Tutorial - Using Addressables

0. Why use Addressables?

It can initialise assets asynchronously in the background, and stores Unity-type objects like Sprites and CardData. This means multiple mods can initialise at the same time without stealing 10 seconds (or minutes if you don't have the Harmony Suppressor subscribed. See section 4.2 Harmony Suppressor's code). However, this comes with the downside that they can get a bit large.

In this tutorial, we won't be exploring ways to use Addressables asynchronously. You will still be able to see the difference, especially if your mod has many image files.

Table of contents >

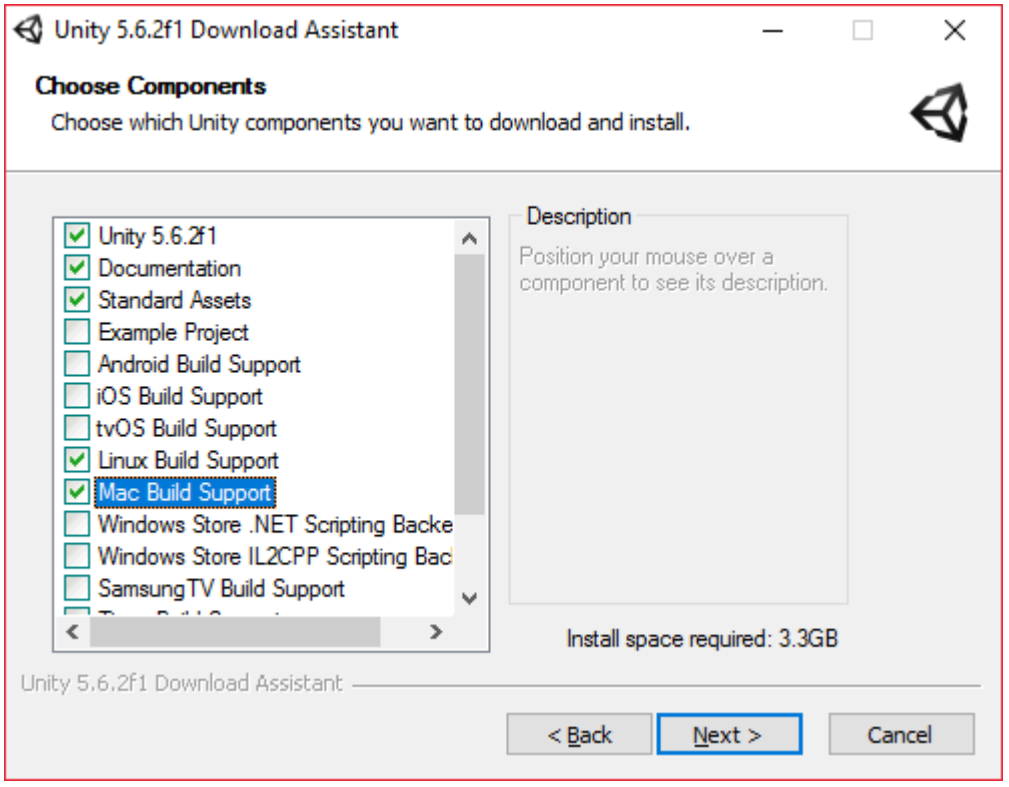
1. Prior Setup

1.1 Installing Unity v2021.3.16f1

The first thing you need is Unity. The version that Wildfrost uses is Unity 2021.3.16f1.

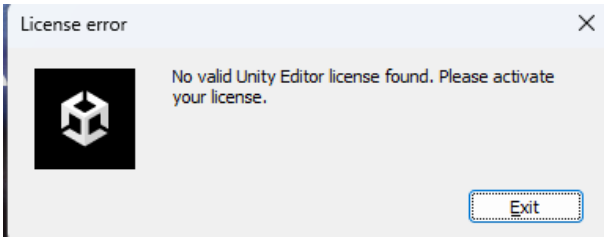
> ⚠ **Wrong Unity Version**
>
> Using the wrong version of Unity will result in possible issues in your exported AssetBundles, if it works at all.

Run the installer and you will be asked which components to include. Be sure **Windows, Linux and Mac** are all checked. Take note of the installation directory as you will need it for the next step.

