FACTORY FRAMEWORK

F.A.Q. and Guide

Latest version can be found
https://docs.google.com/document/d/1U2zVDWSDcqG6s1sXl5qliNWZ3GclAbTwtYuxOxIfalg/edit?usp=sharing

For further questions, check the CtrlAltBees discord:
https://discord.gg/9tnKg9XPpV

Buildings, Belts, Items, Recipes

Welcome, and thank you for the support. Please do not hesitate to reach out with questions, problems, or suggestions to support@ctrlaltbees.com

The Factory Framework is a modular and highly-configurable system. **Items** are Scriptable Objects that hold visual elements and other data. **Recipes** define combinations of items to produce output items. **Buildings** can store, produce, or process items and recipes. **Belts** tie everything together by transporting items to and from buildings.

You can find more details in each section below.

Belts

Basics

The included mesh generator will create procedural conveyor belt meshes along a path. A path requires a start point, start direction, end point, and end direction. The PathFactory class GeneratePath method is used to create a path between the start and end points. This method will reference the global CLS settings for path-solving parameters.

Once a path is generated, the next step is to call BeltMeshGenerator.Generate(), which takes an IPath reference, a BeltMeshSO, the number of segments to generate (controlling the resolution of the mesh), and a scale factor to scale the mesh by.

Paths

Three different path types exist within FactoryFramework:

Smart path:

This path solver will connect the belt start and end points using a combination of arcs and straight line segments. These are tangent lines between circles position by the start and end vectors, using a smart solver to find the best approach.

Settings:

BELT_TURN_RADIUS - Controls the minimum turn radius for horizontal arcs
BELT_RAMP_RADIUS - Controls the minimum turn radius for vertical arcs (ramps)
BELT_VERTICAL_TOLERANCE - When the start and end points differ in height by more than this value, the solver will add vertical arcs (ramps) into the path.

Spline:

This path solver will construct a cubic bezier curve between the start and end points.

Settings:

BELT_TURN_RADIUS - Controls the distance at which bezier control points are placed along the curve

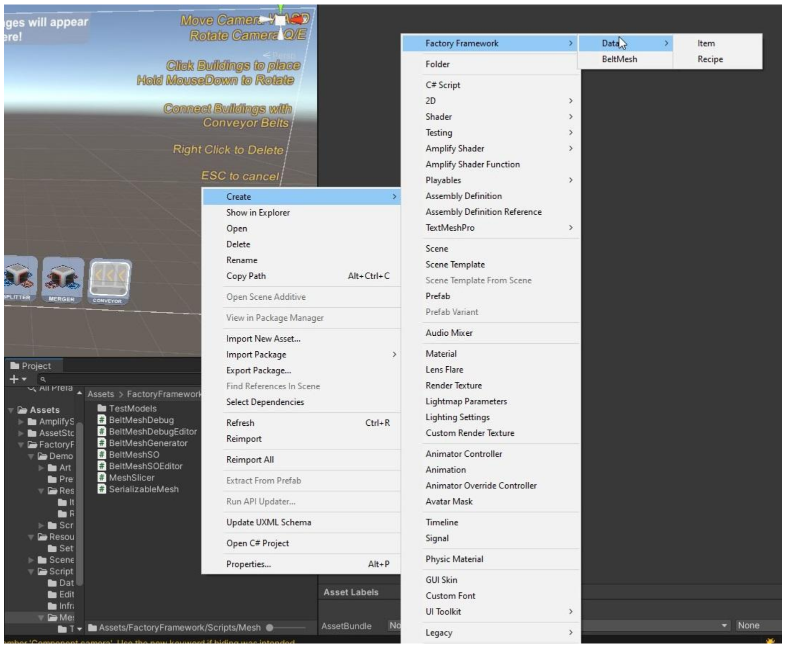
Segment:

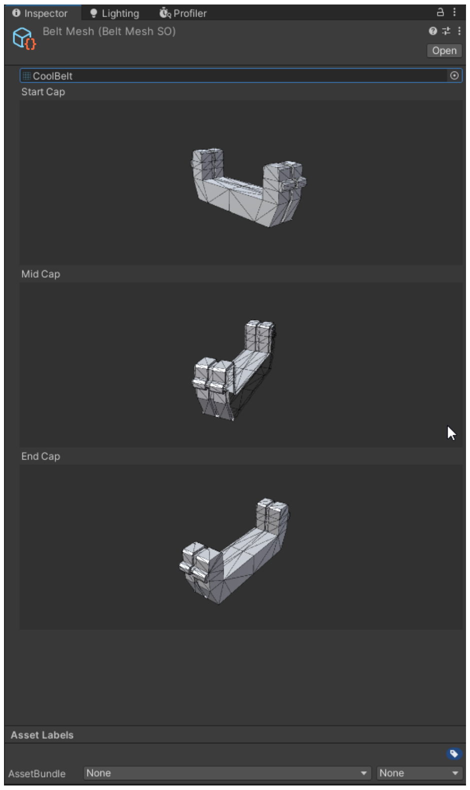
This path solver will simply construct a linear line segment from start to end. This is the *most performant* option but does not handle turns gracefully. Consider this option for **mobile** or more **grid-based** linear factory setups that don't require belts to turn sharply.

BeltMeshSO

Using this tool you can customize the model used for the conveyor belts.

The BeltMeshSO class functions as a holder for your conveyor belt models. To create a BeltMeshSO, right click anywhere in your Project window and select Create->Factory Framework->Belt Mesh:





Upon selecting the BeltMeshSO asset, you will see an editor window similar to the image shown on the right.

