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:

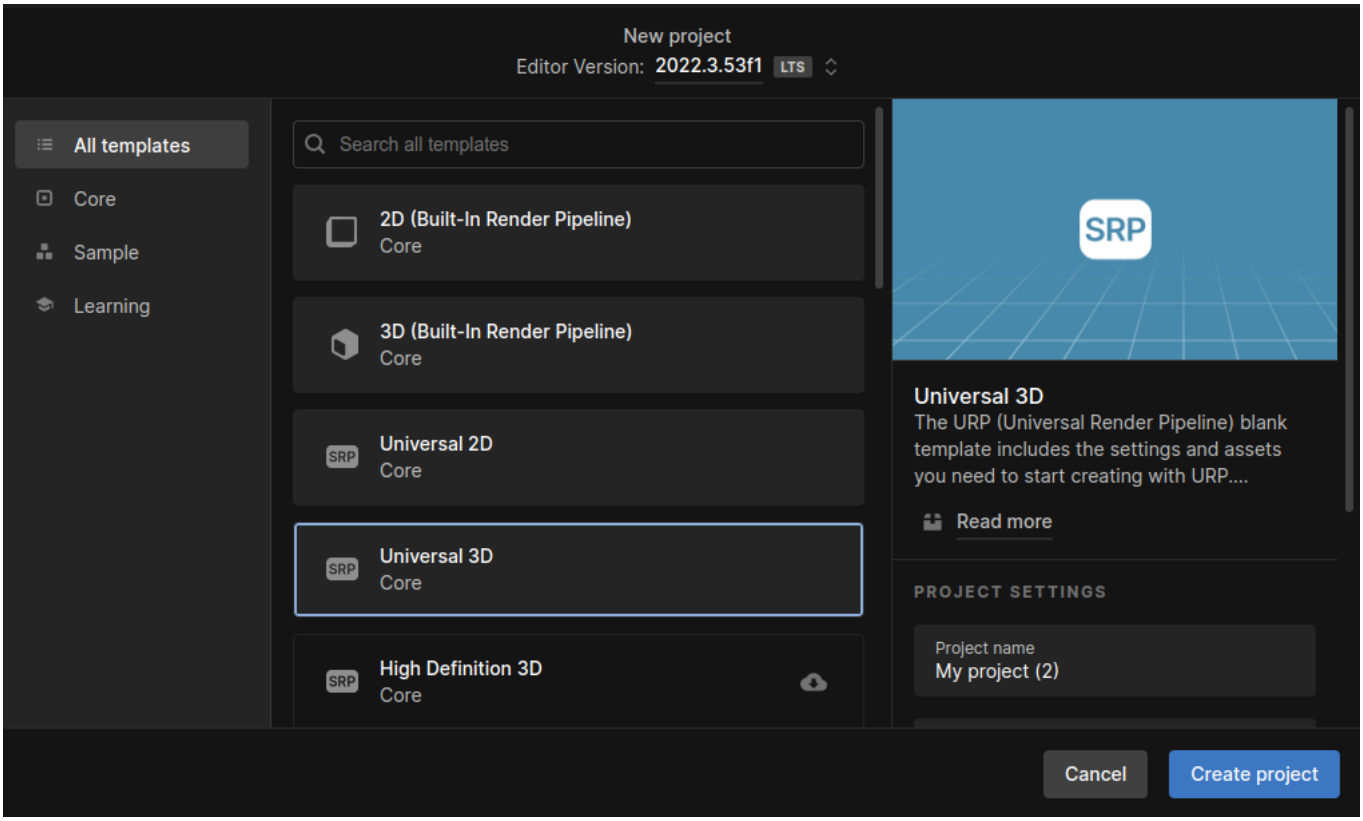
- **Studio objects** are items that can be spawned and manipulated in the studio.
- **Character objects** are items that are attached to characters, like hair, clothing, accessories and textures.
- **Scene mods** are scenes that are used as 3D backgrounds.
- **Animation mods** are used on both characters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

Features

- Studio object mods
- Character object mods
- Scene mods
- Animation mods
- Custom code for modded objects
- Multiple modded objects in one mod
- Custom shader support
- Forward Kinematics support
- Physics Simulation support

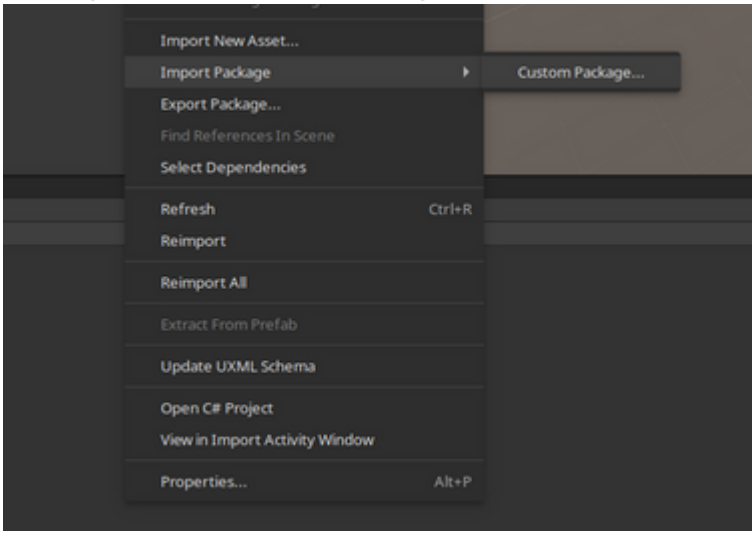
Setting up the Unity Project

Create an Unity 3D (URP) (**2022.3.53f1**) project and proceed with the next steps.
Our shaders use Unity URP, make sure that the project template is correct.

