPriority when collection is used, otherwise Profile.backgroundLoadingPriority.

SceneManagerBase sceneManager { get; }

The scene manager that requested this operation.

bool? clearUnusedAssets { get; }

Specifies whatever unused assets should be cleared. See also: Resources.UnloadUnusedAssets.

bool doCollectionCallbacks { get; }

Specifies whatever ICollectionOpen and ICollectionClose should be called.

Action loadingScreenCallback { get; }

Specifies callbackBeforeBegin parameter for LoadingScreenUtility.ShowLoadingScreen()

ReadOnlyCollection actions { get; }

The generated actions.

float totalProgress { get; }

The total progress of the operation.

SceneOperation openedLoadingScreen { get; }

Gets the loading screen that was opened.

SceneAction current { get; }

The current action that is executing.

bool cancelled { get; }

Gets if this scene operation is cancelled.

Methods

`Cancel(Action callbackWhenFullyCancelled = null)`

Cancel this operation. Note that the operation might not be cancelled immediately, if user defined callbacks are currently running (`WithAction()`, `WithCallback()`) they will run to completion before operation is cancelled. `SceneOperation.cancelled` can be used in callbacks to check whatever a operation is cancelled.

The following methods use the fluent api pattern, and does as such return the operation itself, because of this return values have been omitted here for redundancy and clarity.

`IgnoreQueue(bool ignore = true)`

Specifies that this operation should ignore queue.

`Open(params Scene[] scene)`

Adds scenes to be opened.

`Close(params OpenSceneInfo[] scenes)`

Adds scenes to be closed.

`Reopen(params OpenSceneInfo[] scene)`

Adds scenes to be reopened.

`Open(IEnumerable scene)`

Adds scenes to be opened.

`Close(IEnumerable scenes)`

Adds scenes to be closed.

`Reopen(IEnumerable scene)`

Adds scenes to be reopened.

`WithCollection(SceneCollection collection, bool withCallbacks = false)`

Specifies associated collection and whatever `ICollectionOpen` and `ICollectionClose` callbacks should be called.

`WithLoadingScreen(bool use)`

Prevents loading screen from opening on this operation if false.

`WithLoadingScreen(Scene scene)`

Specifies loading screen to use for this operation, overrides collection loading screen, if one is defined.

`WithAction(params SceneAction[] actions)`

Adds `SceneActions` to be performed after generated actions are done. Called before loading screen closes, if one was opened.

`WithAction(params Action[] actions)`

Adds `System.Action` to be performed after generated actions are done. Called before loading screen closes, if one was opened.

`WithAction(params Func[] actions)`

Adds `IEnumerator` callbacks to be performed after generated actions are done. Called before loading screen closes, if one was opened.

`WithClearUnusedAssets(bool enable)`

Specifies whatever unused assets should be cleared. See also: `Resources.UnloadUnusedAssets`.

`WithLoadingScreenCallback(Action callback)`

Specifies `callbackBeforeBegin` parameter for `LoadingScreenUtility.ShowLoadingScreen()`

`WithLoadingPriority(ThreadPriority priority)`

Sets `Application.backgroundLoadingPriority` for the duration of this operation, value is reset when operation is done or cancelled.

If 'loading thread priority' is disabled in settings, then this will also be disabled.

Also note that operations that skip the queue cannot change

`Application.backgroundLoadingPriority`, which means that setting this value will then have no effect.

Scene Action

SceneAction is an action that is to be performed by the scene manager, such as opening a scene. Used in SceneOperation.

Default actions

Actions used by SceneOperation

SceneLoadAction

Loads a scene, but does not activate it. This is the same as preload.

SceneFinishLoadAction

Finishes loading a scene, calls Start() and Awake().

SceneUnloadAction

Unloads a scene.

SceneOpenCallbackAction

Calls all ISceneOpen callbacks on a scene.

SceneCloseCallbackAction

Calls all ISceneClose callbacks on a scene.

OpenAndRunCallbackAction

Opens the scene and finds the first script instance that implements T and runs a callback defined on T.

Closes scene if not found.

Makes use of SceneOpenAction to open scene.

RunCallbackAndCloseAction

Runs a callback on T and optionally closes the scene.

Makes use of SceneCloseAction to close scene.

Startup and quit

StartupAction

Represents the startup process in ASM. AggregateAction, executes the following actions:

1. FadeOut (CallbackAction) 2. SetProfileInBuild (CallbackAction) 3. ReloadAssets (CallbackAction) 4. CloseAllUnityScenesAction 5. PlaySplashScreenAction 6. ShowStartupLoadingScreen (CallbackAction) 7.

OpenCollectionsAndScenesFlaggedToOpenAtStartAction 8. HideStartupLoadingScreen (CallbackAction)

CloseAllUnityScenesAction

Closes all scenes, regardless of whatever they are tracked or not. Since we cannot know if standalone scene manager has finished collecting scenes yet, lets just assume it has not, and clear scene manager lists manually.

PlaySplashScreenAction

Plays the splash screen specified in the current profile, can also hide loading screen, if passed in constructor. Loading screen is hid regardless of whatever splash screen is set or not in profile.

OpenCollectionsAndScenesFlaggedToOpenAtStartAction

Finds all scenes and collections that are flagged to be opened during startup and open them.

QuitAction

Runs callbacks registered through SceneManager.runtime.RegisterQuitCallback(), and then quits the game. Can be cancelled through SceneManager.runtime.CancelQuit().

Other actions

Beyond the ones listed above, there are a few others:

`abstract OverridableAction`

Represents an action that can be overridden. `SceneLoadAction`, `SceneFinishLoadAction`, `SceneUnloadAction`, are overridable in order to support the addressables support package. Please take great care when overriding default ASM actions in user code as this may cause issues.

`abstract AggregateAction`

Represents a list of actions that are run in sequence. This is mostly used for convenience actions such as `SceneOpenAction` and `SceneCloseAction`, but is also used for `StartupAction`.

`SceneOpenAction`

Opens a scene. Convenience action to execute the following actions: 1. `SceneLoadAction` 2. `SceneFinishLoadAction` 3. `SceneOpenCallbackAction`

`SceneCloseAction`

Closes a scene. Convenience action to execute the following actions: 1. `SceneCloseCallbackAction` 2. `SceneUnloadAction`

`CallbackAction`

Runs an `IEnumerator` callback.

Custom actions

Custom actions have to inherit from `SceneAction` and can be added to a `SceneOperation` through `SceneOperation.WithAction(SceneAction)`.

Use constructor to define and receive properties, if bad properties are received then call `Done()` to indicate that action should not run.

Methods

`abstract IEnumerator DoAction(SceneManagerBase _sceneManager)`

Do what you want to do here. `Done()` must be called before returning (at end of method and before any return statements).

`Done(OpenSceneInfo openScene = null)`

Should be called in `DoAction` when done. If a scene was opened, you may want to pass the `OpenSceneInfo`.

`RegisterCallback(Action action)`

Register a callback when scene action is done.

`UnregisterCallback(Action action)`

Remove an registered callback when scene action is done.

Properties

`OpenSceneInfo openScene { get; set; }`

`SceneOpenAction`: The scene that was opened.

`SceneCloseAction`: The scene that is about to be closed, is set to null when done.

`Scene scene { get; protected set; }`

The scene to perform the action on. Null if no value set in constructor.

`SceneCollection collection { get; protected set; }`

The collection that is being opened.

`bool isDone { get; protected set; }`

Is this scene action done? Use `Done()` to set this to true.

`float progress { get; protected set; }`

The progress of this scene action.

Asset Management

The `AssetManagement` class is responsible for managing scene and collection assets in ASM.

Usage:

```
AdvancedSceneManager.SceneManager.assetManagement
```

Events

The following events are only available in the editor.

Action `AssetsChanged`

Called when assets changed.

Action `AssetsCleared`

Called when assets are cleared, by either `Clear()` or from ui.

Properties

`ReadOnlyCollection` `collections { get; }`

The collections in this project.

`ReadOnlyCollection` `scenes { get; }`

The scenes in this project.

`bool` `isInitialized { get; }`

Returns whatever all assets has been loaded on startup.

`bool` `allowAssetRefresh { get; set; }`

If false, then assets will not be refreshed, this will mean that no `Scene ScriptableObject` will be created when a `SceneAsset` added, and a `Scene` will also not be removed when its associated `SceneAsset` is removed.

`SceneUtility` is unaffected, since it creates `Scene ScriptReference` directly.

Methods

The following methods are only available in editor.

`void` `DuplicateProfileAndAssign()`

Duplicates the active profile and assigns it as active.

`void` `CreateProfileAndAssign()`

Creates a new empty profile and assigns it as active.

`Profile` `DuplicateProfile()`

Duplicates the active profile.

`Profile` `CreateProfile()`

Creates a new empty profile.

`T` `FindAssetById<T>(string assetID)` where `T : Object`

Find the asset with the associated asset ID. This only finds assets which are managed by ASM.

`T` `FindAssetByPath<T>(string path)` where `T : Object`

Find the asset with the specified path. This only finds assets which are managed by ASM.

`Scene` `FindSceneByPath(string path)`

Find the scene with the associated path (this is the path to the `SceneAsset`).