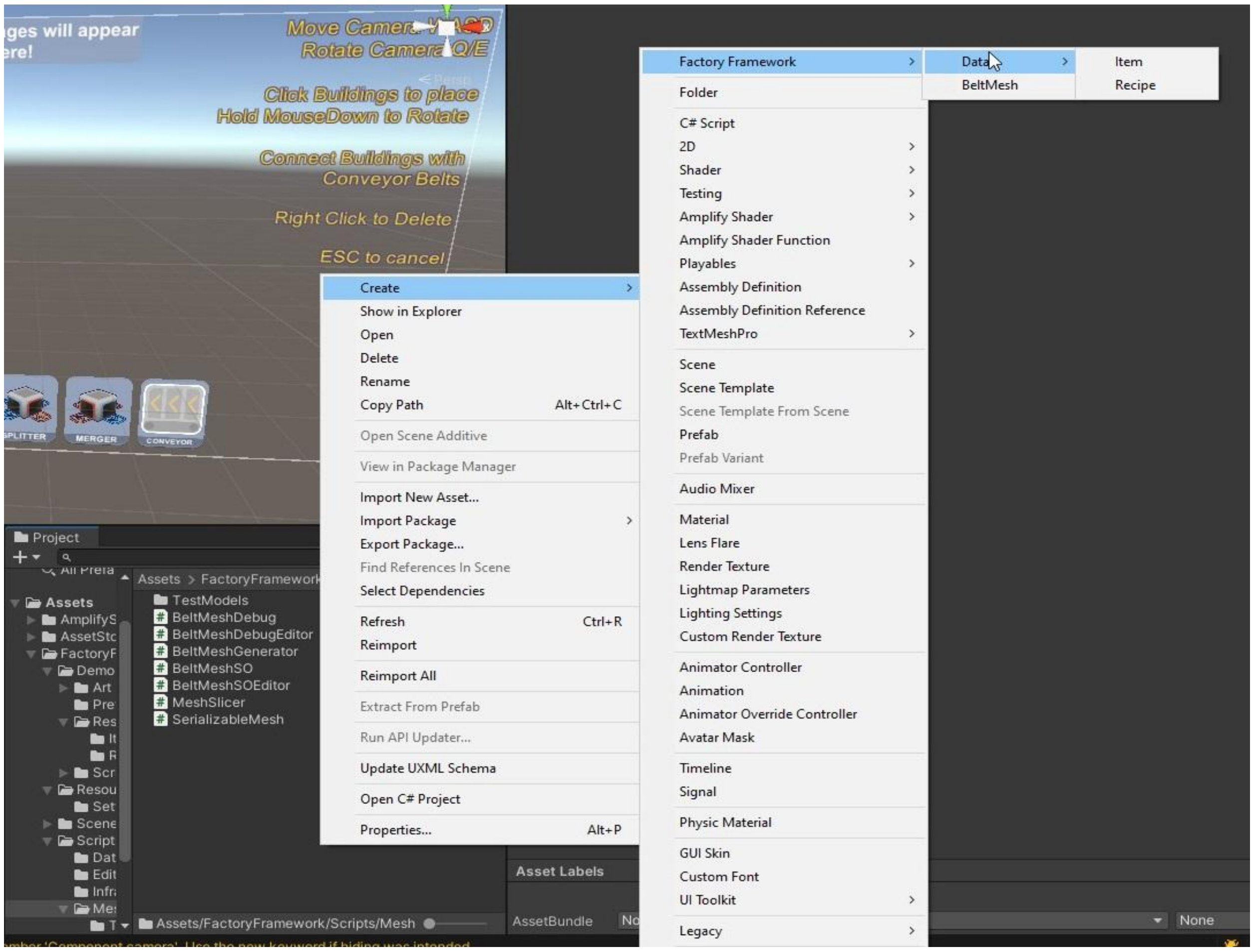
The first field in this Editor is where you will assign your belt mesh model. Once a model is selected, CLS will automatically slice it into thirds along the Z axis, creating a start, middle, and end. The middle segment will be copied along the length of your path, creating the conveyor belt mesh.

If you wish to have an animated belt, please use two separate BeltMeshSO assets- one for the frame, and one for the belt itself. This pattern is demonstrated in the CLS demo scene. Please also reference the attached fbx file for a template.

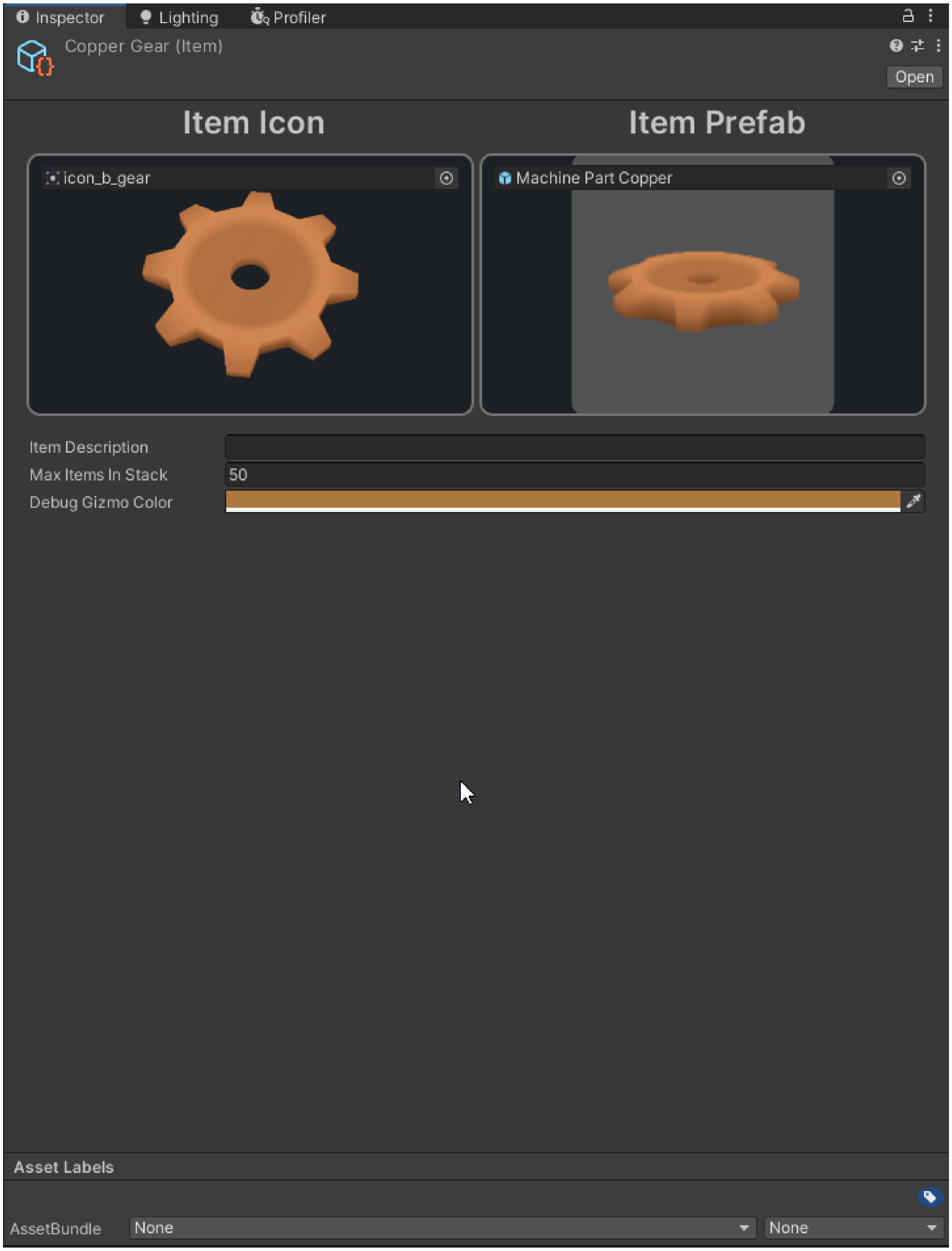
As these meshes will be extruded along the whole path, please keep the polygon count in mind - generating the conveyor mesh can be costly if the polygon count is too high.

Items

Buildings and belts interact with Items. An Item can be defined by right clicking in your "Project" window, then selecting Create -> Factory Framework -> Item.
Important: Item ScriptableObjects must be stored in a [Resources](#)folder to facilitate save/load functionality!