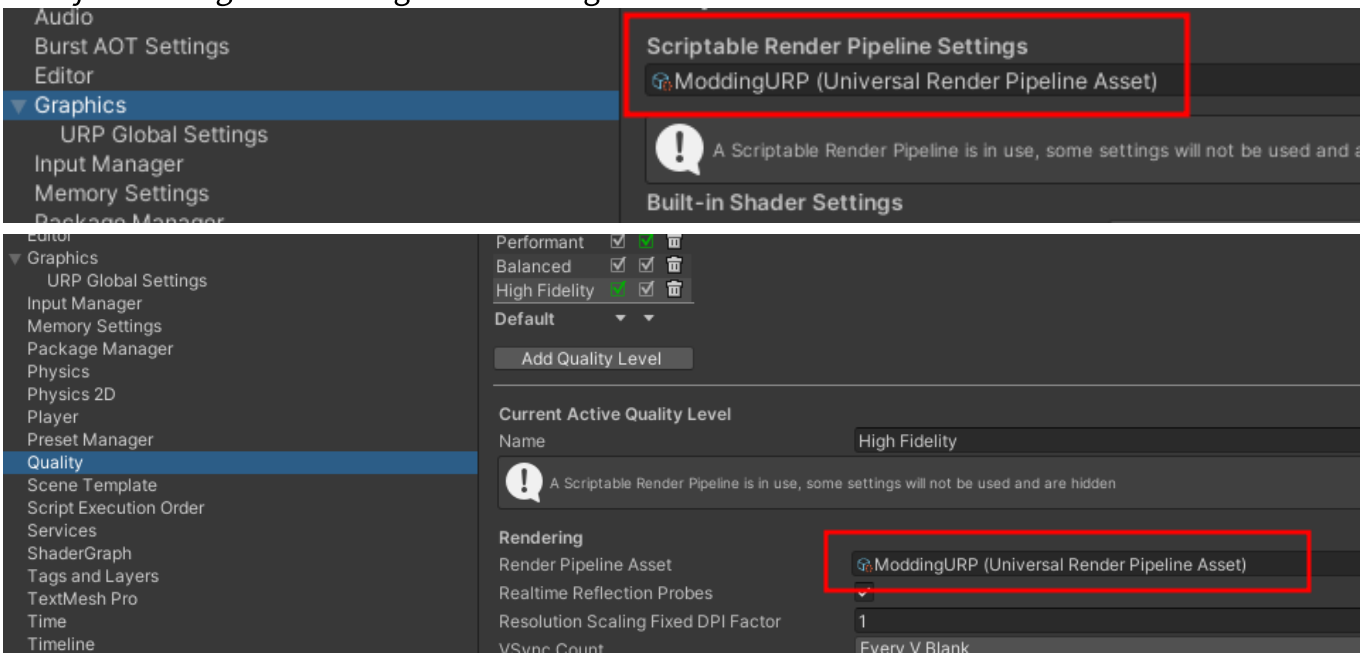


Installation

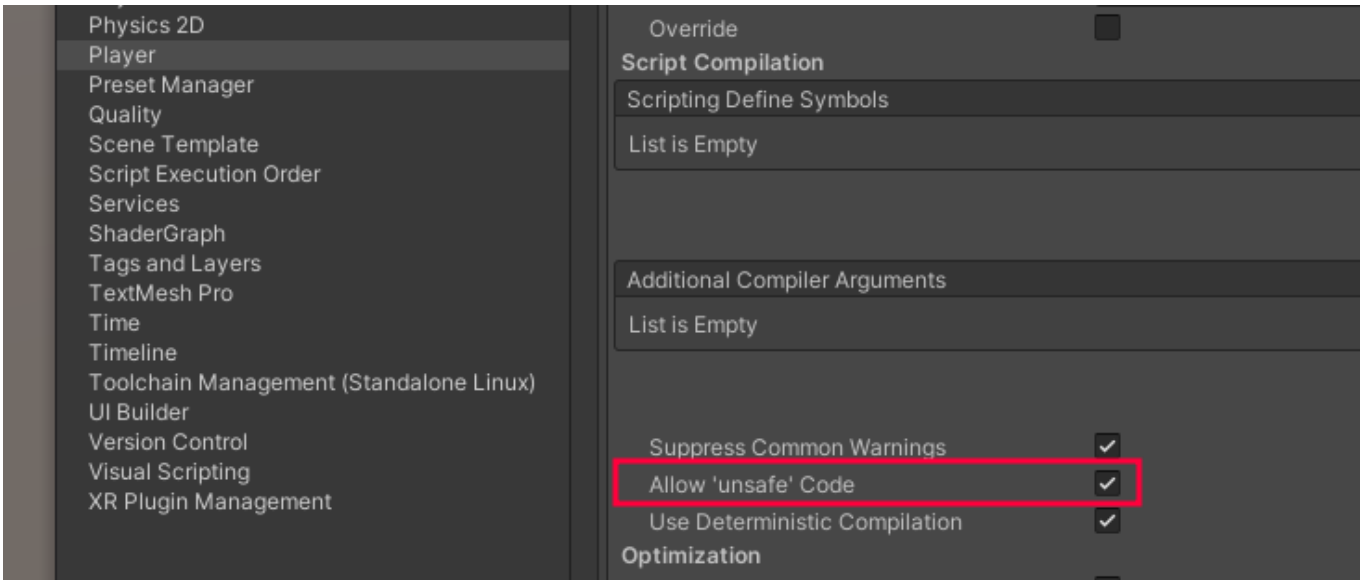
Import the included unitypackage by right clicking the Assets view -> *Import Package -> Custom Package*, point it to the package.



Make sure the correct URP Render Asset is active by opening the *Project Settings by clicking Edit -> Project Settings* and setting the following fields:

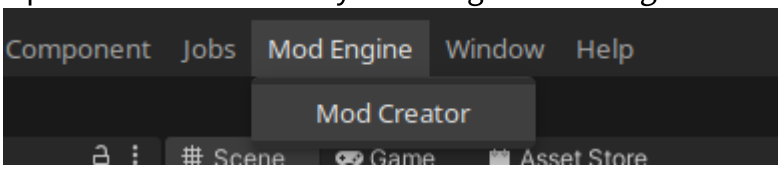


Make sure unsafe code is allowed in *Player -> Other Settings -> Allow 'unsafe' Code* by ticking the checkbox.

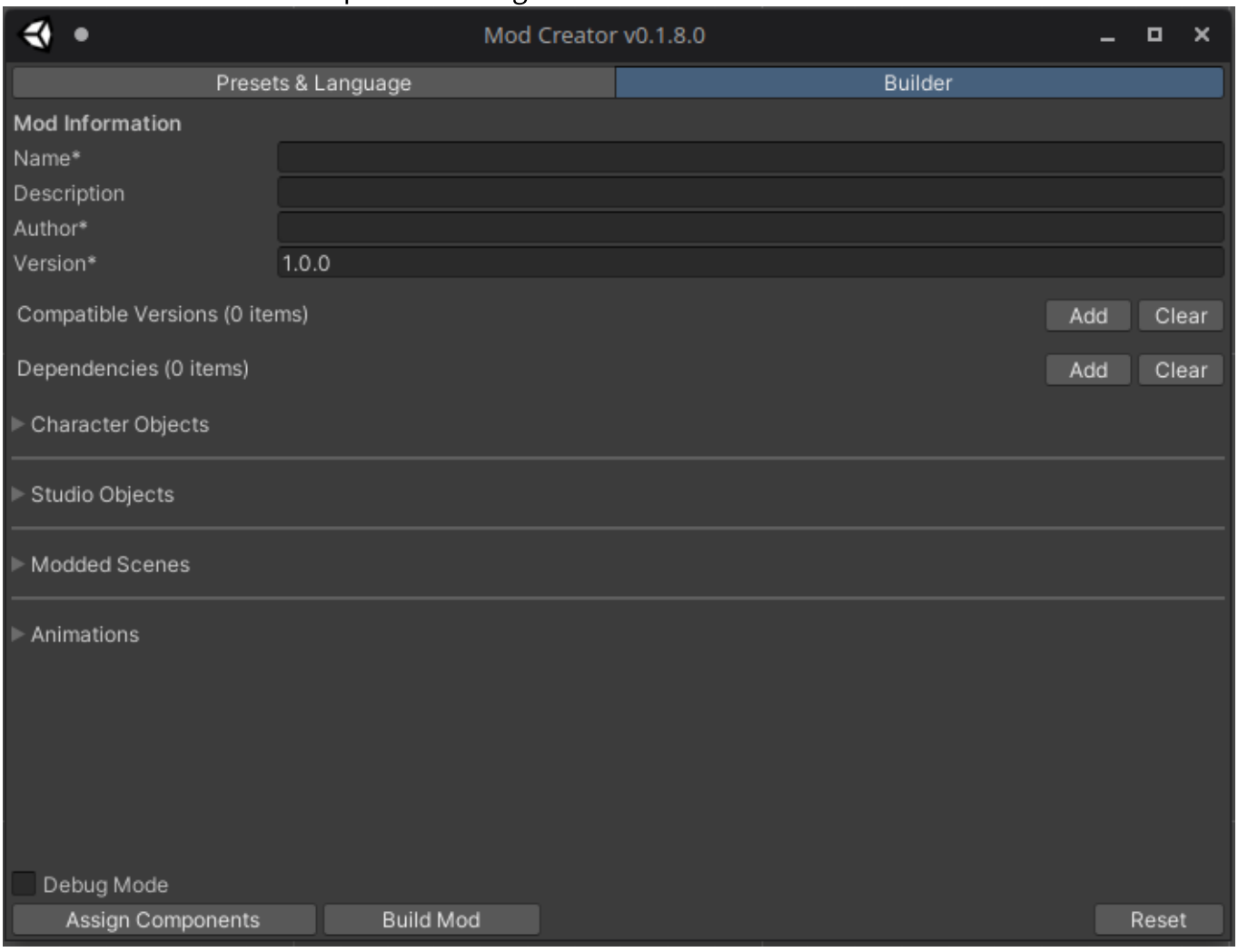


UI Overview

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



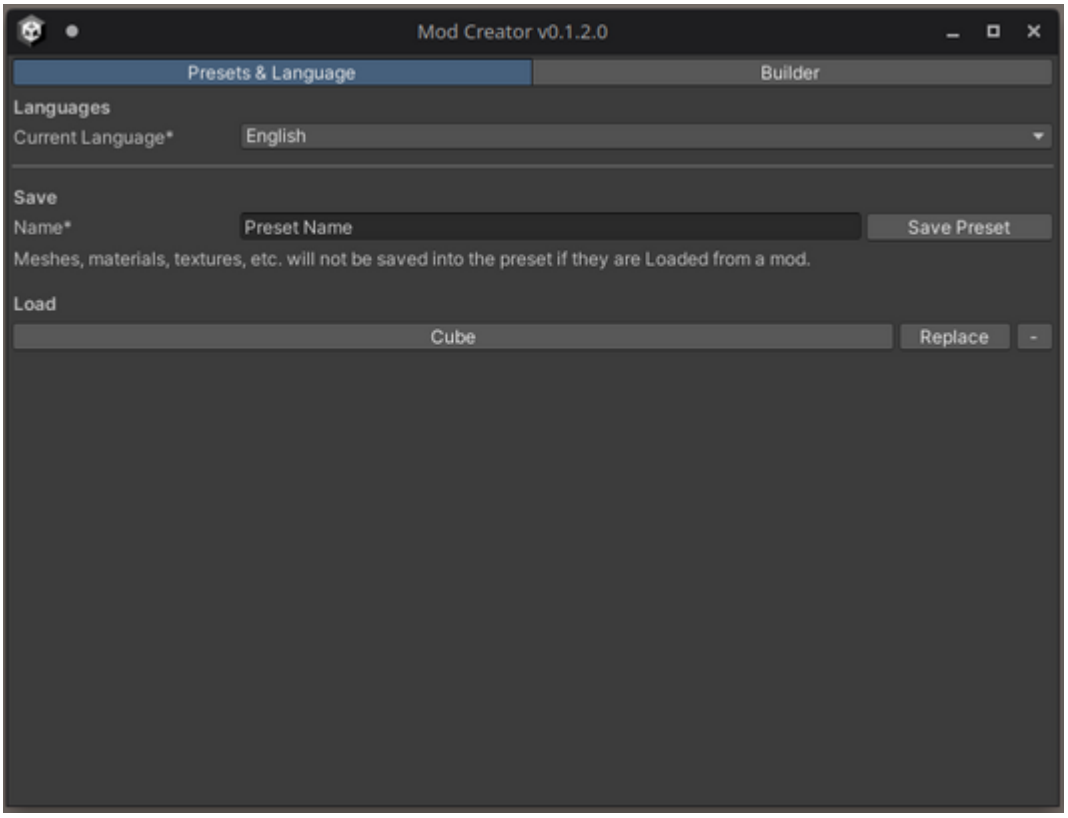
The Mod Creator window opens revealing the default view of the creator.