`Add<T>(T obj, Profile profile = null, bool import = true)` where `T : ScriptableObject, ISceneObject`

Adds the asset.

Add(SceneAsset scene)

Adds the SceneAsset to asm. Returns existing Scene if already exists.

Remove<T>(T obj) where T : ScriptableObject, ISceneObject

Removes the asset.

T Create<T>(string name, Profile profile = null, Action initializeBeforeSave = null) where T : ScriptableObject, ISceneObject

Create and adds an asset to the specified profile.

T Create<T>(string name, Action initializeBeforeSave = null) where T : ScriptableObject, ISceneObject

Creates and adds an asset to the active profile.

Clear()

Clear all assets, that are managed by ASM.

Open Scene Info

Open scene info represents an open scene at runtime.

Properties

Scene scene { get; }

The scene that this OpenSceneInfo is associated with.

UnityEngine.SceneManagement.Scene? unityScene { get; }

The UnityEngine.SceneManagement.Scene that this OpenSceneInfo is associated with.

bool isPreloaded { get; }

Gets whatever this scene is preloaded.

bool isOpen { get; }

Gets whatever this scene is currently open.

bool isPersistent { get; }

Gets whatever this scene is persistent. See PersistentUtility for more details.

SceneManagerBase sceneManager { get; }

The scene manager associated with this OpenSceneInfo.

Preloaded Scene Helper

Preloaded scene helper is a class that helps with scenes that have been preloaded, and is returned from any methods that preloads scenes.

Please note that Unity only supports a single preloaded scene at a time, and that all subsequent scene operations will be halted until this scene is fully loaded and activated, or closed.

```
//using UnityEngine;
//using AdvancedSceneManager;
//using AdvancedSceneManager.Core;
//using AdvancedSceneManager.Utility;

public class Preloader : MonoBehaviour
{
    public Scene sceneToPreload;
    public SceneOperation<PreloadedSceneHelper> preloadedScene;

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
            preloadedScene = SceneManager.standalone.Preload(sceneToPreload);
    }

    public void FinishPreload()
    {
        preloadedScene.value.FinishLoading();
    }
}
```

Properties

OpenSceneInfo scene { get; }
The OpenSceneInfo that is associated with this scene.

bool isInPreloadedState { get; }
Get if the scene is still in a preloaded state.

Methods

SceneOperation FinishLoading()
Finishes loading scene.

SceneOperation Discard()
Closes the scene.

Scene Utility

Scene utility contains some useful methods for working with scenes.

```
using AdvancedSceneManager;
using AdvancedSceneManager.Utility;
using AdvancedSceneManager.Models;

public async void GenerateScenes(int count)
{
    var sc = SceneCollection.Find("GeneratedScenes")
    ? SceneCollection.Find("GeneratedScenes")
    : SceneCollectionUtility.Create("GeneratedScenes");

    SceneManager.assetManagement.allowAutoRefresh = false;

    for (int i = 0; i < count; i++)
    {
        EditorUtility.DisplayProgressBar(
            title: "Creating scenes...",
            info: "Creating scene: " + i,
            progress: i / (float)count);

        await SceneUtility.Create($"Scenes/{i}", sc);
    }

    SceneManager.assetManagement.allowAutoRefresh = true;
    EditorUtility.ClearProgressBar();
}

public void RemoveGeneratedScenes()
{
    var sc = SceneCollection.Find("GeneratedScenes")
    ? SceneCollection.Find("GeneratedScenes")
    : SceneCollectionUtility.Create("GeneratedScenes");

    SceneManager.assetManagement.allowAutoRefresh = false;

    foreach (var scene in scenes)
    {
        EditorUtility.DisplayProgressBar(
            title: "Removing scenes...",
            info: "Removing scene: " + i,
            progress: i / (float)count);

        SceneUtility.Remove(scene.path);
    }

    SceneManager.assetManagement.allowAutoRefresh = true;
    EditorUtility.ClearProgressBar();
}
```

Methods

`IEnumerable<scene> GetAllOpenUnityScenes()`

Get all open unity scenes.

`bool hasAnyScenes`

Gets if there are any scenes open that are not dynamically created, and not yet saved to disk.

`int sceneCount`

The total number of currently loaded scenes.

`OpenSceneInfo CreateDynamic(string name, LocalPhysicsMode localPhysicsMode = LocalPhysicsMode.None)`

Creates a scene at runtime, that is not saved to disk.

`Task<Scene> Create(SceneCollection collection = null, int? index = null, bool replaceIndex = false, bool save = true)`

Creates a scene, using save prompt for path. Returns null if save dialog cancelled. Only usable in editor.

collection: The collection to add the scene to.

index: The index of the scene in collection, no effect if collection: is null.

replaceIndex: Replaces the scene at the specified index, rather than insert it.

save: Save collection to disk.

`Task<Scene> Create(string path, SceneCollection collection = null, int? index = null, bool replaceIndex = false, bool save = true)`

Creates a scene at the specified path. path: The path that the scene should be saved to.

collection: The collection to add the scene to.

index: The index of the scene in collection, no effect if collection is null.

replaceIndex: Replaces the scene at the specified index, rather than insert it.

save: Save collection to disk.

Note that calling this method in quick succession produces a lot of asset refresh triggers, causing asm to attempt to create scene ScriptableObject for newly created scenes, even though this method does so automatically, it may be desirable to set `SceneManager.assetManagement.allowAutoRefresh` to false before calling this method multiple times, and then re-enabling it again afterwards.

`void Remove(string path)`

Removes the SceneAsset at the specified path and its associated Scene, and removes any references to it from any SceneCollection.

`void Remove(Scene scene)`

Removes the scene and its associated SceneAsset, and removes any references to it from any SceneCollection.

`IEnumerable<Scene> FindOpen(string name)`

Find open scenes by name.

`IEnumerable<Scene> FindOpen(Func<Scene, bool> predicate)`

Find open scenes by predicate.

`IEnumerable<Scene> Find(string name, SceneCollection inCollection = null, Profile inProfile = null)`

Find scenes by name, in the specified collection or profile, if defined.

`IEnumerable<Scene> Find(Func<Scene, bool> predicate, SceneCollection inCollection = null, Profile inProfile = null)`

Find scenes by predicate, in the specified collection or profile, if defined.

`void MoveToNewScene(params GameObject[] objects)`

Moves the object to a new scene.

Only available in editor.

`void MergeScenes(params string[] scenes)`

Merges the scenes together, the first scene in the list will be the output scene.

Scene Collection Utility

An utility class to perform actions on collections.

`SceneCollection Create(string name, Profile profile = null)` Creates a SceneCollection.

name: The name of the collection.

profile: The profile to add this collection to. Defaults to Profile.current.

`Remove(SceneCollection collection)`

Removes a collection.

`RemoveNullScenes(SceneCollection collection)` Removes all null scenes in the collection.

Scene Helper

The scene helper is a scriptable object that can be used in UnityEvent to easily open scenes or collections, since it might be easier to find scenes or collections using it.

It may also be used through a singleton instance:

```
AdvancedSceneManager.Utility.SceneHelper.current
```

Methods

(SceneCollection collection, bool asLoadingScreen[]) FindCollections(Scene scene)

Finds the collections that are associated with this scene.

SceneOperation Preload(Scene scene)

Preloads the scene.

Use PreloadedSceneHelper.FinishLoading to finish loading scene.

bool IsOpen(SceneCollection collection) Gets whatever the collection is currently open.

IsOpenReturnValue IsOpen(Scene scene)

Gets whatever the scene is open, either as part of a collection, or as stand-alone.

The following methods do not have any return values, since they are meant to be used through UnityEvent, which does not expose methods with return values.

Open(SceneCollection collection)

Opens the collection.

ReopenCollection()

Reopens the current collection

Open(Scene scene)

Opens a scene.

Throws a OpenSceneException if the scene cannot be opened by the current collection.

Reopen(Scene scene)

Reopens the scene.

OpenSingle(Scene scene)

Close existing scenes and open the specified one.

This will close the current collection.

CloseCollection()

Closes the current collection.

Close(Scene scene)

Closes a scene regardless of whatever it is associated with a collection, or is was opened as stand-alone.

Toggle(SceneCollection collection)

Toggles the collection.

Toggle(SceneCollection collection, bool enabled)

Toggles the collection.

Toggle(Scene scene)

Toggles the scene.

Toggle(Scene scene, bool enabled)

Toggles the scene.

`SetActiveScene(Scene scene)`
Sets the scene as the activate scene.

`Quit()`
Quits the game.

`Restart()`
Restarts game and plays startup sequence again.
Enters playmode if in editor.

`RestartCollection()`
Reopens the current collection

Coroutine Utility

CoroutineUtility is a helper class that runs coroutines in DontDestroyOnLoad, since running coroutines for opening and closing scenes won't work very well otherwise.

