Mod Creator

Introduction

The Mod Creator is a simple tool for the Unity Editor intended to make modding Passion Eye easier.

As of right now, Passion Eye supports only the following types of mods:

- **Studio objects** are items that can be spawned and manipulated in the studio.
- **Character objects** are items that are attached to characters, like hair, clothing, accessories and textures.
- **Scene mods** are scenes that are used as 3D backgrounds.

Features

- Studio object mods
- Character object mods
- Scene mods
- Custom code for modded objects
- Multiple modded objects in one mod
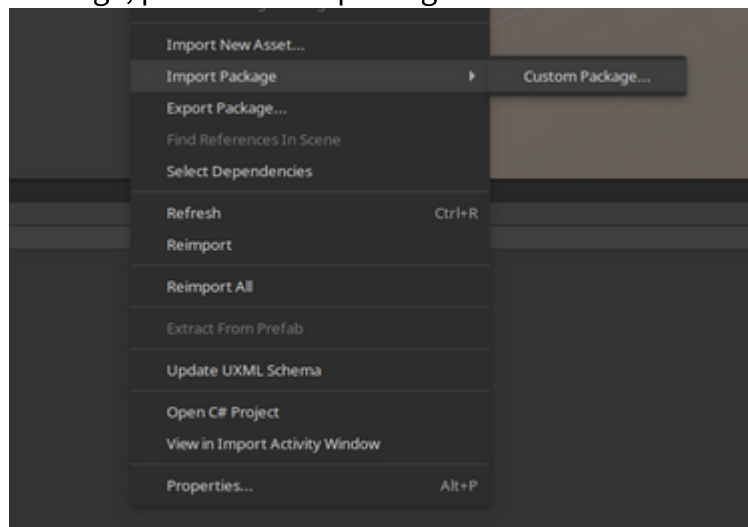- Custom shader support

Setting up the Unity Project

Create an Unity 3D (URP) (**2022.3.20f1**) project and proceed with the next steps.
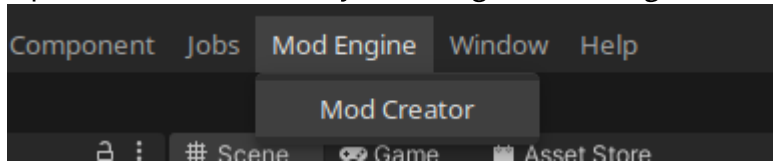Our shaders use Unity URP, make sure that the project template is correct.
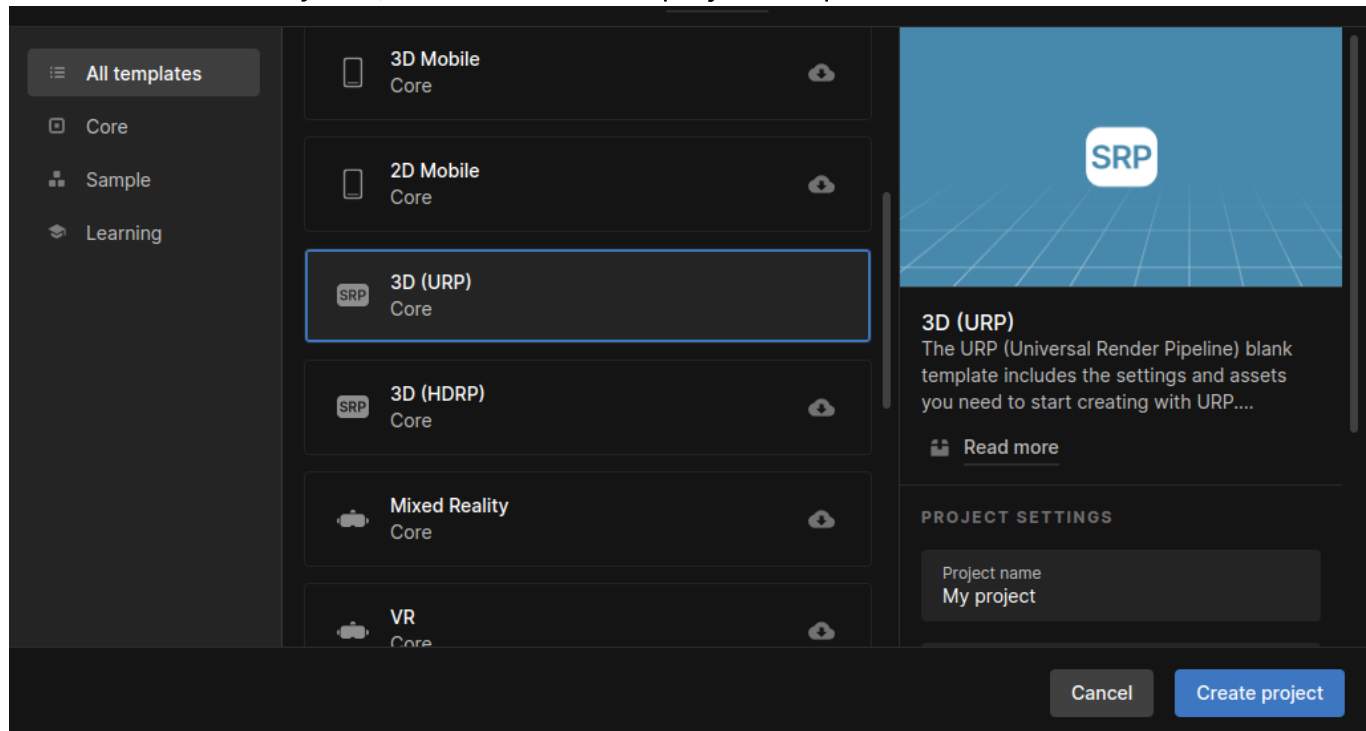
Installation

