Concrete And Steel Modding SDK 0.2

April 2016 - Matterhorn Software LTD

Changelog

0.1.0 27/04/2016 - Initial Alpha release to testers.
0.1.1 28/04/2016 - Added Network Identity / LPA information
0.2.0 04/05/2016 - Added description, author and version number to the Buildable script, added paintables. Reorganised the documentation to allow for paintables.

Introduction & Prerequisites

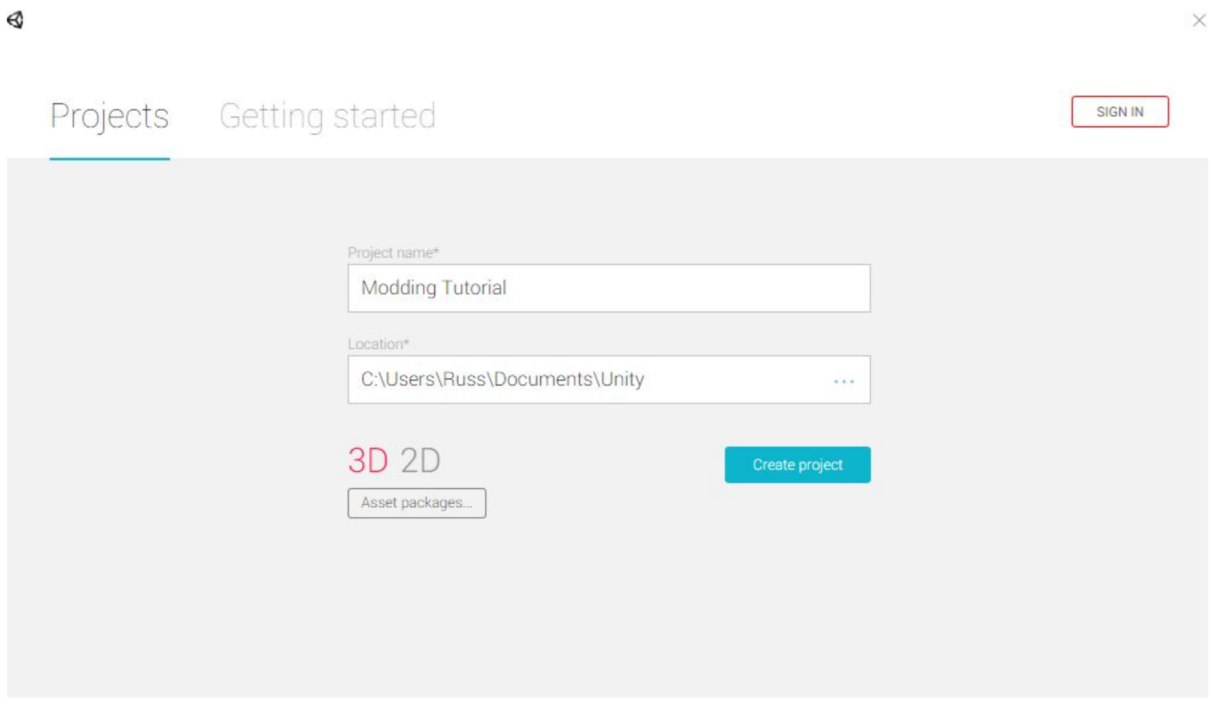
This documentation primarily explains the concepts for creating a Concrete And Steel “buildable object”, including setting up the editor for operations which are essential for composing the mod.

A “buildable object” is an object within the Concrete And Steel game which is used for construction. The object’s purpose may be for decorative effect or as part of a structure.

You should already have composed the 3d object you wish to import into the game. Furthermore, this documentation does not go into how to texture your object, set the hierarchy of the objects or import and apply materials or textures. It will really help if you already have a good grasp of how the Unity editor works.

Set Up Unity

First you need to install Unity editor. Once it's installed, run Unity, create a new project - you can name this whatever you want. Choose 3D, with no asset packages.



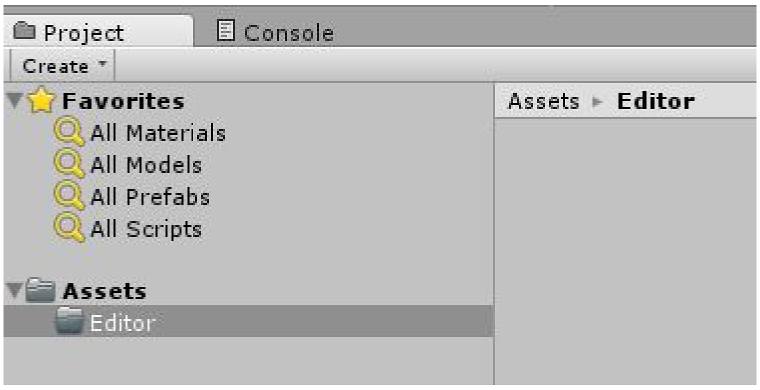
Project name*
Modding Tutorial

Location*
C:\Users\Russ\Documents\Unity

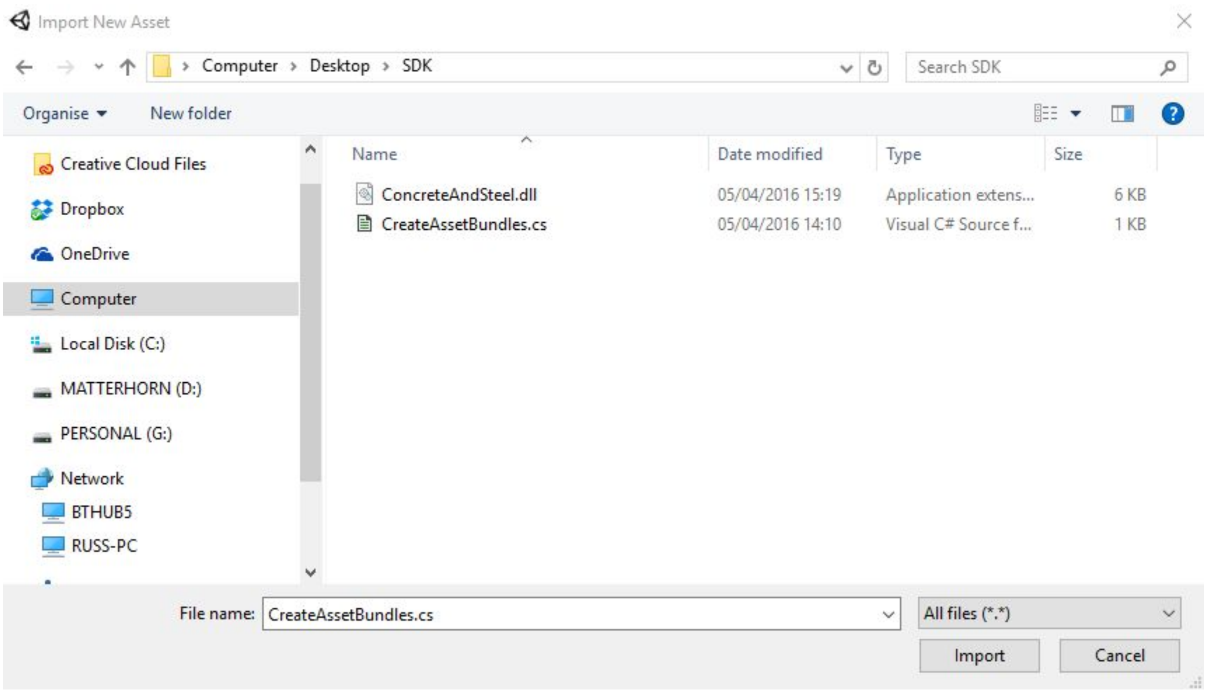
3D 2D

Import Editor Script

In order to export this mod as an "assetbundle", we need an editor script. Create a folder under "**Assets**" called "**Editor**"