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.

In the case of Modded Scenes, the object reference is the Unity Scene itself. Loading a preset will assign the reference, and use the current objects in that scene instead of copying them to the workspace.

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

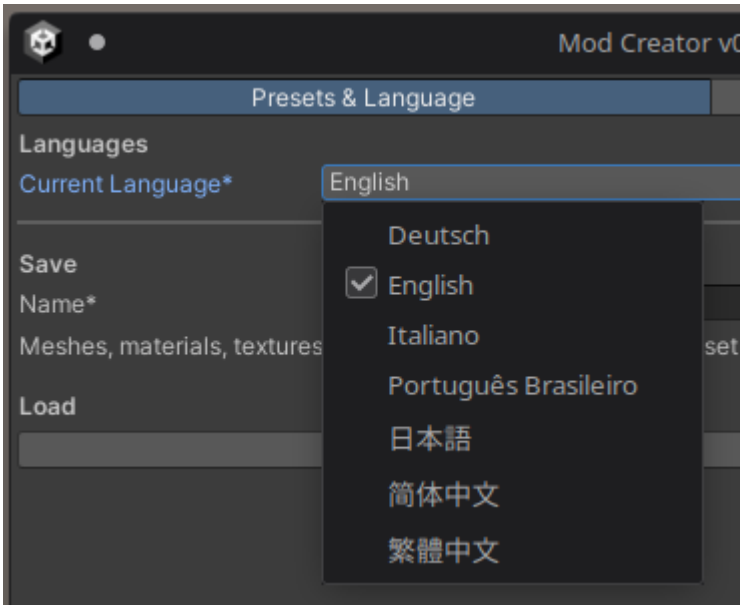
Removing the preset is done by clicking the **-** button next to the preset name, replacing a preset overwrites it, keeping the same name.

Closing Mod Creator will automatically save a preset ***previous_state***.

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.

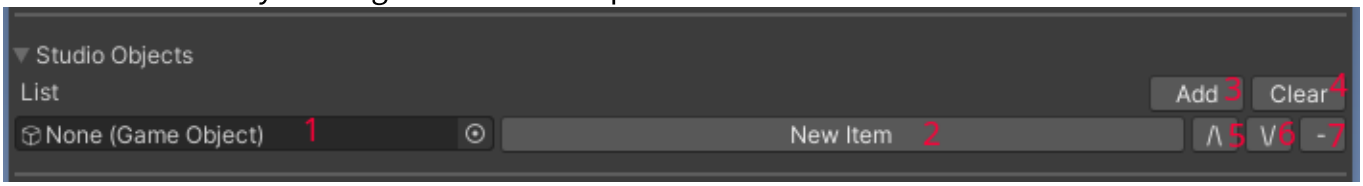


Mod Creation Setup

Enter the provided Modding Scene 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.

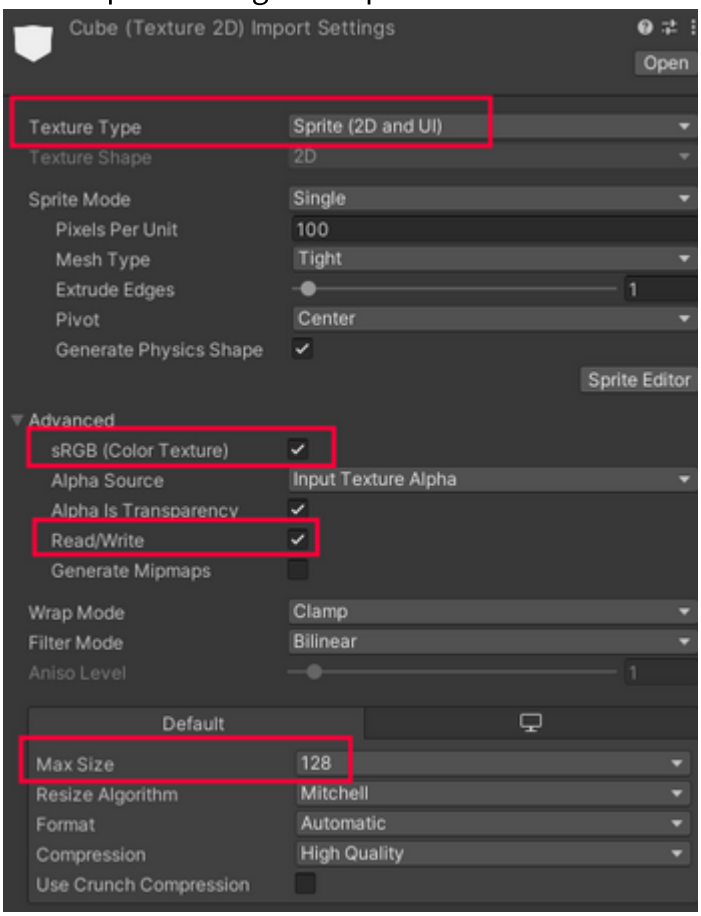


1. Reference to the mod object GameObject
2. Configuration and component information of the mod object
3. Add a mod object
4. Remove all mod objects under given category
5. Move the mod object up the list
6. Move the mod object down the list
7. Remove a mod object

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.
- Images using transparency have `Alpha Is Transparency` checked in their import window.
- Images have their `Compression` level set appropriately in their 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.
- Skinned meshes used in mods are exported without leaf bones.
- Skinned mesh armatures are named "Armature" and placed at the root of the object.
- If clothing fitting for the item does not work, make sure to `Apply Transforms` in your modelling application.
- Animations have the correct Rig setting. Humanoid animations should have Humanoid selected.

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.

Hair Highlights

To enable hair highlights, make sure the `_HairHighlightToggle` property is enabled on the hair material. This should only be enabled for the hair material itself, not any ribbon or decoration that is attached to it.

Specifying a texture to the `_PerlinNoiseTex` property will adjust how the hair highlights are drawn. Leaving it empty will automatically pick a default in-game.

Make sure that all the hair materials on the same object use the same highlight settings as they will be matched to any material on the object that has the highlights enabled.

To simplify the set-up process, use the included BaseHair parent material for your hair material.

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.

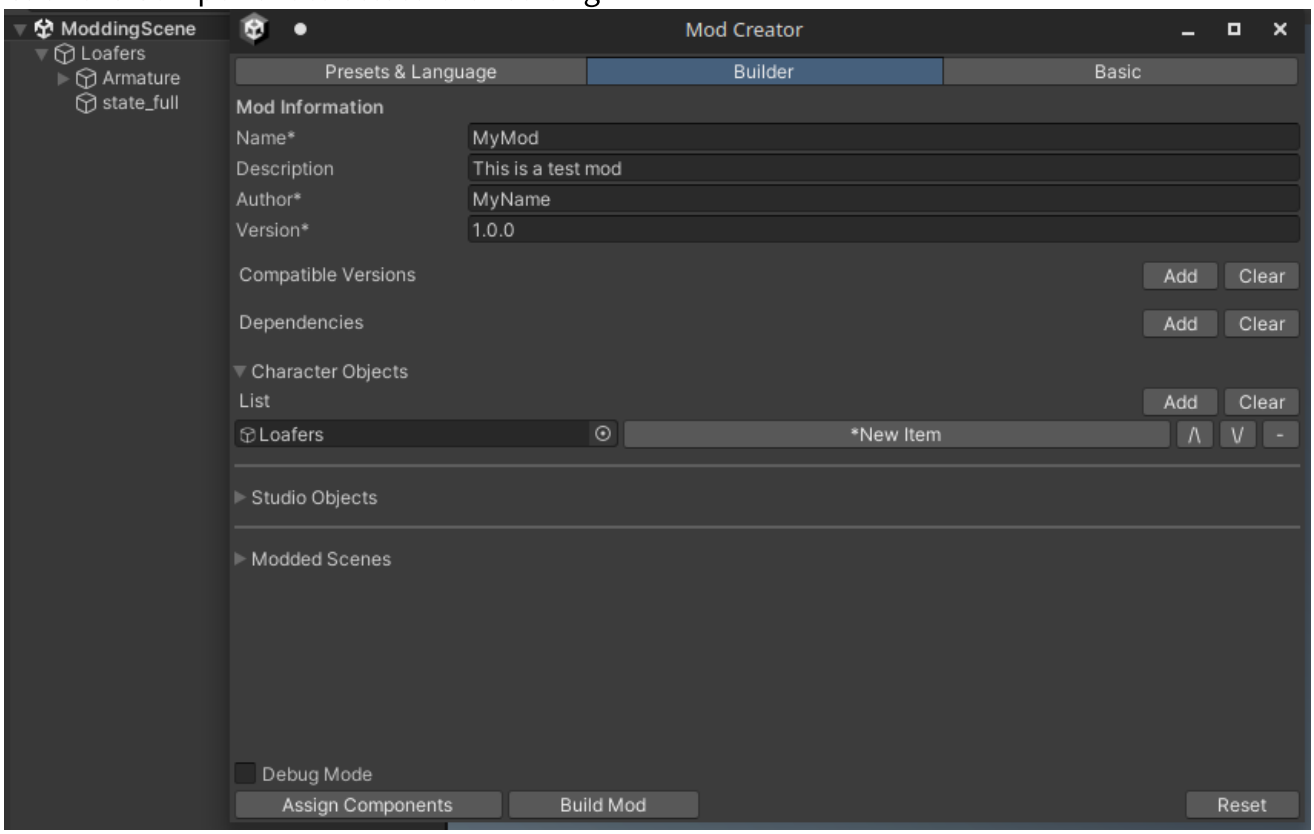
Rotation serialized into hair is overwritten to the head bone rotation, which means any corrections should be done in the 3D software before exporting the hair mesh.

