Asset Link 1.0.0

Asset link is a full replacement of Unity's Resources system. It enables users to find and use assets more efficiently than Unity's built in system. It is also more advanced, adding features like storage of game constants, object pooling, and retrieval of instanced assets at runtime. Most importantly assets are referenced directly through code, rather than string references, so broken asset links can be discovered at compile time, rather than at runtime.

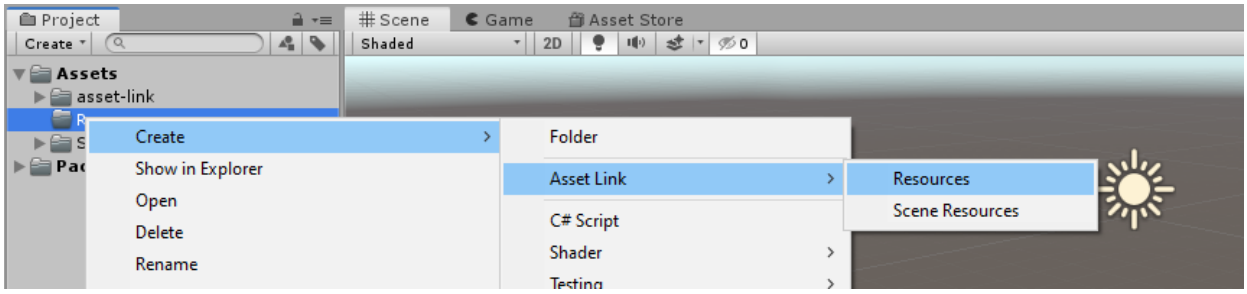
Setting up Asset Link

Asset Link requires two assets to be created to function. The Resources asset is responsible for maintaining references to assets in the Project folder. The Scene Resources asset maintains references to assets that are in scenes.

Setting Up the Resources Asset

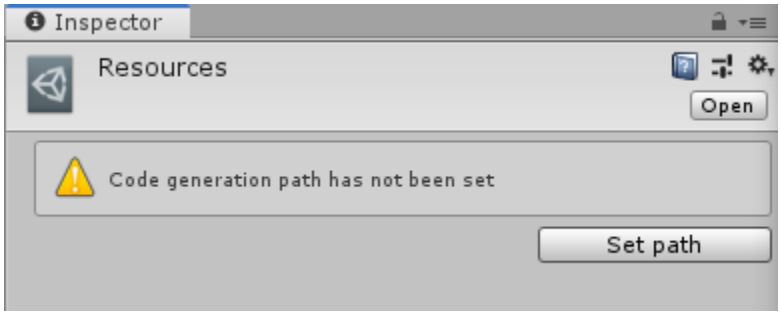
To create a Resources asset, right click on a Resources folder and click:

Create → Asset Link → Resources



Note that Resources assets must be contained in a folder named Resources.

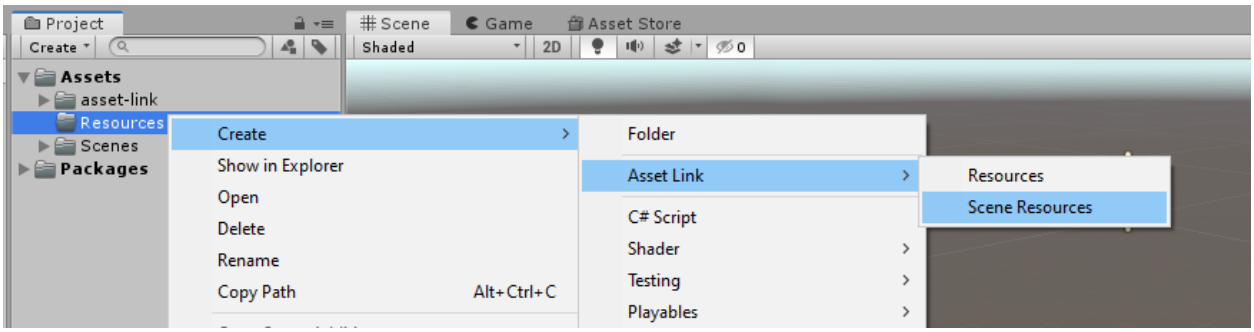
Once the asset has been created, a code generation path must be set. Click on the new asset, and then click the Set Path button. Anywhere will do, as long as it's in the Assets folder. The generated code will be used to access resources.