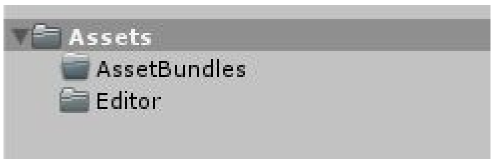
Next, on Editor, right click and select “Import New Asset”. We’re going to import the C# script “**CreateAssetBundles.cs**”



Name	Date modified	Type	Size
ConcreteAndSteel.dll	05/04/2016 15:19	Application extens...	6 KB
CreateAssetBundles.cs	05/04/2016 14:10	Visual C# Source f...	1 KB

File name: CreateAssetBundles.cs

We now also need to create a folder called “**AssetBundles”** under “**Assets**”:



This is where our mod will get exported to.

Import DLL

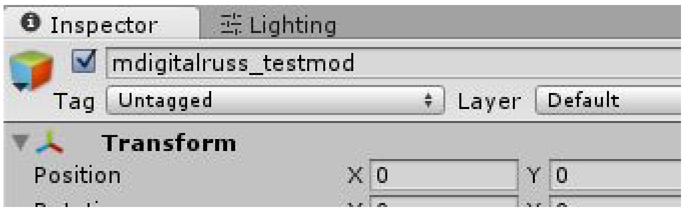
To use Concrete And Steel's features, we need the special Concrete And Steel modding DLL. Under Assets, right click and select "**Import New Asset**" and choose "**ConcreteAndSteel.dll**" (in the same way as we imported the C# script).

Create a Buildable object

In Concrete And Steel there are two types of asset: **Buildable** and **Paintable**. Buildable objects are meshes that can be spawned in the game world, while Paintable items are paints that can be applied to those Buildables.

In this chapter I will explain how to create a Buildable object, and in the next chapter I will explain paintables.

In Unity Editor, in the main toolbar, select "**GameObject > Create Empty**". This empty object will be our Buildable container. In the **Inspector,** rename the object to something unique. I recommend you prefix it with your username, to avoid accidentally overwriting someone else's mod. I'm going to call this mod "***mdigitalruss_testmod***". Press Enter to make it happen.



Set Object tree

Our mod needs a structure like below:

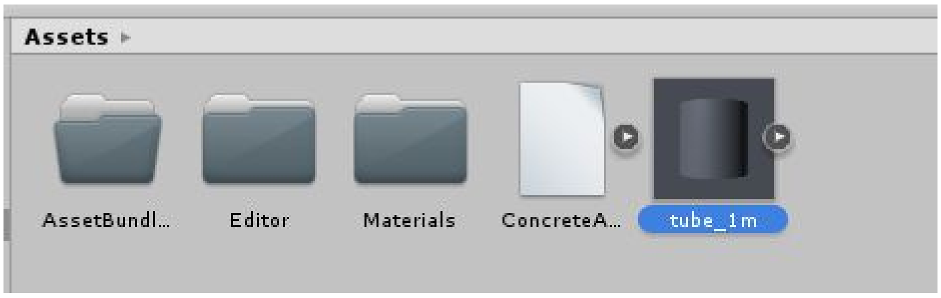
- Root Object
  - Aim Root
    - Aimer object
  - Built Root
    - Built Object

Under our mod, we should now create another empty gameobject. Right click on our root object we just made (in my case, *mdigitalruss_testmod*), and click “**Create Empty”**, and name it **“Built”.**

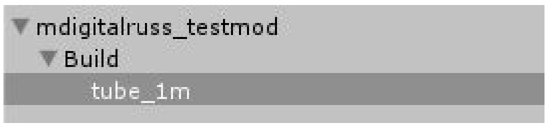
Import objects

Back in our Project view, under “**Assets**”, right click and select “**Import New Asset”**. In this example, I’m going to import a maya file called “***tube_1m.mb*”.** This takes a while, as Unity is painfully slow at importing maya files. It will automatically import the materials from Maya into the “**Materials**” folder.

Here’s my project so far:

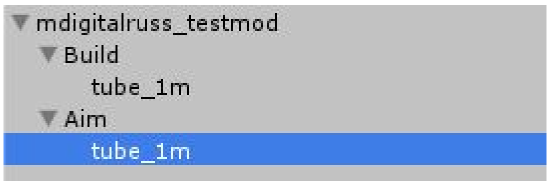


I’m going to drag “**tube_1m**” onto our “**Build”** gameobject that we just made:

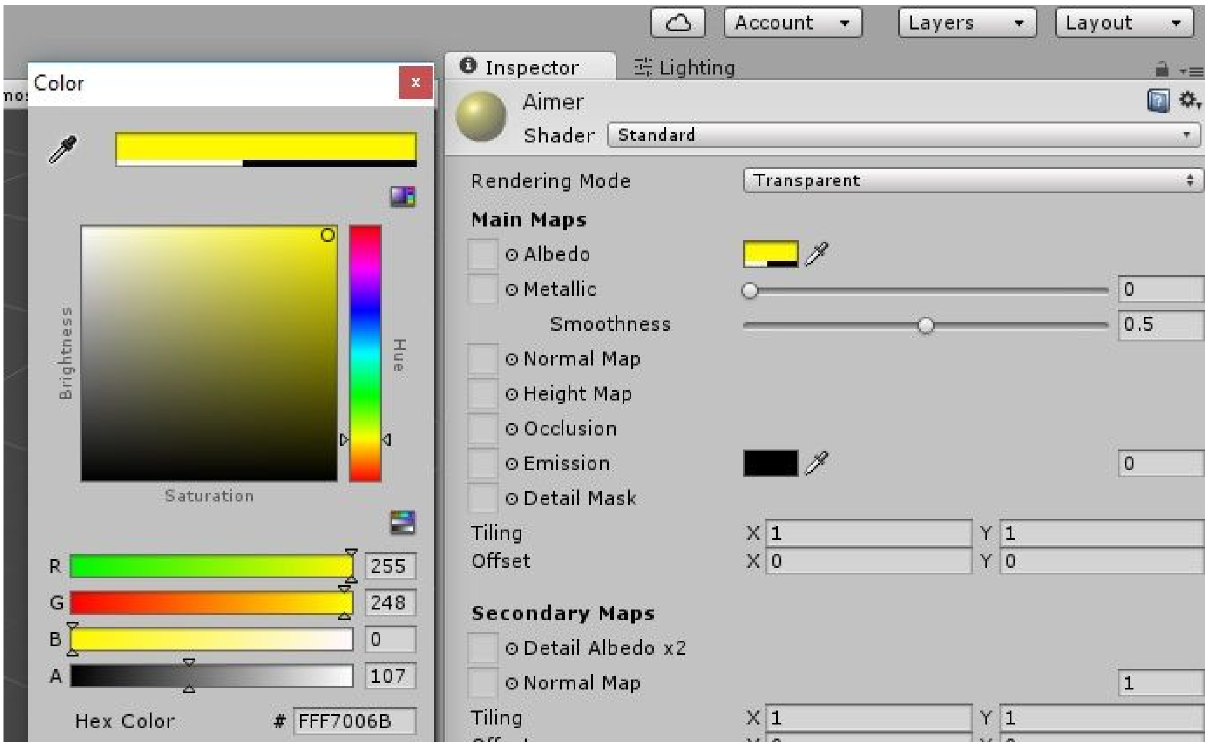


Create an Aimer object

So far so good! Before we carry on, we need to create an Aim object. This is the object that will appear while we’re aiming the object, before we build it. Duplicate what we’ve done so far by selecting “**Build”** and pressing **CTRL+D.** Rename it to “**Aim**”.



Now, in the project view, right click and select “**Create > Material**”. Name the material “**Aimer**”. In the Inspector view, set “**Rendering Mode**” to “**Transparent**”, set the **Albedo** colour to something like #***FFED00AB*** (In the Hex Colour box at the bottom):



Color
R 255
G 248
B 0
A 107
Hex Color # FFF7006B

Aimer
Shader Standard
Rendering Mode Transparent
Main Maps
Albedo
Metallic 0
Smoothness 0.5
Normal Map
Height Map
Occlusion
Emission 0
Detail Mask
Tiling X 1 Y 1
Offset X 0 Y 0
Secondary Maps
Detail Albedo x2
Normal Map 1
Tiling X 1 Y 1

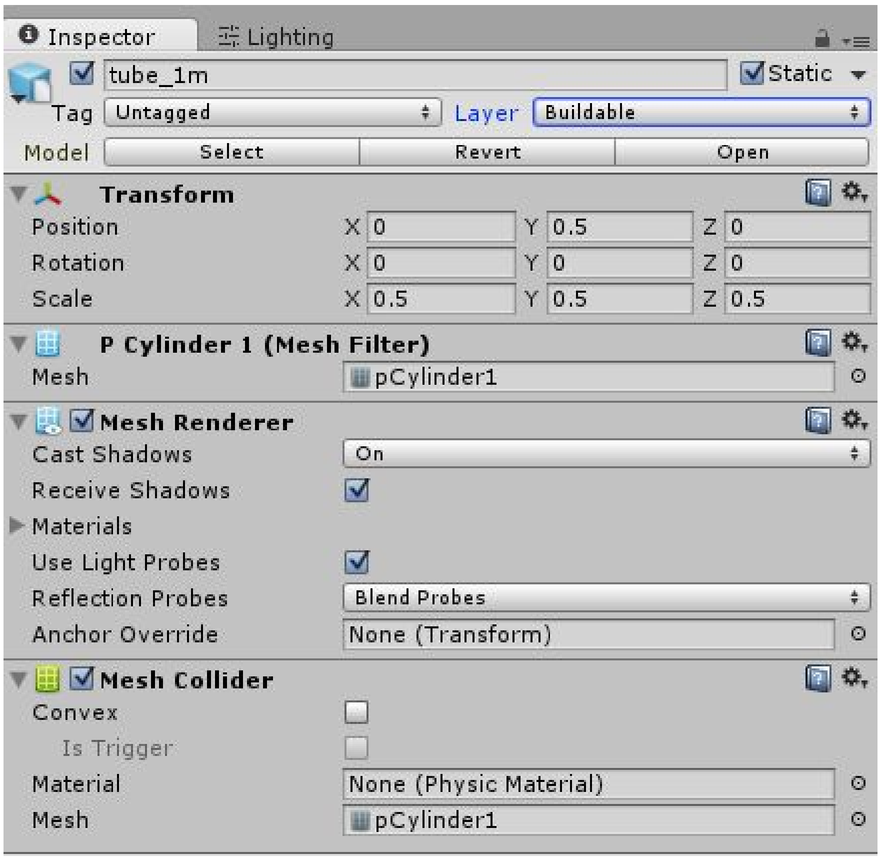
Now drag this material onto our “**Aim**” object in the **Hierarchy view**. This material will render the “**Aim”** object in a highlighted and semi-transparent way.

Make the Buildable

Now we need to configure our buildable object and assign the script.

With “***tube_1m*”** (under “**Build**”) selected, in the Inspector tick “**Static**”. Then select “**Layer > Add Layer”,** and in the box next to User Layer 8 type “**Buildable**”. Then go back to tube_1m, back to the inspector, and click “**Layer > Buildable”**.

At the bottom of the inspector, click **“Add Component”** and type **“Mesh Collider”** and press enter. The object is now buildable upon!


Inspector | Lighting

☑ tube_1m ☑ Static

Tag: Untagged | Layer: Buildable

Model: Select | Revert | Open

Transform

	X	Y	Z
Position	0	0.5	0
Rotation	0	0	0
Scale	0.5	0.5	0.5

P Cylinder 1 (Mesh Filter)

Mesh: pCylinder1

☑ **Mesh Renderer**

Cast Shadows	On
Receive Shadows	✓
Materials	
Use Light Probes	✓
Reflection Probes	Blend Probes
Anchor Override	None (Transform)

☑ **Mesh Collider**

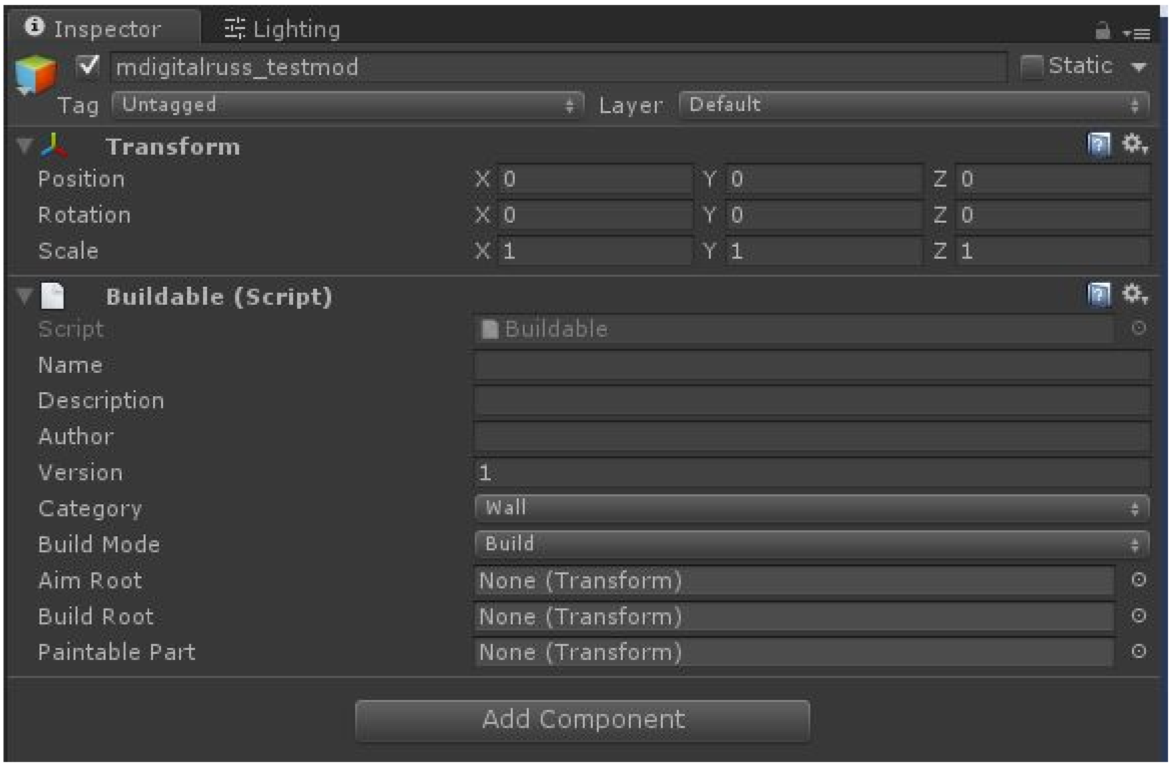
Convex	☐
Is Trigger	☐
Material	None (Physic Material)
Mesh	pCylinder1