Hair and accessories are parented directly to the armature bones of the character.
If you are making an accessory that is not skinned or a hair item - make sure their fbx transform origins are at 000, otherwise they might be floating once exported in game.

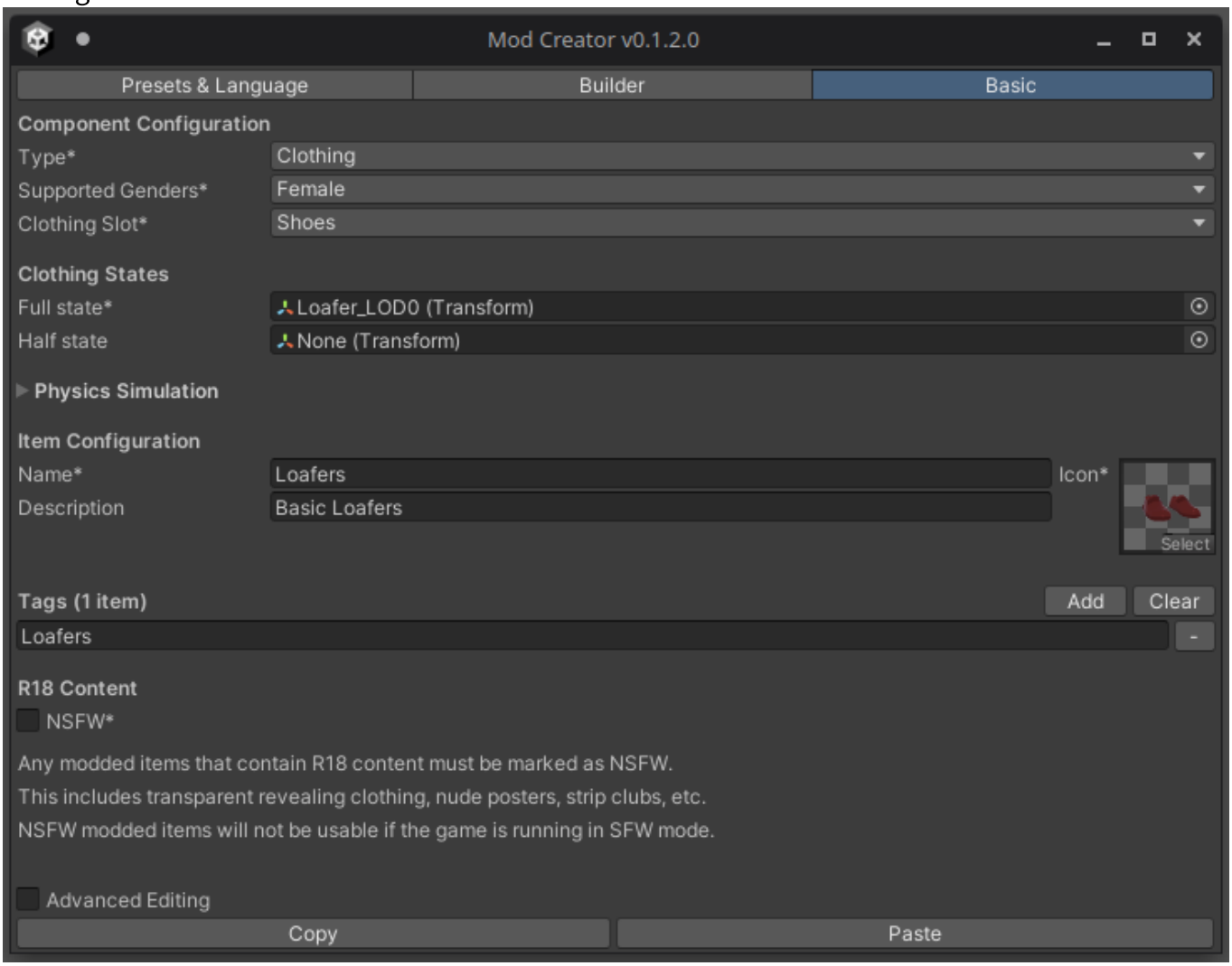
Creation of a Character Object Mod

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. For Texture mods, create an empty GameObject and assign it instead.
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.

Physics Simulation

Character objects that would benefit from **Physics Simulation** can utilize a `Simulation` array on clothing, accessories and hair. Please note that this feature is still experimental.

We currently rely on **Magica Cloth 2** for handling the simulation inside the game. We strive to abstract from its complex setup by offering you a simplified interface.

Multiple simulations can be run on the same character object. This means that skirt physics can run independently from, say, an oversized belt that is part of the same mesh.

Please refer to the included `Physics Simulation` document for a description of properties, usage, examples and setup of physics.

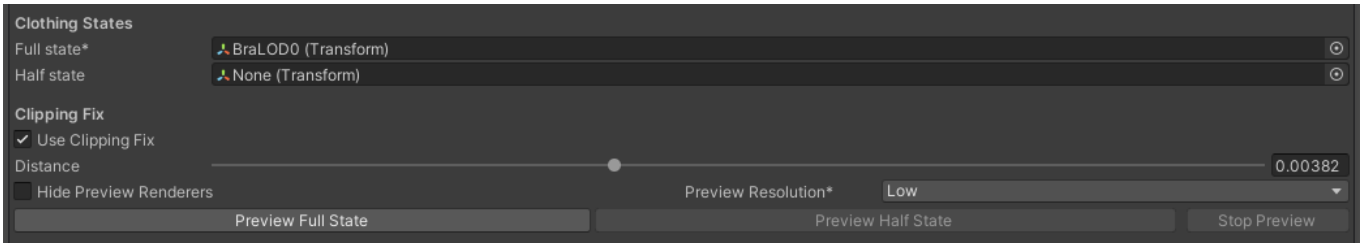
Clothing

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. These transforms should contain the mesh renderers.
The full state is what is shown when the clothing is worn normally and is required.

If you have not created a half state, there is no need to assign anything as it is optional.
Multiple states can not use the same object, so make sure the same object is not assigned to both the full and half states.

Clipping Fix



"Clipping fix" is our in-house developed technique for attenuating small clipping issues and simplifying the process of skinning.

⚠ Some preliminary warnings and limitations: ⚠

- Clipping fix is meant to solve small clipping issues that show up during animations in places where clothing is skintight. It cannot fix clipping if your clothing piece has clipping issues in bind pose
- Clipping fix cannot be used to mask clothing whose bind pose is not a T-Pose
- Clipping fix is not meant to work with clothing featuring semi-transparency

Usage

- Check the "Use Clipping Fix" box on Mod Creator's object page.
- Press "Preview Full State" or "Preview Half State" depending on which clothing state you want to generate Clipping Fix data for.
  - A short pre-generation step will be automatically performed, just wait it out.
  - You should then see a body mesh with your selected clothing piece in the aforementioned state.
- Use the distance slider to change which distance is considered to be clipped.
  - The units are in Meters: 0.01 = 1cm
  - Use the highest value possible before from the outside you can see holes. Look into the borders between your clothing and the body.
  - The preview will update in real time as you slide the slider knob.
    - Use the highest Preview Resolution possible without your editor slowing down too much. The lower resolutions will display in game when texture quality is lowered as they are faster to compute. In general any modern graphics card should not have issues at the highest preview quality in the mod creator.
    - You want to prioritize having a good clipping on highest texture quality rather than lowest. Lowest quality will mask less anyways (which is safe as we prefer small clipping rather than visible holes)
  - If you want to easily hide the clothing piece to check what is being clipped, simply check the "Hide Preview Renderers" checkbox to do so.
- When done proceed to set the rest of your mod as you want, the settings will be retained.