The item definition will appear in your current folder. Selecting it will reveal the item editor:



Copper Gear (Item)

icon_b_gear

Machine Part Copper

Item Description
Max Items In Stack 50
Debug Gizmo Color

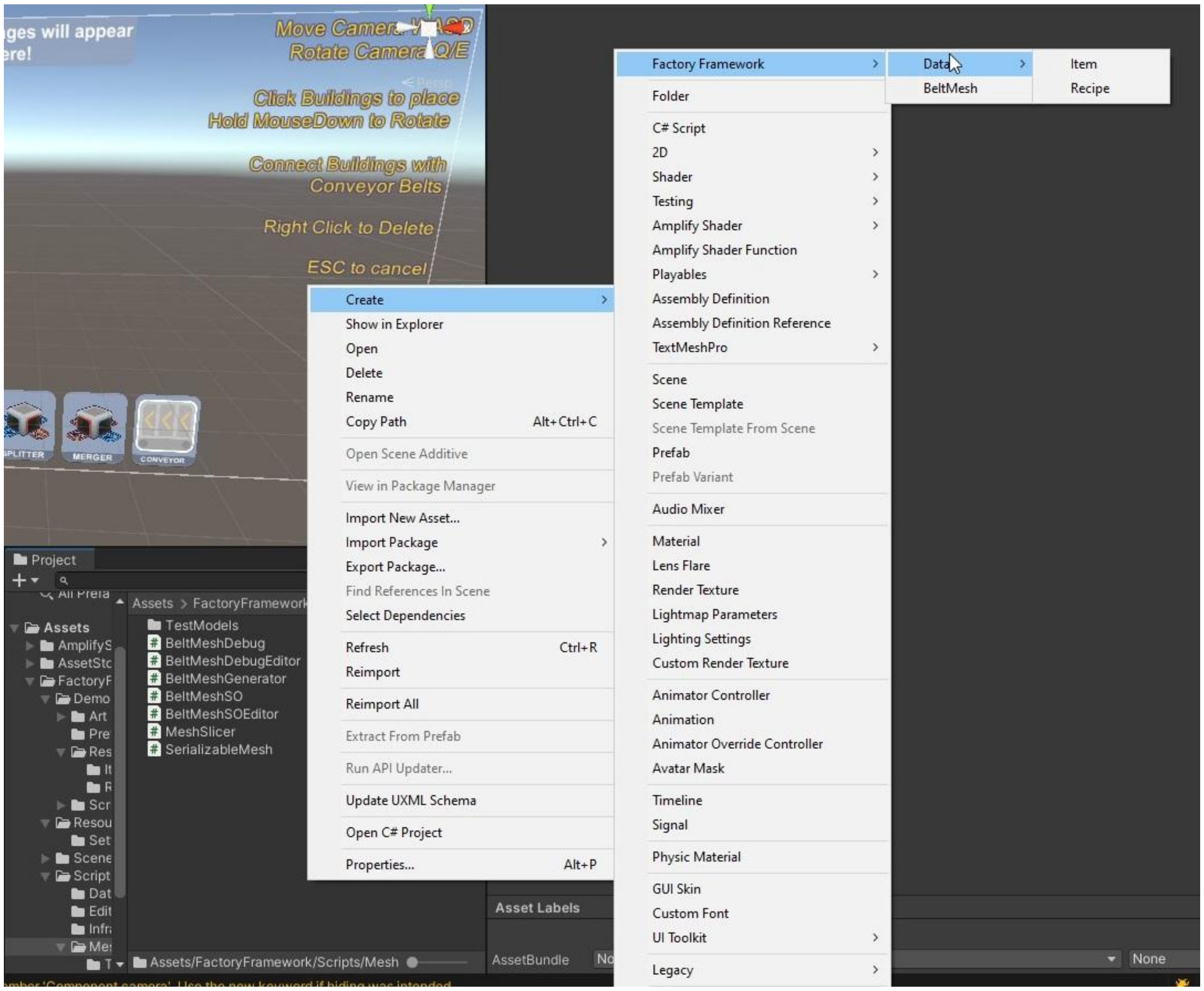
AssetBundle None None

The first preview window on the left shows the *item icon*. This should be an image file, and is intended to be used in factory building UIs. The preview window on the right shows your *item prefab*, which is the prefab that will be shown on belts while the item is moving. This should be a simple prefab with a mesh renderer and mesh filter.

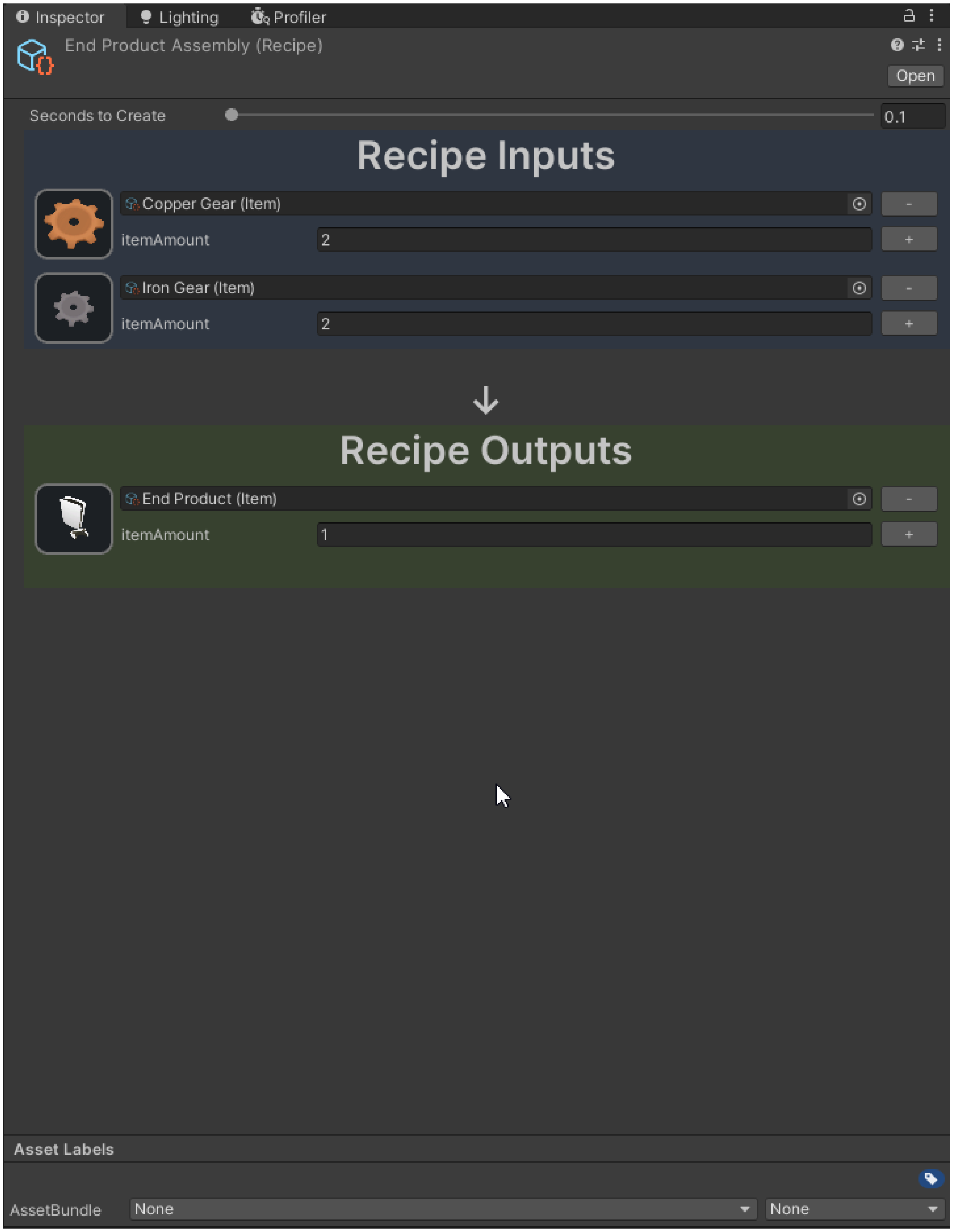
Beneath the preview windows are fields for item description, the max stack size for an item, and the color of the debug gizmo that can be shown on belts.