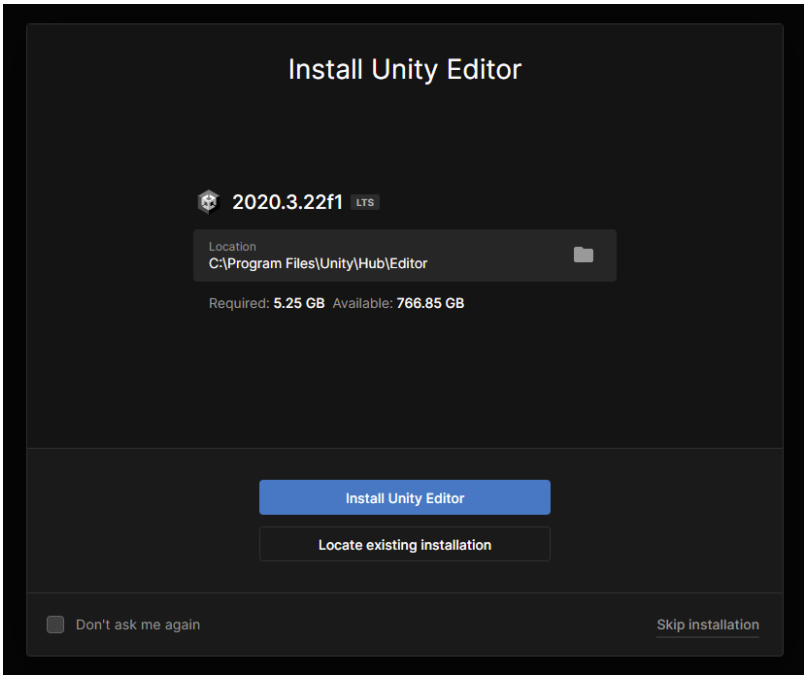


1.2 Download Unity Hub

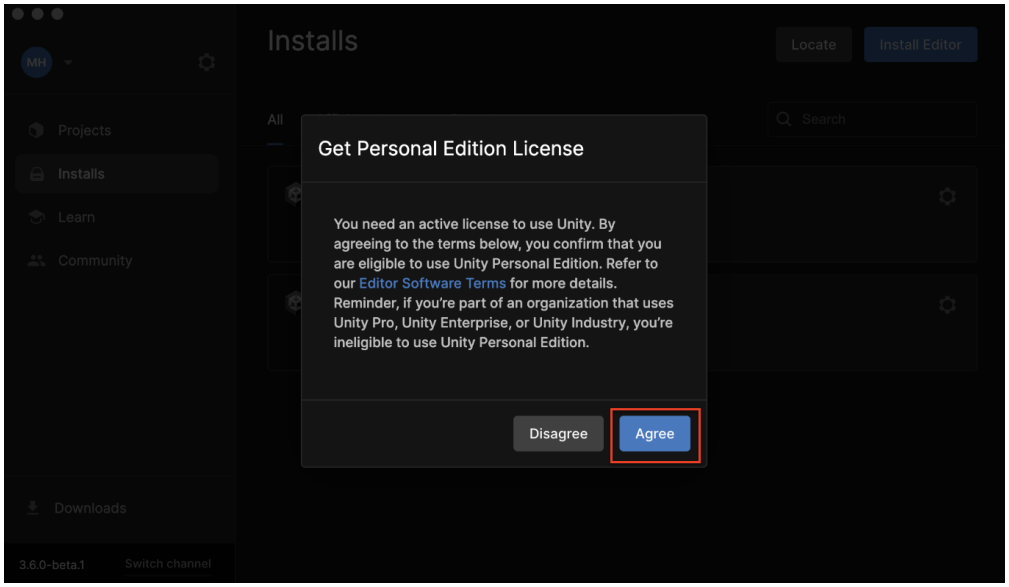
If you tried to open Unity before this step, you'd come across an error stating you don't have a valid Unity Editor license.



Not to worry, as you can get a personal license for free just by creating a Unity account and signing into it via the Unity Hub. When you reach this point in setting up Unity Hub, select **Locate existing installation** and find the Unity executable from the previous step.

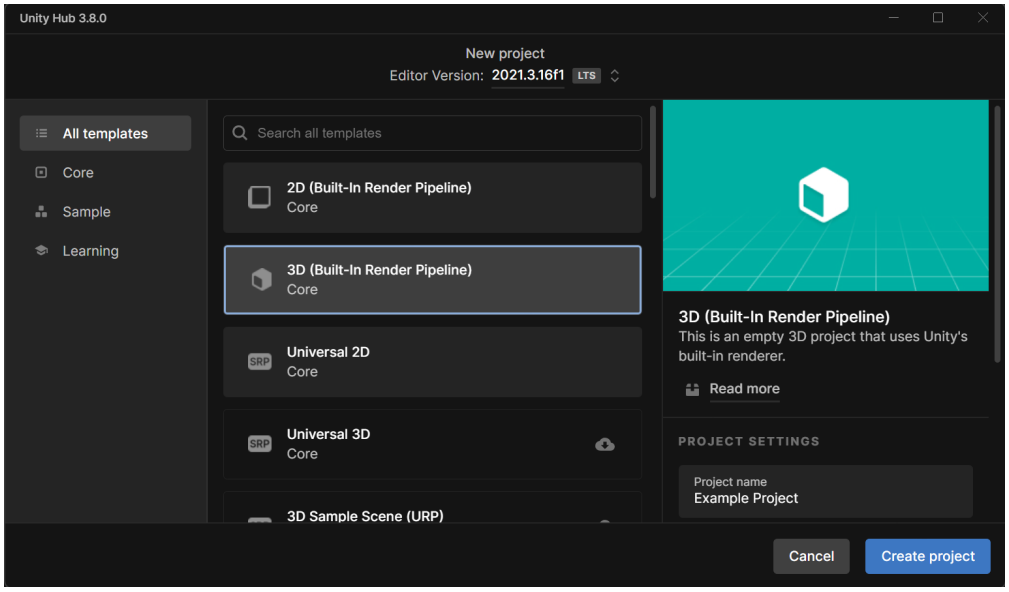


Review and agree to the **Editor Software Terms**. Your account will now have the Personal Edition License



2. Unity Project Setup

From Unity Hub, you can now choose to make a new project. For this tutorial, we will go with a 3D template. Choose a project path you can easily access and select **Create project**.

