ULTIMATE SOCKET SYSTEM

Documentation

Video Guides

Quick Start: https://youtu.be/iiktvp8drl4
Item Placement: https://youtu.be/3zP6H5kpmDo
Highlighters: https://youtu.be/k6HT3KtX8Fk
Placement Criteria: https://youtu.be/aYGq16OEn_4
Stacking: https://youtu.be/jZnEz7Gp12o
Preview Objects: https://youtu.be/_eqlLJ1Ya_4

Support Email: support@hypertonicgames.com
Discord: https://discord.gg/ueysMDxNwW

Table of Contents

Overview

The Ultimate Socket System was developed to provide a fantastic user experience when it comes to placing items into sockets. It streamlines configuration of placeable items and sockets to allow for fast and fluid development in your apps. The asset is primarily focused on XR applications, however, it can be used for non XR applications too.

Compatibility

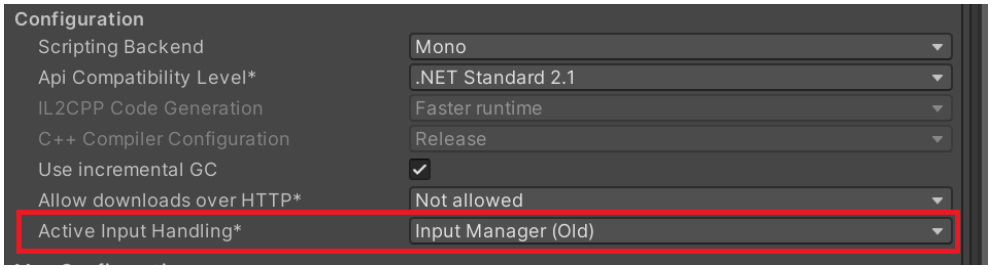
The asset is compatible with Unity **2021** LTS and later.

Demo Scene compatibility

Input Handling

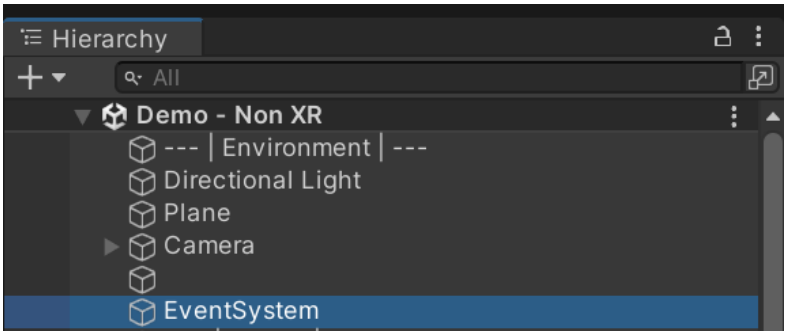
If you are still using the old input input handling system: **Input Manager (old)** then you'll need to replace the input handling on the EventSystem game object in the Hierachy in the demo scenes.

You can check your Active Input Handling settings by finding it in Player Settings.

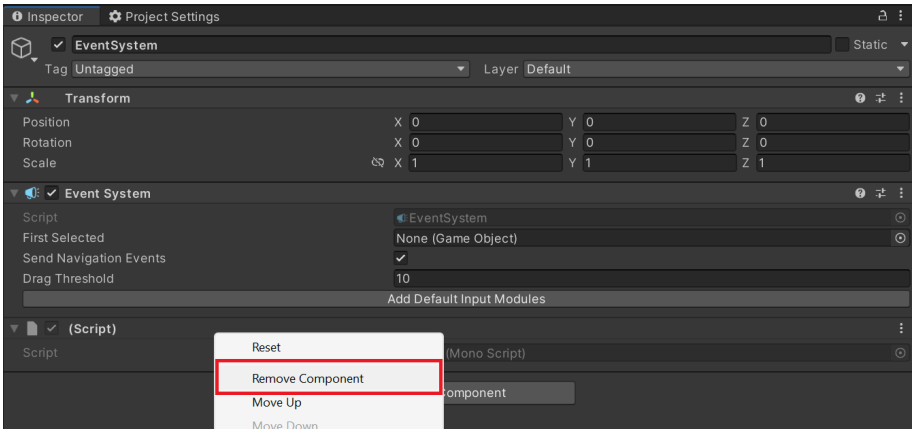


To get the event system to work with the old Input Manager follow the following steps:

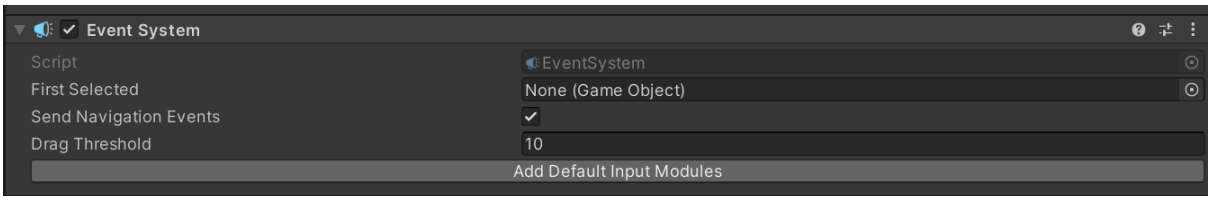
1. Locate the EventSystem gameobject



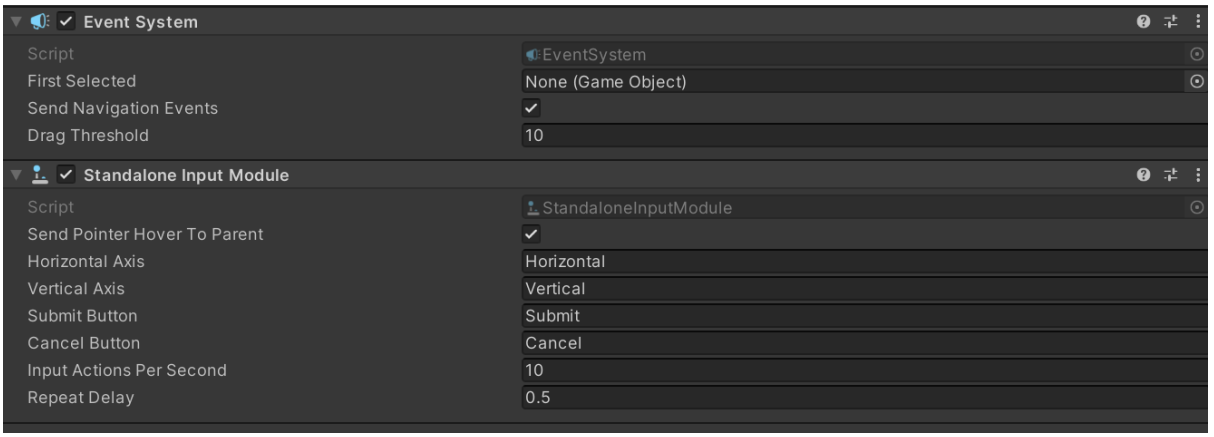
2. Remove the missing script



3. Click the “Add Default Input Modules” button on the Event System component



4. Your Event System should now have the Standalone Input Module attached



Getting Started

Installation

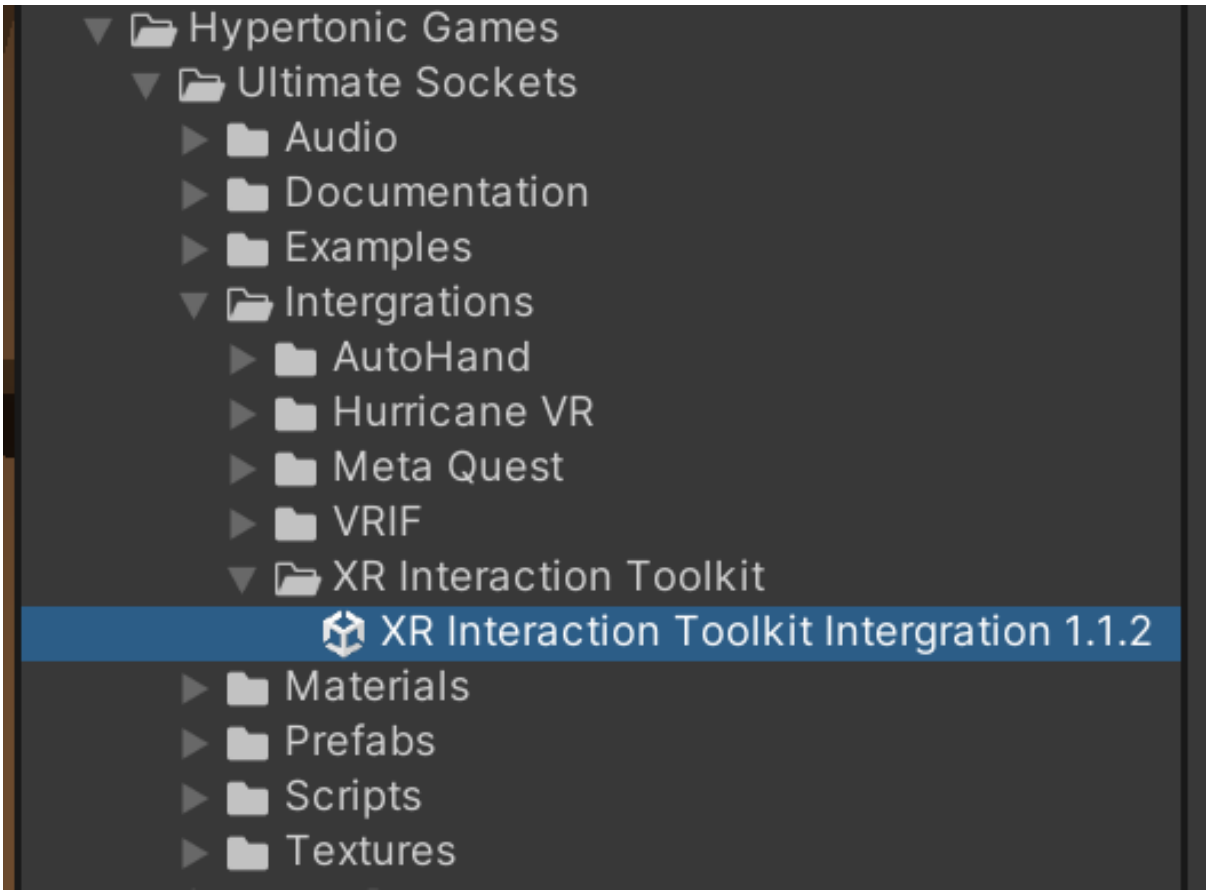
1. Open your Unity project.
2. Locate the asset from the Package Manager window.
3. Click the Import button.
4. If it’s your first time using the package it’s recommended to install the examples folder too.

