Managing AssetBundles assets references in Unity

Pierrick Bignet Follow
Aug 14, 2018 · 6 min read

If you use asset bundles, you probably sacrificed the possibility to drag and drop assets in the editor and learned to use strings instead to reference your assets. In this article I'll explain the solution we put in place to mitigate this issue.

AssetBundles ?

Despite being often overlooked because of their shady documentation and tools, AssetBundles are pretty awesome, for several reasons:

- They **decrease your build size** by allowing you to split your assets into packages and download only those needed on the player device

need to push a new version to the store (Steam, Google Play, etc.)

- They **decrease "EnterPlayMode" time** by freeing up space in your "Resources" folder

The concept is simple:

1. You tag your assets with a bundle name
2. All the assets with the same bundle name are then packaged together, creating ... a bundle !
3. Instead of using Resource.Load() to load an asset, you will have to first load the bundle and then get the asset from the loaded bundle

Bundles are stored remotely (ie, on a simple HTTP server) and they all have a version number. When you load a bundle, the system will first check if there is one loaded locally (stored in the local storage) and if the local version matches the remote version. If not, the new bundle will be downloaded.

It took me some time to set-up an efficient process, and some tools helped me greatly :

- Sad Panda Studio AssetBundleManager is better and cleaner than Unity's, and now supports Async/Await. It allows you to easily load bundles, even while testing in Editor
- Unity's AssetBundleBrowser is a must have to set-up and check your bundle in Editor

Bundle creation and dependency management will probably be the topic of another post

Quick How-to

Here is a small code snippet from our BundleManager in City Invaders (sorry for the singleton pattern there):

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using Unity2Async;

public class BundleManager {
    private static BundleManager _instance;
    private Dictionary<string, AssetBundle> _assetBundles;
    private Dictionary<string, TaskCompletionSource> _loadingAssetBundles;
    private bool _init;

    private AssetBundleManager _assetBundleManager;

    private AssetBundleManager AssetBundleManager =>
        _assetBundleManager ?? (_assetBundleManager = new AssetBundleManager());

    public static BundleManager Instance => _instance ?? (_instance = new BundleManager());

    private async Task Init() {
        _assetBundleManager = new AssetBundleManager();

        if (Application.isEditor) {
            //_assetBundleManager.SetBaseUrl(Config.Instance.AssetBundleURL);
            _assetBundleManager.UseSimulatedMode();
        } else {
            _assetBundleManager.SetBaseUrl("http://mothership.com");
        }

        await _assetBundleManager.Initialize();
        _assetBundles = new Dictionary<string, AssetBundle>();
        _loadingAssetBundles = new Dictionary<string, TaskCompletionSource>();
        _init = true;
    }

    /// <summary>
    /// Return an AssetBundle using its name
    /// </summary>
    /// <param name="assetBundleName">The name of the asset bundle to load</param>
    /// <returns></returns>
    /// <exception cref="Exception">Will throw an exception if the bundle does not exists</exception>
    public async Task<AssetBundle> GetBundle(string assetBundleName) {
        if (!_init) await Init();

        var lowerName = assetBundleName.ToLower();

        if (_loadingAssetBundles.ContainsKey(lowerName)) {
            return await _loadingAssetBundles[lowerName].Task;
        }

        if (!_assetBundles.Contains(lowerName)) {
            var loadingTask = AssetBundleManager.GetBundle(lowerName);
            _loadingAssetBundles.Add(lowerName, loadingTask);

            var bundle = await loadingTask;
            _loadingAssetBundles.Remove(lowerName);

            if (bundle == null) {
                throw new Exception("Failed to load AssetBundle");
            }

            // we check again for concurrency reasons
            _assetBundles.Add(lowerName, bundle);
        }

        return _assetBundles[lowerName];
    }

    public async Task<T> LoadAssetFromString<T>(string assetBundleName, string assetName) where T : UnityEngine.Object {
        if (!_init) await Init();

        var bundle = await GetBundle(assetBundleName);
        var asset = bundle.LoadAsset<T>(assetName);
        if (asset == null) Debug.LogError("Failed to load Asset {0} from {1} ({2})");
        return asset;
    }
}
```


Note: This code uses .NET 3.5+ features, such as async/await.

This allows us to easily (down)load bundles and assets within them:

```
var sprite = await BundleManager.Instance.LoadAsset<Sprite>
("spriteBundle", "mySprite");
```

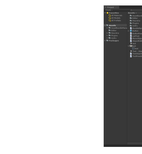
AssetBundle and references

After playing a while with AssetBundles, I realized that I was missing one big, simple feature: the possibility to simply drag & drop an asset in the editor.

When one **asset in a scene references another asset in the project, this asset will automatically be packaged in your game**. This goes against the asset bundle purpose. So, by default, when you want to load an asset within an asset bundle you will need 2 strings:

- The asset bundle name
- The asset name (and its type)

This makes asset management more cumbersome, and for many teams it can be a deal breaker, especially for artists and level designers. It also makes the system more error prone, even if can manage all this using Enums.



Can you imagine living without this ?

The solution

To benefit from the best of both worlds, I started thinking about a specific object reference class which would keep the link to the asset in the editor, and replace it by the asset's name and bundle name.

At runtime, I would load the actual asset using the previously mentioned BundleManager. And voilà !

Here is the piece of code I'm using for that purpose. Note that part of the file is compiled only for the editor (UNITY_EDITOR). You will also probably notice that the AssetReference getter is calling the Setter method to force refresh the asset reference name and bundle name.

Disclaimer: because I'm lazy and I learned to hate Unity's editor code, this example will use Odin (but not Odin's serialization). I may write an example with a custom editor / propertydrawer (spoiler: I won't anytime soon)

Quick usage example :

```
public class MonoObjectRefTest : MonoBehaviour {
    public SpriteBundleReference SpriteRef;
    public Image Image;

    private async void Start() {
        var sprite = await SpriteRef.LoadAssetFromBundle();
        if (Image) Image.sprite = sprite;
    }
}
```

Conclusion

I'm pretty happy with our solution and it serves its purpose quite well for now. I'm curious to hear about the solutions that other people may have found to handle asset bundles nicely. I'm convinced that this kind of implementation could be done way better, and I think that Unity should provide it by default considering that asset bundles are mandatory for almost every mobile games nowadays.

I would like to thank the Sirenix team for tidying up my code and helping me figuring out Odin's possibility.

* * *

Pierrick Bignet is a cofounder & CEO at Lone Stone. Lone Stone is currently hiring web and Unity developers in Nantes, France. The studio is specialised in Javascript and uses ReactJS, VueJS, Angular, Node and Typescript. Feel free to contact us.

Thanks to GoodKey.

Unity Unity3d Game Development Gamedev

Pierrick Bignet
CEO, Game Designer, Developer and best Ast2 player at @LoneStoneStudio
Follow