Tip: Ensure the Aim object never has a collider and is never set as Static!

Assign Script

Now we need to assign the script which tells Concrete And Steel to load this object, and tells it which is the Build and which is the Aim object.

On the root object (“*mdigitalruss_testmod”* in my case), in the Inspector, click Add Component.Type “**Buildable**” and press enter.


Inspector | Lighting

mdigitalruss_testmod | Static

Tag: Untagged | Layer: Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

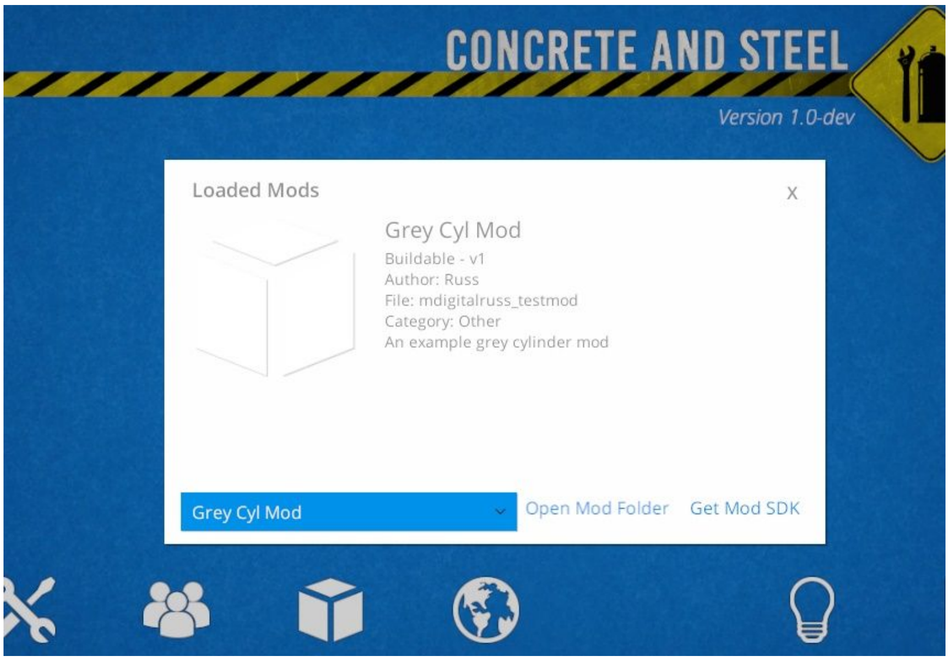
Buildable (Script)

Script	Buildable
Name	
Description	
Author	
Version	1
Category	Wall
Build Mode	Build
Aim Root	None (Transform)
Build Root	None (Transform)
Paintable Part	None (Transform)

Add Component

In the **name** field, type the name of your mod. I'm going to call this "Grey Cyl Mod". This is what users will see in the build menu.

You can specify a **description**, **author** and **version** here too. These details will be displayed in the Mods menu in the game.



Grey Cyl Mod
Buildable - v1
Author: Russ
File: mdigitalruss_testmod
Category: Other
An example grey cylinder mod

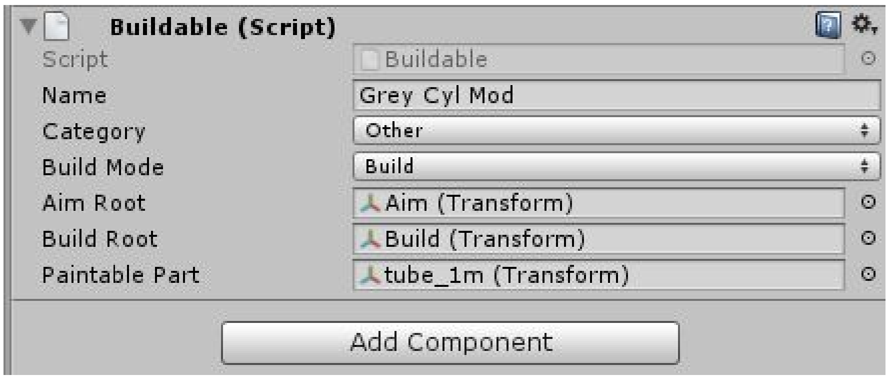
For the **version** number, if it's your public release just use 1.0, and each small change is 1.1, 1.2, etc. For big changes that are incompatible with previous versions, move up to to 2.0, 2.1, etc. For testing and pre-release, use a number below 1.0, like 0.1, 0.2, etc.

Under **category**, I’m just going to select Other. Pick the one most relevant to your mod. “Unlisted” means it won’t appear in the build menu, so it’s not wise to select that category.

For build mode you have two options, “Build” and “Place”. “Build” constrains the object to the 10mm grid and 90-degree rotations, whereas “Place” allows the object to be placed in any position and any location. It’s advisable to set structural components to “**Build**”, and props or decorations as “Place”.

Now drag the “**Aim**” object (from under the root object) onto the “**Aim Root**” field. Repeat this for Build Root, dragging the “**Build**” object onto that.