Setting Up the Scene Resources Asset

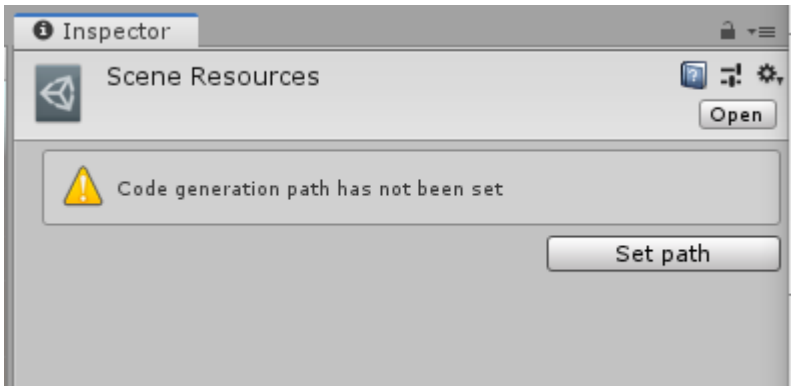
To create a Scene Resources asset, right click on a Resources folder and click:

Create → Asset Link → Scene Resources



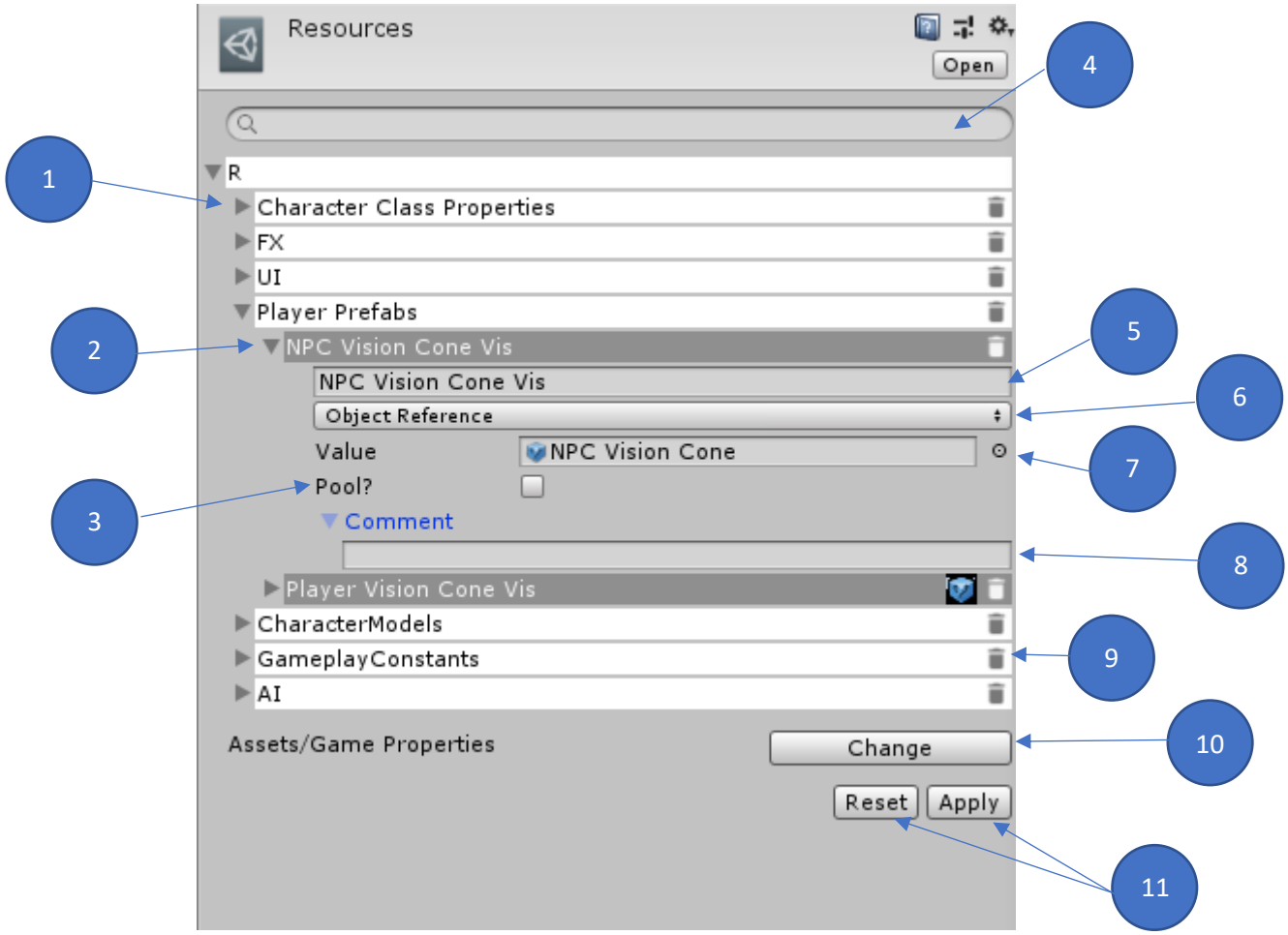
Note that Scene Resources assets must be contained in a folder named Resources.