Coroutines can be run by CoroutineUtility like this:

```
public class RunCoroutineScript : MonoBehaviour
{
    void Start()
    {
        //Just run it
        RunCoroutine().StartCoroutine();

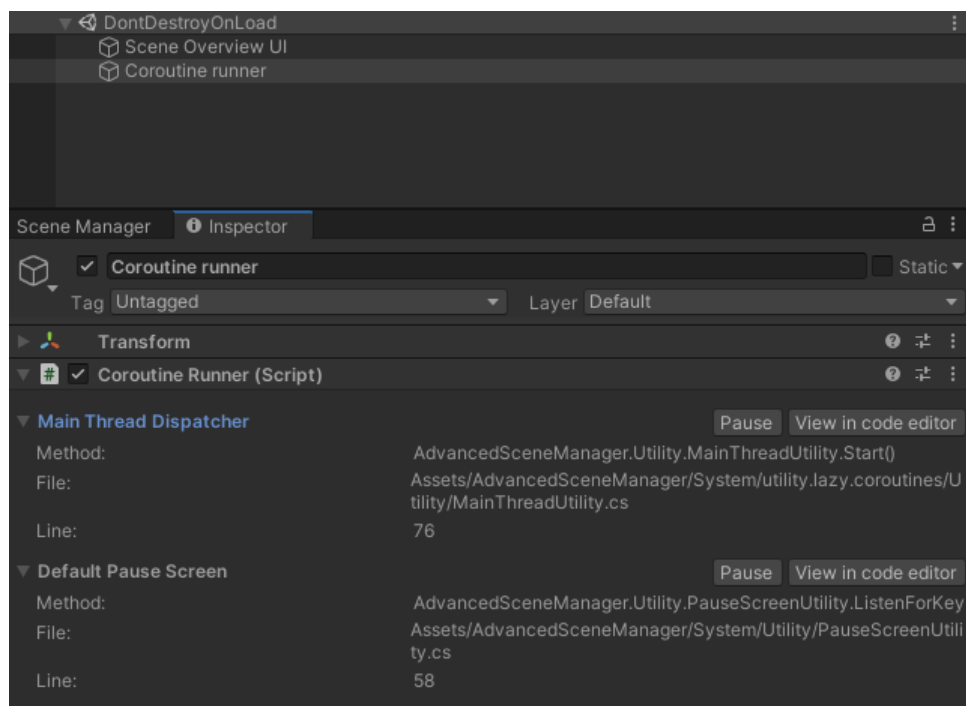
        //If callback when done is desired
        RunCoroutine().StartCoroutine(OnComplete);

        //Presents a more readable message in the editor, more info below.
        RunCoroutine().StartCoroutine(debugText: "Waiting for 10 seconds.");
    }

    IEnumerator RunCoroutine()
    {
        yield return new WaitForSeconds(10);
    }

    void OnComplete()
    {
        Debug.Log("Wait complete!");
    }
}
```

The coroutines that are currently in progress using coroutine utility can be easily viewed by having a look at the object created in DontDestroyOnLoad.



The open button opens the script in the editor at the line that called `StartCoroutine()`.

The text displayed is the name of the method that returns the `IEnumerator`. This can be overridden by specifying `debugText` parameter in `StartCoroutine()`.

Loading Screen Utility

Loading screen utility manages loading screens in ASM.

Properties

`ReadOnlyCollection<LoadingScreen> loadingScreens { get; }`
The currently open loading screens.

`bool IsLoadingScreenOpen { get; }`
Gets if any loading screens are open.

`Scene fade { get; }`
Finds the default fade loading screens.

Methods

The following methods that return `SceneOperation` are all non-blocking, and skip the queue to be run independently.

`Scene FindLoadingScreen(SceneCollection collection)`
Find the loading screens that is associated with this collection. If collection uses default, then default is returned.

`SceneOperation<LoadingScreen> ShowLoadingScreen(SceneCollection collection, float? timeout = null, Action<LoadingScreen> callbackBeforeBegin = null)`
Shows the loading screen associated with this collection.

`SceneOperation<LoadingScreen> ShowLoadingScreen(Scene scene, float? timeout = null)`
Shows a loading screen.

`SceneOperation HideLoadingScreen>LoadingScreen loadingScreen)`
Hide the loading screen.

`SceneOperation HideAll()`
Hide all loading screens.

`bool IsLoadingScreen(Scene scene)`
Gets if this scene is a loading screen. This is only true for scenes that are currently open and were opened with `LoadingScreenUtility`.

`SceneOperation DoActionWithFade(Func<IEnumerator> action, float duration = 1, Color? color = null)`
Fades screen out, performs action and fades in again.

`SceneOperation DoActionWithFade(Action action, float duration = 1, Color? color = null)`
Fades screen out, performs action and fades in again.

`SceneOperation<LoadingScreen> FadeOut(float duration = 1, Color? color = null)`
Fades out the screen.

`SceneOperation FadeIn>LoadingScreen loadingScreen, float duration = 1, Color? color = null)`
Fades in the screen. Loading screen from `FadeOut()` must be passed.

`SceneOperation DoAction(Scene scene, Action action)`
Opens loading screen, performs action and hides loading screen again.

`SceneOperation DoAction(Scene scene, Func<IEnumerator> coroutine)`
Opens loading screen, performs action and hides loading screen again.
Throws `OpenSceneException` if scene is null.

Persistent Utility

Persistent utility is responsible for managing persistent scenes.

A SceneCloseBehavior can be set automatically on scenes using tags.

Methods

Set(UnityEngine.SceneManagement.Scene scene, SceneCloseBehavior behavior = SceneCloseBehavior.KeepOpenAlways)
Set SceneCloseBehavior for this UnityEngine.SceneManagement.Scene.

Unset(UnityEngine.SceneManagement.Scene scene)
Unset and revert to default SceneCloseBehavior for this UnityEngine.SceneManagement.Scene.

UnsetAll()
Unsets SceneCloseBehavior for all UnityEngine.SceneManagement.Scene[s].

SceneCloseBehavior GetPersistentOption(UnityEngine.SceneManagement.Scene scene)
Gets the SceneCloseBehavior that is set for this UnityEngine.SceneManagement.Scene.

Canvas Sort Order Utility

Canvas order utility is responsible for making sure that canvases is in the correct order.

`PutOnTop(this Canvas canvas)`

Sets the sort order on this canvas to be on top of all other canvases managed by `CanvasSortOrderUtility`.

`PutAtBottom(this Canvas canvas)`

Sets the sort order on this canvas to be on bottom of all other canvases managed by `CanvasSortOrderUtility`.

`MakeSure(this Canvas canvas, Canvas above = null, Canvas below = null)`

Adds a constraint on the sort order of this canvas based on one or two other canvases.

above: Makes sure that the canvas is always above this one.

below: Makes sure that the canvas is always below this one.

Throws `ArgumentException` if above and below is the same, or canvas is same as below or above.

Asset Refresh Utility

Asset refresh utility runs after asset import to ensure that scenes and collections are updated. This unfortunately results in more progress bars to wait for during import and save. This process is quick however and we are working to reduce this as much as possible.

The following tasks are the ones currently implemented: > RefreshDeletedFiles
Remove scenes that have had its associated SceneAsset removed.

RefreshAddedFiles

Add scenes for newly added SceneAssets.

RefreshInvalidPaths

Update path for scenes where the associated SceneAsset has moved.

RefreshRenamed

Update name of scenes that have had its associated SceneAsset renamed.

UpdateLabels

Updates asset labels for scenes to easily identify scenes based on collection.

Example: ASM:NewCollection

RefreshProfileCollectionReferences

Fixes references for collections in profile, since they will in certain circumstances be lost, and this fixes most cases.

FixInvalidSceneReferences

Makes sure that paths for scenes are correct.

FixDuplicatesOutsideOfSettingsFolder

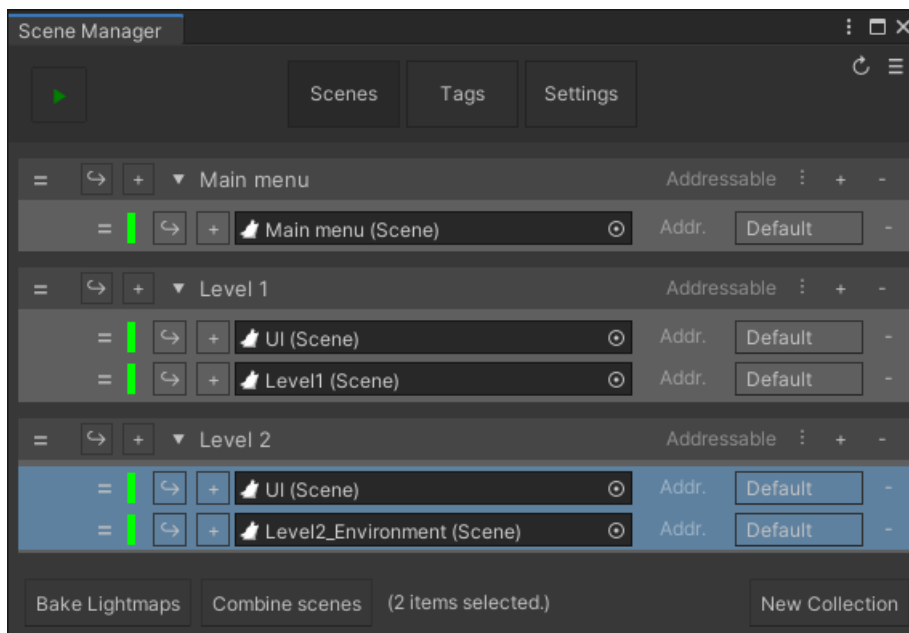
Prevents scenes from getting created in regular assets folder when a tutorial package provides them.