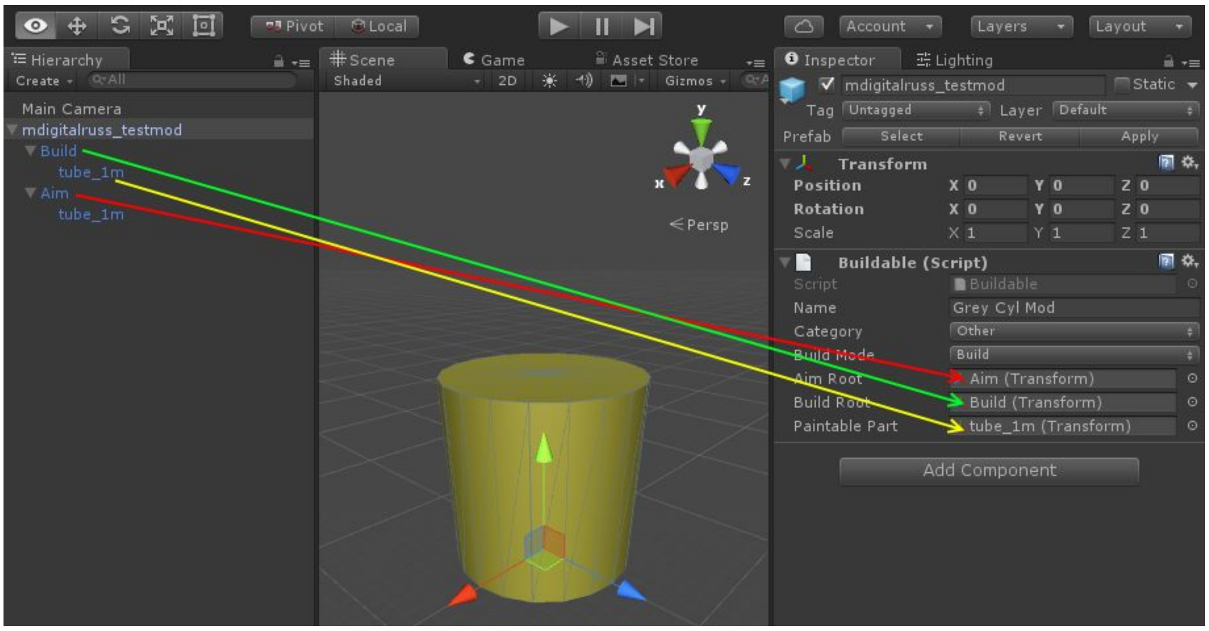
For paintable part, this is the object we’re going to see change colour when we paint it with the Paint tool. This can only be one mesh currently. Drag “***tube_1m***” from under “**Build**” to the Paintable Part field. If you don’t want any part to be paintable, leave that field blank.



Buildable (Script)

Script	Buildable
Name	Grey Cyl Mod
Category	Other
Build Mode	Build
Aim Root	Aim (Transform)
Build Root	Build (Transform)
Paintable Part	tube_1m (Transform)

Add Component



Hierarchy
- Main Camera
- mdigitalruss_testmod
  - Build
    - tube_1m
  - Aim
    - tube_1m

Inspector: mdigitalruss_testmod — Tag Untagged, Layer Default

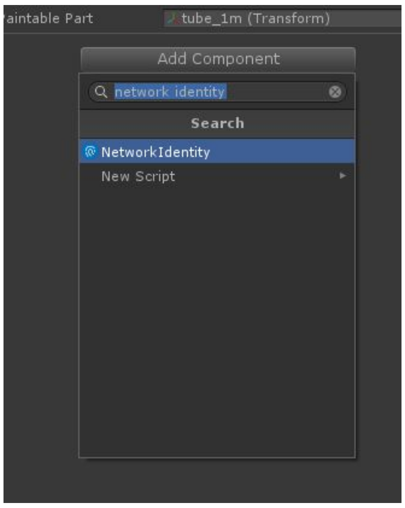
Transform

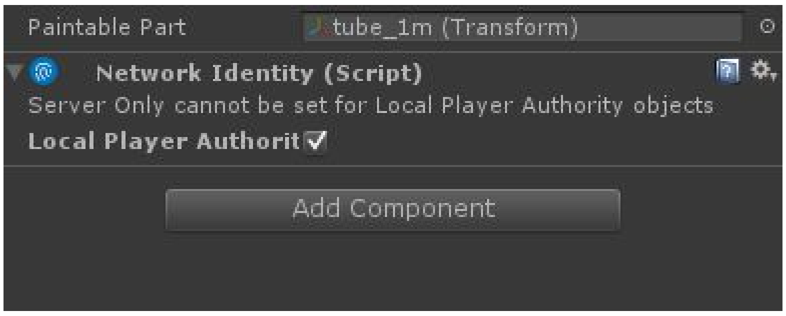
Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Buildable (Script)

Script	Buildable
Name	Grey Cyl Mod
Category	Other
Build Mode	Build
Aim Root	Aim (Transform)
Build Root	Build (Transform)
Paintable Part	tube_1m (Transform)

Finally, add a “**Network Identity**” component under the Buildable script, and tick “**Local Player Authority**”. This is important for making the object paintable.

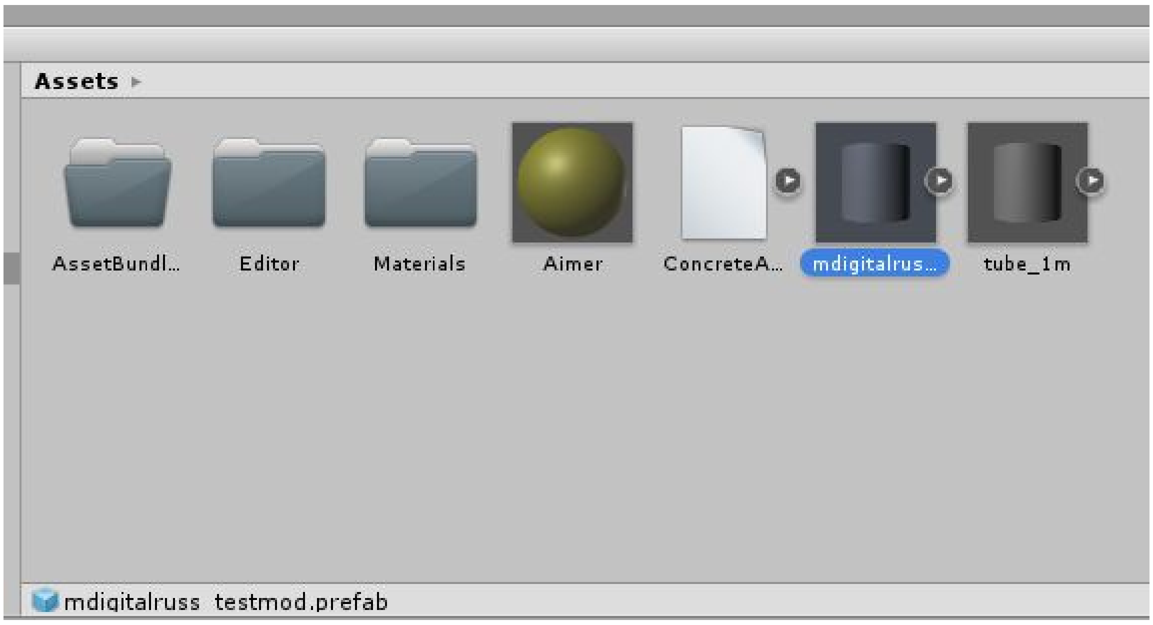




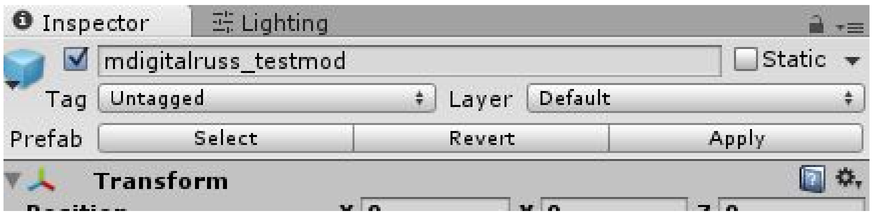
Server Only cannot be set for Local Player Authority objects

Create Prefab

Now we need to turn this object into a prefab. Drag your root object (in my case, mdigitalruss_testmod) to under “**Assets**” in the **Project** view. This will create another object called “*mdigitalruss_testmod.prefab*” or whatever you named your mod.