Import the included unitypackage by right clicking the Assets view -> Import Package -> Custom Package, point it to the package.
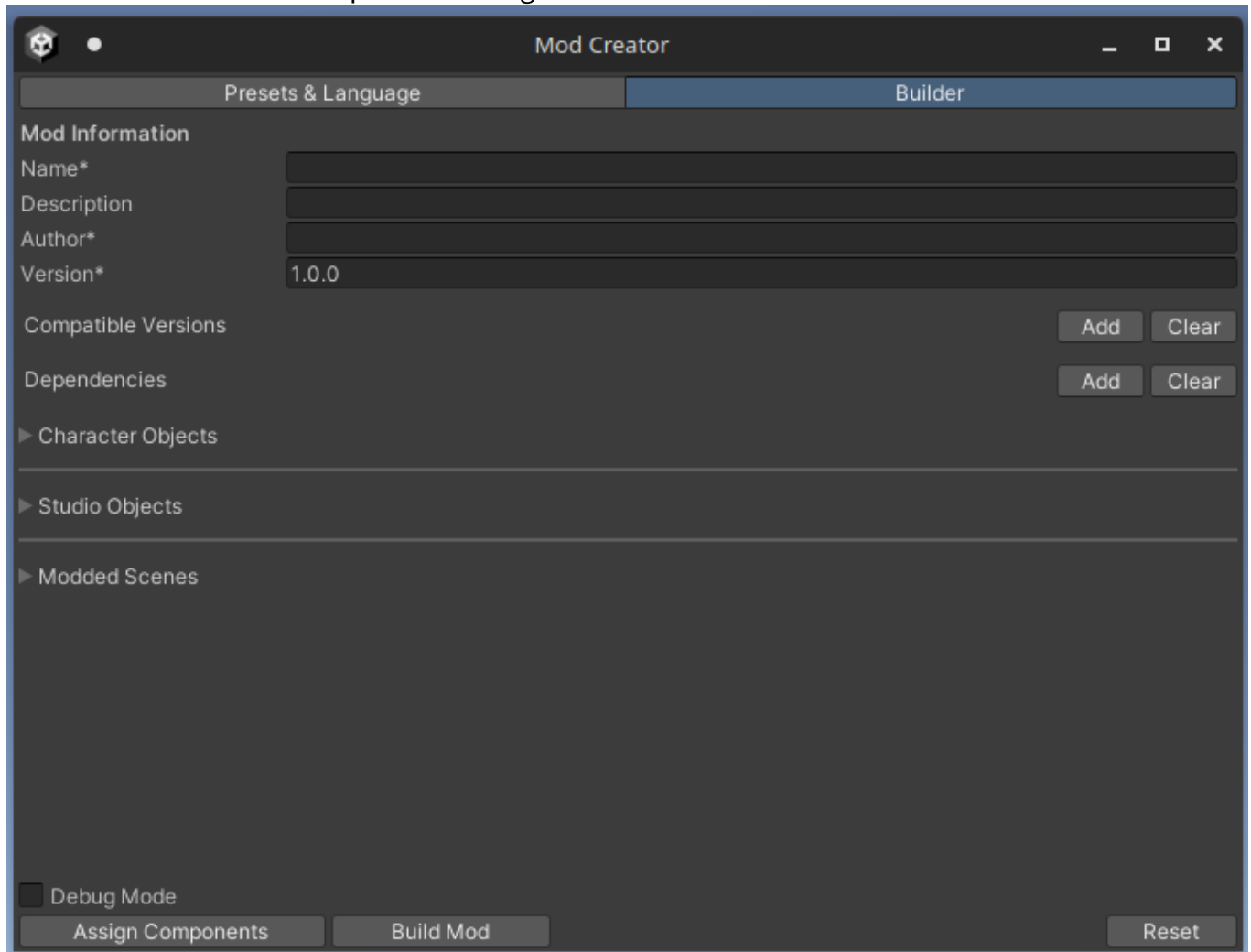
UI Overview

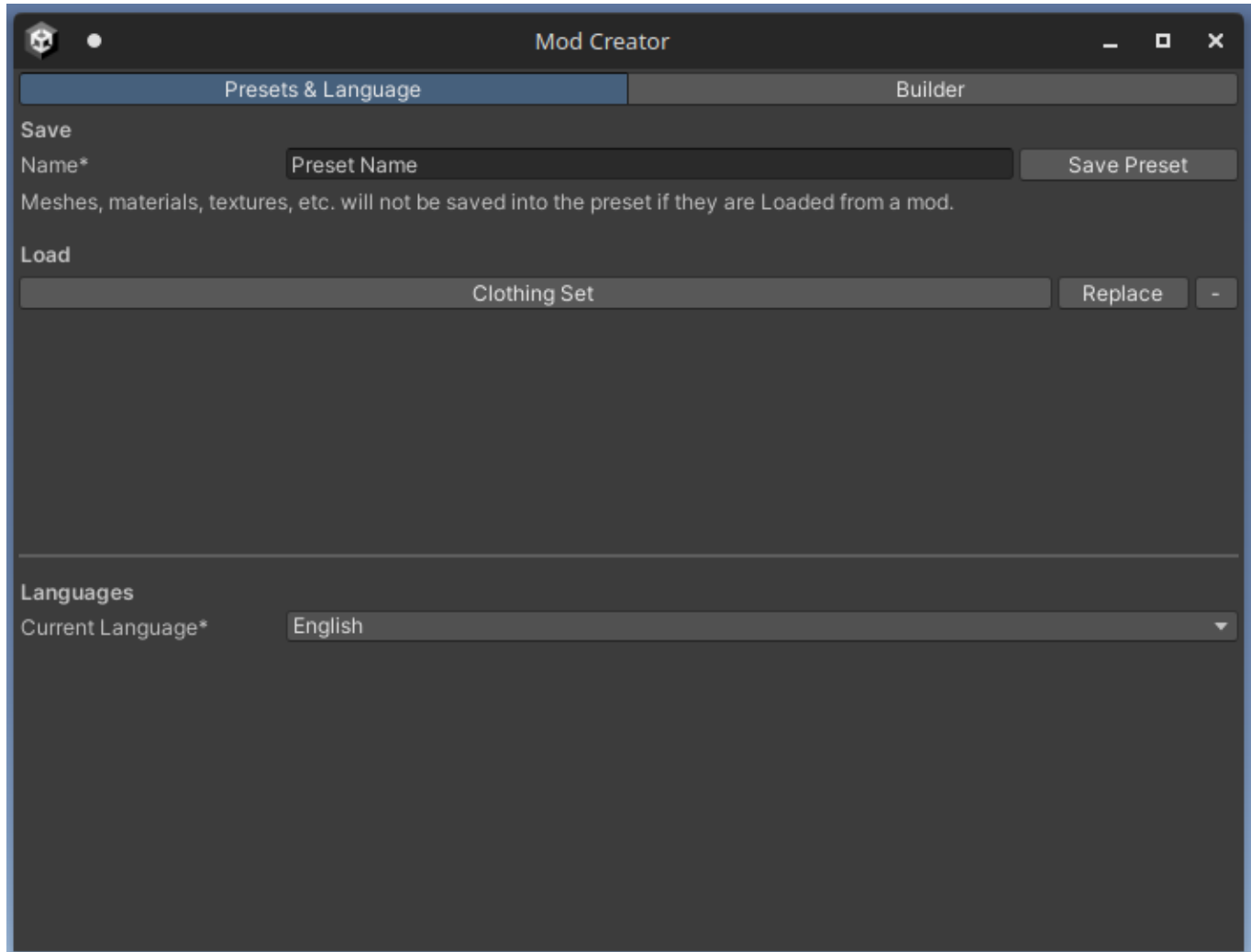Open the Mod Creator by selecting the *ModEngine -> Mod Creator* tab.

The Mod Creator window opens revealing the default view of the creator.

Presets

To make modding easier, there is an option to save and load the complete Mod Creator state by using Presets.

Using this, you can quickly update a mod by skipping most of the set up, done with a single click.

Entering a name and clicking **Save Preset** will save the current mod creator state to disk.
This will save everything in the Mod Creator: basic mod info, each component set up, assigned game objects, advanced mode code, etc.

The presets reference assets in the Unity project, so any changes to the assets themselves will be reflected onto the presets.

Once there are saved presets, simply click on the name button and it will load everything associated with it.

Removing the preset is done by clicking the **-** button next to the preset name, replacing a preset overwrites it, keeping the same name.

Language

We strive to support as many languages as we can in the game, that also includes the mod creator.