Scene Merge Split

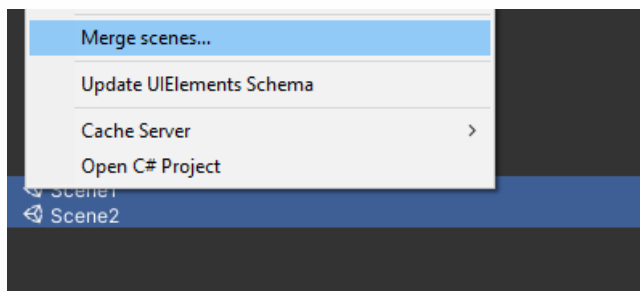
Scene merge

Merging scenes can be done in two ways:
(API methods are contained in SceneUtility)

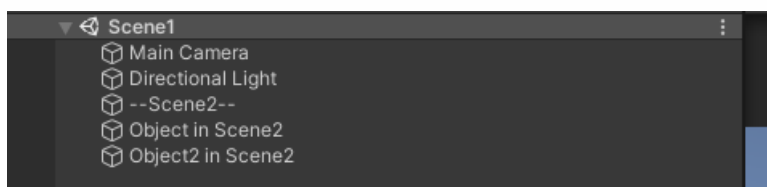
Pressing the **Combine scenes** button:



Or by selecting two or more SceneAsset in project window:

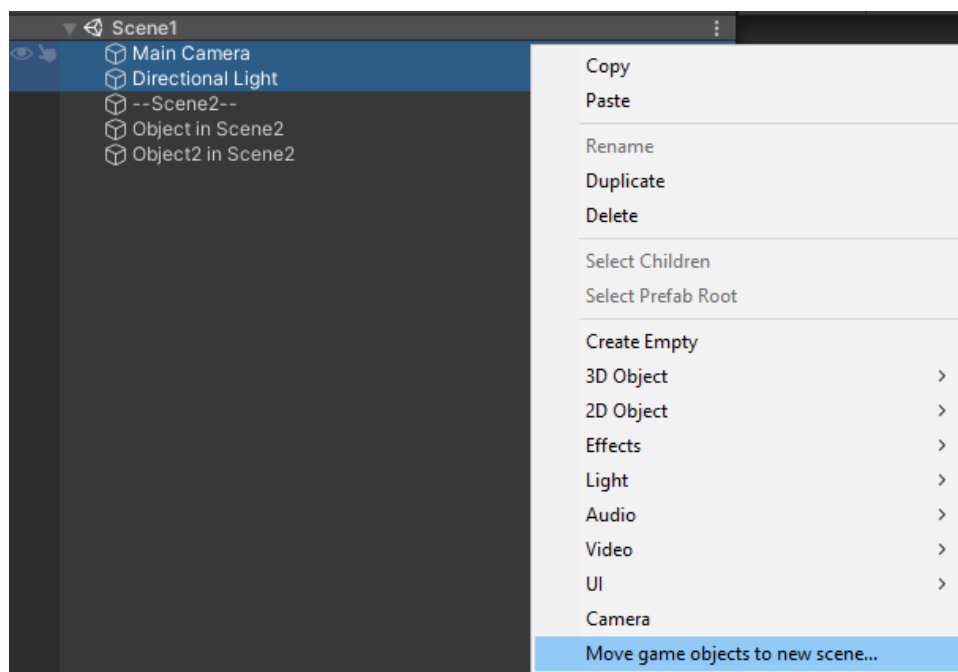


The scenes will then be merged like this:

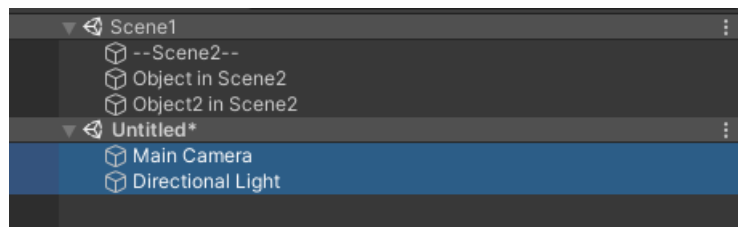


Scene split

Scenes can be split by selecting two GameObjects in the hierarchy:

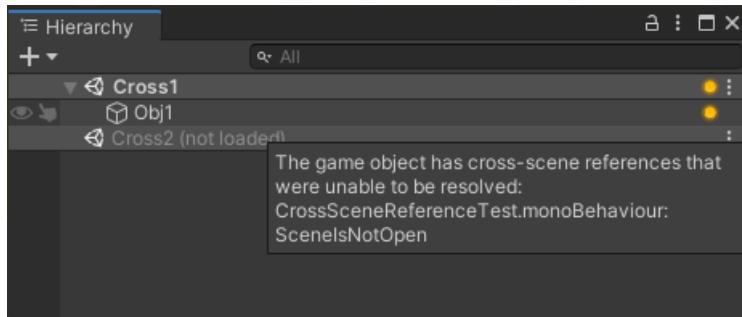


And will be split like this:



Hierarchy GUIUtility

HierarchyGUIUtility is a wrapper around `EditorApplication.hierarchyWindowItemOnGUI`, which automatically manages the bounds of each registered callback, so that adding a button for example, does not require manual positioning and avoids overlapping other callback rects.



Cross scene makes use of HierarchyGUIUtility with its yellow indicator icon, which has a tooltip.

Delegates

`bool HierarchySceneGUI(Rect position, Scene scene)`

Called after reserving a rect in hierarchy scene field.

Return true to indicate that something was drawn, false means that the rect will not be allocated and next OnGUI callback may use the area.

`bool HierarchyGameObjectGUI(Rect position, GameObject gameObject)`

Called after reserving a rect in hierarchy game object field.

Return true to indicate that something was drawn, false means that the rect will not be allocated and next OnGUI callback may use the area.

Methods

`void AddSceneGUI(HierarchySceneGUI onGUI, float? width = null, int? index = null)`

Adds a onGUI call for Scene fields.

Passing null to width means that width will be the same as height.

Setting index can help sort icons, when multiple is displayed. Note that order is reversed, higher index means icon will be placed further to the right.

`void AddGameObjectGUI(HierarchyGameObjectGUI onGUI, float? width = null, int? index = null)`

Adds a onGUI call for GameObject fields.

Passing null to width means that width will be the same as height.

Setting index can help sort icons, when multiple is displayed. Note that order is reversed, higher index means icon will be placed further to the right.

`void RemoveSceneGUI(HierarchySceneGUI onGUI)`

Remove a OnGUI call for a Scene.

`void RemoveGameObjectGUI(HierarchyGameObjectGUI onGUI)`

Remove a OnGUI call for a GameObject.

Scene Data Utility

SceneDataUtility provides functions to store data that is to be associated with a scene. The utility automatically reassociates the data when the scene is renamed or moved.

```
IEnumerable<T> Enumerate(string key)
```

Enumerates all T on all scenes.

```
T Get<T>(Scene scene, string key, T defaultValue = default)
```

Gets the value with the specified key, for the specified scene.

```
T Get<T>(string scene, string key, T defaultValue = default)
```

Gets the value with the specified key, for the specified scene.

```
void Set<T>(Scene scene, string key, T value)
```

Sets the value with the specified key, for the specified scene.

```
void Set<T>(string scene, string key, T value)
```

Sets the value with the specified key, for the specified scene.

```
string GetDirect(Scene scene, string key)
```

Gets the value with the specified key, for the specified scene. This is the direct version, all values are stores as string, which means Get<T>(string, string, T) must convert value beforehand, this method doesn't.

```
string GetDirect(string scene, string key)
```

Gets the value with the specified key, for the specified scene. This is the direct version, all values are stores as string, which means Get<T>(string, string, T) must convert value beforehand, this method doesn't.

```
void SetDirect(Scene scene, string key, string value)
```

Sets the value with the specified key, for the specified scene. This is the direct version, all values are stores as string, which means Get<T>(string, string, T) must convert value beforehand, this method doesn't.

```
void SetDirect(string scene, string key, string value)
```

Sets the value with the specified key, for the specified scene. This is the direct version, all values are stores as string, which means Get<T>(string, string, T) must convert value beforehand, this method doesn't.

```
void Unset(Scene scene, string key)
```

Unsets the value with the specified key, for the specified scene.

```
void Unset(string scene, string key)
```

Unsets the value with the specified key, for the specified scene.

Guid Reference Utility

GuidReferenceUtility provides functions for referencing game object globally. The utility makes use of GuidReference script, which may be manually applied to a game object, or may be applied automatically by ASM when cross-scene references are used.



GuidReference works by adding itself to GuidReferenceUtility, in Start(), and removes itself in OnDestroy(), this means that there is minimal overhead to say, scanning the hierarchy every time a scene opens or closes.

GuidReference uses guid strings internally, which has been shortened to only use 22 bytes, minimizing memory usage when a lot of references are used.

Note that GuidReference scripts will not be removed automatically, even when ASM it has been automatically applied by ASM, this is because we cannot know whatever a reference is being used, since GuidReferenceUtility is public.