Recipes

Recipes define how items are transformed into other items. To create a recipe definition, right click in your "Project" window, then select Create -> Factory Framework -> Recipe.
Important: Recipe ScriptableObjects must be stored in a Resources folder to facilitate save/load functionality!



The recipe definition will appear in your current project folder. Selecting it will reveal the Recipe inspector:



End Product Assembly (Recipe)

Seconds to Create 0.1

Recipe Inputs

Copper Gear (Item)
itemAmount 2

Iron Gear (Item)
itemAmount 2

Recipe Outputs

End Product (Item)
itemAmount 1

Asset Labels

AssetBundle None None

At the top of the editor is a value to define the base processing time for the recipe to be completed. Beneath that is a list of input items and a list of output items. By using the + and - buttons, items can be added to the lists.

Buildings

A Building in Factory Framework is used to store, move, or process items. Each buildingFactory Framework includes a number of Building scripts for your convenience. A list of these scripts is included below. Buildings are extensible if you need to add your own. Use these buildings as a template for your needs. All buildings must have an array of input and output sockets. In most cases, this is 1 input and 1 output.
Important: Building Prefabs must be stored in a Resources folder to facilitate save/load functionality!

Producer

This script will supply a steady stream of an item, ideal for use with buildings that will gather resources from the environment. This is the starting point for all items. The amount produced per second is configurable through the inspector window.

Processor

This script will take item(s) as input for recipes and output the processed item(s). The valid recipes, number of inputs, and number of outputs can be configured through the inspector. It will attempt to match the incoming item with any of the valid recipes.

Storage

This script represents a simple storage system. The only configurable option is the max capacity. This capacity is in stacks of items. That is to say, if an item can stack up to 50, and the storage has 50 slots, it can hold 2500 of that item.

Merger

This script will take from all input sockets as evenly as possible, presenting them for output.

Splitter

This script will split its input into multiple different outputs as evenly as possible.

Socket

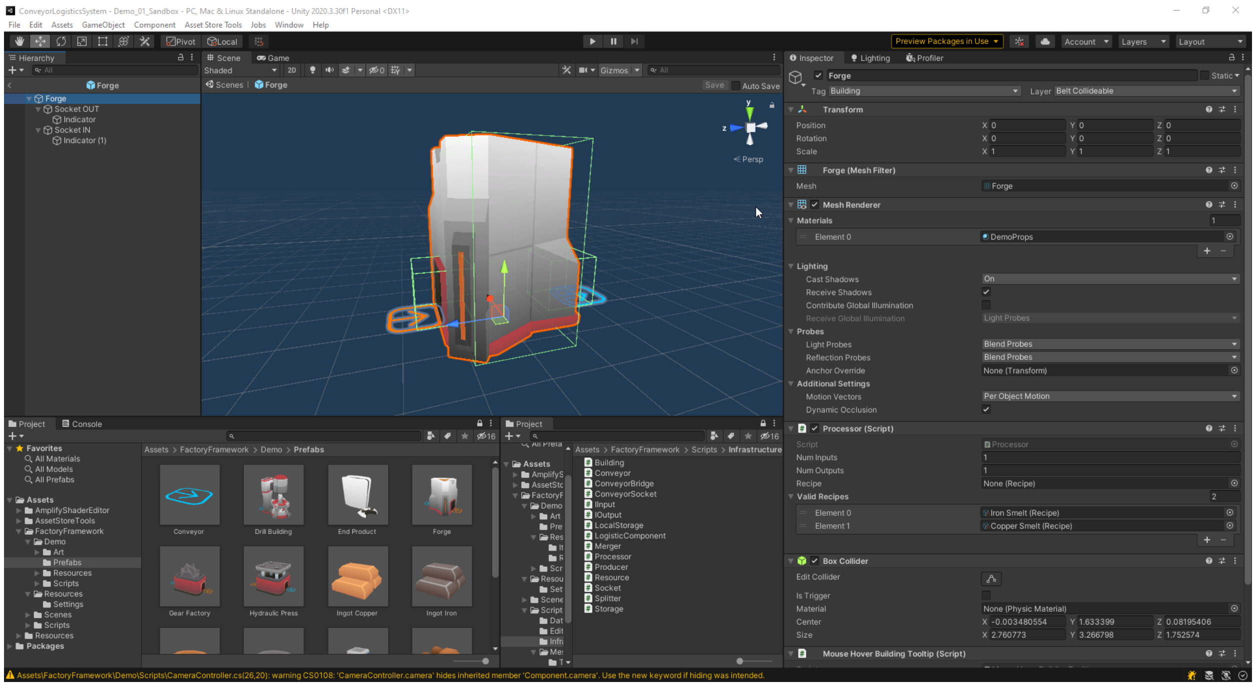
Sockets are what connect belts to other things. The two options in the demo scene show belts connecting to other belts, and belts connecting to or from buildings. The Socket facilitates checking if the *source* can GiveOutput() and the *sink* can TakeOutput(). If all conditions are satisfied, the socket moves the item from the source to the sink.

Sockets also have an optional indicator gameobject to provide feedback to players showing open sockets and their direction.

Look to the demo scene prefabs for an example.

Building Prefab Example

Please check the Demo prefabs for examples of all building types.
As an example, consider the Forge building:



Forge — Static
Tag: Building — Layer: Belt Collideable

Transform

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

Forge (Mesh Filter)

Mesh	Forge

Mesh Renderer

Materials	1
Element 0	DemoProps
Lighting	
Cast Shadows	On
Receive Shadows	✓
Contribute Global Illumination	
Receive Global Illumination	Light Probes
Probes	
Light Probes	Blend Probes
Reflection Probes	Blend Probes
Anchor Override	None (Transform)
Additional Settings	
Motion Vectors	Per Object Motion
Dynamic Occlusion	✓

Processor (Script)

Script	Processor
Num Inputs	1
Num Outputs	1
Recipe	None (Recipe)
Valid Recipes	2
Element 0	Iron Smelt (Recipe)
Element 1	Copper Smelt (Recipe)