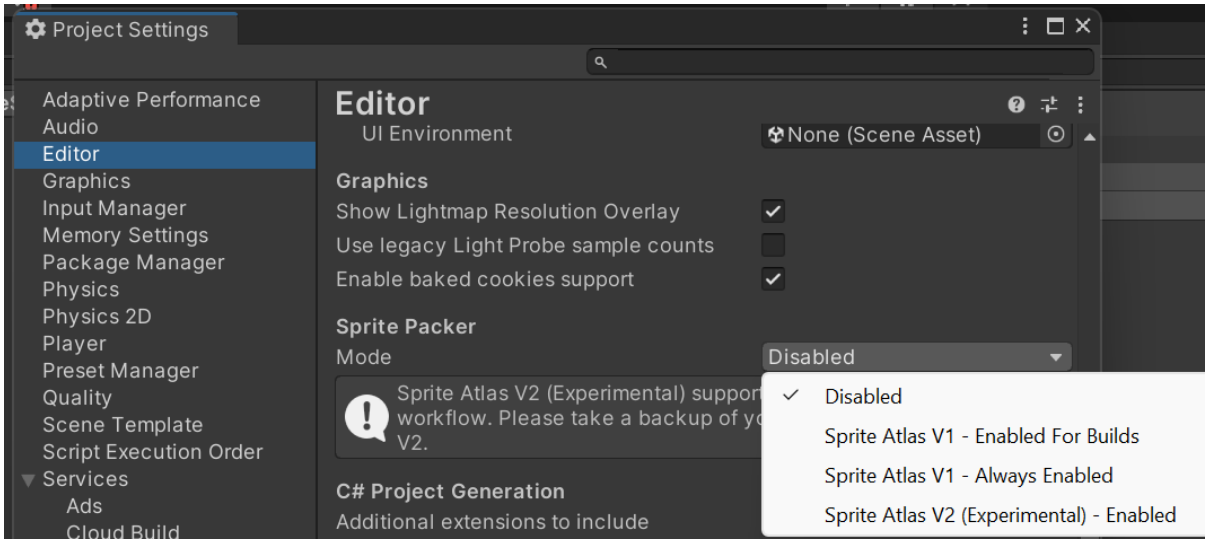


> ⚠ **Hold on!**
>
> Unity can take a while (a few minutes) to import files to create each new project. Note that the above image is from **Unity Hub**, but the project will open in another application: **Unity Editor**.

You'll come across an empty SampleScene, and it will stay mostly empty for this tutorial. Experiment with creating or moving GameObjects around as you wish!

2.1 Enable Sprite Packing

We'll do the easy bit first. Navigate to **Edit > Project Settigs > Editor > Sprite Packer** and find this setting. Just set it to **Always Enabled**.

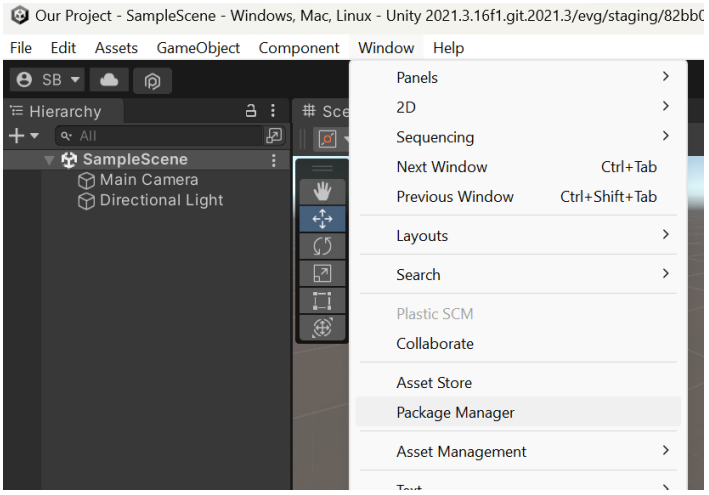


2.2 Required Unity Packages

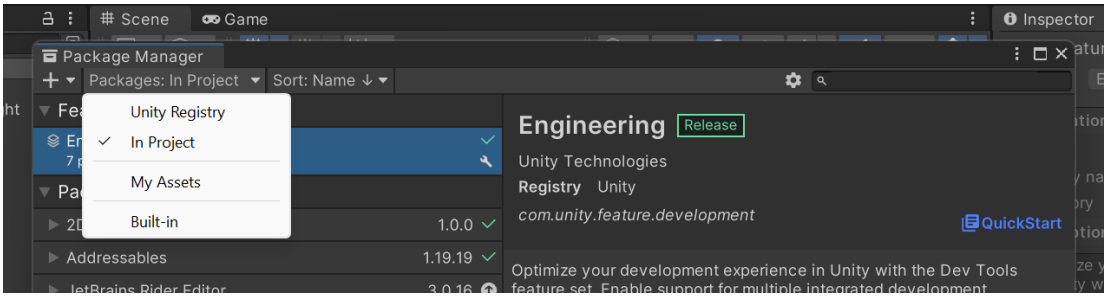
Now, we need a few Unity packages:

- Addressables
- 2D Sprite: Enables using Sprite Atlases. (Optional but highly recommended)

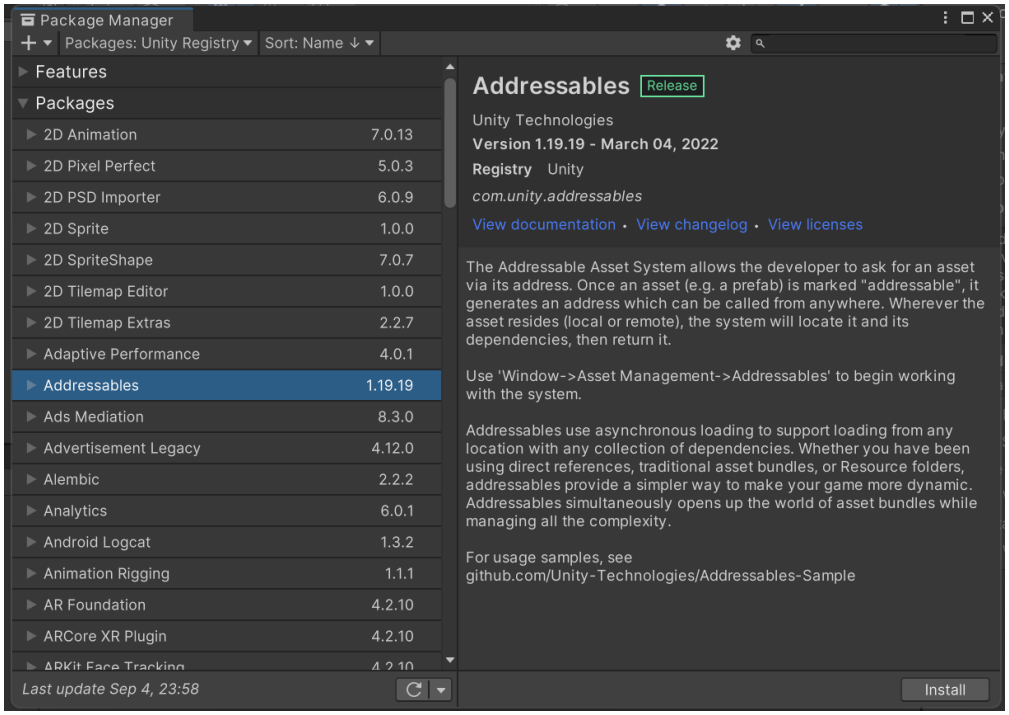
To get these packages, navigate to **Window > Package Manager**



On the Package Manager, select **Packages: In Project**, and switch it to **Unity Registry**.