Best practices when modeling

To ensure your clothing pieces can take the maximum advantage possible from clipping fix:

- Make borders between your clothing and the base mesh closer when possible.
- In general ensure just a few mms of distance from the skin if something is meant to be skintight. Setting Clipping fix at that distance plus some extra small value will ensure that all of the covered body will be hidden, making the skinning process easier.
- In general model tightly, and where possible follow the polygon density of the underlying base mesh as when transferring weights this will make the skinning more accurate and less problematic, alongside making it easier to keep a constant distance from the body itself.

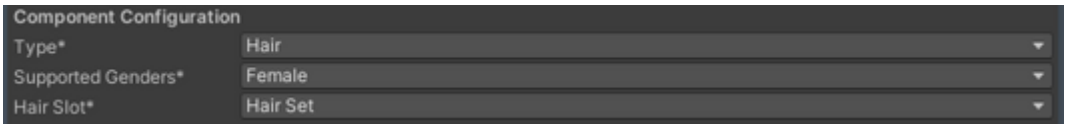
Accessory

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

If the accessory being added does not use simulations that fully remap bones, enabling **Reparentable** parents the accessory to the specified characters armature transform by default and allows users to change the parent in game.

Choosing to not enable **Reparentable** will parent the accessory directly to the character, not part of any bone. This should mainly be used with full bone remapped simulations.

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.

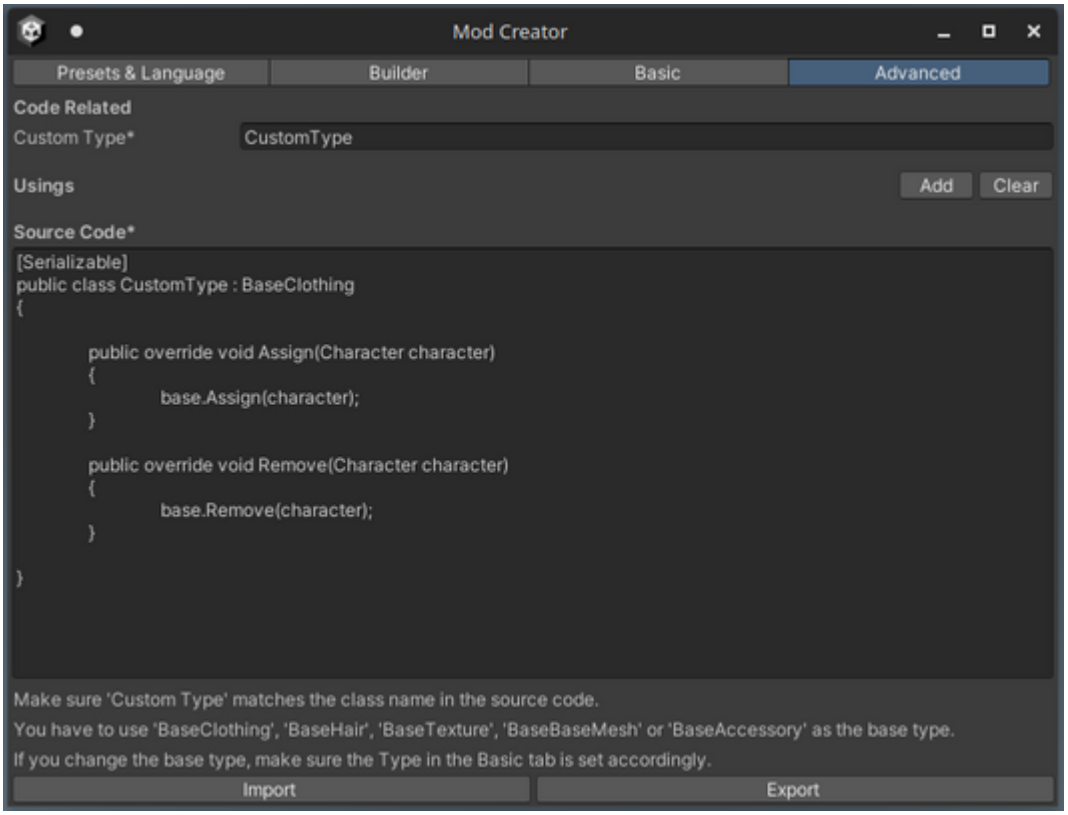
The texture assigned to the Texture field will be used and applied to the character.

Eye (Inner iris, Outer iris, Pupil) textures should be 512x512 to keep quality consistency.

Overlay texture types (including Nipples and Lips) are overlaid on top of the parent texture (face or body), so make sure unused areas are fully transparent.

Additionally, overlay texture types allow you to specify a default color which will be multiplied to them in game. A pure white color will result in the exact provided image.

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.

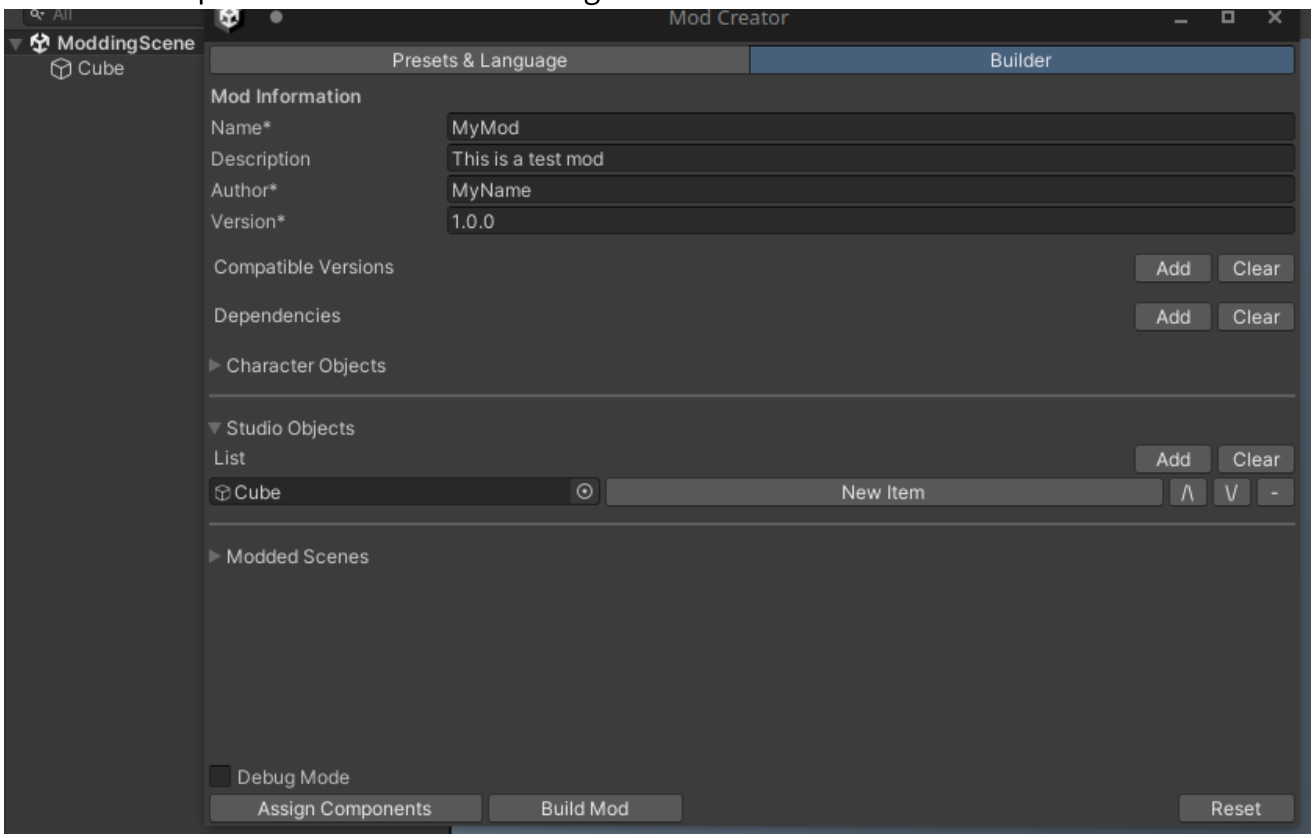
For this example, **Advanced Editing** will remain unchecked.

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.