Selecting a language from the languages dropdown immediately applies the selected language to the entire interface.
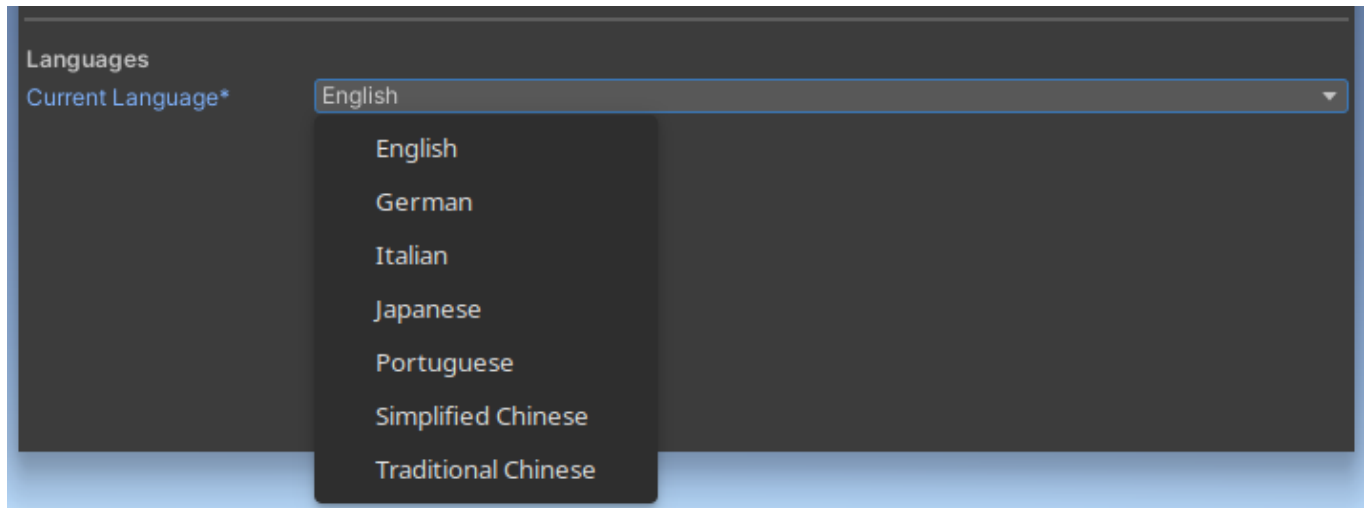
Mod Creation Setup

Enter an empty scene you intend to use for modding and open the Mod Creator.

To successfully make a mod, you need to specify its **Name**, **Author** and **Version** as a minimum. Filling in the **Description** is recommended.

Once you have set up the basic information, proceed to add a Character object, a Studio object or a Modded Scene by clicking **Add** to their respective list.
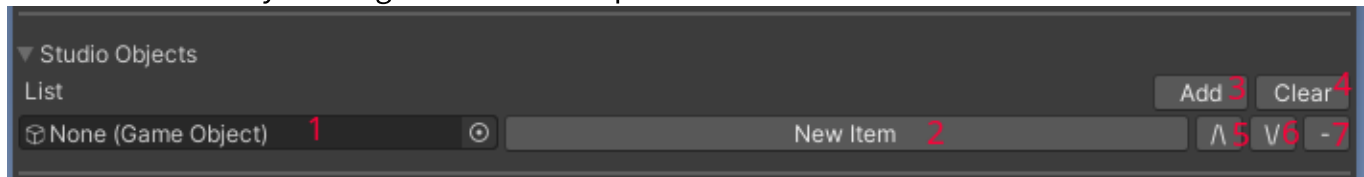

1. Reference to the mod object GameObject
2. Configuration and component information of the mod object
3. Add a mod object
4. Remove all mod objects under given category
5. Move the mod object up the list
6. Move the mod object down the list
7. Remove a mod object

Checklist

- Make sure that the objects you will use for your mod are in the scene.
- The objects that you assign to modded objects are in the **root** of the scene.
- All meshes, images and textures used in mods are marked as `read/write` in the import window.
- Image width/height are in a multiple of 4.
- Images have `sRGB mode` enabled in the import window.
- Icons have their import window `Texture Type` set to `Sprite (2D and UI)`
- Icons have their import window `Max Size` set to `128` (or `256` for Modded Scenes)
- Materials use shaders that are provided in the Mod Creator package or use custom shaders in the `Mod/` namespace.

Icon import settings example:
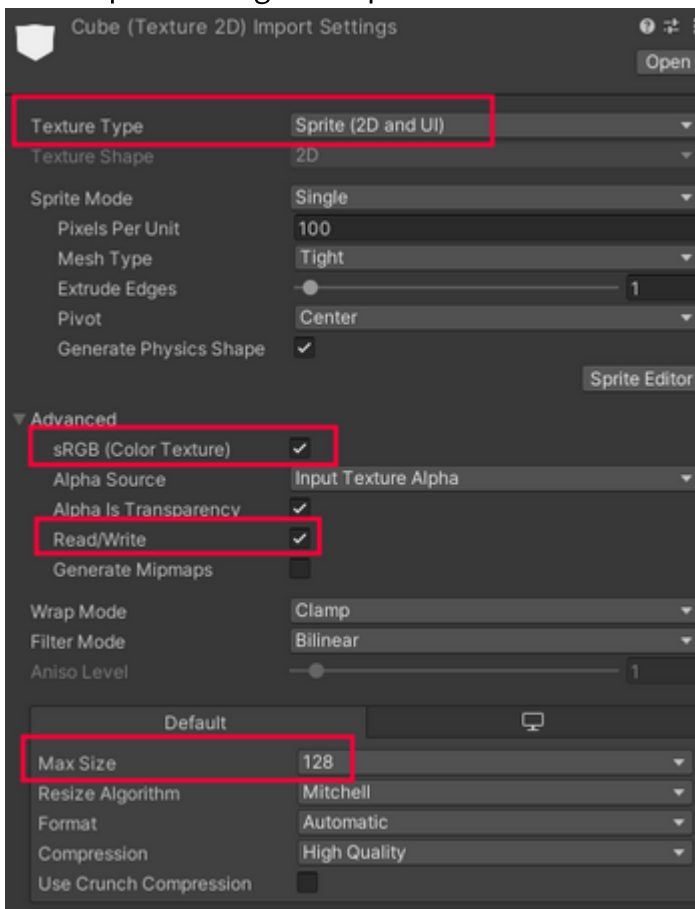
Material used shader example:
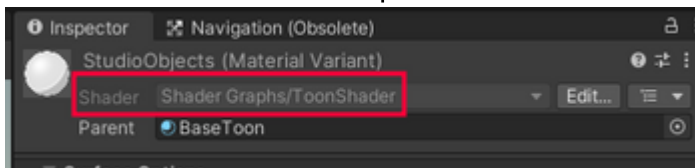
Custom Shaders

- Make sure your custom shader is in the `Mod/` namespace or it will not be included in the mod.
- Custom shaders are built for both Windows and Linux platforms and included in the mod, loaded for the correct platform during runtime.
- If you want your shader to be editable in the Material Customizer, add `Toon/` to its namespace, for example the resulting name would be: `Mod/Toon/MyShader`

Usage of our Toon shaders is highly recommended to maintain consistent visual style and allow them to be edited in the Material Customizer during gameplay.

Proceed to the next section depending on what mod you are making.

LODs

- If you want to use LODs, make sure to have at least three, otherwise the modded object might not appear correctly on different graphics settings.
- Make sure the LOD objects are named appropriately: `LOD0`, `LOD1`, `LOD2`, etc.
- Upon building, Mod Creator will automatically create LOD groups and assign the correctly named LODs if they exist.

NSFW

All modded items must be specified if they are NSFW or not. Any items that are marked as NSFW will be hidden from the SFW mode of the game.

Transforms

Modded item transforms are serialized as they are, which means any offset of the object in the modding environment is also reflected in game.

Creation of a Character Object Mod

Begin creation of a character object mod by clicking **Add** in the Character Object list.