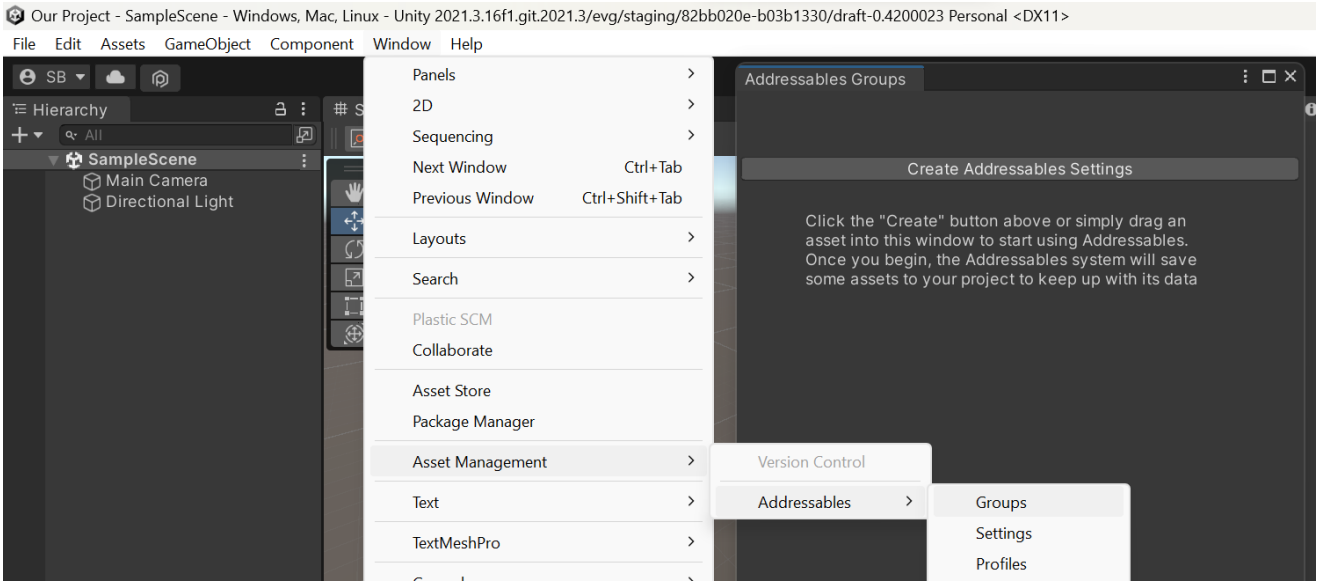


You will now be shown all the packages you can install. Either by searching for their name or scrolling down, **Install** the required packages.



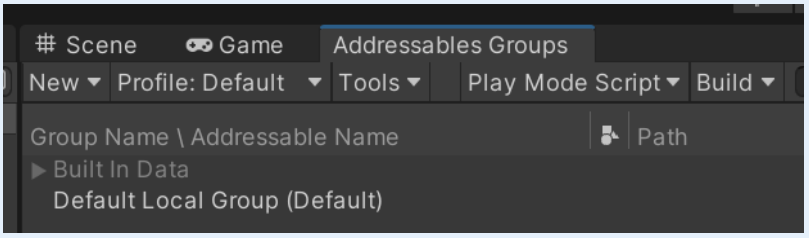
2.3 Create Addressables Settings

Once the Addressables package is installed, first go through **Window > Asset Management > Addressables > Groups** to open this window and select **Create Addressables Settings**.

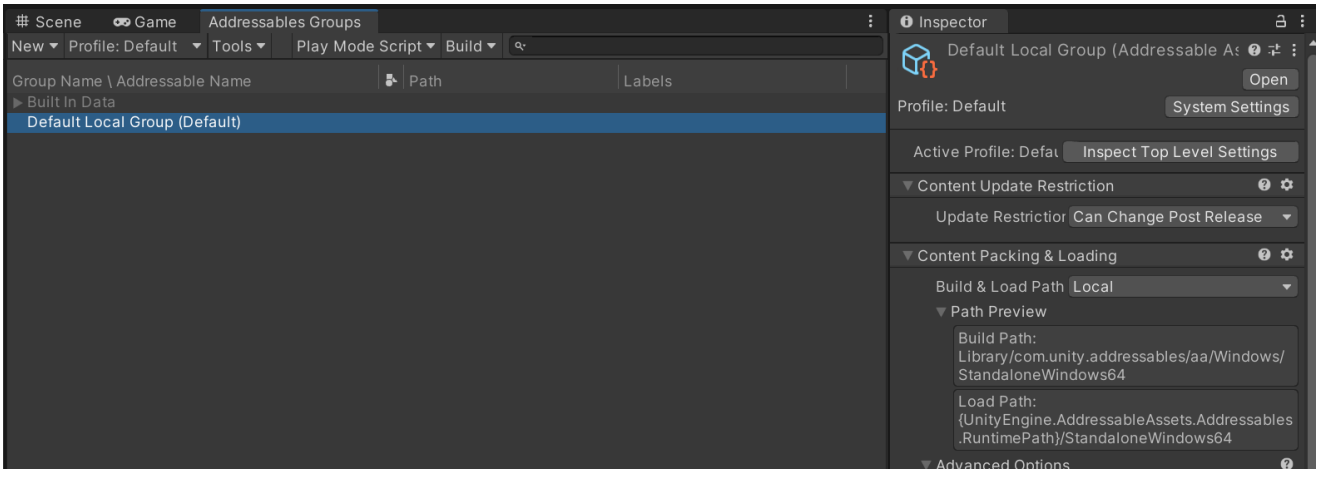


ⓘ Addressables Window

Since you'll be going back to this tab often, drag the label **Addressables Groups** into the row with **Scene** and **Game** to dock the window



Select the **Default Local Group** and take note of its `Build Path` as you'll be coming back to it later. Within the project directory, the `StandaloneWindows64` folder is where you'll find the **.bundle** files after building them. We're really more interested in its parent folder, which will contain the **catalog.json** that tells the game how to load its assets.