Once the asset has been created, a code generation path must be set. Click on the new asset, and then click the Set Path button. Anywhere will do, as long as it's in the Assets folder. The generated code will be used to access scene resources.



Using the Resources Asset

The Resources asset is responsible for maintaining references to assets in the project folder. Here are the components of the Resources asset:

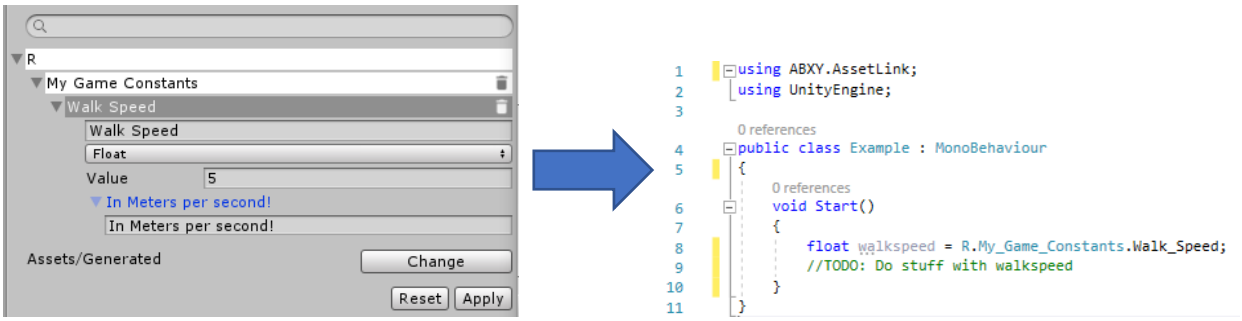


1. Namespace – Namespaces are like folders – they can contain assets and other namespaces. Right click to rename, delete, or add child namespaces and assets. Namespaces can be dragged to move them into other namespaces.
2. Asset – These are assets you'd like to reference. They can be Objects like prefabs, textures, or materials, or constants like ints, floats, enums, and strings. Assets can be dragged into namespaces
3. Pool – If this asset is a Prefab, this checkmark appears. See *Object Pooling* for more information
4. Search – Quickly find the asset you are looking for
5. Name – The name of this asset
6. Type – The type of this asset
7. Value – The value of this asset
8. Comment – Useful for providing explanatory information about an asset. When code for accessing the asset is generated, this comment will be attached to the code and will show in Intellisense
9. Delete – Deletes the Asset or Namespace

10. Change Directory – Changes where generated code is stored. This code will be used to access the resources defined in this prefab
11. Apply/Reset – Click Apply to save changes, or Reset to discard them

Accessing Resources Via Code

Once a resource has been added to the Resources asset, it can be accessed via code using the static class "R". For example:



Accessing Resources Via ResourceSelector

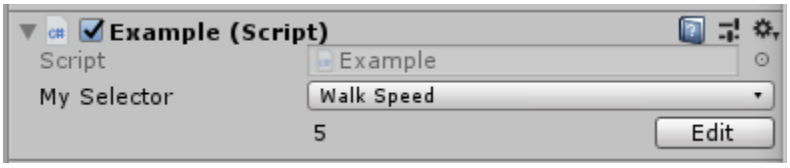
ResourceSelector is a utility class that exposes a selector GUI in the inspector. For example:

```
using ABXY.AssetLink;
using UnityEngine;

0 references
public class Example : MonoBehaviour
{
    [SerializeField]
    ResourceSelector mySelector = new ResourceSelector();

    0 references
    void Start()
    {
        Debug.Log(mySelector.GetFloatValue());
    }
}
```

Will produce this:



Clicking the dropdown enables the user to specify a resource in editor and reference it at runtime. You can also narrow down the types of assets that are allowed to be selected. For example, a selector produced by:

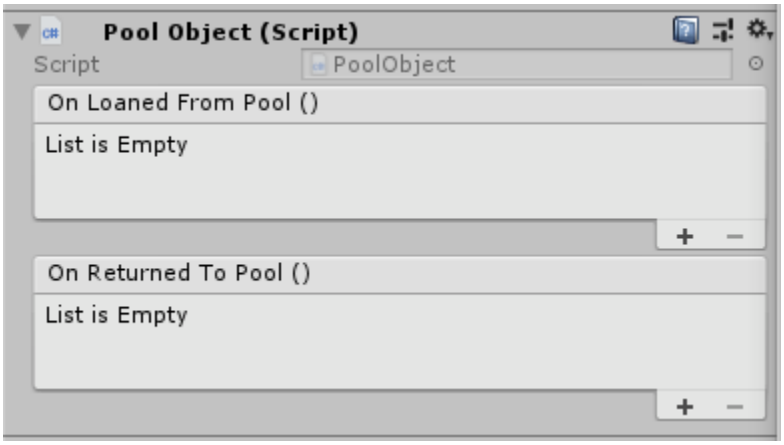
```
ResourceSelector mySelector = new ResourceSelector(typeof(string));
```

Will only show string assets.

Object Pooling

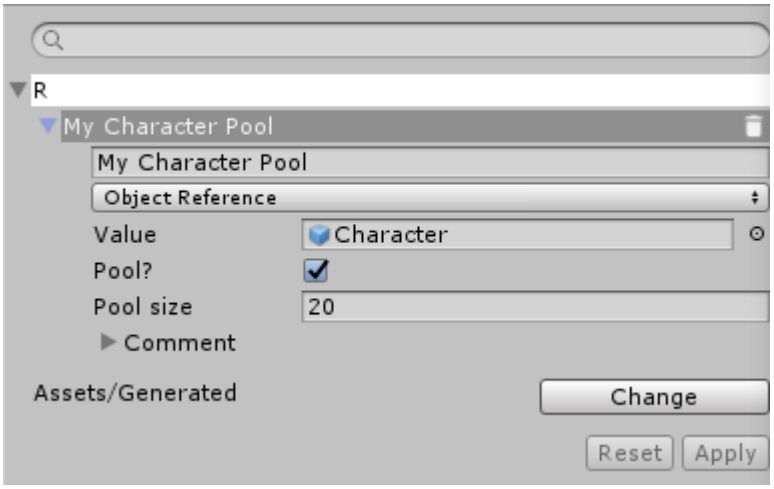
Object pooling with Asset Link is easy! Here's how it works:

1. Create a prefab for your asset that will be pooled
2. Add the PoolObject component to the root GameObject of your Prefab
3. The PoolObject component provides two callbacks, OnLoanedFromPool and OnReturnedToPool. Use these callbacks to hide and reset the various components you've created in your asset



4. Add an asset in your Resources asset, connect it to your prefab, enable pooling, and set your pool size

Using the pool is easy. As an example, take this asset:



A new instance from the pool is accessible through:

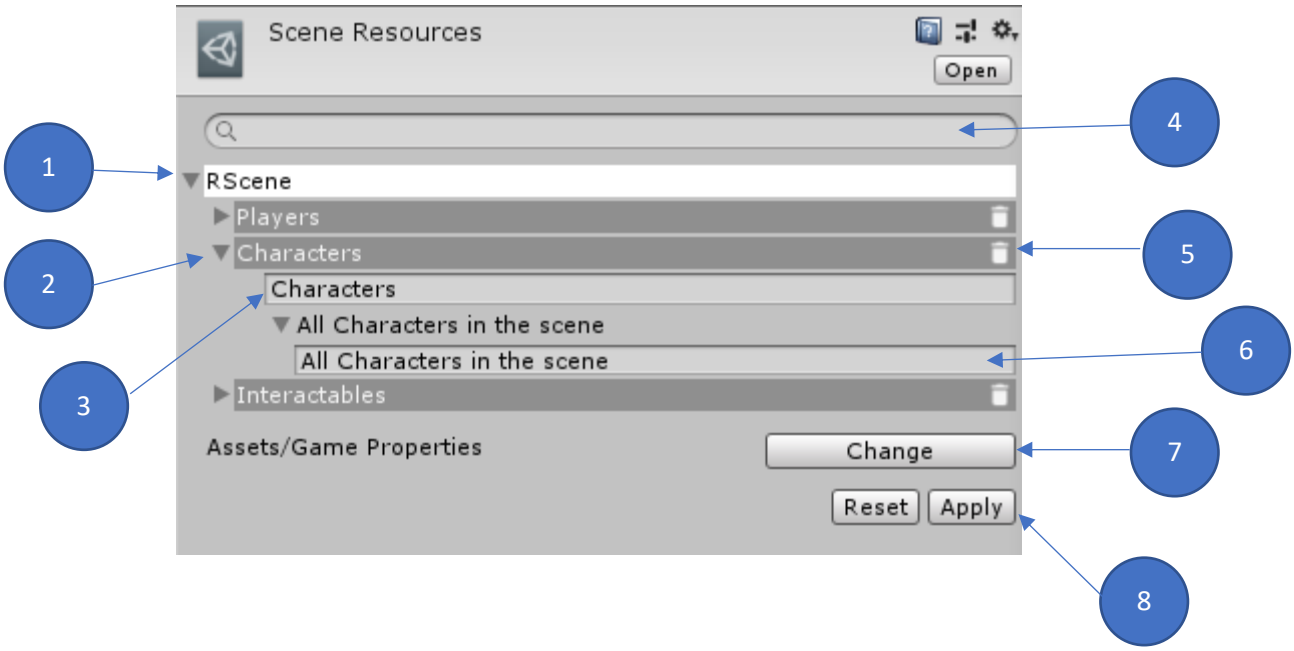
```
GameObject poolInstance = R.My_Character_Pool.GetPooledInstance();
```