1. Assign a GameObject by dragging one from the scene or selecting it manually.
2. Click the Component to select it for editing
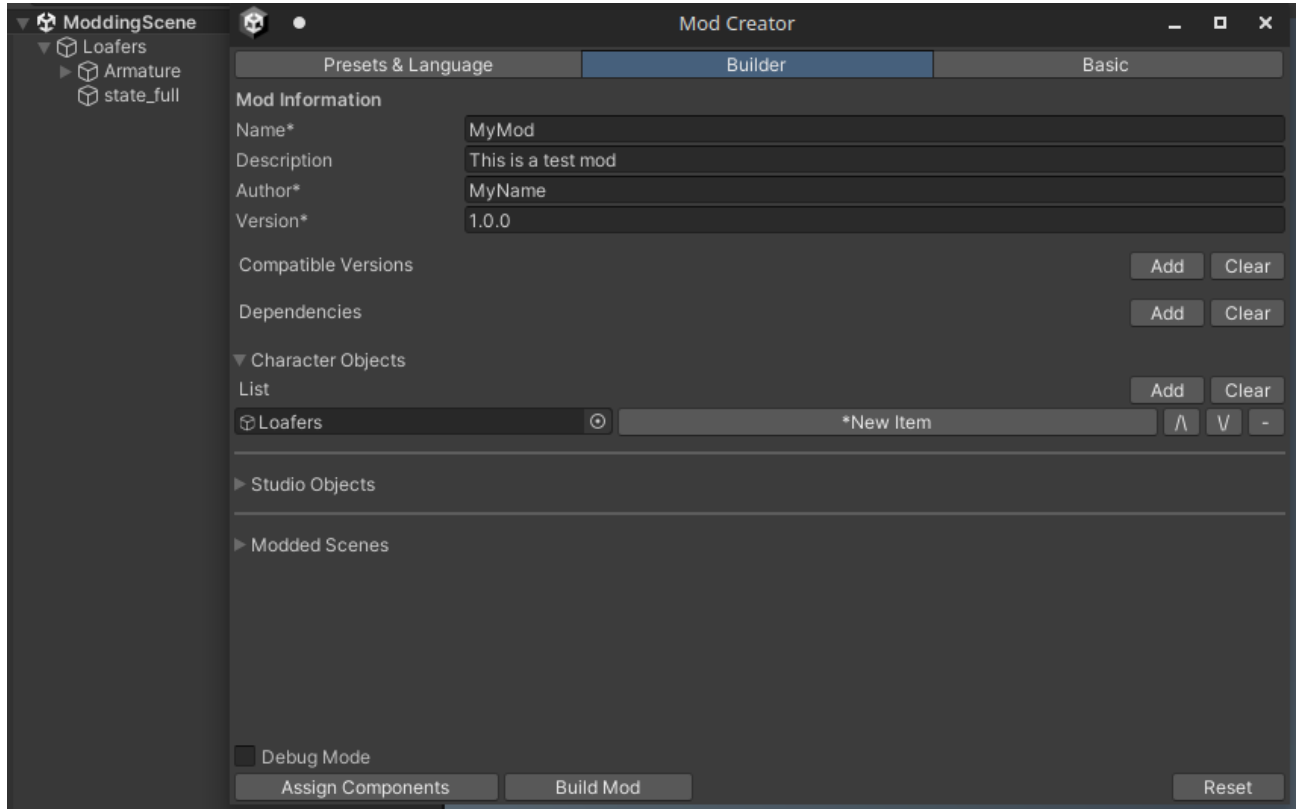
Once the character object Component is selected, the **Basic** tab for editing the Components settings becomes available.
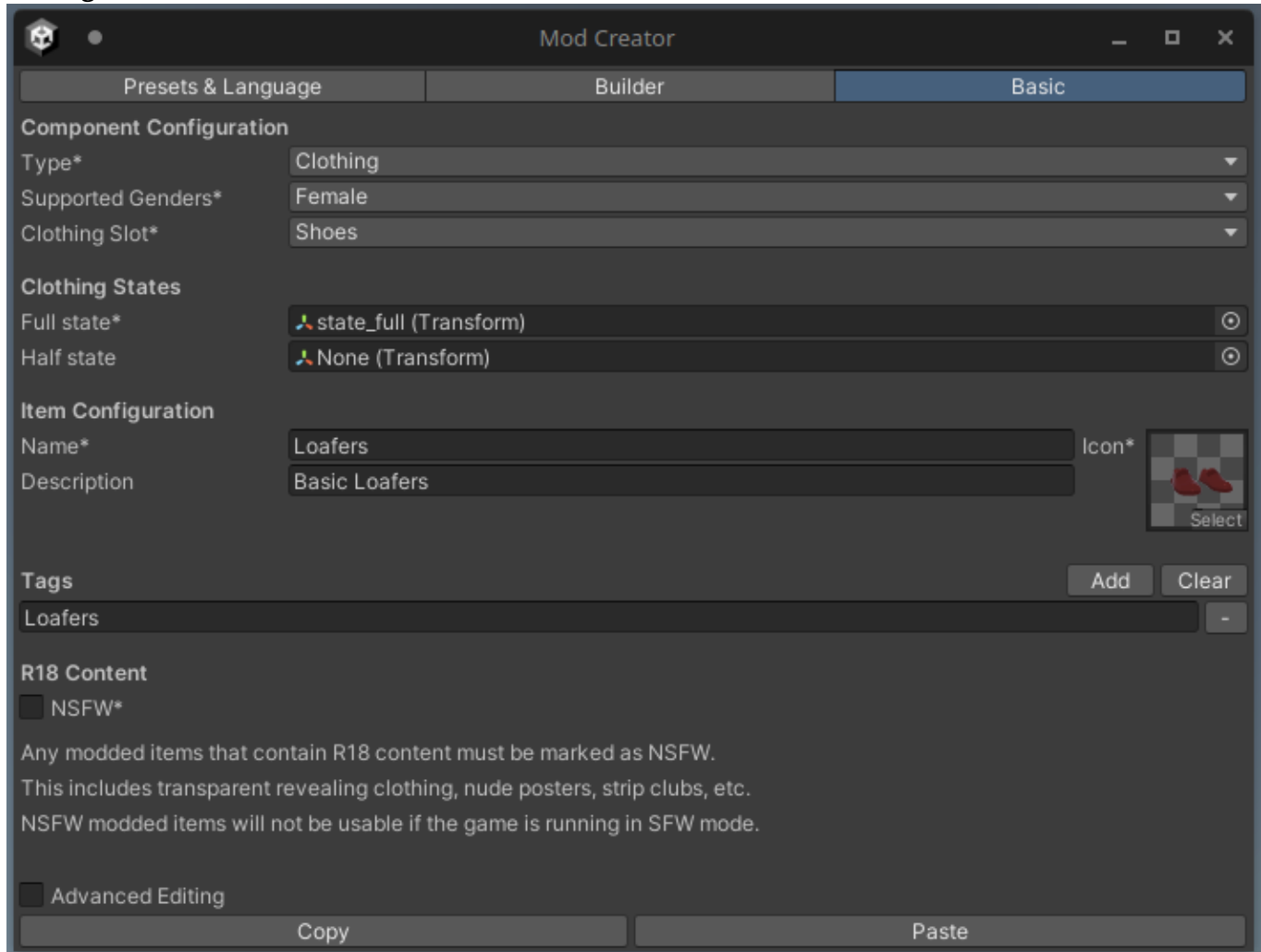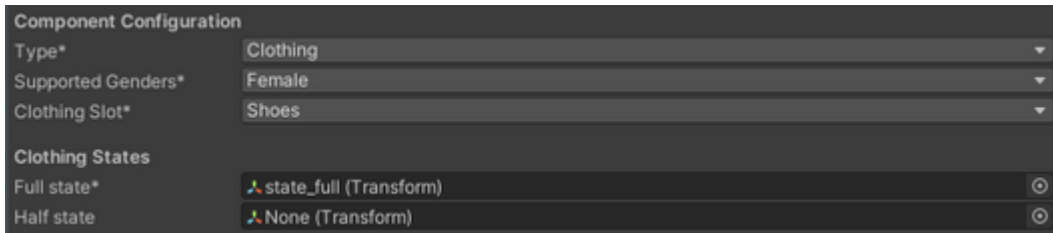

In the **Basic** tab you can set the objects name, description, icon and tags, types all of which will be used and visible in the game UI.