Quick Start

Prerequisites:

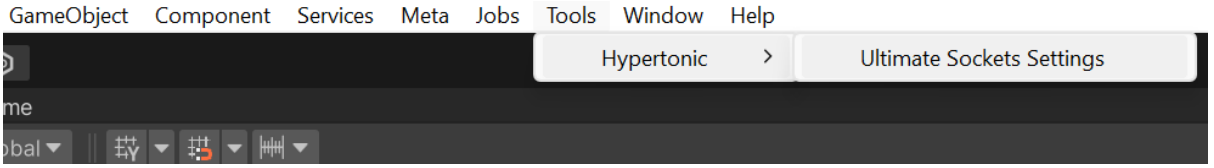
- You have a Unity project with your XR framework of choice setup (E.G Unity's XR Interaction toolkit, Meta XR All-in-One SDK, etc)
- You have an item you can grab and release using the installed XR Framework.
- You have an item that you'd like to use as a socket for the placeable item.
- Either the item or socket has a rigidbody attached.

1. This quick start guide will use Unity's XR Interaction Toolkit Package. If you're using a different XR Framework you can still follow along, just use the integration script required for your framework of choice.
2. First, in the project window navigate to Hypertonic Games > Ultimate Sockets > Integrations. Then double-click your integration of choice. For this guide, we will select XR Interaction Toolkit.

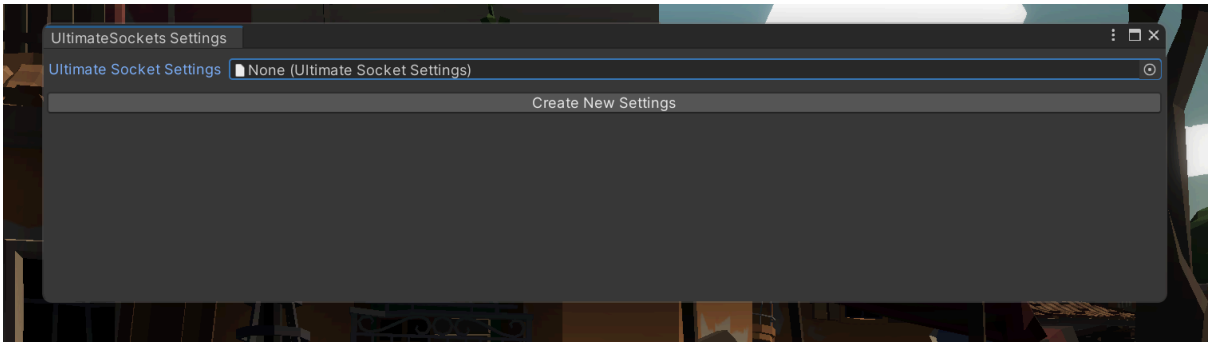


3. The XR Interaction Toolkit comes with some sample assets used in the demo scene. The import file is the XRITGrabbableItem.cs script. This is used to bridge the Socket System to a grabbable item.