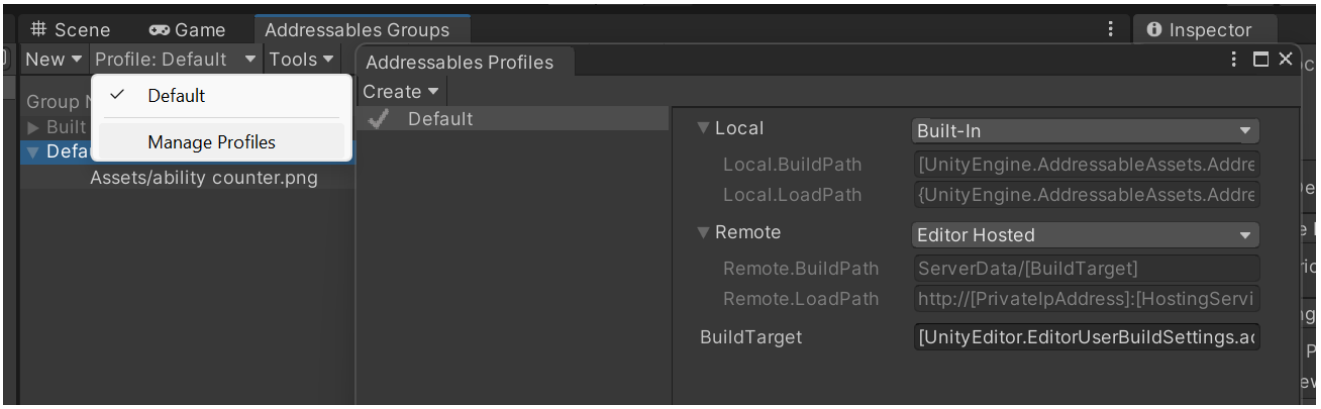


2.4 Create Addressables Profile for your Mod

In your mod code, have something along these lines

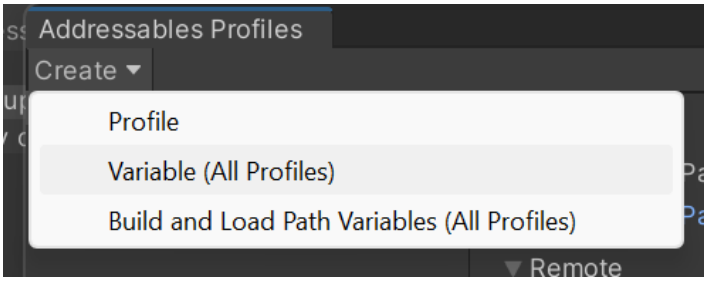
```
using UnityEngine.AddressableAssets;
using UnityEngine.AddressableAssets.ResourceLocators;

public class MyMod : WildfrostMod
{
    public MyMod(string modDirectory) : base(modDirectory)
    {
        instance = this;
    }

    public static MyMod instance;

    // Change "Windows" to whatever you want it named
    // This is where the addressables will be stored
    public static string CatalogFolder
            => Path.Combine(instance.ModDirectory, "Windows");

        // A helpful shortcut
    public static string CatalogPath
            => Path.Combine(CatalogFolder, "catalog.json");

    // rest of your code
}
```

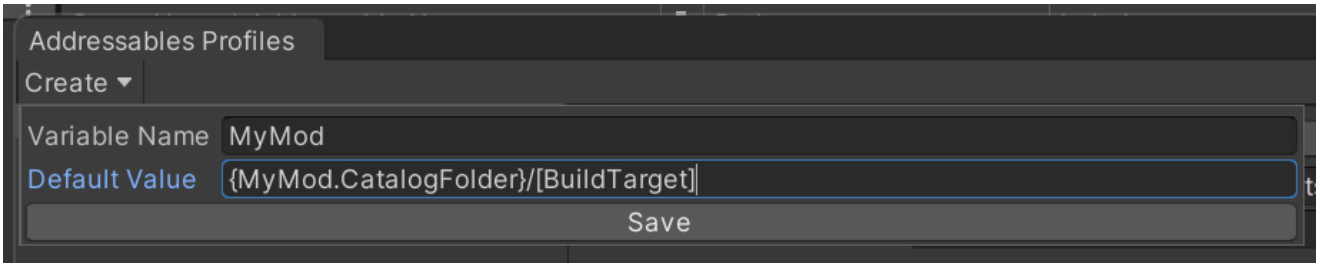
After setting these up, either create an empty folder with in your mod directory or remember it later. Back to the **Addressables Groups** tab, select **Manage Profiles** to open a new window.



By default, these settings will make it so the bundles only get read if they're at *the usual place* for this game (and it does work!). But you'd probably rather they be in your mod folder.

Create a new **Variable** with whatever name is easy to type, and set its value to the *fully qualified name* of the **CatalogFolder** property in your mod, surrounded by curly braces `{ }` followed by `/[BuildTarget]`.