To define a character objects compatibility, you can set it to support either male or female or all genders.

Setting different component Types will show different component configuration options.

Clothing

Specifying the clothing slot puts it to the correct category in the character creators interface.

Clothing states identify what transforms should be shown or hidden when showing or partially removing clothing.
The full state is what is shown when the clothing is worn normally and is required.

Accessory

Specifying the clothing slot puts it to the correct category in the character creators interface.

Default parent indicates what transform of the characters armature the accessory should parent itself to by default. This can be overriden in game.

Hair

Specifying the hair slot puts it to the correct category in the character creators interface.

Texture

Specifying the texture type puts it to the correct category in the character creators interface.

The texture assigned to the Texture field will be used and applied to the character.

Going back to the **Basic** tab, by enabling the **Advanced Editing** checkbox, an additional **Advanced** tab becomes available.
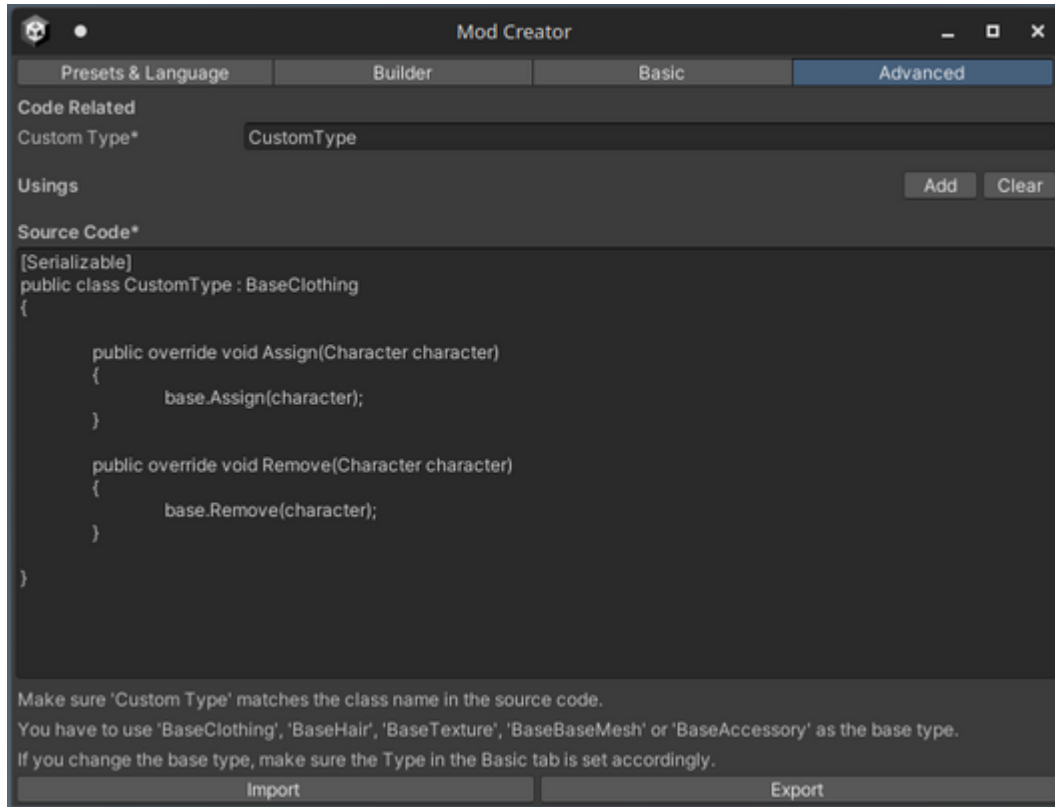
In this tab you can edit custom code for your character object component to add features or modify how it works.

You can either edit the code in the editor UI, or click the **Export** button and edit it in your IDE, and then **Import** it back to the editor.

Having **Advanced Editing** mode enabled creates a custom class for the character object, otherwise using the Base class with just changed parameters. This makes things a bit more complicated as you need to create a name for the class and set up any code accordingly.

Any instances of `using` need to be placed inside the **Usings** list.

The resulting class will be in the `Author.ModName` namespace next to other custom classes from the same mod.

For this example, **Advanced Editing** will remain unchecked.

Upon inputting your desired values, proceed back to the **Builder** tab and repeat the steps if you want to have more character objects in the mod.

Creation of a Studio Object Mod

Begin creation of a studio mod by clicking **Add** in the Studio Object list.