IMPORTANT: Once you’ve made the prefab, if you need to go back and make changes to the mod, remember to select the root object of your mod and click “**Apply”** to make the changes save to the Prefab.

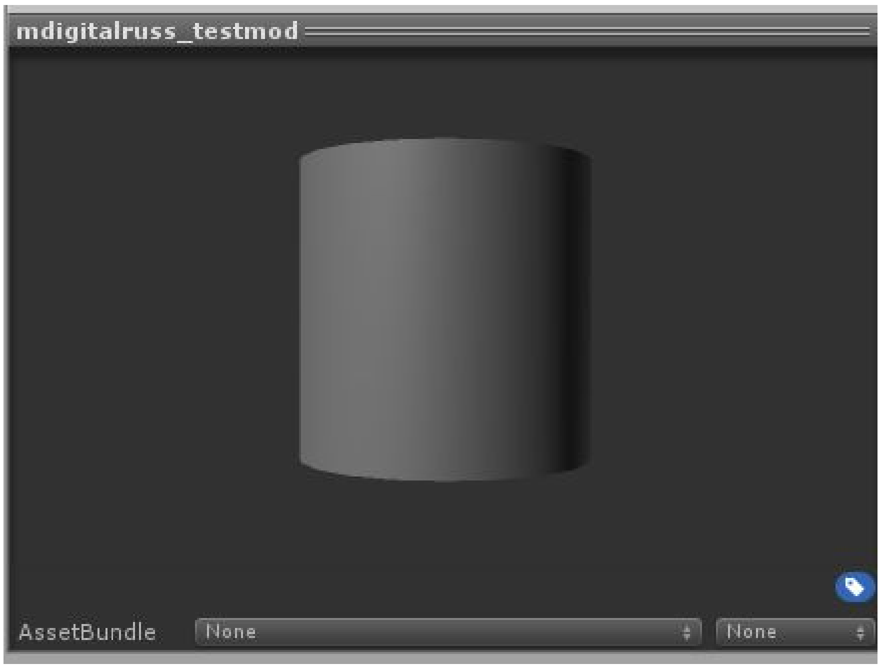


If you don't click "**Apply"**, your changes won't get saved to the Prefab, and therefore won't get exported.

Create Assetbundle

If you're still alive by this point, and not bored to death, we need to finally set an assetbundle.

Select the prefab you just made. In the bottom right you will see a preview:



In the "**AssetBundle**" drop down menu, click "**New**..." and type the name of your mod (in my case, you guessed it, *mdigitalruss_testmod*) and **press enter** to make it so.

If you already have an AssetBundle, you can just select that from the list instead of making a new one.

Create A Paintable

In Concrete And Steel you can also make a **Paintable**. This is a paint that can be applied to any in-game object which has a surface that can take paint. In the context of modded Buildables, the paint is applied to the GameObject referenced as “Paintable Part” in the Buildable script.

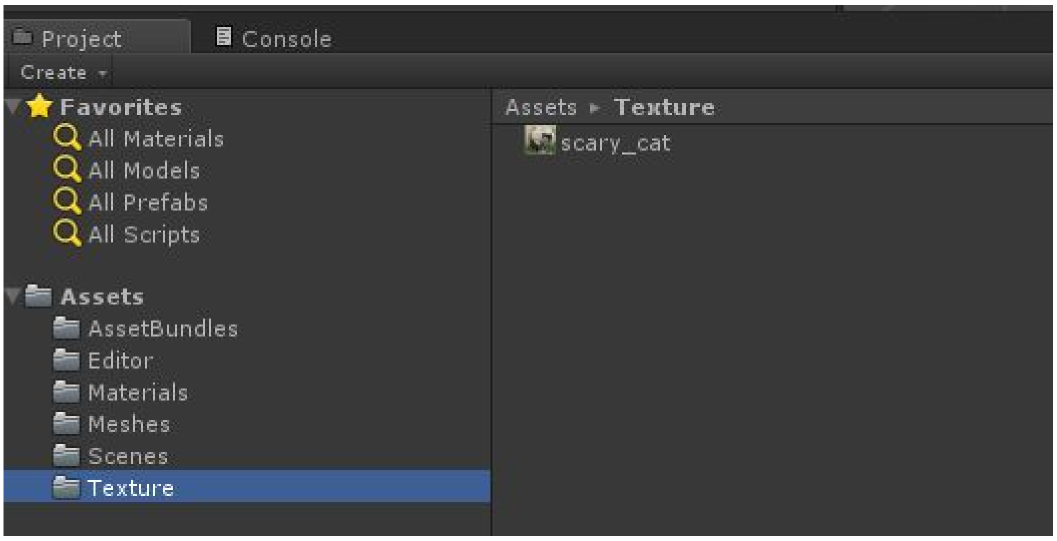
In Unity, in the main toolbar, select “**GameObject > Create Empty**”. This empty object will be our Paintable container. In the **Inspector,** rename the object to something unique. I recommend you prefix it with your username, to avoid accidentally overwriting someone else’s mod. I’m going to call this mod “***mdigitalruss_testpaint***”. Press Enter to make it so.