Callbacks

The following callbacks are called by the scene manager when a scene is opened or closed, or a collection is opened or closed. The callbacks are called only for the scene or collection they are attached to, and are called before loading screens are closed, if one was opened, and is waited for. So any actions you may want to happen before loading screens is closed, these callbacks may be what you want.

Note that these callbacks are only invoked if scene loading is actually done through asm. They will not be called when regular unity play button is used.

ISceneOpened

Called when the scene that the script is attached to is opened.

ISceneClosed

Called when the scene that the script is attached to is closed.

ICollectionOpen

Called when a collection containing a scene with this script is opened.

ICollectionClose

Called when a collection containing a scene with this script is closed.

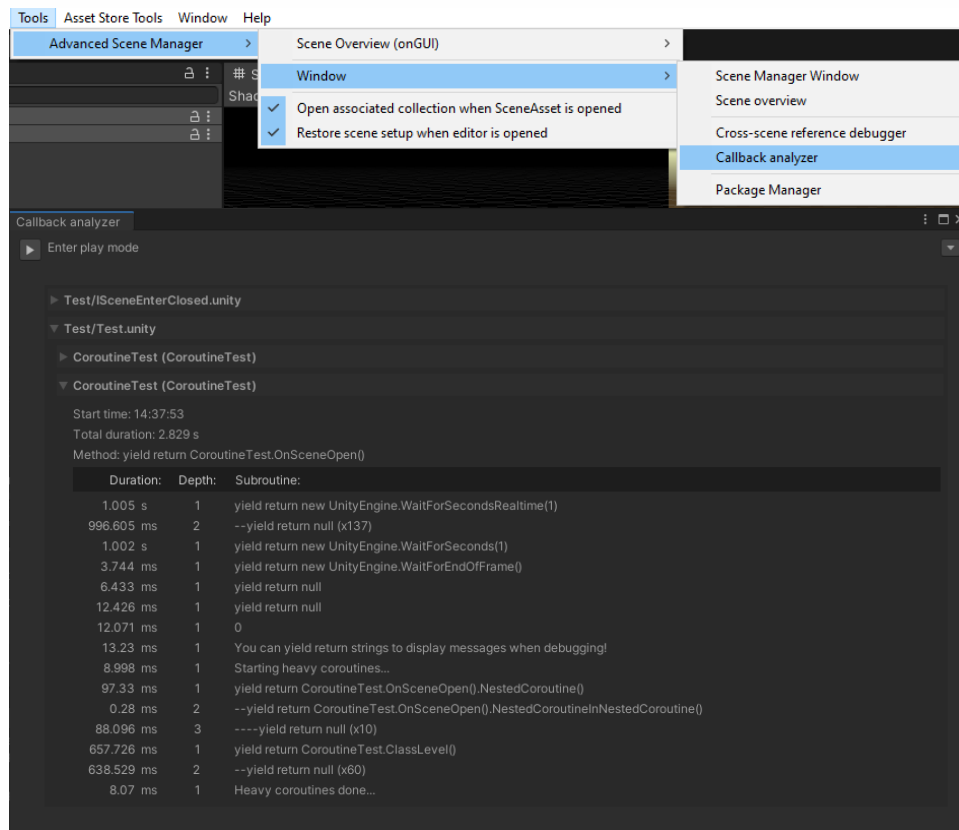
Callback Utility

Callback utility can track coroutines and display an overview over how they perform. Only available in editor.

Note that tracking is not 100% accurate and does on top of this, introduce overhead, so the callback analyzer is more of a quick way to see if there are any performance issues that needs resolving.

For any serious performance optimization, callback analyzer is not the tool for the job, and industry tools that focuses only on diagnosing performance is recommended.

The callback analyzer window can be opened from:



Yield return null is not combined by default, yield return CombineNull.value can be used to tell callback analyzer that this and subsequent nulls should be combined.

Default Unity yield instructions, such as WaitForSecondsRealtime, are automatically combined, even though they technically do not return CombineNull.value.

By default, only ISceneOpen, ISceneClose, ICollectionOpen, ICollectionClose callbacks are tracked, but user coroutines can also be tracked by setting enableDiag to true in StartCoroutine():

```
MyCoroutine().StartCoroutine(enableDiag: true);

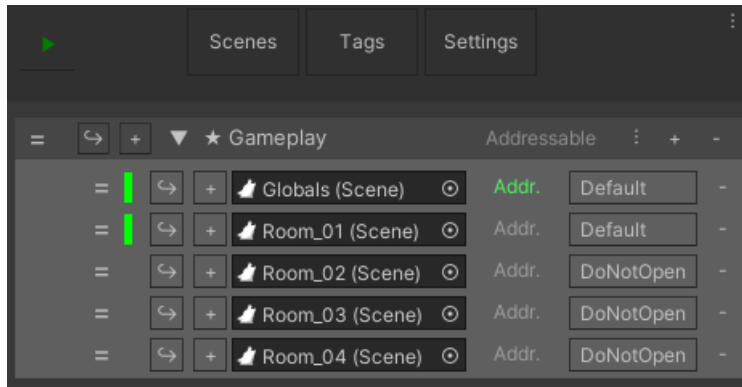
IEnumerator MyCoroutine()
{ ... }
```

Addressables Support

Addressables support package (install instructions here) adds support to open and close addressable scenes in ASM.

This package automatically finds all scenes that are addressable and overrides default scene open and close behavior. Which means, after tweaking settings in addressables, ASM should just work like it usually does.

In addition to scene open and close support, a button is added to scenes and collections in ui.



Pressing 'Addr.' button on a scene toggles the scene as addressable.

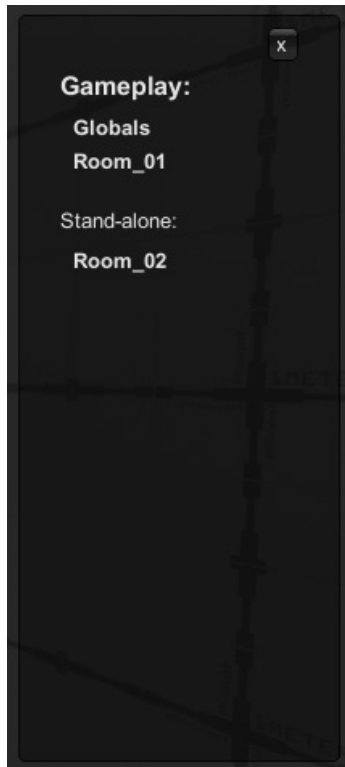
Pressing 'Addressable' on collection will toggle all scenes in collection as addressable.

Please note: > When ASM adds a scene to addressables, it will add it under a default group, named after the collection the scene is contained in. This group will not have a schema assigned to it by ASM, and is expected to be assigned manually, or have all scenes reassigned to other groups.

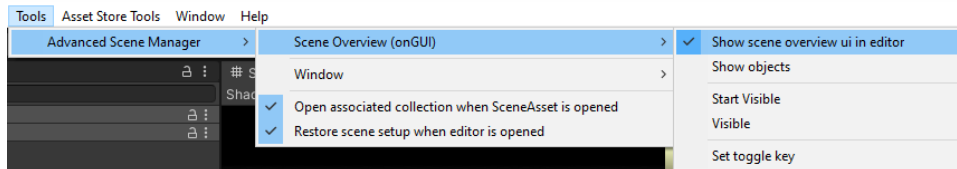
Scene Overview UI

The scene overview ui helps to get an overview over the currently open scenes.

While this might not have much use in editor, it may be invaluable when debugging build in certain circumstances.



The scene overview ui can be enabled through the menus:



or through the toggle key, which by default is set to F6 (though there is some weird issue with different unity installations / PCs that cause default key not to work, in this case you may have to change it manually first).

Toggle key can be changed using 'Set toggle key'.

Note that 'Show objects' may be slow, please be aware of this when debugging.

Default Pause Screen

The default pause screen adds a pause screen out-of-the-box. This is to prevent those annoying moments when you create a build, only to forget that you do not have a pause screen, so in order to restart you have to press alt-f4 and open exe again.

Note that only esc is supported and can as such not be opened on anything beyond a keyboard by default. If InputSystem is installed and enabled then gamepad support is available. Note that `PauseScreenUtility.Show()` may be called to open it manually.

The default pause screen can be disabled in scene manager window.



Static methods

`Show()`

Shows the pause screen.

`Hide(bool ignoreAnimations = false)`

Hides the pause screen.

`Toggle()`

Toggles the pause screen.

ISceneObject

ISceneObject is an interface for identifying and provides access to some common stuff in scene and collection assets.

Properties

```
string name { get; }  
The name of the asset.
```

Methods

```
SetName(string name)  
Sets the name of the asset.
```

Don't use this in user code, as this is only used internally. For collections you can use `SceneCollection.title` instead (which calls this). Don't worry about scenes, as they are automatically renamed when the associated `SceneAsset` is renamed.

```
OnPropertyChanged()  
Invokes INotifyPropertyChanged.PropertyChanged event. Used in ui to automatically  
update ui when property changes.
```


Loading Screen

Loading screens can be opened either manually through `LoadingScreenUtility` or automatically when opening collections.

Two default loading screens are provided out-of-the-box, which can be found in 'AdvancedSceneManager/System/Defaults': > Fade loading screen
Fades the screen out and fades in again after operation finished.