1. Assign a GameObject by dragging one from the scene or selecting it manually.
2. Click the Component to select it for editing
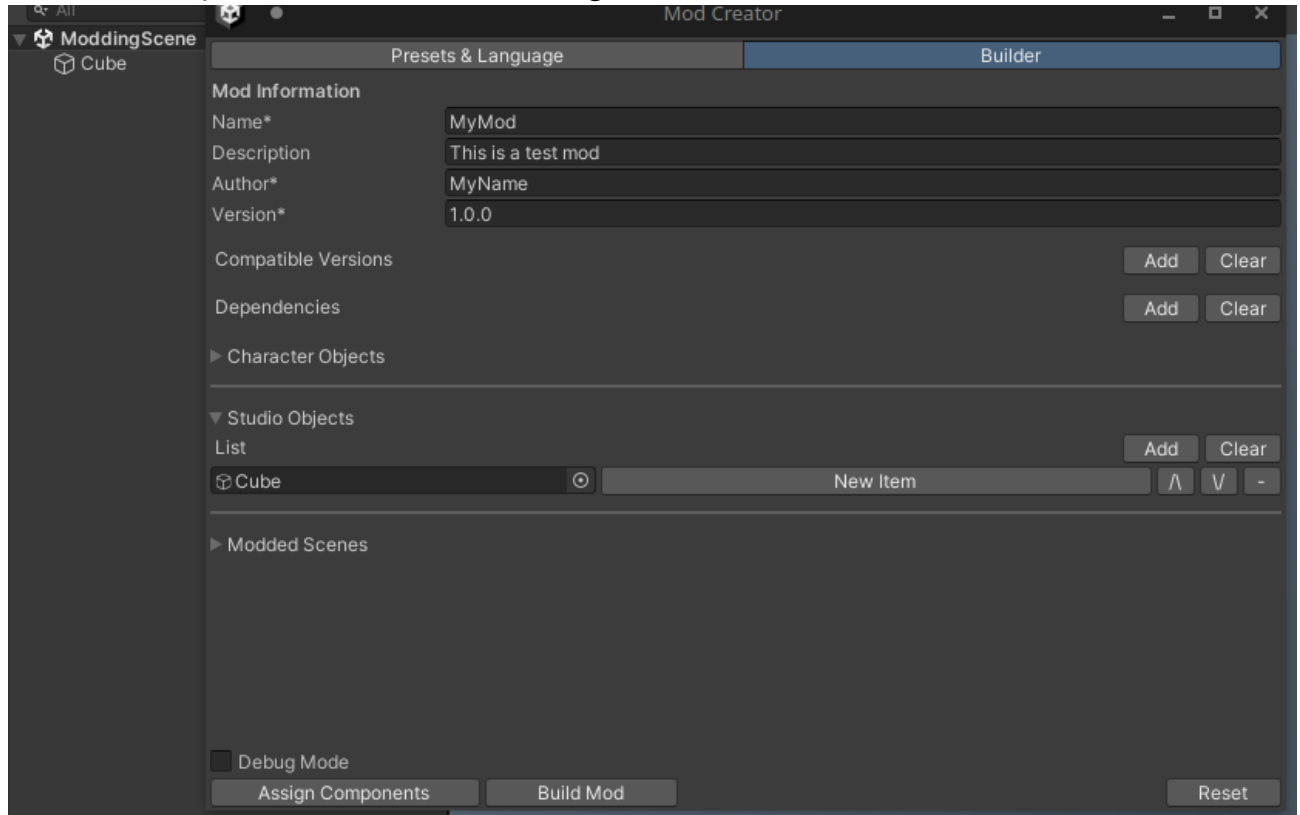
Once the studio object Component is selected, the **Basic** tab for editing the Components settings becomes available.
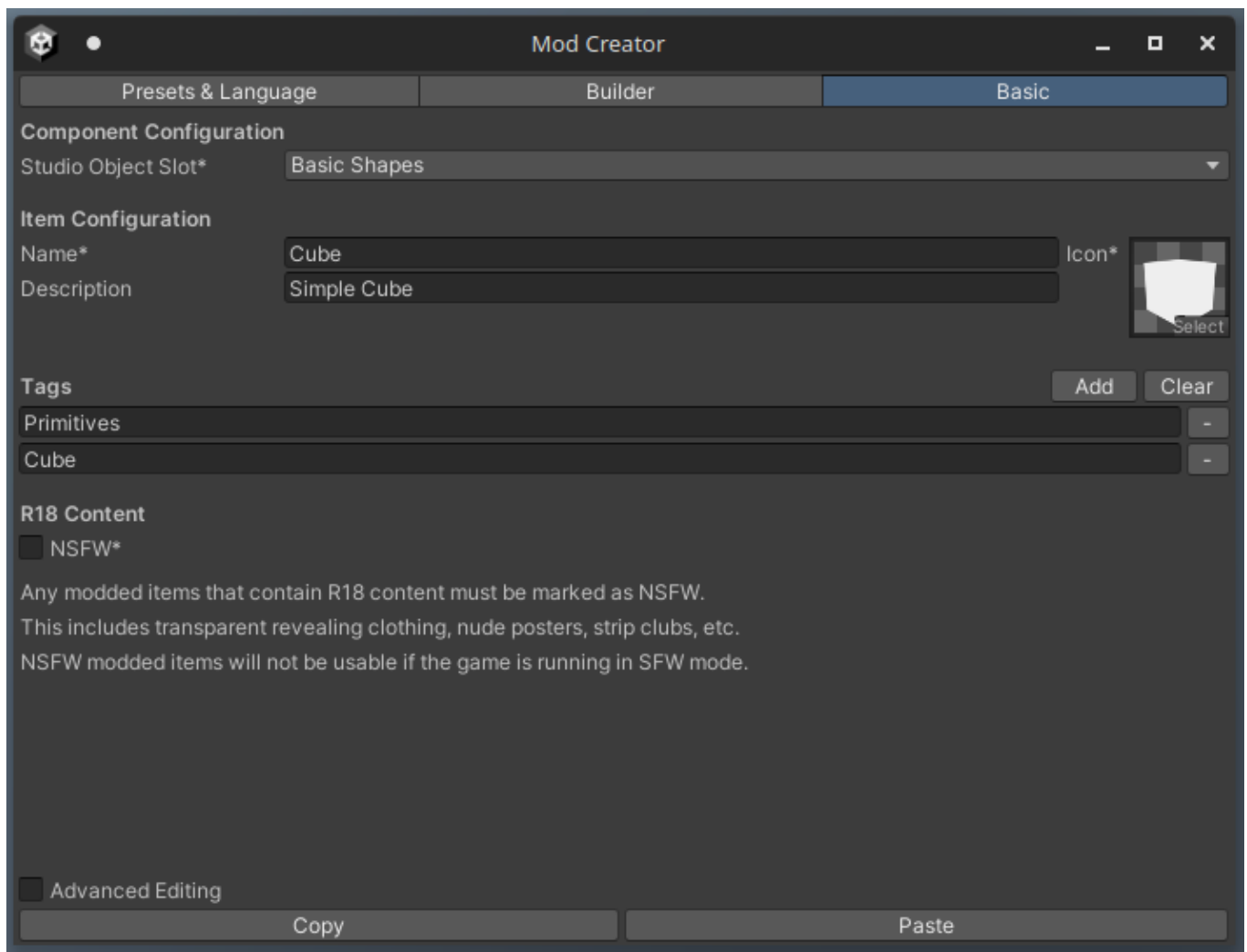
In the **Basic** tab you can set the objects name, description, icon and tags, types all of which will be used and visible in the game UI.

Specifying the studio object slot puts it to the correct category in the studios interface.

By enabling the **Advanced Editing** checkbox, an additional **Advanced** tab becomes available.
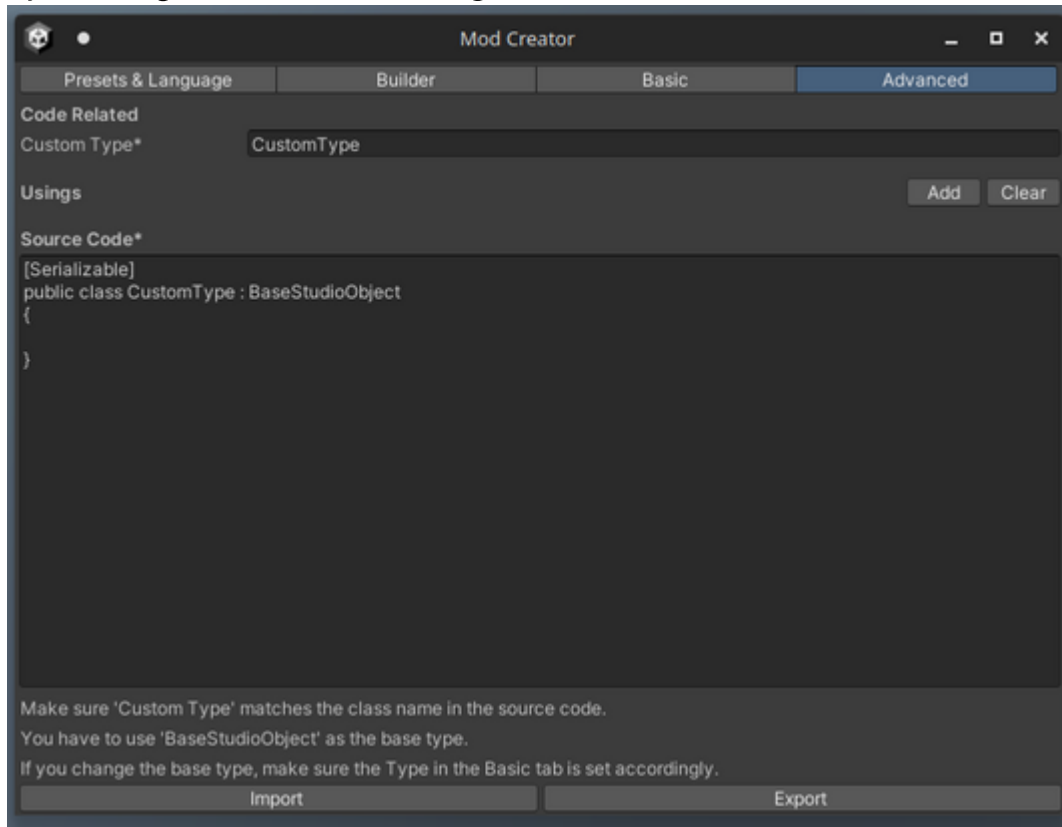
In this tab you can edit custom code for your studio object component to add features or modify how it works.

You can either edit the code in the editor UI, or click the **Export** button and edit it in your IDE, and then **Import** it back to the editor.

Having **Advanced Editing** mode enabled creates a custom class for the studio object, otherwise using the Base class with just changed parameters. This makes things a bit more complicated as you need to create a name for the class and set up any code accordingly.