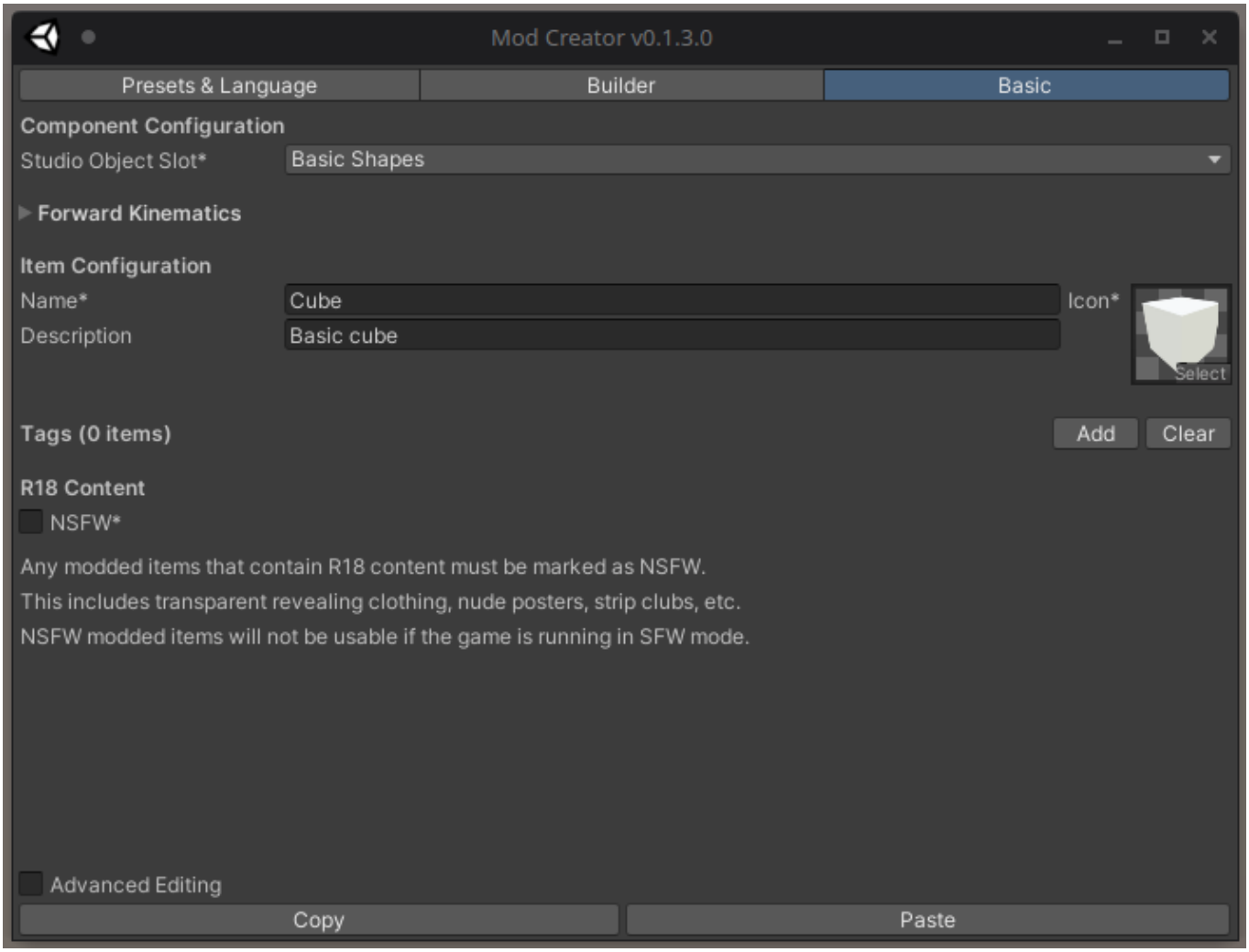
Creation of a Studio Object Mod

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.

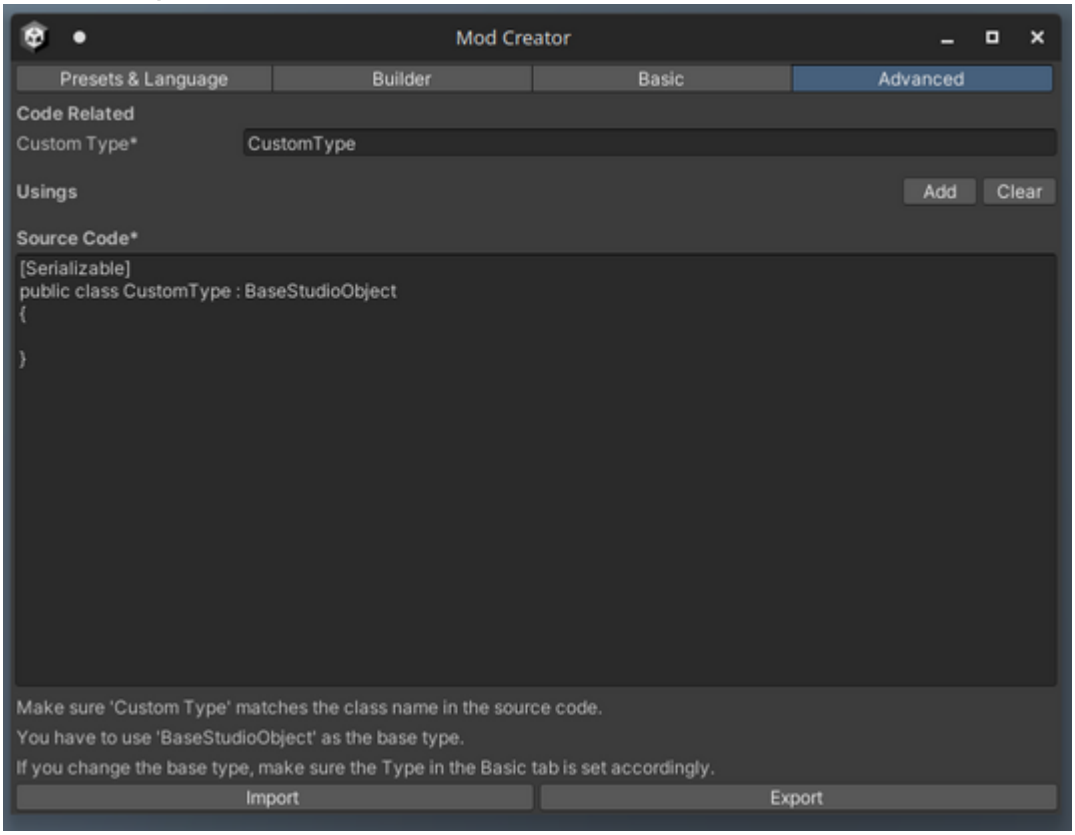
Specifying the studio object slot puts it to the correct category in the studios interface.

Forward Kinematics

Studio objects that include bones can utilize **Forward Kinematics**.

Please refer to the included `Forward Kinematics` document for a description of properties, usage, examples and setup of FK.

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.

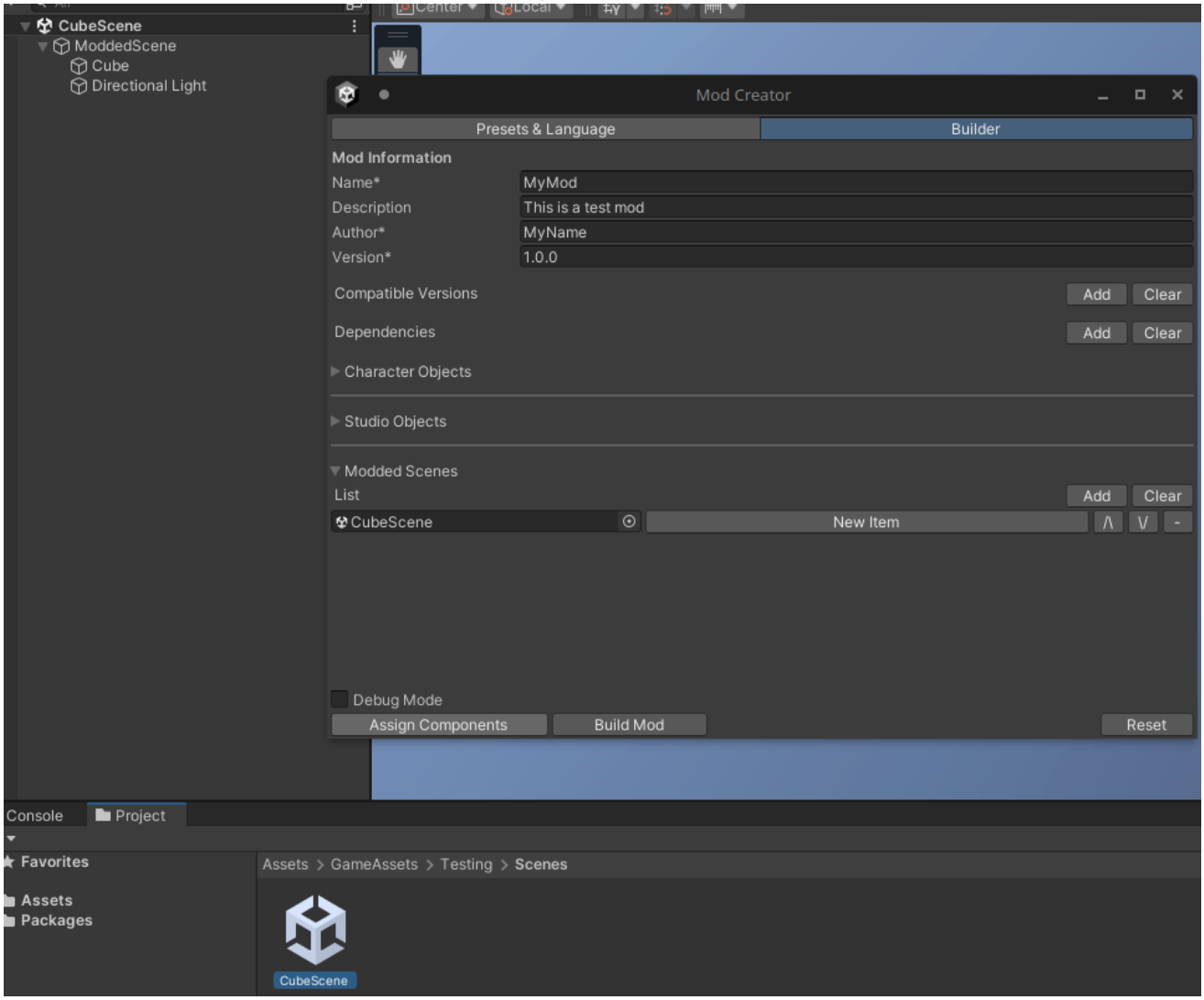
For this example, **Advanced Editing** will remain checked.