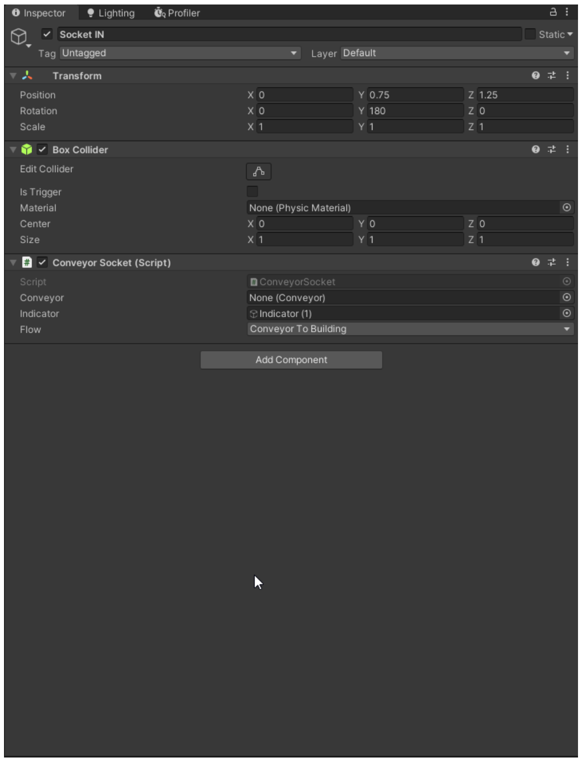
Box Collider

	X	Y	Z
Edit Collider			
Is Trigger			
Material	None (Physic Material)		
Center	-0.003480554	1.633399	0.08195406
Size	2.760773	3.266798	1.752574

Mouse Hover Building Tooltip (Script)

```
Assets\FactoryFramework\Demo\Scripts\CameraController.cs(26,20): warning CS0108: 'CameraController.camera' hides inherited member 'Component.camera'. Use the new keyword if hiding was intended.
```

The main gameobject has a Processor script set to accept two recipes, iron ore -> ingot and copper ore -> ingot. The building has two children which are both sockets- one for input, one for output

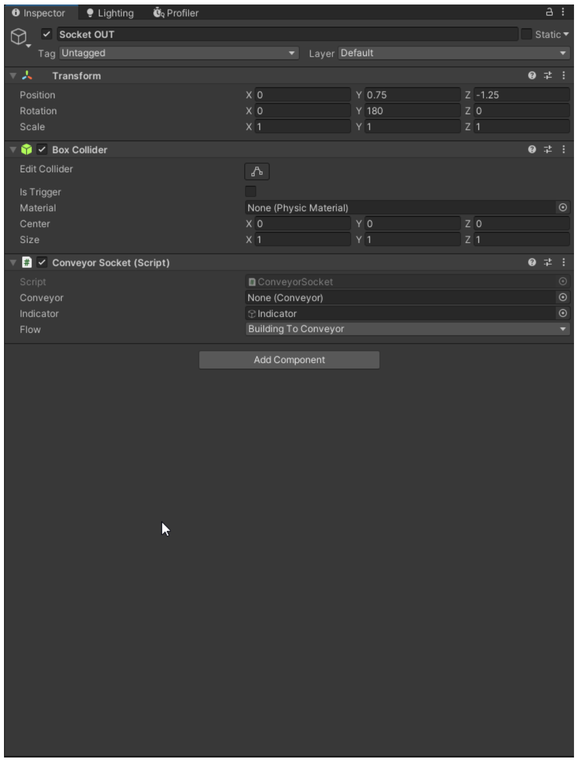


Socket IN
Tag Untagged Layer Default

Transform
Position X 0 Y 0.75 Z 1.25
Rotation X 0 Y 180 Z 0
Scale X 1 Y 1 Z 1

Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

Conveyor Socket (Script)
Script ConveyorSocket
Conveyor None (Conveyor)
Indicator Indicator (1)
Flow Conveyor To Building



Socket OUT
Tag Untagged Layer Default

Transform
Position X 0 Y 0.75 Z -1.25
Rotation X 0 Y 180 Z 0
Scale X 1 Y 1 Z 1

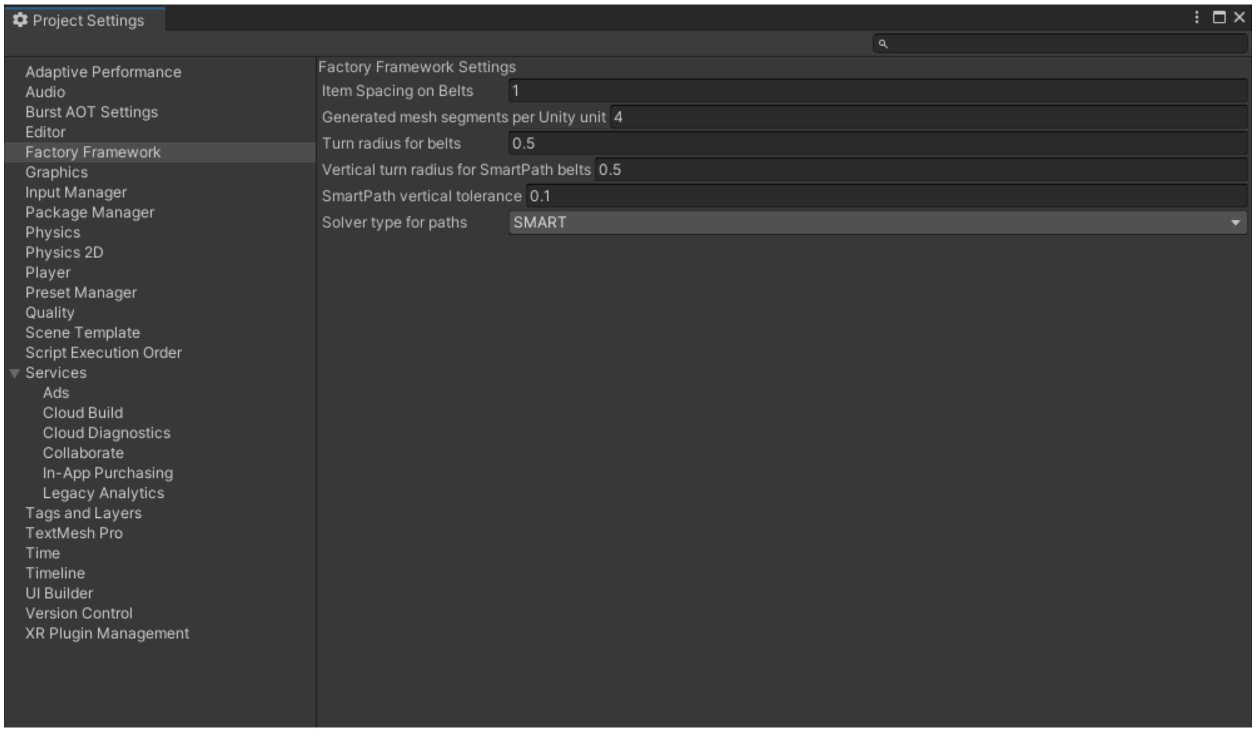
Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1

Conveyor Socket (Script)
Script ConveyorSocket
Conveyor None (Conveyor)
Indicator Indicator
Flow Building To Conveyor

.