Once the instance is no longer needed, the instance can be returned by calling

```
R.My_Character_Pool.ReturnPooledInstance(poolInstance);
```

Using the Scene Resources Asset

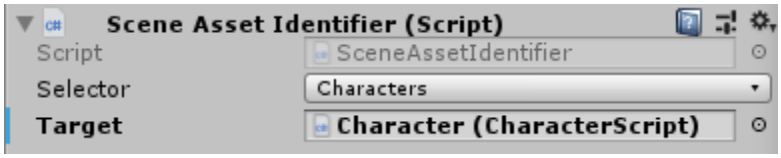
The Scene Resources asset is responsible for maintaining references to assets currently loaded in the scene. The Scene Resources system enables you to define categories of objects, tag objects with categories, and find objects by category at runtime. Here are the components of the Scene Resources asset:



1. Namespace – Namespaces are like folders – they can contain asset groups and other namespaces. Right click to rename, delete, or add child namespaces and assets. Namespaces can be dragged to move them into other namespaces.
2. Asset Group– These are scene assets you'd like to reference. Think of them like categories – at runtime you can refer to the "Characters" asset group and get a collection objects currently in the scene that fall in that category
3. Name – The name of this asset group
4. Search – Quickly find the asset group you are looking for
5. Delete – Deletes the Asset or Namespace
6. Comment – Useful for providing explanatory information about an asset group. When code for accessing the asset group is generated, this comment will be attached to the code and will show in Intellisense
7. Change Directory – Changes where generated code is stored. This code will be used to access the Asset Groups defined in this prefab
8. Apply/Reset – Click Apply to save changes, or Reset to discard them

Tagging Objects as Scene Resources

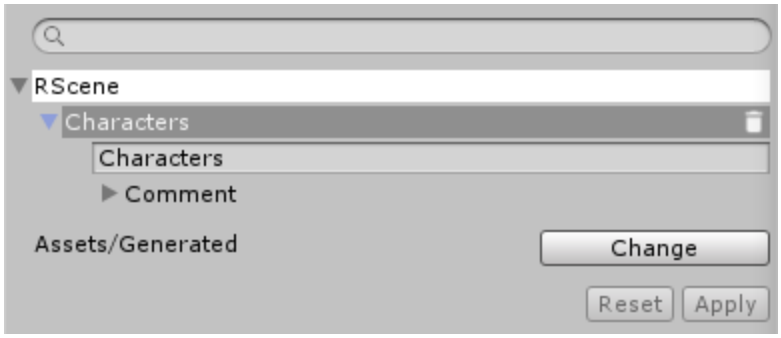
To identify an object with an asset group, add the Scene Asset Identifier component to that object.



Use the selector property to identify which category the object belongs in. Then set the Target property to a component on the Object. If this object is in the scene, searches for this category will return that component.

Accessing Scene Resources Via Code

Scene resources that are added to a category are available by code in the “RScene” static class . For example, given:



A call to:

```
Component[] component = RScene.Characters.GetAll();
```

will get all components currently in the scene identified by the “Characters” category.

There are other selectors generated in the Characters namespace as well. For example, a call to:

```
Component[] component = RScene.Characters.GetAll<CharacterScript>();
```

retrieves all CharacterScripts in the scene identified by the “Characters” category, and a call to:

```
Component[] component = RScene.Characters.GetAllWhere<CharacterScript>(x=>x.name == "A particular object");
```

Retrieves all CharacterScripts in the scene identified by the “Characters” category, where the name of the GameObject is “A particular object”.