Progress bar loading screen
Fades in and out like fade, but displays and updates a progress bar.

Custom loading screens

Custom loading screens can be created by creating a script that inherits from

```
AdvancedSceneManager.Callbacks.LoadingScreen
```

and placing it in a dedicated scene.

Fields

Canvas canvas
This can be set to automatically register canvas with `CanvasSortOrderUtility`.

Properties

`ISceneOperation operation { get; }`
The associated scene operation that opened this loading screen. May be null if loading screen opened manually.

Methods

`abstract IEnumerator OnOpen()`
Called when the associated scene operation is about to start. Use this to show your loading screen.

`abstract IEnumerator OnClose()`
Called when the associated scene operation has ended. Use this to hide your loading screen.

`abstract void OnProgressChanged(ISceneOperation progress)`
Called when the associated scene operation has made progress.

`virtual OnScenePhaseChanged(ISceneOperation operation, SceneOperation.Phase previousPhase, SceneOperation.Phase nextPhase)`
Called when the associated scene operation changes phase.

`virtual void OnCancel(ISceneOperation operation)`
Called when the associated scene operation is cancelled. Note that `OnClose(ISceneOperation)` is not called.

Guide

Setting up a custom loading screen isn't difficult, but it isn't obvious either, we'll cover how to create a custom loading screen and how to use it in a collection here.

A loading screen in ASM is implemented as a `MonoBehaviour` (through `LoadingScreen` class) with a few `IEnumerator` callbacks that are called when a collection is opened or closed (or manually opened through `LoadingScreenUtility`). Callbacks are waited for, and execution of ASM is stopped until a callback is done, note that `OnProgressChanged()` is not a callback, it is managed by `LoadingScreen`.

Each loading scene script is then placed in a dedicated scene, which is then automatically opened by ASM.

Code

Create a new script called 'CustomLoadingScreen' and make it inherit from LoadingScreen, you may have to import 'AdvancedSceneManager.Callbacks'.

Remove default methods generated by unity, and override methods required by LoadingScreen.

Now, you may want the loading screen to fade in, rather than just 'pop' in. You can do this easily by adding a CanvasGroup, and using the Fade() extension method provided by ASM.

```
//using AdvancedSceneManager.Utility;

public CanvasGroup group;

public override IEnumerator OnOpen()
{
    yield return group.Fade(1, 1);
}

public override IEnumerator OnClose()
{
    yield return group.Fade(0, 1);
}
```

This will fade the canvas in when loading screen opens and fade out when loading is finished.

Now, for showing progress, add a Slider variable and update it using OnProgressChanged().

```
public Slider slider;

public override void OnProgressChanged(SceneOperation progress)
{
    slider.value = progress.totalProgress;
}
```

This will update a slider when loading progress changes, note that this will probably not be visible when opening smaller scenes since they may load too quickly, so don't worry too much about it for this tutorial, just know that it exists so that you may use it in the future.

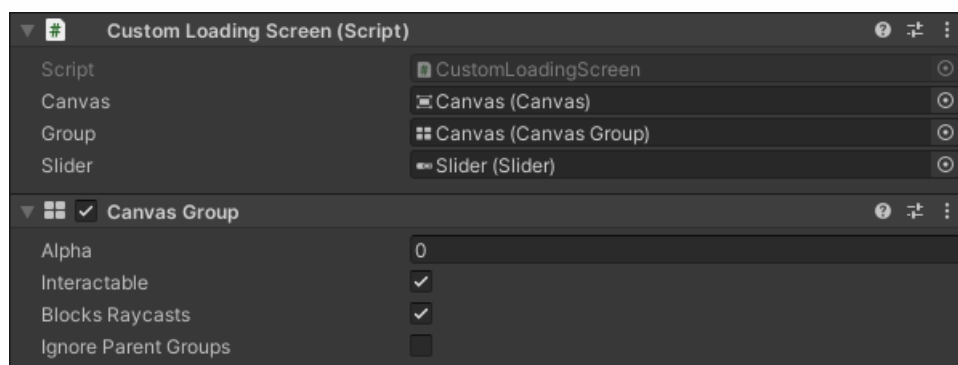
Now that we have a functional implementation of a loading screen, we may begin the graphical part.

UI

Create a new scene called 'Custom loading screen'.


Add a canvas and a darkish background image, you may add a bright background instead if you are so inclined. Add a slider, which you may style and position as you would like.

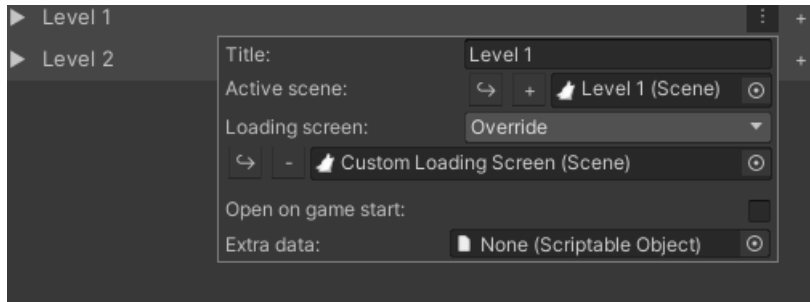
Next, add a CanvasGroup to canvas and then the 'CustomLoadingScreen' script that we created earlier, assign canvas, group, and slider variables.



Setting alpha to 0 in CanvasGroup may prevent potential flickering.

Actually using it

Now for actually using it, we can assign it to a collection in the Scene Manager Window, by pressing the  on a collection, and changing 'Loading screen' to 'Override' and assigning the scene to the field that appears directly underneath.



And with that, we're done! You may now open the collection that has the loading screen assigned to it and the screen should start to fade in the ui you created, and when fade is done, the current collection should close and open the new collection, and then fade your ui out.

Splash Screen

Splash screen

A splash screen in ASM is implemented as a script inheriting

```
AdvancedSceneManager.Callbacks.SplashScreen
```

and placing it in a dedicated scene, and assigning it in the ASM settings.

One splash is provided out-of-the-box, which, while functional, is just intended as a reference, and can be found in 'AdvancedSceneManager/System/Defaults'.

ASM Splash Screen

Displays the text 'Advanced Scene Manager', with fade animations.

Fields

Canvas canvas

This can be set to automatically register canvas with CanvasSortOrderUtility.

Properties

SceneOperation progress { get; }

Inherited from LoadingScreen, will always be null.

Methods

abstract IEnumerator DisplaySplashScreen()

Called during startup process when the scene manager is ready to display the splash screen.

Guide

Splash screens in ASM are implemented as MonoBehaviour (through SplashScreen class) with a callback that manages everything that should happen during the splash screen. Callback is waited for, and execution of ASM is stopped until callback done.

This means that, in difference from Unity's own splash screens, you may do whatever you want during the splash screen, such as play video and/or download extra data from the internet or just load assets, like an initial loading screen.

Code

Create a script called 'CustomSplashScreen' and make it inherit from SplashScreen, you may have to import 'AdvancedSceneManager.Callbacks'.

Remove default methods generated by unity, and override DisplaySplashScreen().

In this guide we'll create a splash screen that fades in some text, then out, to do this we can create add public CanvasGroup variable, and use the extension method Fade() provided by ASM. In between fade in and out, add a delay.

```
//using AdvancedSceneManager.Utility;
public CanvasGroup group;

public override IEnumerator DisplaySplashScreen()
{
    yield return group.Fade(1, 1);
    yield return new WaitForSecondsRealtime(4);
    yield return group.Fade(0, 1);
}
```

This will fade the splash screen in, wait for 4 seconds, and then fade out again, and then ASM will continue with startup.

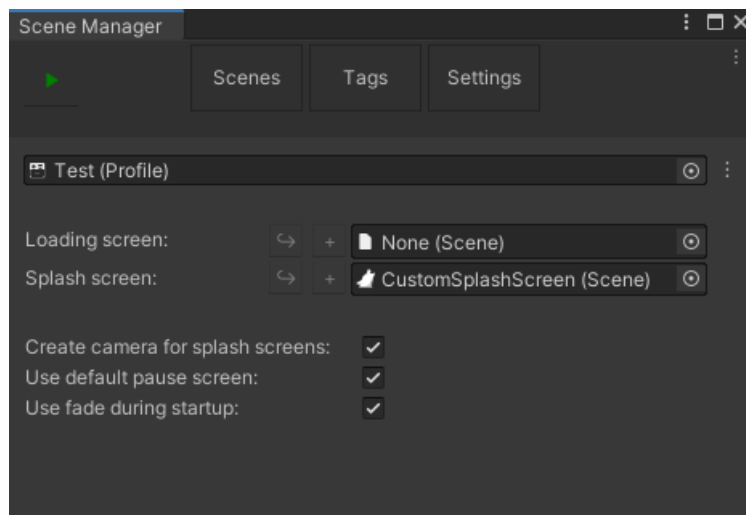
UI


Create a new scene called 'CustomSplashScreen'. Add a Canvas, a black background image to cover what's behind ui, then a CanvasGroup on a child object to background, set alpha on CanvasGroup to 0 to prevent potential flickering. Add some text in the center, with whatever you want, perhaps developer or publisher name?

Now add the 'CustomSplashScreen' script we created earlier to the canvas, then assign canvas and group.

