## Upgrading from previous versions

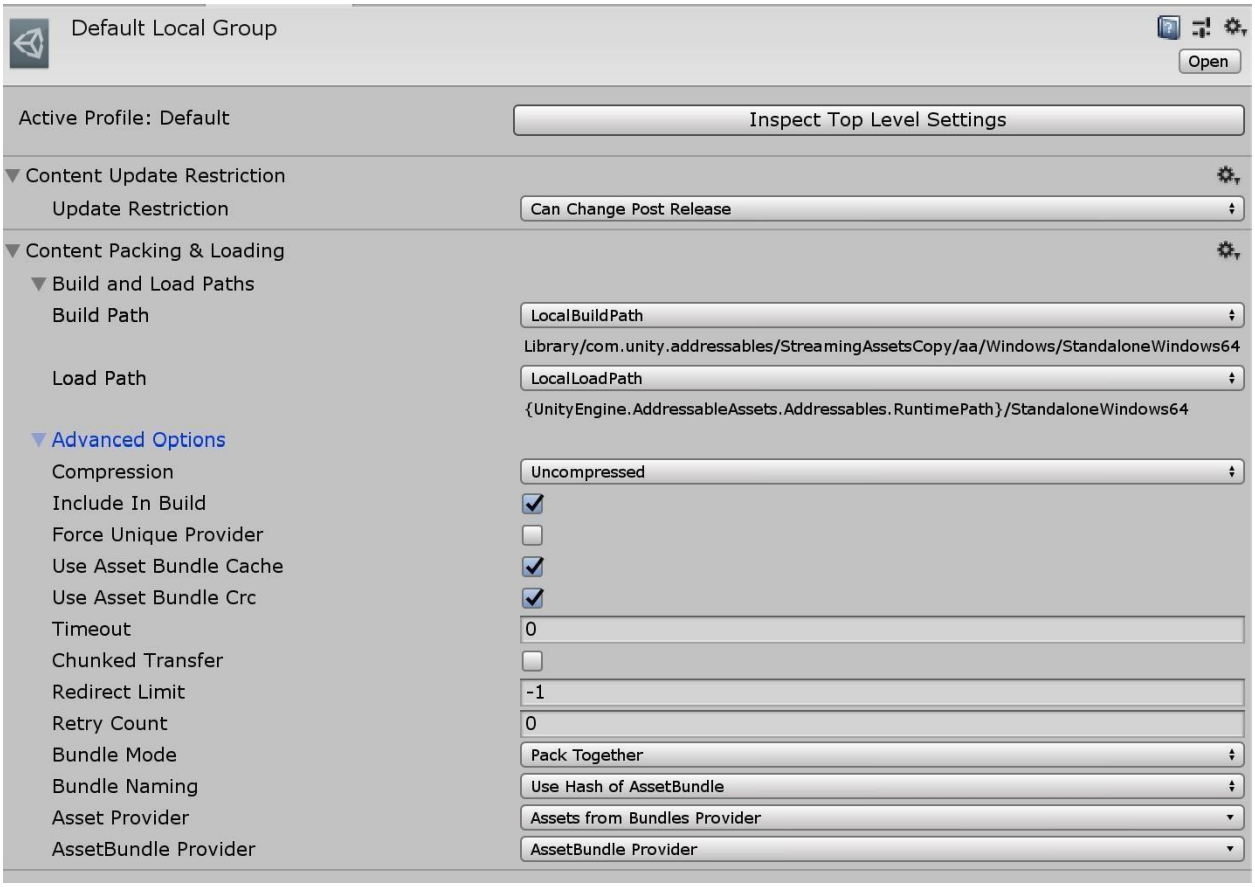
You should remove the previous UMA folder before importing! If this is not possible, at the very least, remove the CORE and EXAMPLES folders.

To use this version of UMA with Addressables (very much recommended), you must have the *Addressables* package 1.18.19 or later installed from the package manager.

**UMA will not download raw asset bundles in this version**. If you want to use raw Asset Bundles, you will need to download them, and add the contents to the global library using **UMAAssetIndexer.Instance.AddFromAssetBundle(yournewbundle);** or you can add each item manually using UMAAssetIndexer.Instance.ProcessNewItem(item) for each item in the bundle. You would need to manage the asset bundle manually.

To upgrade a project:

- Install Addressables **1.18.19** or greater
- Open the preferences window, and enable addressables in the UMA preferences section.
- In your scene, delete the UMA_DCS prefab. This is very important. You should not use this older prefab when using Addressables.
- Add the UMA_GLIB prefab from the "UMA/Getting Started" folder.
- Open the Addressable groups window and dock it. (Windows/AssetManagement/Addressables/Groups)
- Edit the default settings in the default group as needed. These settings will be used for the generated groups. Here is the UMA Default:




- Open the new "Global Library" window. This is new for this version of UMA.
- Select the Generate/Generate Single Group option from the Addressables menu in the Global Library. The system will generate the addressable groups. This may take some time.
- In the addressable groups screen, you should build the asset bundles: Build/New Build/Default Build Script.
- Select the “play mode script”. I use “use existing build”, as that validates the bundles are generated correctly.

Note: In your build script, you should add two functions, and call the first function before your build starts, and the second function after your build is completed.

```
        // Call this right before you build your bundles!
        public static void GenerateUMAAddressables()
        {
            // Clear the index, rebuild the type arrays, and then query to project for the indexed types, and
            // add everything to the index. Do not add the text assets (only needed if loading characters from resources)
            Debug.Log("Rebuilding asset index.");
            UMAAssetIndexer assetIndex = UMAAssetIndexer.Instance;
            try
            {
                // This is the function that makes sure everything is clean and tidy
                assetIndex.PrepareBuild();
            }
            catch (Exception ex)
            {
                Debug.LogException(ex);
            }
            Debug.Log($"Generating UMA addressable labels.");

            // Generate all UMA addressable labels by recipe. Every recipe gets a unique label, so when that
            // recipe needs to be loaded, all bundles that contain that item are demand loaded into memory.
            // they are unloaded when there is no active character using any of the assets.
            UMAAddressablesSupport.Instance.GenerateAddressables(new SingleGroupGenerator { ClearMaterials = true });

            // Make sure that the global library has a reference to every item that is not addressable.
            // This ensures that they item is included in resources. (Since the items are built dynamically,
```

```
            // they must be able to be loaded at runtime either through addressable bundles or
resources).
            Debug.Log($"Adding UMA resource references");
            assetIndex.AddReferences();
        }


        // Call this right after your bundles have completed building!
        /// <summary>
        /// This will reset the materials on the assets by looking up the materials in the library.
        /// This needs to happen after the bundles are built.
        /// </summary>
        public static void UMAPostBuildMaterialUpdate()
        {
            Debug.Log($"PostProcessBuild - Adding UMA resource references");
            try
            {
                UMAAssetIndexer.Instance.PostBuildMaterialFixup();
            }
            catch (Exception ex)
            {
                Debug.Log($"PostProcessBuild - Adding UMA resource references failed with exception
{ex.Message}");
            }
        }
```