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.

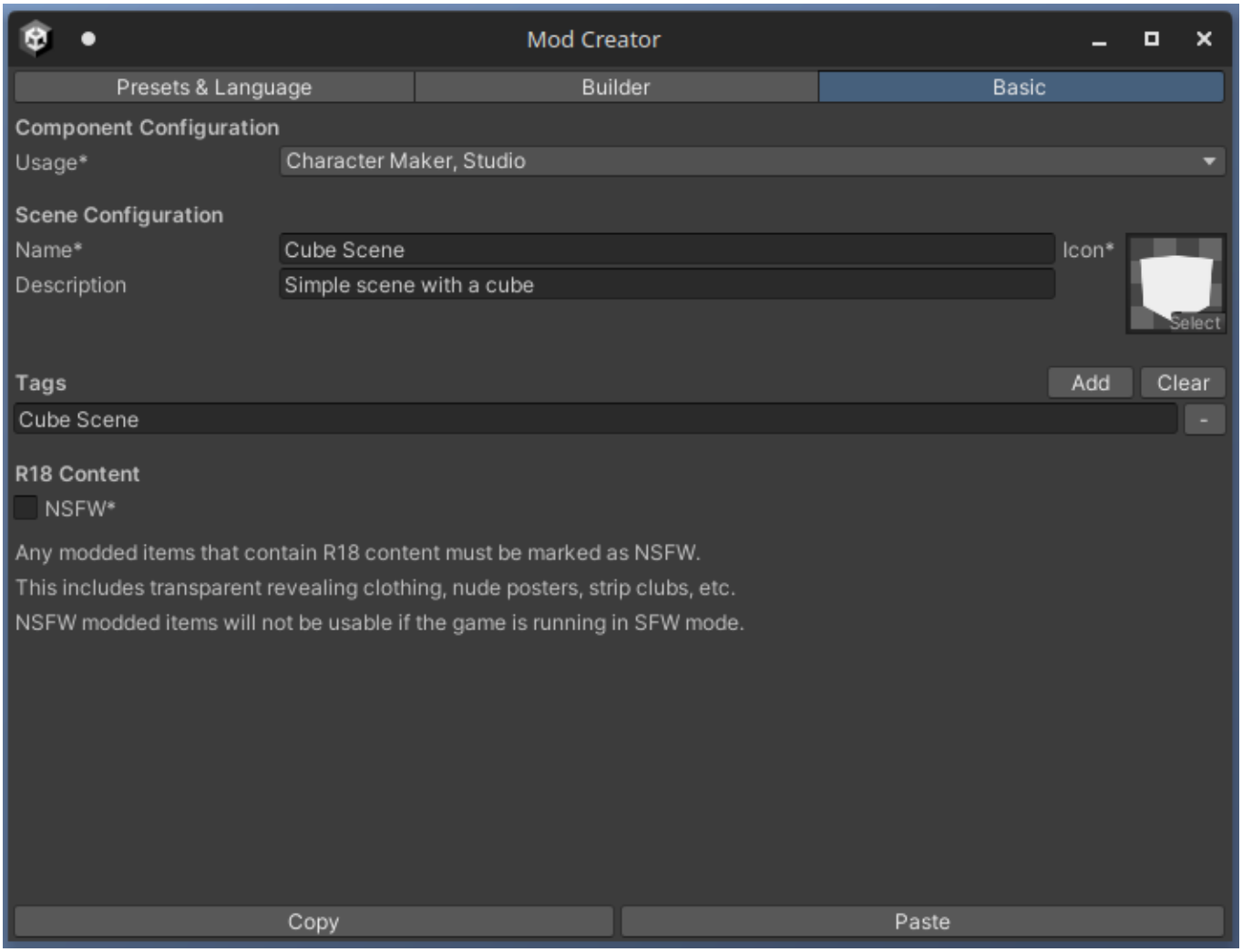
Creation of a Custom Scene Mod

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.

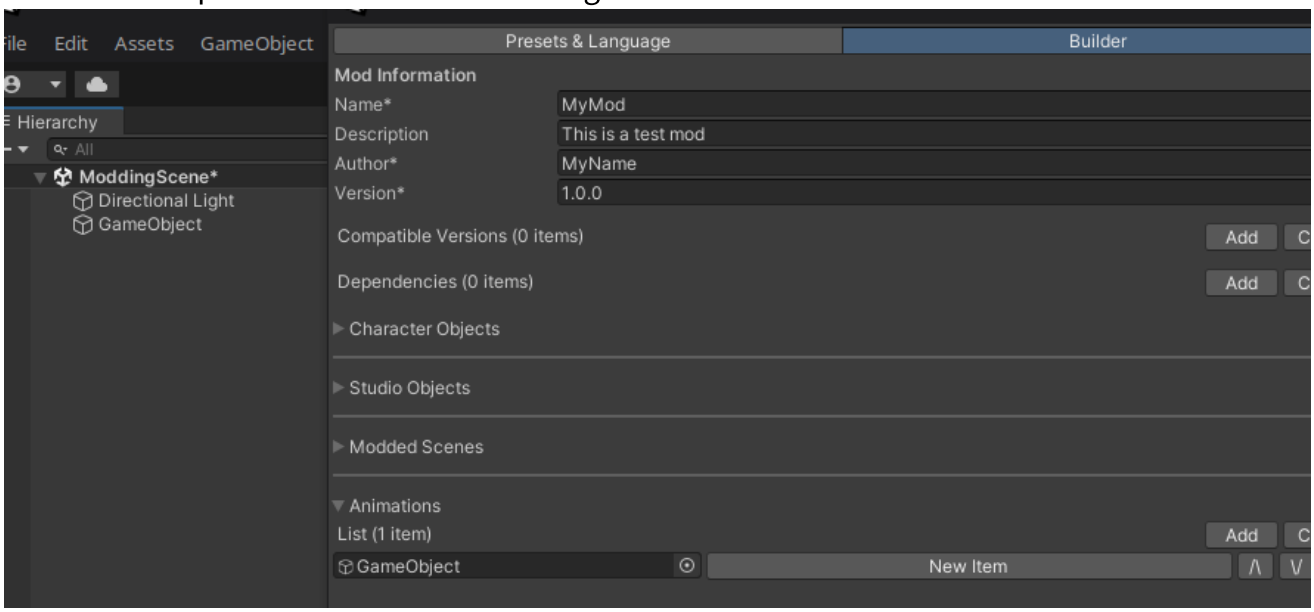
The root object of the scene must be called `ModdedScene` and should contain all objects that you want the scene to contain.

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.