Actually using it

Making ASM actually use it is as simple as assigning it in the settings tab in Scene Manager Window.



And with this, we're done! You may now use the  in Scene Manager Window to start game as if it was a build, since splash screen won't play otherwise, and you should see the text fade in, wait for a few seconds, and then fade out again, and then ASM will continue startup process, and open your collections and scenes.

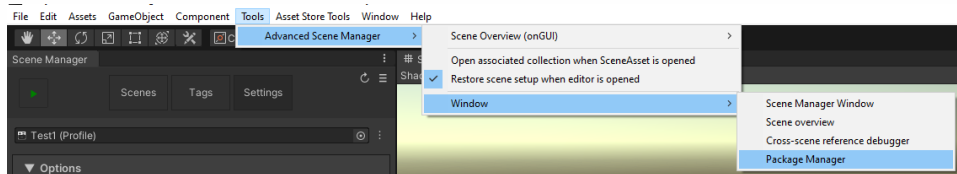
Support Packages

Support packages are packages that can be downloaded that either add additional functionality, like addressables package, or they can add example projects.

The example projects are designed to be as non-destructive as possible, and installing them in an existing project should not pose any problems but please make sure everything is backed up before downloading any, just in case.

Also remember to switch profile, since each example has its own profile.

These packages can be downloaded through the package manager:



The following packages currently exist:

Addressables support

Adds support for addressables unity package.

Lock scenes and collections

Adds the ability to lock scenes and collections from editing, from within unity.

Level select example

An example of how to create a level select menu.

Preloading example

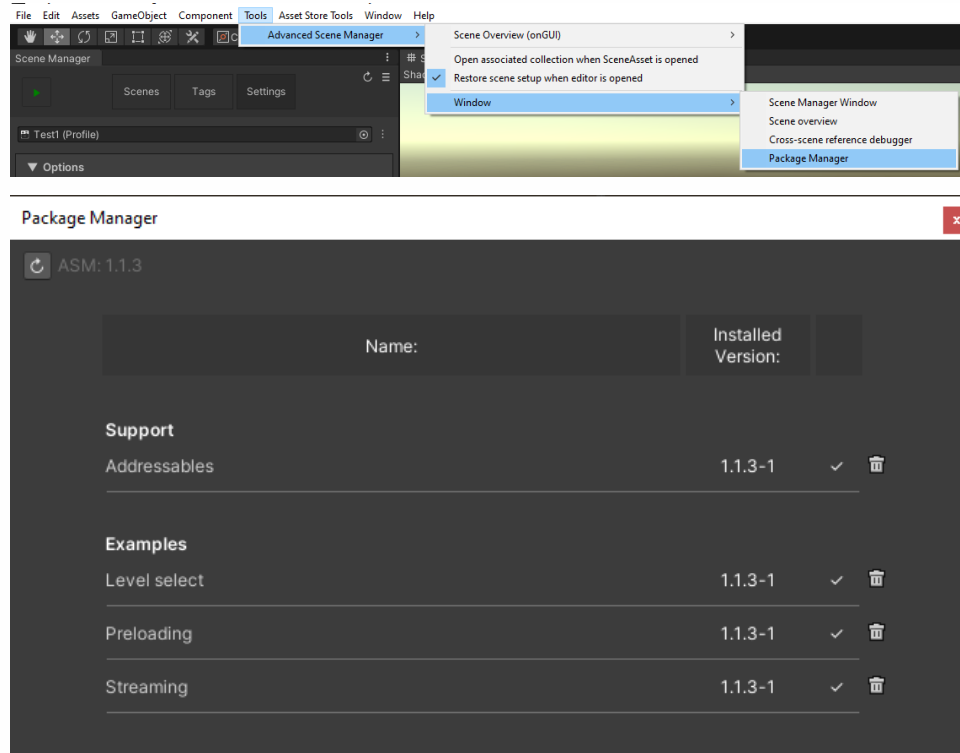
An example of how to preload scenes.

Streaming example

An example of how to implement streaming scenes.

Package Manager

The package manager is responsible for providing an overview over available support packages for ASM and to download and install them.



The package manager will tell you if a package is available for update / install, or up-to-date. If an update available, then a button will be presented to install.

It will also display the current version of ASM, and display a update button if one is available, which will open the unity package manager.

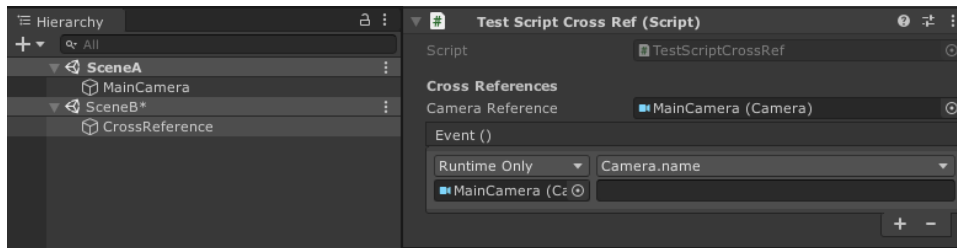
When ASM is updated, the package manager will also attempt to automatically update all previously installed packages to the newest version.

Note that while releases may be available directly from github.com immediately after release, they may not be available in package manager for up to about 10 minutes, this is likely due to an API restriction.

Cross Scene References

Cross scene references is one of those small things that should be supported by Unity, but just isn't. Fret not however, since as of ASM 1.1, that functionality is provided out-of-the-box, and may be enabled in settings.

Note that cross-scene references may not be restored until after `Start()` and `Awake()` is called, `ISceneOpen` or `ICollection` is as such recommended, since they are invoked by asm after cross-scene references has been restored.



Note that cross-scene references is still experimental, and may not work in all circumstances.

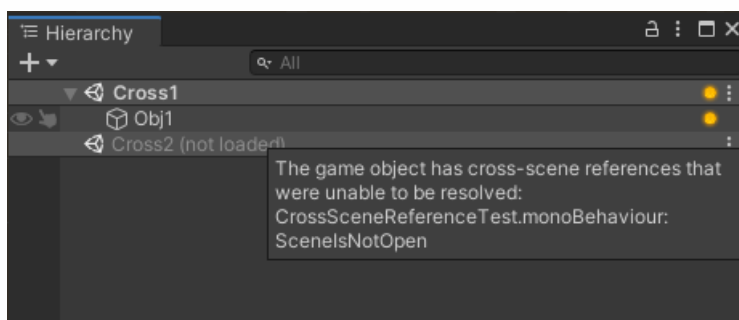
Note that while we do suppress warnings associated with cross-scene references (they cannot be disabled completely, all we can do is to prevent them from triggering where we can), please be aware that there will still occasionally be warnings, which you may ignore.

Problems with resolving

There may cases when a reference does not resolve correctly, there is then two different ways you may be notified:

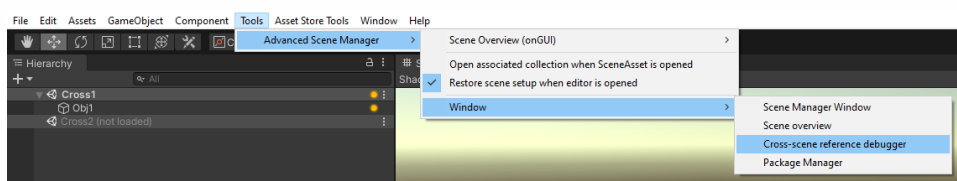
A warning (or error in build) may be logged in console, if enabled in settings. Pressing it will highlight the offending gameobject.

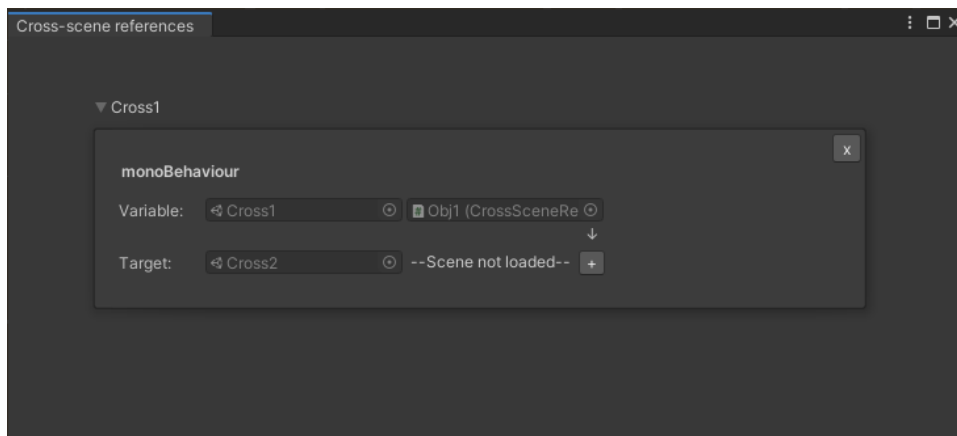
Or, in ASM 1.3, there is now a new icon in the hierarchy window, that will be displayed on a scene when any object in the scene was unable to be resolved. Another icon will also be displayed on the offending gameobject itself, this one will have a tooltip displaying a list of all references that could not be resolved and the reason why.