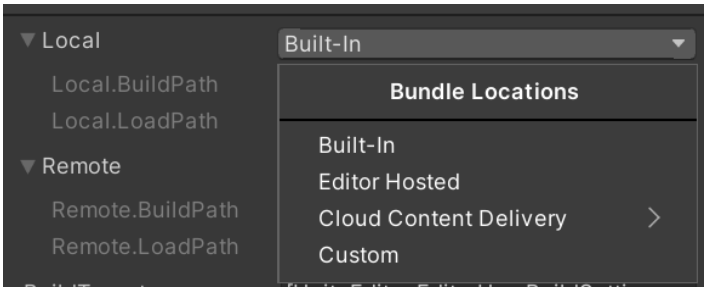


ⓘ Fully Qualified Name

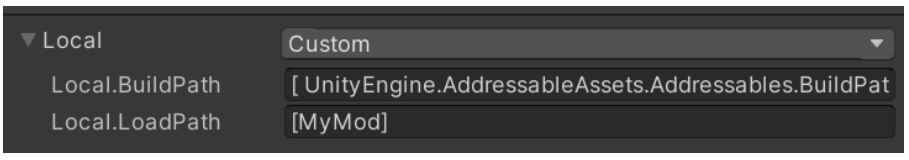
The fully qualified name is usually in the form of `<namespace>.<className>.<propertyName>`

Your namespace is usually specified in the mod code after all `using`-s and before the `class`-es. If you don't have one, then just use `<className>.<propertyName>`

Change the **Bundle Locations** from **Built-in** to **Custom**.



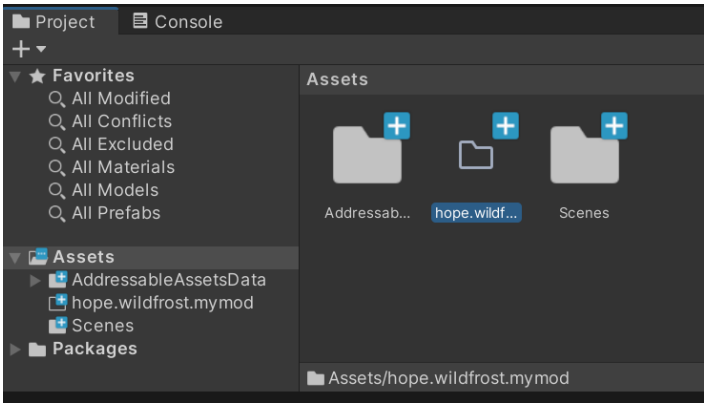
Now, add a space in the BuildPath after the `[`, and
Use the previous variable we defined, surrounded by square braces `[ ]` for the LoadPath



That should conclude all the necessary setup before we can play with addressables!

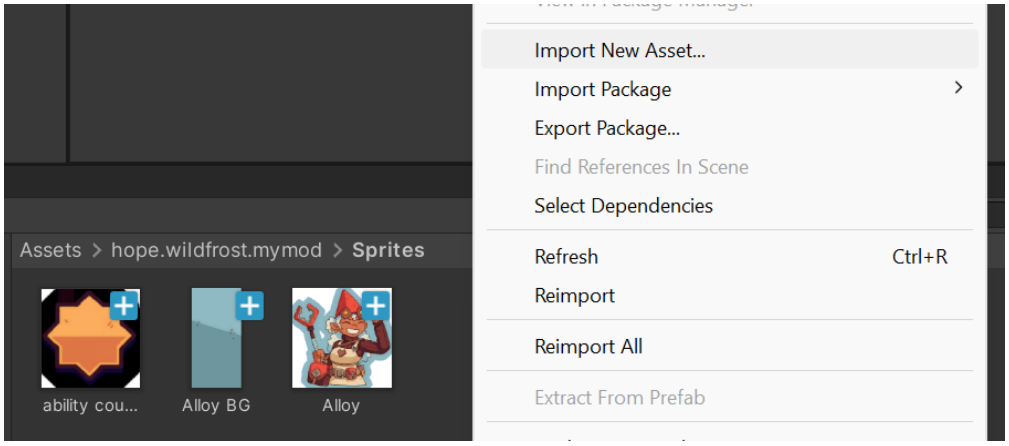
3. Addressables for real

Before anything, you should make a new folder with a name that won't be used by anyone else. Just be smart about it, or use the mod's GUID to be absolutely safe. I'll refer to this as **the main folder**.



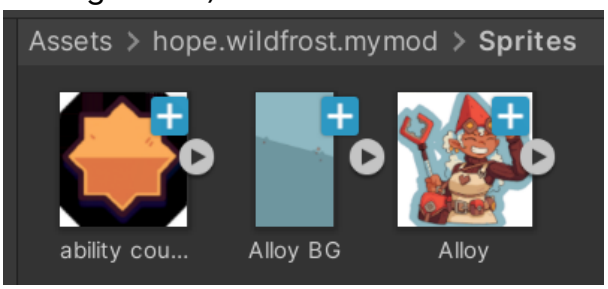
3.1 Create Sprites

Create another folder inside the main to put all our sprites in - as many as you want for your orgnization. Import your files into your folder(s) either by drag-and-dropping them, or **Right click > Import New Assets**.



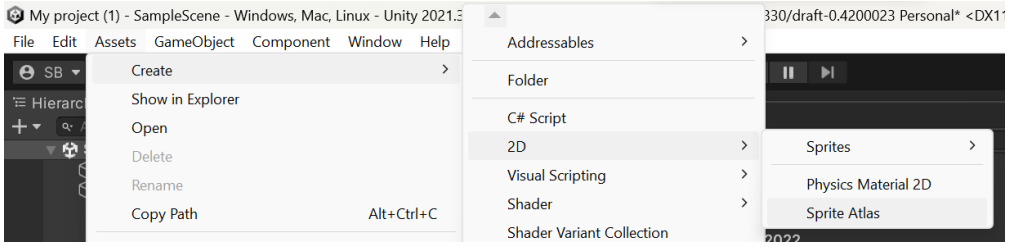
Select all of them either by **CTRL+A** or **Edit > Select All**, so they show up in the **Inspector**. In the inspector window, change the **Texture Type** to **Sprite**, and change the **Sprite Mode** to **Single**. **Apply** the changes.

If you did it correctly, you'll see an arrow next to the files indicating they each contain a single sprite (the entire image itself).