Configure Your Materials And Textures

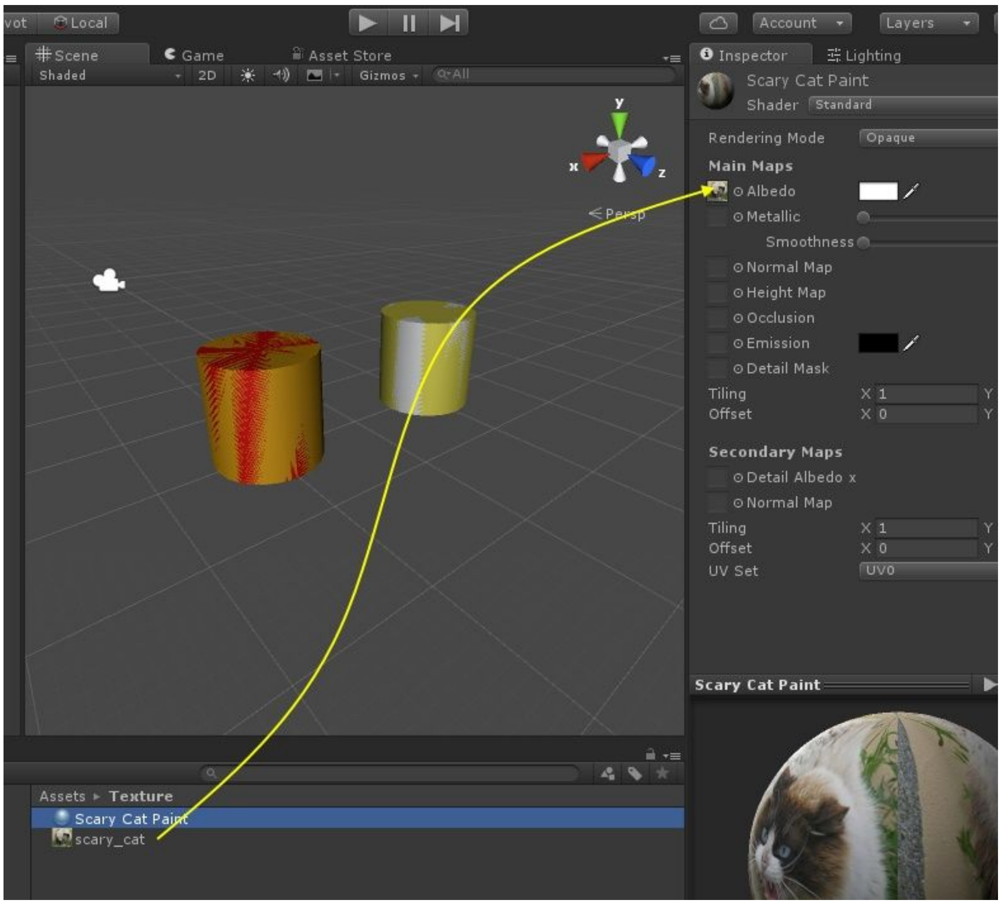
The first thing you will want to do is find a texture. I have found this glorious picture of a cat, and I can’t wait to paint my living room with it.

The image is by Luis Miguel Bugallo Sánchez from Santiago de Compostela, Galicia, taken from Flickr, and the licence is CC BY 2.0. Always make sure you have a licence for assets, and always give credit where credit is due.

In Unity, in the Project window, right click and select **Import New Asset**, and import your texture. Make sure your texture is Power Of Two sized, for example 128x128, 256x256, 512x512, 1024x1024, etc. Anything over 2048x2048 is probably overkill, and will result in poor performance or long load times on low powered computers.

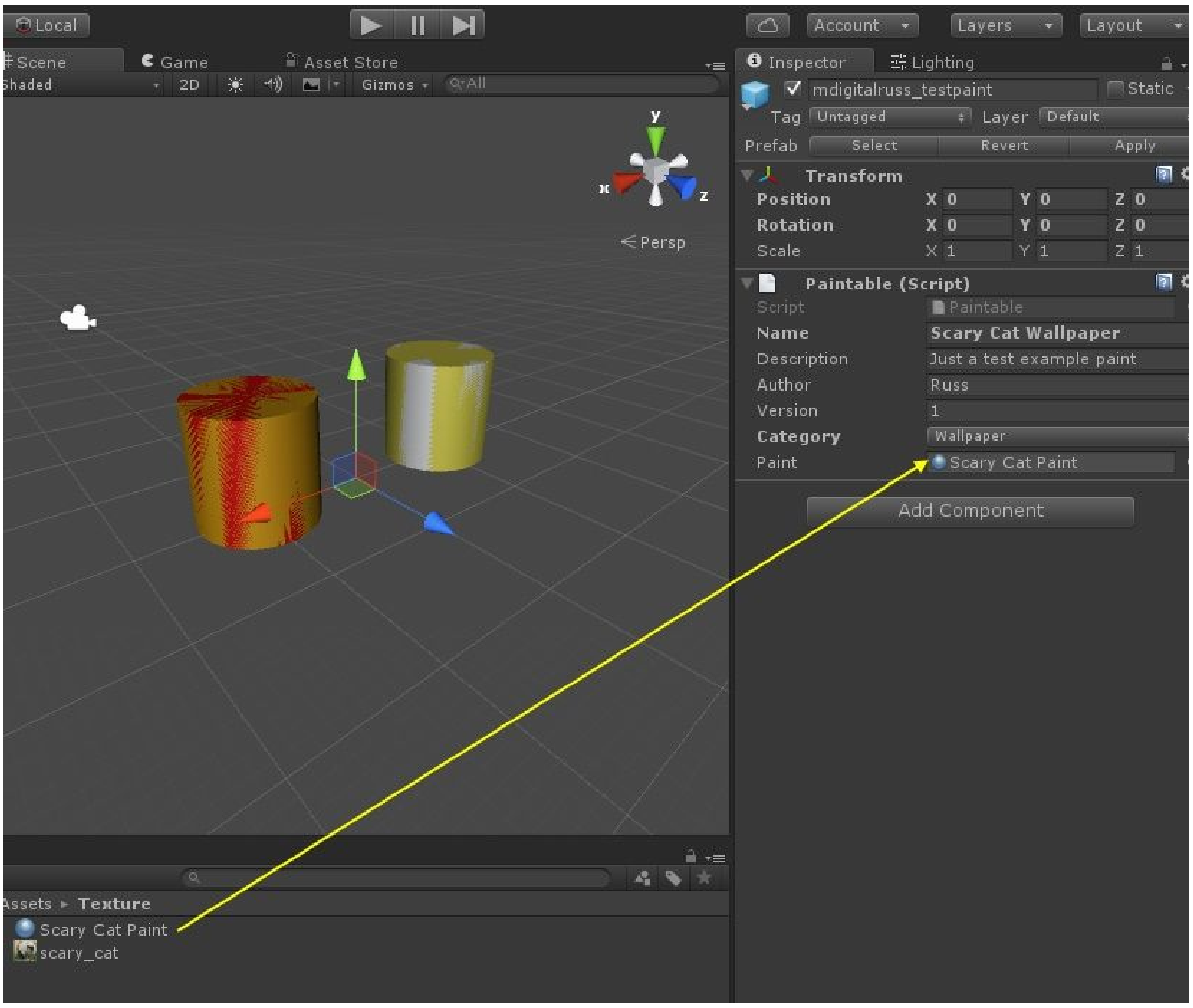


Now we have our Texture imported, we need to make a Material, just like we did for the Aimer material. Right click in the Project window and select **Create > Material.** With this new material selected, drag and drop your texture from the Project window to the square next to “Albedo” in the Inspector window, like this:



Assign the Paintable Script

We don't need an object tree like the Buildable items, so we can go right ahead and assign the script. In the Hierarchy select your empty gameobject again (in my case, "*mdigitalruss_testpaint*") and in the inspector window click **Add Component** and type "**Paintable**"



mdigitalruss_testpaint
Static
Tag Untagged
Layer Default
Prefab Select Revert Apply

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Paintable (Script)

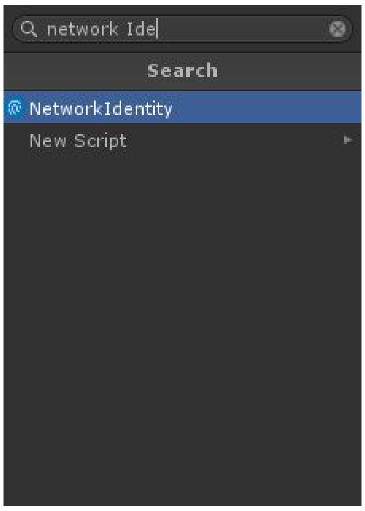
Script	Paintable
Name	Scary Cat Wallpaper
Description	Just a test example paint
Author	Russ
Version	1
Category	Wallpaper
Paint	Scary Cat Paint

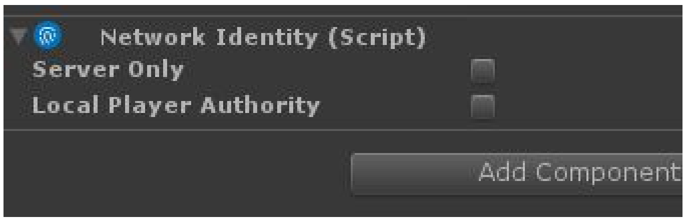
Assets ▸ Texture
Scary Cat Paint
scary_cat

As you can see I've filled out a **name**, **description**, **author** and **version** and I've selected the Wallpaper **category**.