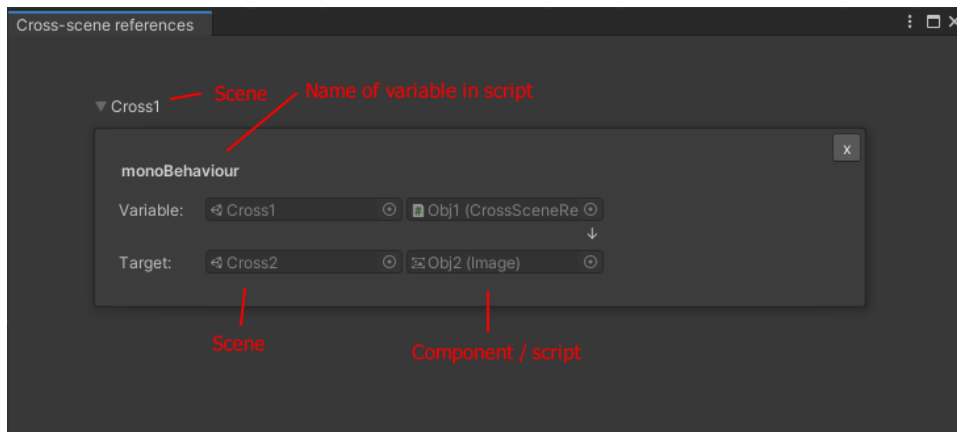
Cross-scene debugger

Sometimes the issue may be as simple as the target scene being unloaded, but when it isn't, the debugger can be used:





Pressing the **+** will load the scene that a reference refers to, so that we may get some more information:



Now, this may be confusing at first glance, but references is grouped by the scene that the variable is defined in.

The text ('monoBehavior' in example above), is the variable name in the script or component. If variable is an element in an array, unity event or similar, then index will be displayed in parenthesis, suffixed to name.

The two scene fields will be the scene that the variable / target object exists in.

The second column of object fields, will display the target script, component or gameobject of the variable or target.

Pressing any of the fields will select asset or object in scene, double pressing will open it in inspector.

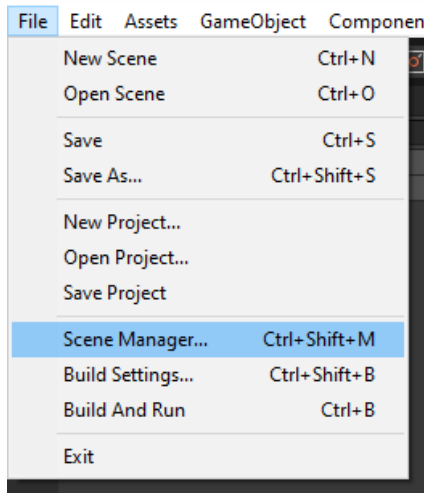
Remove button is also available, which would, in perfect operation, not be used, however there a certain scenarios where asm will not detect that a reference has been removed, and must as such be removed manually (we will keep working on this!).

Quick Start

Easily setting up a scene management solution is exactly the issue that ASM is trying to solve, and we've attempted to make it as easy as possible. But there may still be a slight learning curve.

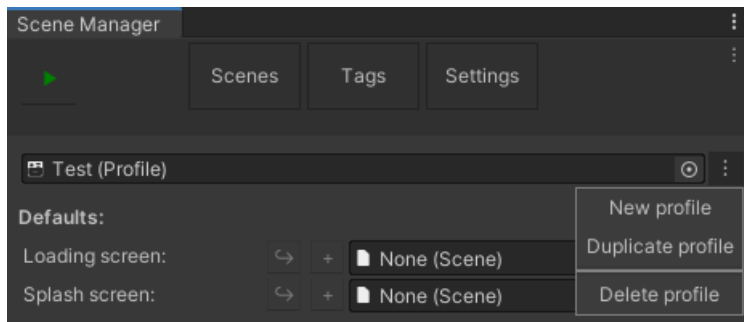
But don't worry, we'll walk through the steps needed to set up a basic scene management solution.

Open up the scene manager window:



Create profile

First, you'll need a profile. To create one, go to the settings tab. Press the menu button next to the top field. Press 'New profile' button in the dropdown. Enter your preferred name.

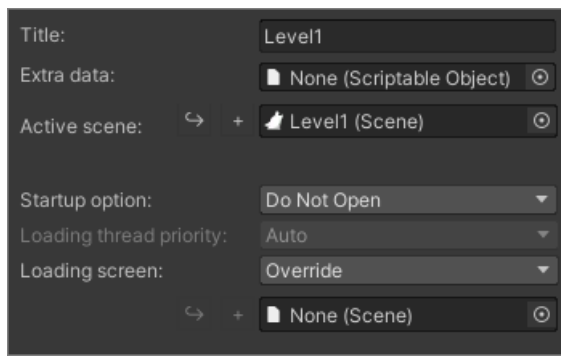


Setting up collections

Now that we've got a profile, we can go to the scenes tab again to actually set the scenes up. Press the 'New Collection' button to create a new collection.

Now, press the menu button on its header to open the edit collection menu. The top field is the title of the collection, change it to 'main menu'.

Next check 'Open on game start' to automatically open it when game starts. Multiple collections may be set to open at startup, and if so, then they'll be opened in order, top to bottom, this can be useful for persistent scenes, but not something we'll worry about in this guide.



While we won't worry about the other settings right now, they are documented over at [SceneManagerWindow](#).

Next, create two more collections, name them '*Level 1*', and '*Level 2*' (don't check '*Open on game start*' this time).

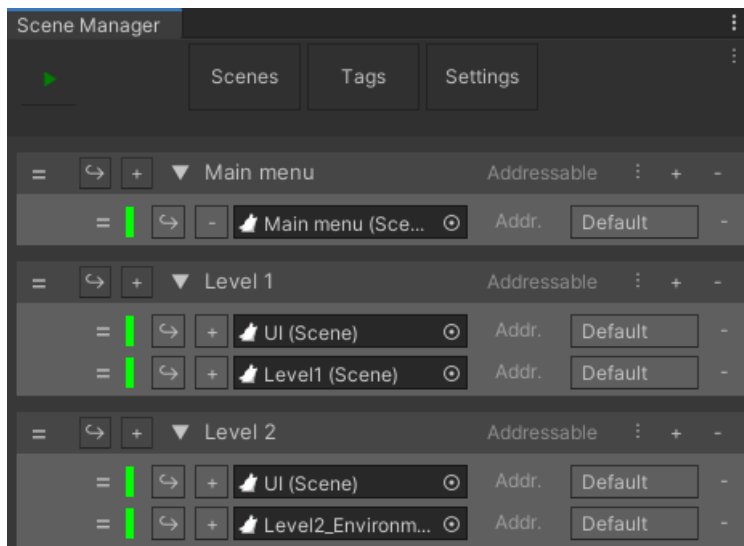
Adding scenes

Now that we've got our collections setup, we can add scenes to them.

Expand all collections by pressing on their titles, and add a scene by pressing the '+' button once on each.


Now create main menu, level 1, and level 2 scenes and assign them to their respective collection. (Just create scenes for now, we'll cover functionality in next segment)

Also create a ui scene.



Setting up main menu

Now that we have a scene manager solution we can easily open the scenes at runtime.

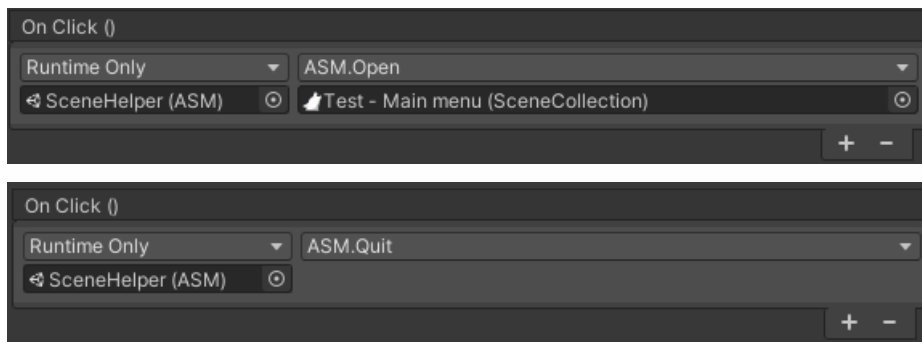
Now open your main menu scene, by pressing the  button. Setup the scene as you wish, but please add three buttons, one for each level, and one for quit.

Level buttons:


On the Button component, create a new **OnClick** action, and assign **SceneHelper** as target, '*ASM > Open (SceneCollection)*' as method, and assign the collection for the level as parameter.

Quit button:

On the Button component, create a new **OnClick** action, and assign **SceneHelper** as target, '*ASM > Quit*' as method.



Setting up level scenes

Now open ui scene, by pressing the  button. This scene may contain any ui objects, such as a health bar, if you wish, but please add a button to go back to main menu. On this button, create a new OnClick action, assign SceneHelper, and select 'ASM > Restart'.

Alternatively you could also set button to open main menu collection directly, the same as level buttons, this would be preferable if any standalone scenes are added, since these would be closed and then opened again, if restart is called.

And with that, we're done!

Press the play button in the scene manager window, now you'll notice that both level buttons open the associated scene, and the ui scene, with a fade animation, and goes back to main menu with a fade.