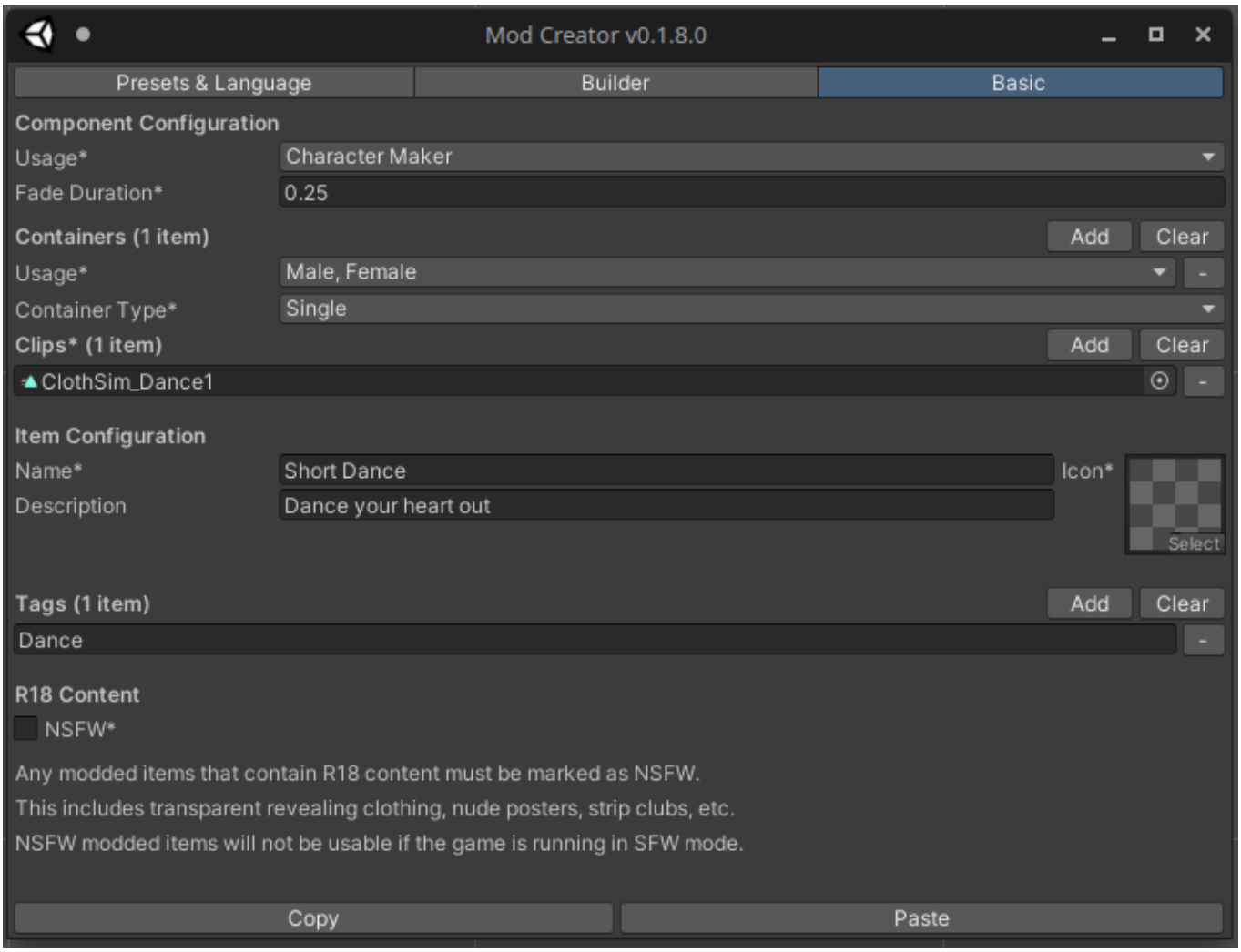
Creation of an Animation Mod

Begin creation of an animation mod by clicking **Add** in the Animation list.

1. Create an empty GameObject and assign it.
2. Click the Component to select it for editing



Once the animation 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.

Component Configuration

These settings affect the animation mod as a whole.

The **Usage** field indicates where the animation mod can be used. The entries *Interaction* and *Object* are currently only related to the H Scene. For example, a character dance animation could be used in *CharacterMaker*, *H Scene*, and as part of an *Interaction*.

A **Fade Duration** greater than *0* indicates that transitions to a container crossfade over the duration specified.

Containers

An animation mod may contain multiple animation clips organized into containers. Each container has settings and options for blending clips.

The **Usage** field primarily helps match containers to the correct animation target. For example, character-related animations usually provide at least one gender. If it is H-related, the actor's role could be passive or active. Entries here will be added with additional usage types as the system grows (based on your feedback). This is especially useful when dealing with multiple containers and creating paired animations.

The **Container Type** determines the amount of animation clips and how these work together:

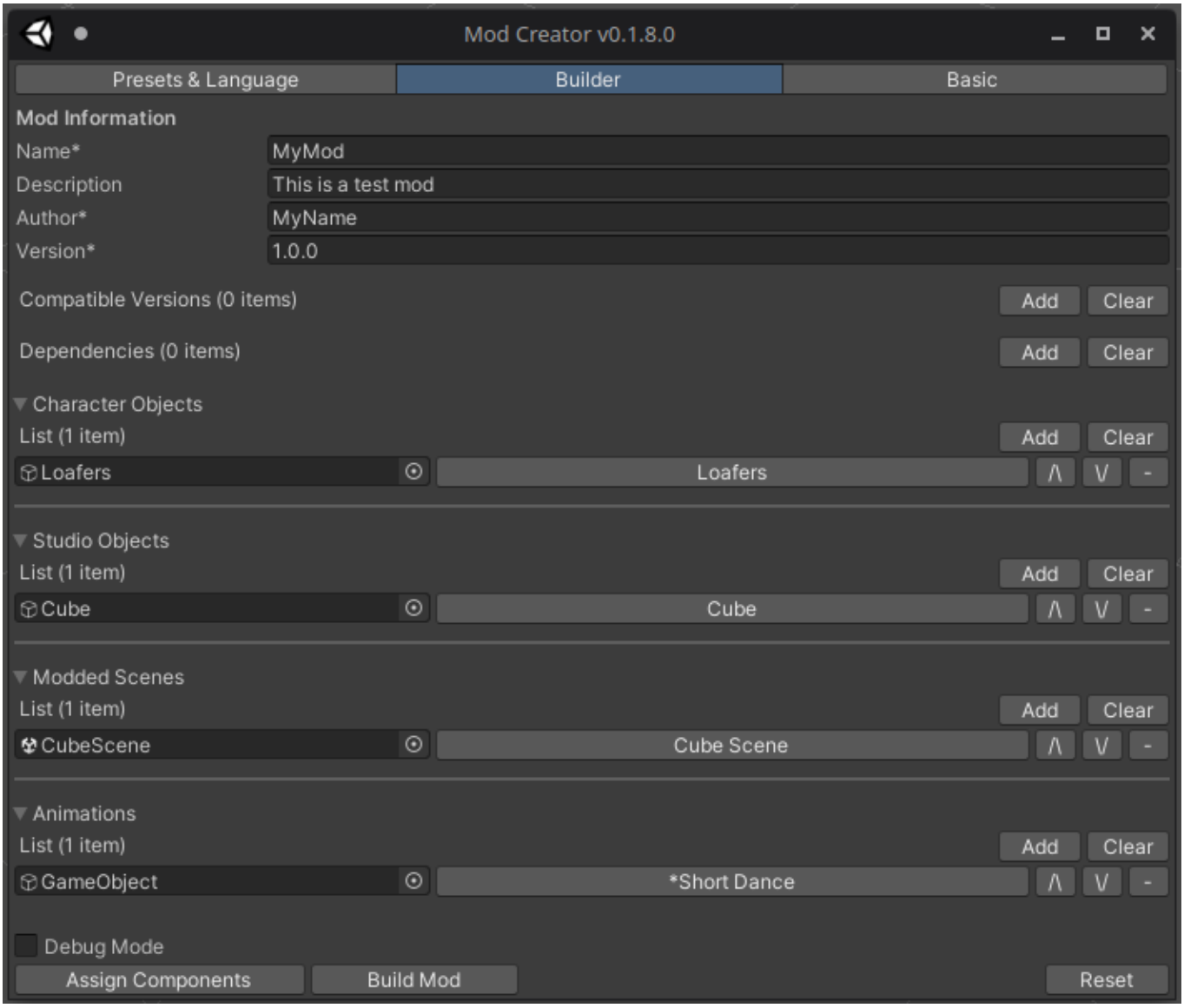
Type	Description
Single	This container contains a single animation clip. No more options are necessary.
Linear	Similar to a 1D blendtree. Each animation clip is assigned a treshold value. Animation clips will then be blended linearly based on a single named parameter between the min and max threshold values defined.
2D	Similar to a 2D (directional) blendtree. Each animation clip is assigned a position value (x,y). Animation clips will then be blended based on two named parameters

Check the links for more information about 1D and 2D blending.

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

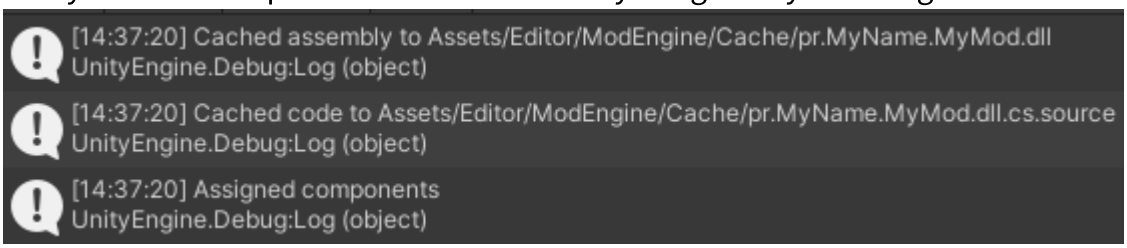
Building the Mod

Once you have set up the mods basic information and added at least one modded item, 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.



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. Behavior in the editor may be different from the game.

Make sure to test the mod in-game 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.
    - Fields/Properties of GameObject type are currently not supported, serialize Transforms instead
- 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.
- LODs
  - Clothing fitting and clothing simulation might misbehave when multiple LODs are present. A solution is work in progress.
- Clothing Fitting
  - Currently only one mesh renderer is considered per clothing state for clothing fitting. Merge multiple meshes into one for it to be fitted properly. A proper solution is work in progress.
- Advanced Mode
  - Using the "Reload Mods" functionality in-game with mods that use Advanced mode will very likely not reload the custom code, requiring a game restart.