Getting Started

After importing Factory Framework, select Edit -> Project Settings from Unity Editor's top navigation bar. A new setting page has been added titled "Factory Framework."

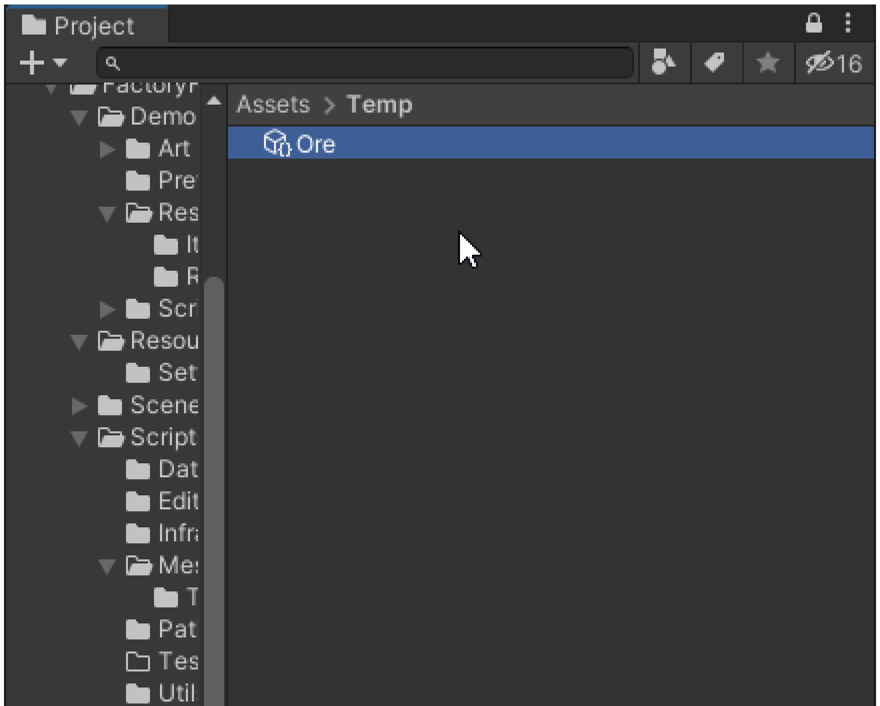


Factory Framework Settings

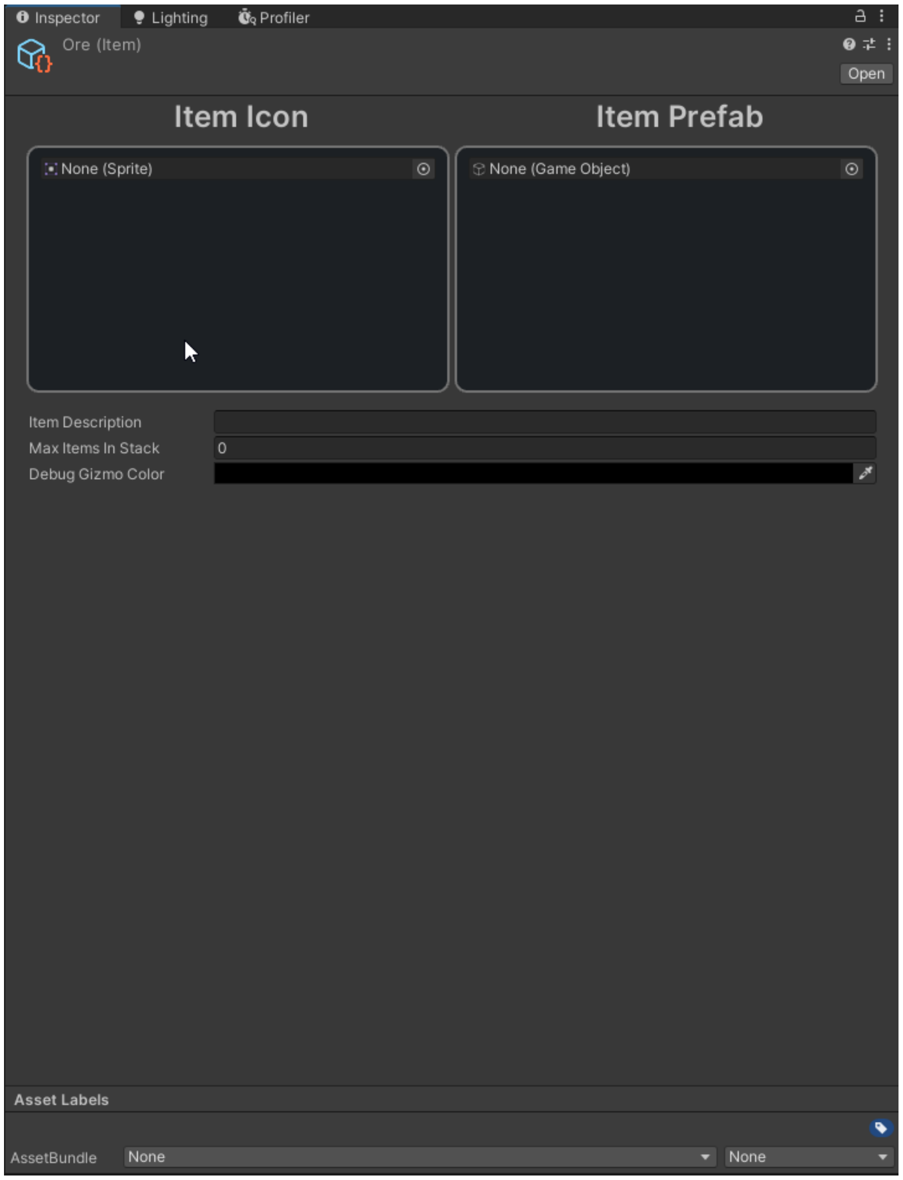
Item Spacing on Belts	1
Generated mesh segments per Unity unit	4
Turn radius for belts	0.5
Vertical turn radius for SmartPath belts	0.5
SmartPath vertical tolerance	0.1
Solver type for paths	SMART

On this page you can configure various options related to Factory Framework. A detailed breakdown of these options can be found in the "Belts" section of this guide. Configure to your liking, then close the settings page.