Drag and drop your Material from the **Project** window onto the **Paint** item in the Inspector.

Finally, don't forget to add a "**Network Identity**" component under the Paintable script. You don't need to tick any boxes in this script for Paintables.





Create a Prefab

Now we need to turn this object into a prefab. Drag your root object (in my case, *mdigitalruss_testpaint*) to under "**Assets**" in the **Project** view. This will create another object called "*mdigitalruss_testpaint.prefab*" or whatever you named your mod.

IMPORTANT: Once you've made the prefab, if you need to go back and make changes to the mod, remember to select the root object of your mod and click "**Apply"** to make the changes save to the Prefab.

Create or Assign an Assetbundle

You can either use the same AssetBundleas you made in the Buildable chapter, or you can create a new one.

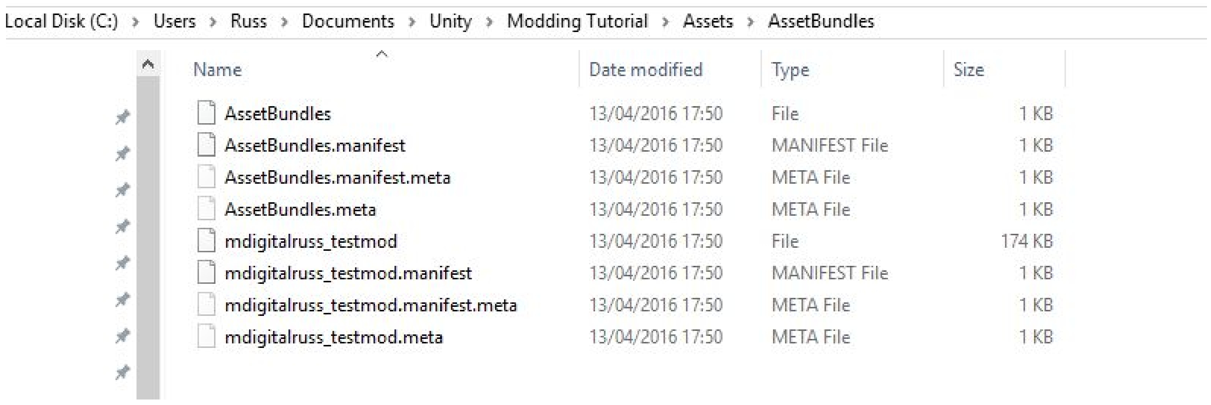
Select the prefab you just made, and down in the bottom right you will see the preview area with the title "**Assetbundle**".

For a new AssetBundle: Under the "**AssetBundle**" drop down box, click "**New**..." and type the name of your mod and **press enter** to make it so.

To use an existing AssetBundle, Just select the name of the AssetBundle you want to use instead of clicking "**New**..".

Export Your Assetbundle

Finally, in the Unity toolbar, select **“Assets > Build AssetBundles”.** You will get a progress bar, and then eventually you will get the mod files in your “**AssetBundles”** folder. On “**AssetBundles”**, right click and select “**Show in Explorer”,** and select the “**AssetBundles”** folder.



Local Disk (C:) › Users › Russ › Documents › Unity › Modding Tutorial › Assets › AssetBundles

Name	Date modified	Type	Size
AssetBundles	13/04/2016 17:50	File	1 KB
AssetBundles.manifest	13/04/2016 17:50	MANIFEST File	1 KB
AssetBundles.manifest.meta	13/04/2016 17:50	META File	1 KB
AssetBundles.meta	13/04/2016 17:50	META File	1 KB
mdigitalruss_testmod	13/04/2016 17:50	File	174 KB
mdigitalruss_testmod.manifest	13/04/2016 17:50	MANIFEST File	1 KB
mdigitalruss_testmod.manifest.meta	13/04/2016 17:50	META File	1 KB
mdigitalruss_testmod.meta	13/04/2016 17:50	META File	1 KB

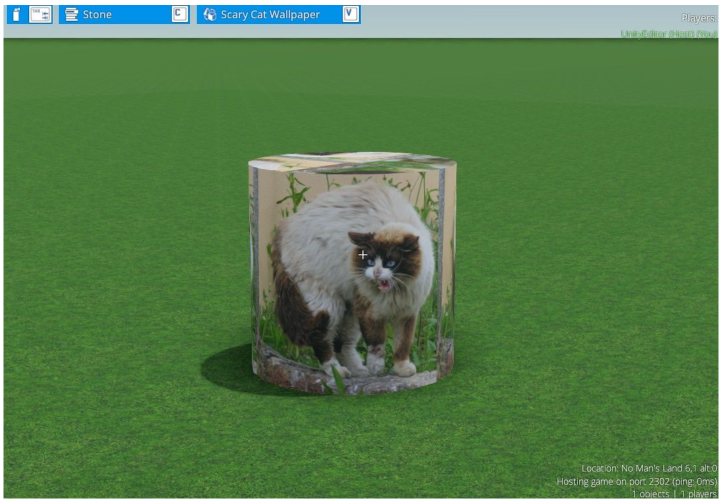
The only file we need from here is the file with the EXACT name as your mod. So in my case, ***mdigitalruss_testmod*** (174kb). You can ignore the other 1kb files.

Copy Mod To Game

Find your mod folder, it’s normally here:

C:\Program Files (x86)\Steam\steamapps\common\Concrete And Steel\ConcreteAndSteel_Data\StreamingAssets

Copy and paste your mod file from the “**AssetBundles”** folder to the “**StreamingAssets”** folder. Now run the game. Your mod should be available in the game!



Further Consideration

- For multiplayer games, both the host and the client need the same mod in their StreamingAssets folder. This also applies to dedicated servers.

- To make changes to your mod, remember to click **Apply** to make the change save to the prefab. Then repeat the build process by clicking “**Assets > Build AssetBundles**” and copying the mod over to your “**StreamingAssets”** folder in the game’s directory. If you do not click apply, the changes won’t be saved!

- If you make changes after the release which could break compatibility, remember to change the **version** number in the **Buildable** configuration script.

- Please obey the EULA for Concrete And Steel when creating and distributing mods. Be aware that you are not, for example, permitted to sell or trade any mods using this SDK for profit or reward.

Authored by Russ Peterson. For help, information and bugs contact software@m.digital.