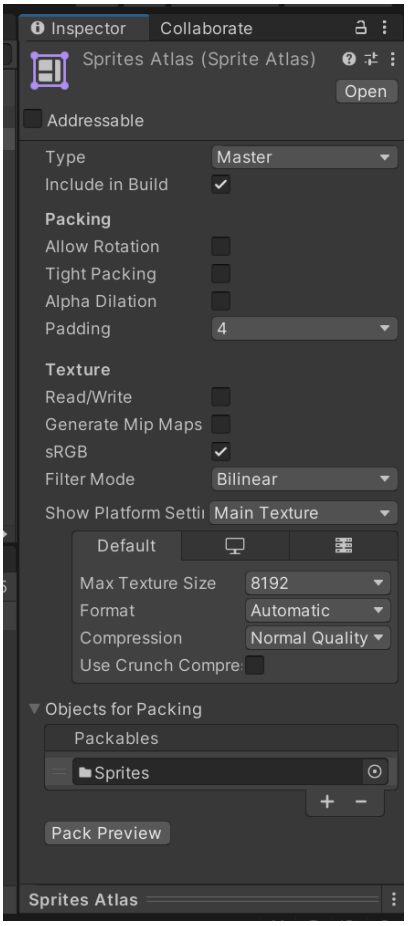


3.2 Create Sprite Atlases

If you installed the **2D Sprite** package from the previous Section 2.2, you'll be able to find new **Create Assets Menus**, specifically for Sprite Atlases.



Create one in the main folder and inspect it. Disable **Allow Rotation** and **Tight Packing**, then set the **Max Texture Size** to 8192 to reduce the number of atlases you make. In **Objects for Packing**, press **+** and search for the folder(s) you put the sprites in. This will automatically pack those sprites.



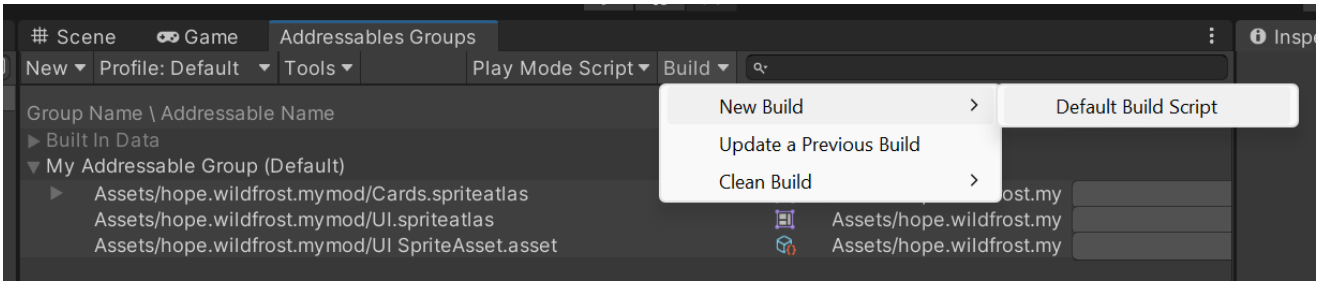
Preview it with **Pack Preview** to make sure it's good. If so, select the **Addressable** option at the top of the inspector to put it in the default group.

ⓘ Transparent padding is preserved

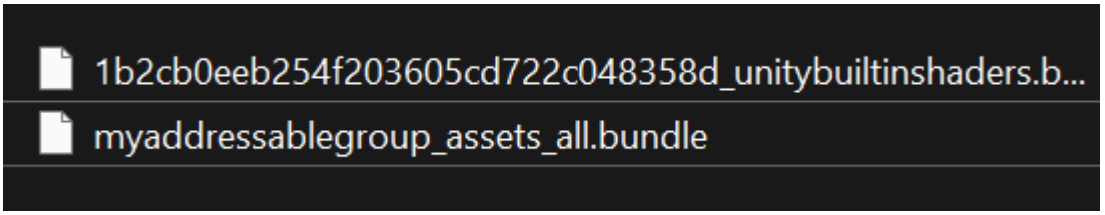
Even without tight packing, the transparent padding of your sprites will still be cut out. They'll be readded when you retrieve them from the atlas, so no changes should be necessary assuming they normally fit ingame.

3.3 Building/Loading the AssetBundle

Once you've made all your sprite atlases, prefabs, and whatevers you wanted addressable, select **Build > New Build > Default Build Script**.



Once that's done, the built bundles will appear in the Build Path you saw earlier (inspect the addressable group again for a reminder).



Normally this would be in a `StandaloneWindows64` folder, within a parent `Windows` folder or whatever platform you have. Copy the `Windows` parent folder into your mod directory - rename it if you want, we just need its contents. To load the addressable assets, first load its content catalog at a suitable place; we will do it at the start of `Load()`.

```
public static string CatalogFolder
        => Path.Combine(instance.ModDirectory, "Windows");
public static string CatalogPath
        => Path.Combine(CatalogFolder, "catalog.json");

public override void Load()
{
    if (!Addressables.ResourceLocators.Any(r => r is ResourceLocationMap map
&& map.LocatorId == CatalogPath))
    Addressables.LoadContentCatalogAsync(CatalogPath).WaitForCompletion();
        // if running this asynchronously, use the lower code
        // await Addressables.LoadContentCatalogAsync(CatalogPath);

    // ...
}
```

The extra `if` statement is just there to prevent the catalog from being reloaded with the mod. The `WaitForCompletion()` here makes the code run synchronously, ie we don't let it run in the background.