In the project window, right click and find Create -> Factory Framework -> Item. This will create a new ScriptableObject instance in the currently open folder. Rename this item to "Ore"



Select this item to reveal the Item inspector:



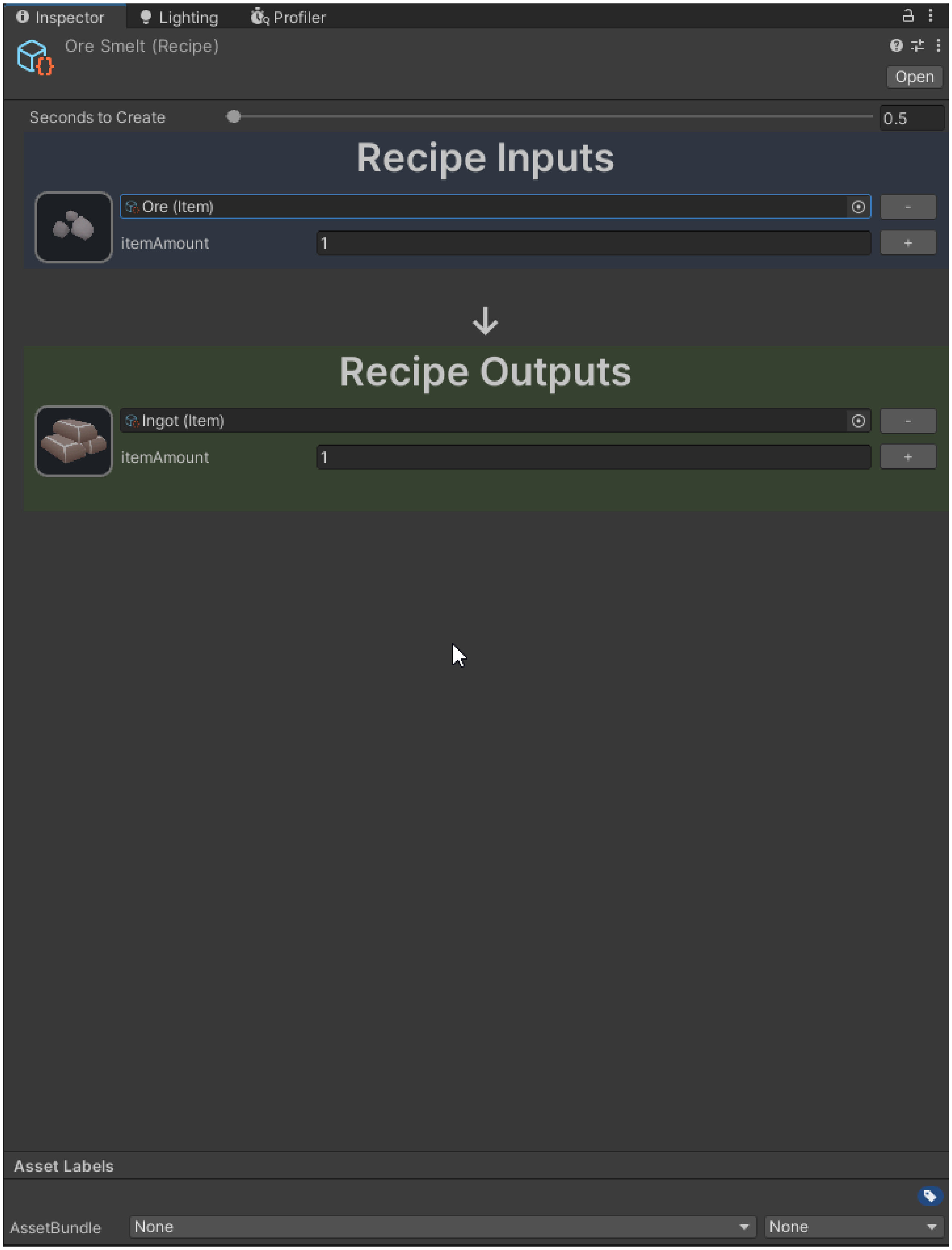
Item Description
Max Items In Stack 0
Debug Gizmo Color

For the icon select icon_raw_a.png and for the prefab select “Raw Mineral Iron”
Set the max items in stack value to 50.

Repeat this process to create one more item named Ingot. Use icon_a_ingot and “Ingot Iron” for this item.

Now that we have two items, we will define a recipe. In the same way as creating the items, right click and find Create -> Factory Framework -> Data -> Recipe. Name this recipe “Ore Smelt”

After creation, select the recipe and use the inspector to configure it as shown below:



Ore Smelt (Recipe)

Seconds to Create 0.5

Recipe Inputs

Ore (Item)

itemAmount 1

Recipe Outputs

Ingot (Item)

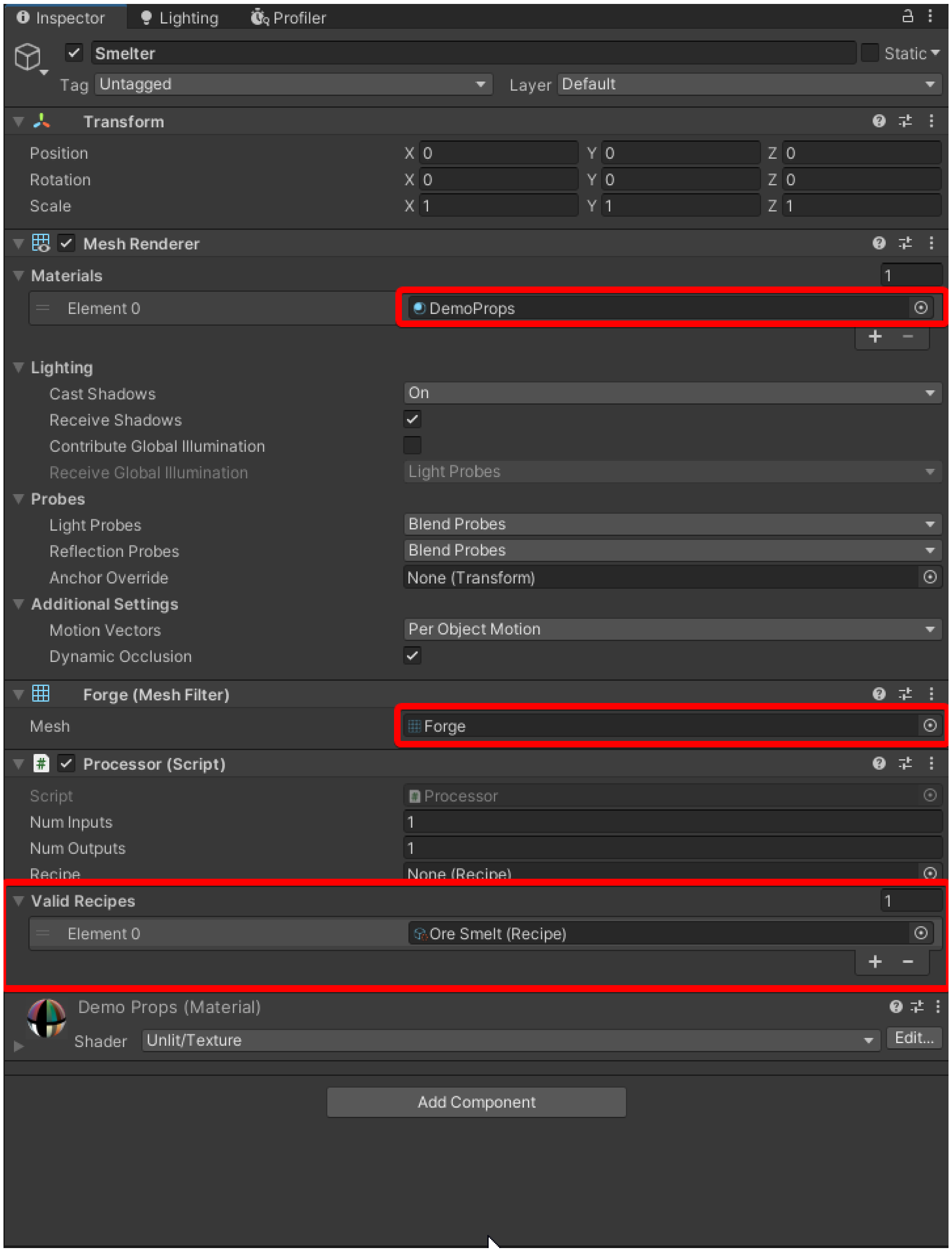
itemAmount 1

Asset Labels

AssetBundle None None

We now have two items defined as well as a recipe to convert from one to the other. Now, we will create a building to process them. Once again, right click, but this time select Create -> Prefab. Name this prefab “Smelter” and double click it to open the prefab inspector.