Any instances of `using` need to be placed inside the **Usings** list.

The resulting class will be in the `Author.ModName` namespace next to other custom classes from the same mod.

For this example, **Advanced Editing** will remain checked.

Upon inputting your desired values, proceed back to the **Builder** tab and repeat the steps if you want to have more studio objects in the mod.

Creation of a Custom Scene Mod

Begin creation of a custom scene mod by clicking **Add** in the Modded Scenes Object list.

1. Assign a scene reference by dragging one from the project view or selecting it manually.
2. Click the Component to select it for editing
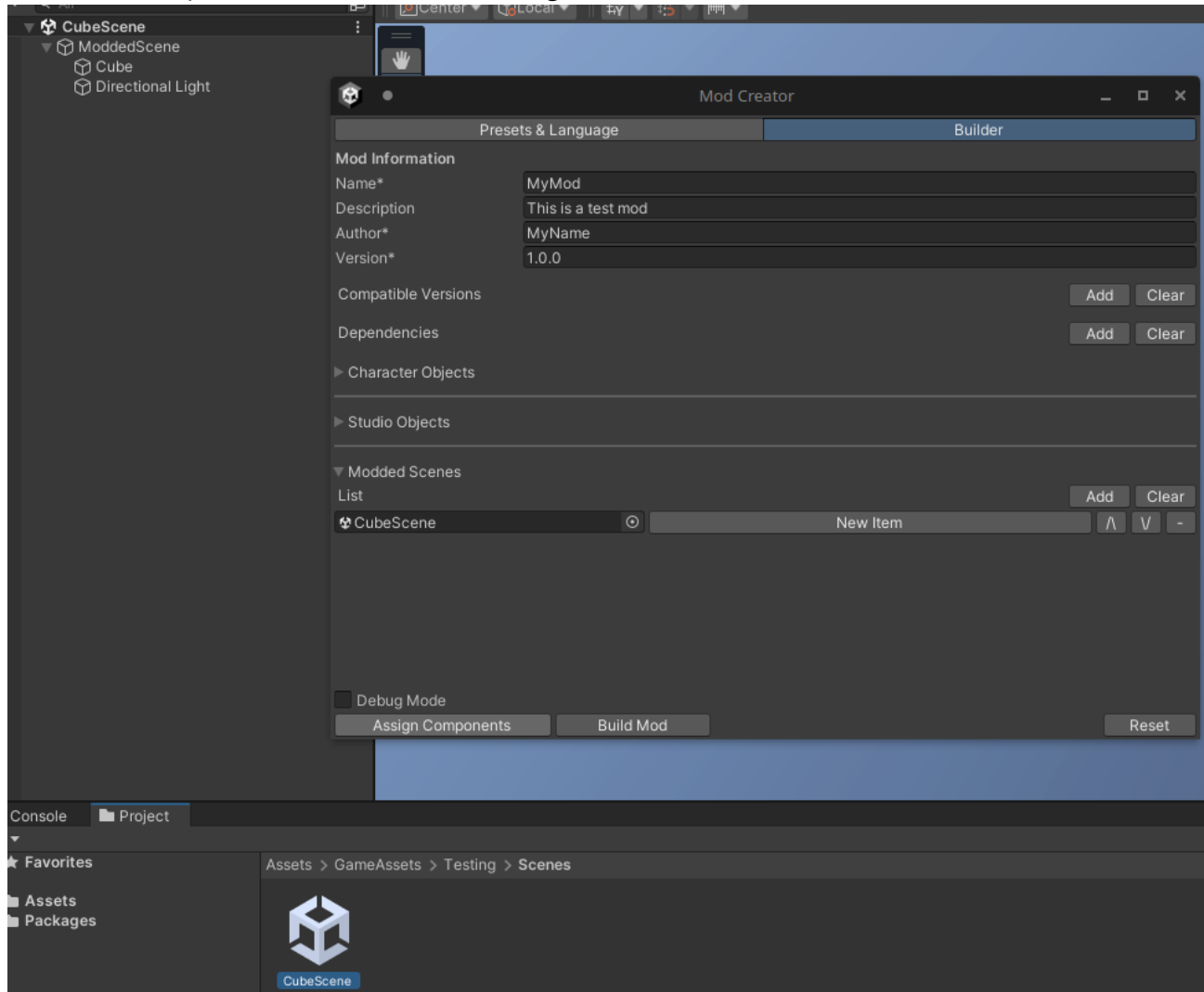
Once the modded scene object Component is selected, the **Basic** tab for editing the Components settings becomes available.

In the **Basic** tab you can set the scenes name, description, icon and tags, types all of which will be used and visible in the game UI.

The **Usage** field indicates where the modded scene will be used. For Maker, it can be used as a 3D background, for the Studio, it can be used as a map.