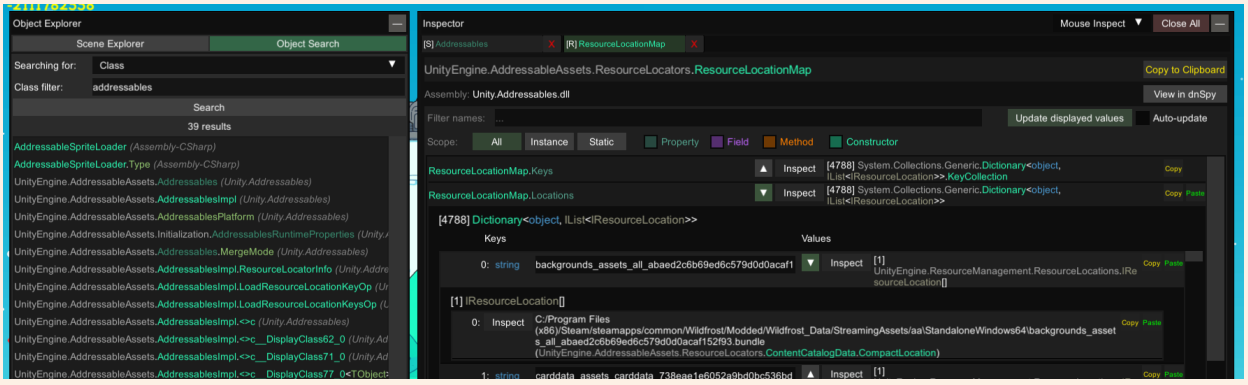
Now for the assets:

```
// ...
public static SpriteAtlas Cards;
public static SpriteAtlas UI;

public override void Load()
{
    if (!Addressables.ResourceLocators.Any(r => r is ResourceLocationMap map
&& map.LocatorId == CatalogPath))
    Addressables.LoadContentCatalogAsync(CatalogPath).WaitForCompletion();
    // You can use UnityEngine.Object as the type as a catch-all
    Cards = (SpriteAtlas)Addressables.LoadAssetAsync<UnityEngine.Object>
($"Assets/{GUID}/Cards.spriteatlas").WaitForCompletion();
    UI = Addressables.LoadAssetAsync<SpriteAtlas>
($"Assets/hope.wildfrost.mymod/UI.spriteatlas").WaitForCompletion();

    // ...
}
```

⚠ "Failed to read data for the AssetBundle ..."

If you get this error, what most likely happened is it couldn't find the bundle file. This happens if its path in the catalog points to the wrong place.

Check that you set it right from Unity Explorer. Search for the **Addressables** class and find the `ResourceLocationMap` that corresponds to yours. Check the value of one of its bundle's Locations - if it's wrong, fix the `Load Path` as in Section 2.4



ϟ Don't change `Addressables.BuildPath`

You may get inspired to directly build the catalog directly in your mod folder. While it's easy to do this, a small mistake can recursively delete many files, as witnessed first-hand by the author. sigh

4. Extras

4.1 Using Sprite Atlases

The syntax for getting a sprite from a sprite atlas is using the `sprite = atlas.GetSprite(nameWithoutExtension);`. Here's a simple extension method to quickly replace on CardDataBuilders, using my sprite atlas `MyMod.Cards`.

```
public static class CardHelpers
{
    public static CardDataBuilder SetAddressableSprites(this CardDataBuilder
builder, string mainSpriteName, string backgroundSpriteName)
    {
        Sprite mainSprite = MyMod.Cards
                .GetSprite(mainSpriteName.Replace(".png", ""));
        Sprite backgroundSprite = MyMod.Cards
                .GetSprite(backgroundSpriteName.Replace(".png", ""));
        return builder.SetSprites(mainSprite, backgroundSprite);
    }
}
//...
// Example usage
new CardDataBuilder(this)
    .CreateUnit("sylveon", "Sylveon")
    .SetStats(10, null, 3)
    .SetAddressableSprites("sylveon", "sylveonBG.png")
    // ".png" doesn't matter anymore
```

4.2 Harmony Suppressor's code

The biggest performance boost I recently found is that subscribing to Harmony Suppressor (hence having its patch run) cuts down load times from 2 minutes to 7 seconds. So, you can either set Harmony Suppressor as a dependency, or include its patch manually.

```
[HarmonyPatch(typeof(WildfrostMod.DebugLoggerTextWriter), "WriteLine")]
class PatchHarmony
{
    static bool Prefix() { Postfix(); return false; }
    static void Postfix() => HarmonyLib.Tools.Logger.ChannelFilter =
(HarmonyLib.Tools.Logger.LogChannel)(8+16);
}
```

and `HarmonyInstance.PatchAll(typeof(PatchHarmony));` in the mod class constructor

4.3 Creating CardData/StatusEffects in Unity (without DataBuilders)

If you recall from Section 3.2, I pointed out specifically that a package added a **Create Asset Menu**. If you've delved into the code for CardData or StatusEffects, you may also have seen similar attributes above the class, eg

```
[CreateAssetMenu(fileName = "CardData", menuName = "Card Data")]
public class CardData : DataFile, ISaveable<CardSaveData>
{//...
}
[CreateAssetMenu(menuName = "Status Effects/Instant/Kill", fileName =
"Kill")]
public class StatusEffectInstantKill : StatusEffectInstant
{//...
}
```

What this actually does is, when referenced in your Unity project, adds a menu option you can access from **Assets > Create**. From there, you can create an instance of your scriptable object that can be set addressable and loaded asynchronously, skipping the whole DataBuilder ordeal. This is heavily simplified, and I won't bother going into the specifics (yet). Maybe in the future?

Somewhat relevant: You can also create SpriteAssets like the one for sprites in text boxes via **Assets > Create > Text > Sprite Asset**, instead of using my VFX Tools mod.

Similarly, using Unity you can create and use **AssetReferences** where the game normally asks for them, like for **routinePrefabRef** for **CampaignNodeTypeEvents**, or for the **LocaleFont** of the **FontSetterSystem**.

5. Resources / References:

- Introduction to Scriptable Objects
  Goes more in depth on this. It contains some materials for trying out scriptable objects, and there's also a long-winded 1 hour YouTube tutorial going through the introduction (link to avoid enabling cookies)
- Risk of Rain 2 mod development using ThunderKit
  ThunderKit seems to be a Unity extension for the convenience of certain aspects of mod development? I haven't tried this yet but Miya (game's mod api developer) says it's good so give it a shot
- Unity's Addressables documentation