To the base game object, add a Mesh Filter, a Mesh Renderer, and a Processor. Configure them as shown below, using standard Unity Editor techniques:



Inspector | Lighting | Profiler

Smelter — Static

Tag: Untagged — Layer: Default

Transform

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

Mesh Renderer

Materials: 1
Element 0: DemoProps

Lighting
Cast Shadows: On
Receive Shadows: ✓
Contribute Global Illumination: ☐
Receive Global Illumination: Light Probes

Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)

Additional Settings
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓

Forge (Mesh Filter)

Mesh: Forge

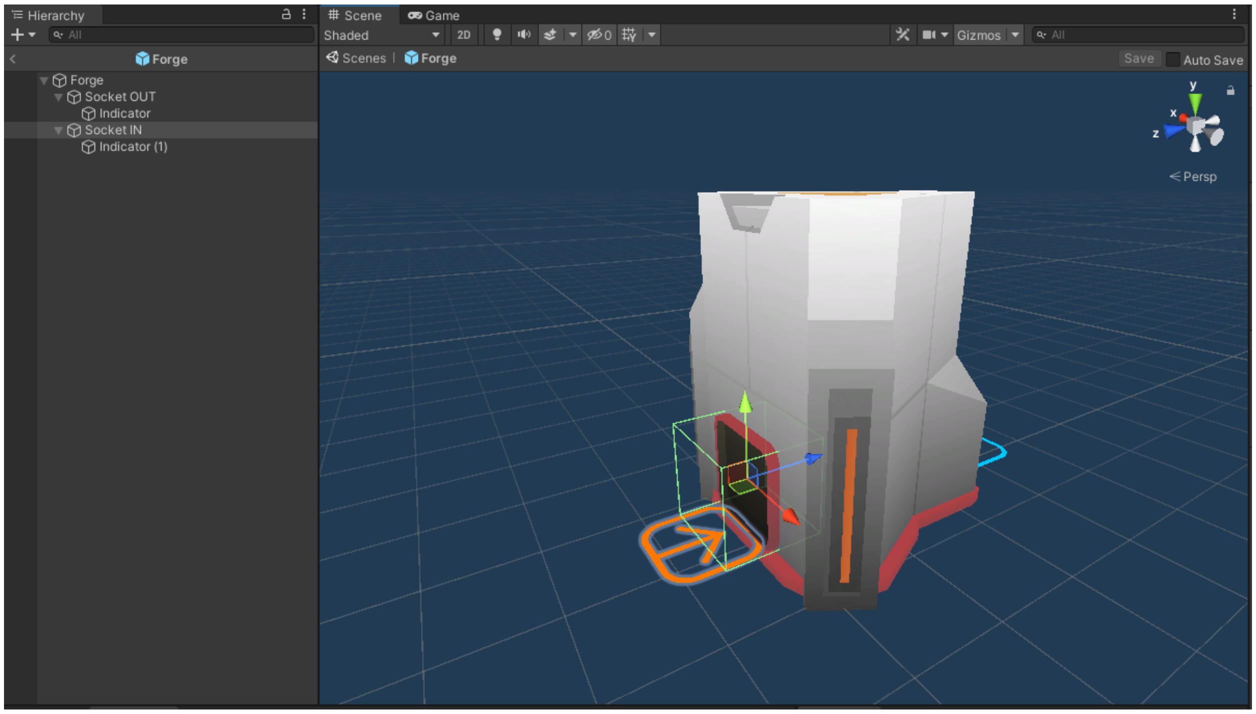
Processor (Script)

Script: Processor
Num Inputs: 1
Num Outputs: 1
Recipe: None (Recipe)
Valid Recipes: 1
Element 0: Ore Smelt (Recipe)

Demo Props (Material)
Shader: Unlit/Texture — Edit...

Add Component

Now we need to give our building the ability to import and export items, so we need to add Sockets. At this point in the guide, it would be best to open the “Forge” prefab building included in the Demo assets to understand how to set this up properly.



Feel free to copy and paste the Socket OUT and Socket IN children from the included prefab into your newly created prefab. If creating the sockets manually, pay special attention to the box collider sizing and the ConveyorSocket “Flow” setting.

At this point, you have created a building, two items, and a recipe. When the “Ore” item is input into this building, it will output the “Ingot” item.

Save and Load Support

With the release of Factory Framework 1.2.0, we've added save and load functionality. Essentially, all instances of the "Building" class, as well as all conveyors, will be serialized when calling the SerializeManager's "save" and "load" method. For this to work, there are a few requirements-

1. The scene must have the "Factory Framework Core" prefab loaded
2. Building prefabs, Item scriptable objects, and Recipe scriptable objects must be stored in a resources folder.
3. Buildings must have their input and output sockets assigned to their inputSockets and outputSockets arrays via the inspector
4. Building prefabs must have only one "Building" script (Producer, Processor, Storage) on their root object.

Ideally, by following these requirements, saving and loading should *just work*.

Extending the Save and Load System

The save and load system is fairly simple, and thus should be fairly simple for the user to extend if more functionality is needed. Both of these functions are handled by the SerializeManager.cs script. Saving occurs through the following process:

1. All "Building" script instances are found
2. "Building" instances are converted into save data, simple structs that hold important data to the building type. Extremely important is the "assetPath" data- this is used to instantiate the building prefabs when loading
3. All "Conveyor" instances are found
4. "Conveyor instances are converted into conveyor save data, which includes the GUIDs of the buildings they are connected to
5. All of the gathered data is stored in a "FactorySaveData" object, which is then serialized and written to the provided filepath

Loading occurs through the following process:

1. The file is opened, read, and closed
2. The FactorySaveData object is deserialized
3. Each list of buildings is iterated over and reinstantiated by loading the prefab stored in "assetPath." At this time, the buildings are also added into a GUID -> LogisticComponent lookup table.
4. The conveyor belts are reinstantiated and have their meshes regenerated
5. The conveyor belts are reconnected to buildings (and other belts) by accessing the lookup table to find references to the appropriate buildings

If you are to make your own building type, you must add a few things to have it serialize properly.

1. A new subclass of “BuildingSaveData” that stores information relevant to your building
2. A new array in FactorySaveData of the building data class you created in step 1
3. Expand the “Save” method to process your new building type
4. Expand the “Load” method to process your new building type

Questions?

If you have further questions, or have a project you want to show off, we are always accessible at our discord server or our email support@ctrlaltbees.com