The root object of the scene must be called `ModdedScene` and should contain all objects that you want the scene to contain.
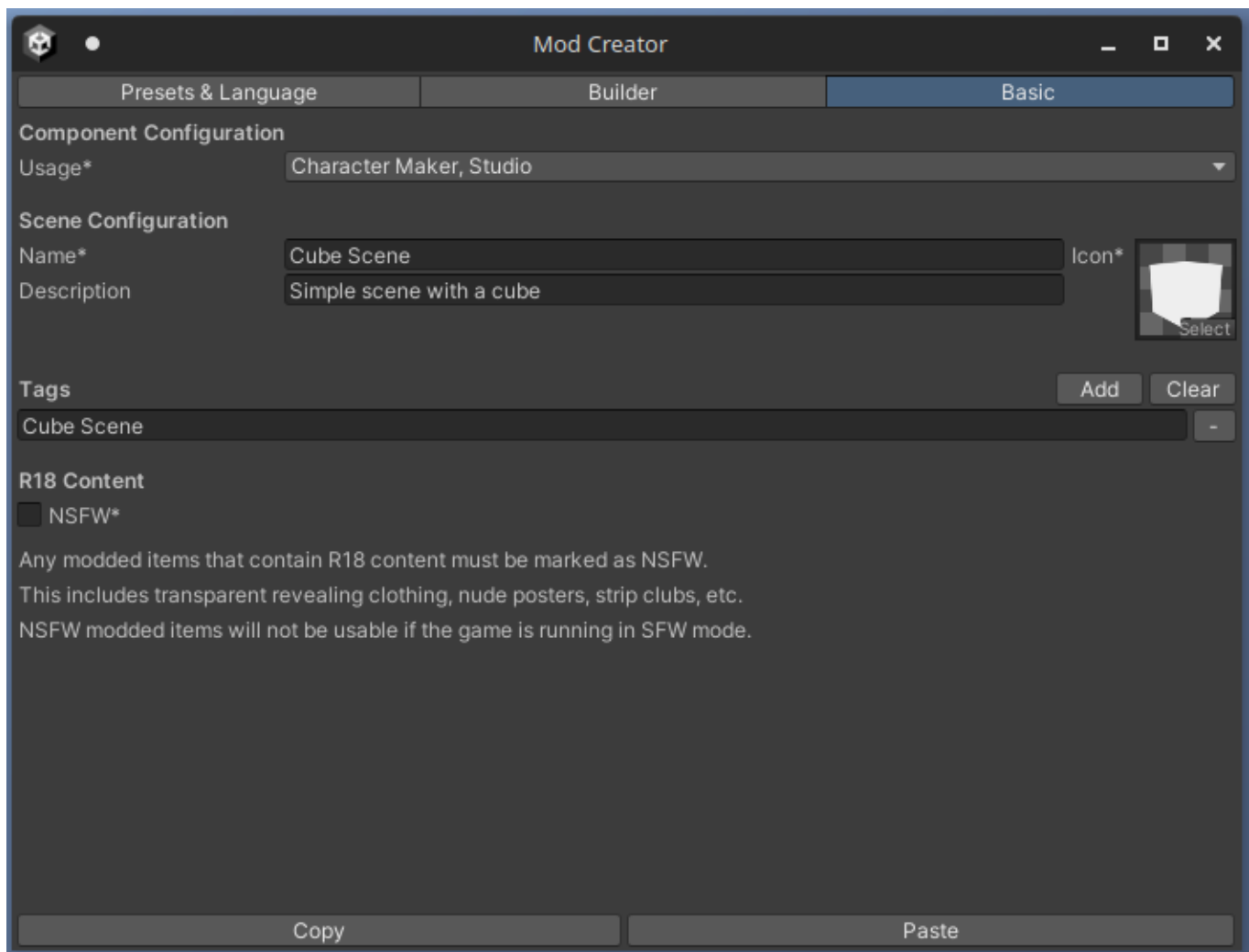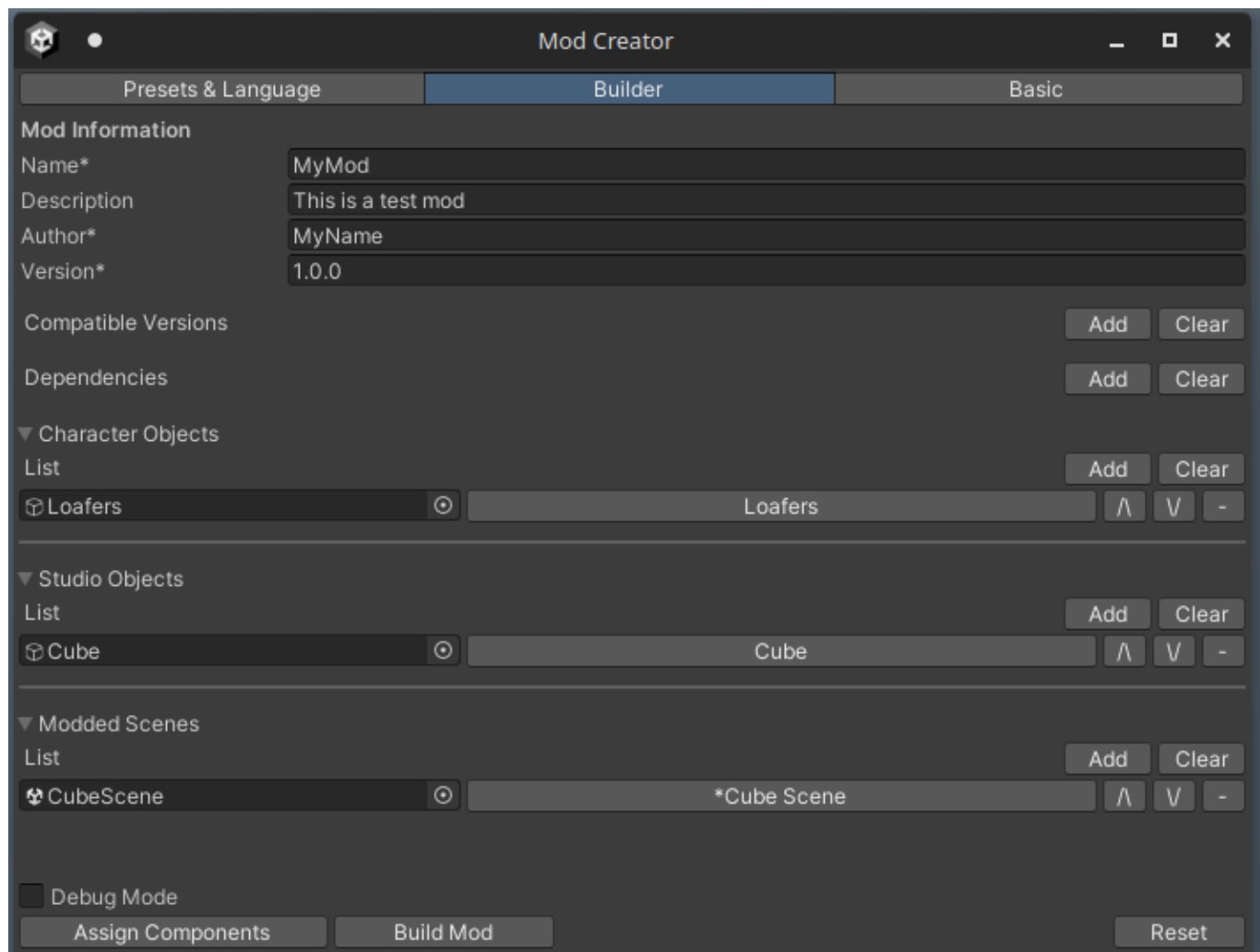
Upon inputting your desired values, proceed back to the **Builder** tab and repeat the steps if you want to have more custom scenes in the mod.
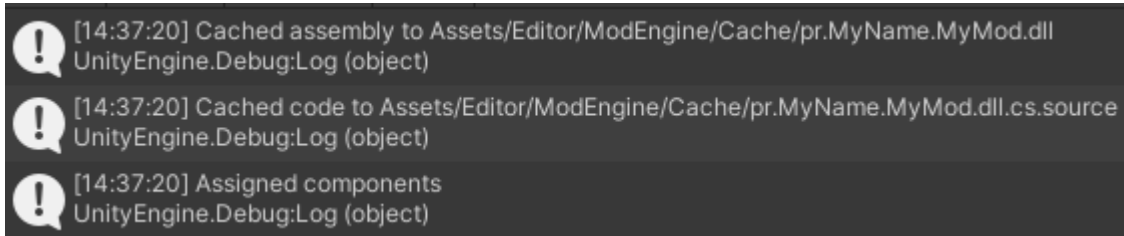
Building the Mod

Once you have set up the mods basic information and added at least one Studio object or Character object or Modded scene, you can now build your mod.

To assign your Components to the GameObjects, click the **Assign Components** button. If you used **Advanced mode**, a custom assembly will be built and included in the mod.

Verify that the components were successfully assigned by checking the console log.


```
[14:37:20] Cached assembly to Assets/Editor/ModEngine/Cache/pr.MyName.MyMod.dll
UnityEngine.Debug:Log (object)

[14:37:20] Cached code to Assets/Editor/ModEngine/Cache/pr.MyName.MyMod.dll.cs.source
UnityEngine.Debug:Log (object)

[14:37:20] Assigned components
UnityEngine.Debug:Log (object)
```

Since we used **Advanced mode** for one of the objects, a custom assembly was built and will be included in the mod. It is cached alongside its source code in the above path for later editing.

If there are no errors or other problems, you can now build the mod.

Now you are ready to click the **Build Mod** button and export the mod. Your default file browser will open, allowing you to pick a location to save your mod.

Testing the Mod

You can load a built mod by clicking the **Load Mod** button in the **Builder** tab with Debug Mode enabled. This will load a mod to the current scene so you can match the loaded values to your specified ones.

Make sure to test the mod in-game as well, by putting it in the games **mods** folder and launching the game to make sure everything works as intended.

Limitations

- Types
  - Some types may not be supported and cause the build to fail.
- Animations
  - Animations are not supported, but meshes can include blendshapes.
- Presets
  - Saving objects that are loaded in memory (loaded from a mod, created on runtime and not written to disk) is not possible and they will be ignored. This can result in missing meshes, materials, textures, etc.
- Modded scenes
  - Lighting information (lightmaps, light probes, baking etc.) is currently not supported.
- Assets
  - All meshes, textures and images must be marked as read/write, otherwise ModEngine will not be able to serialize them into a mod.