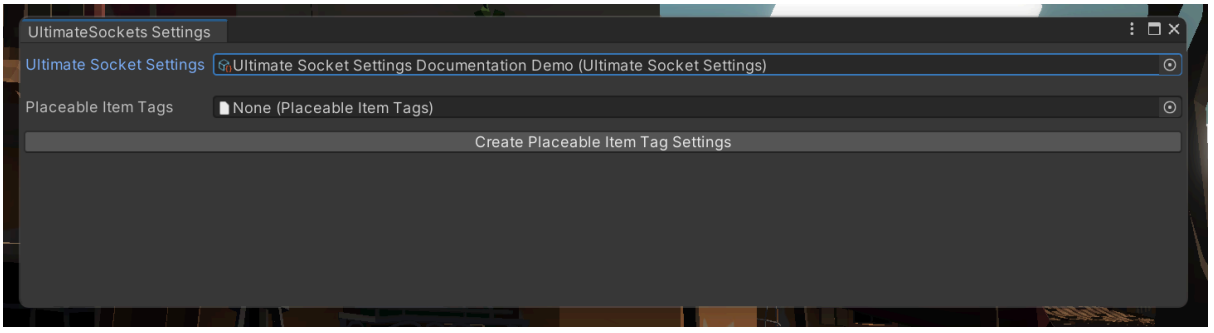
4. From the context menu navigate to Tools > Hypertonic > Ultimate Sockets Settings



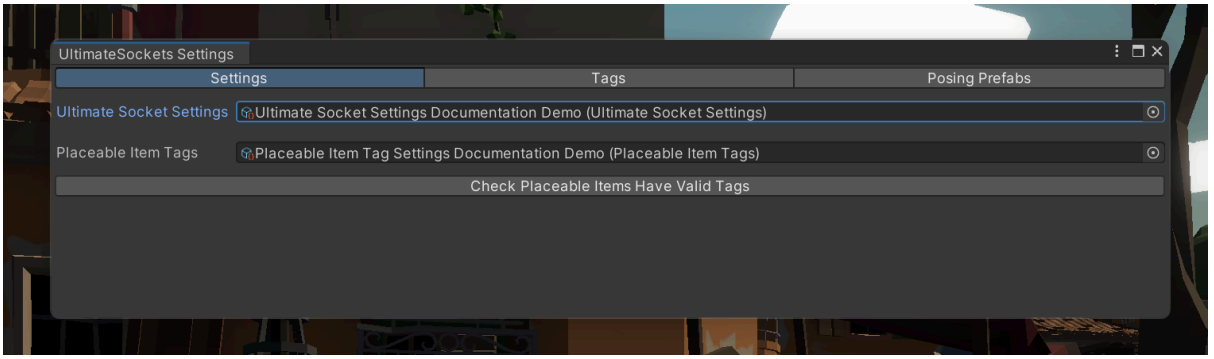
5. Click the Create New Settings button



6. Select a location for the settings configuration file to be saved within your project.
7. Now Click the Create Placeable Item Tag Settings button and select a location for the settings to be saved.

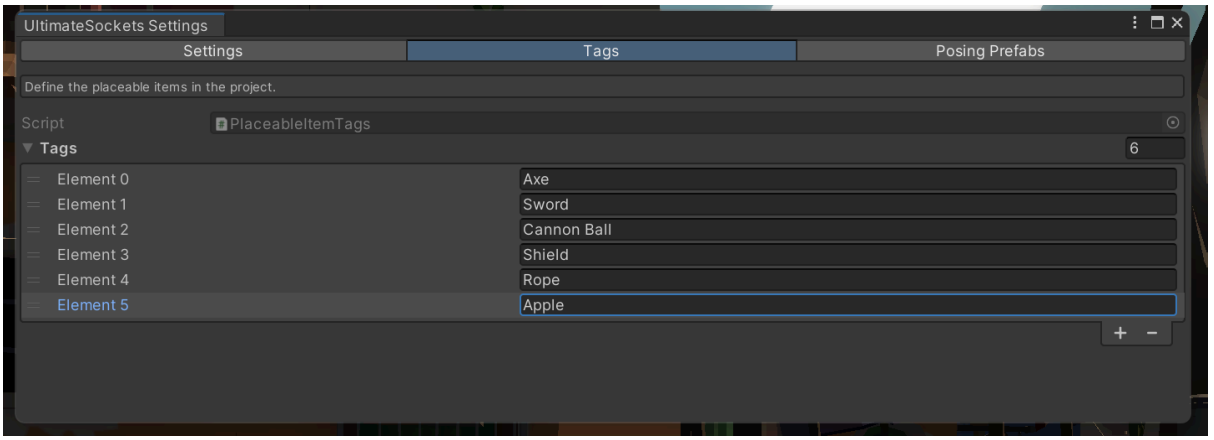


8. Now you have created the Ultimate Socket Settings and the Placeable Item Tag settings the other tabs will display.

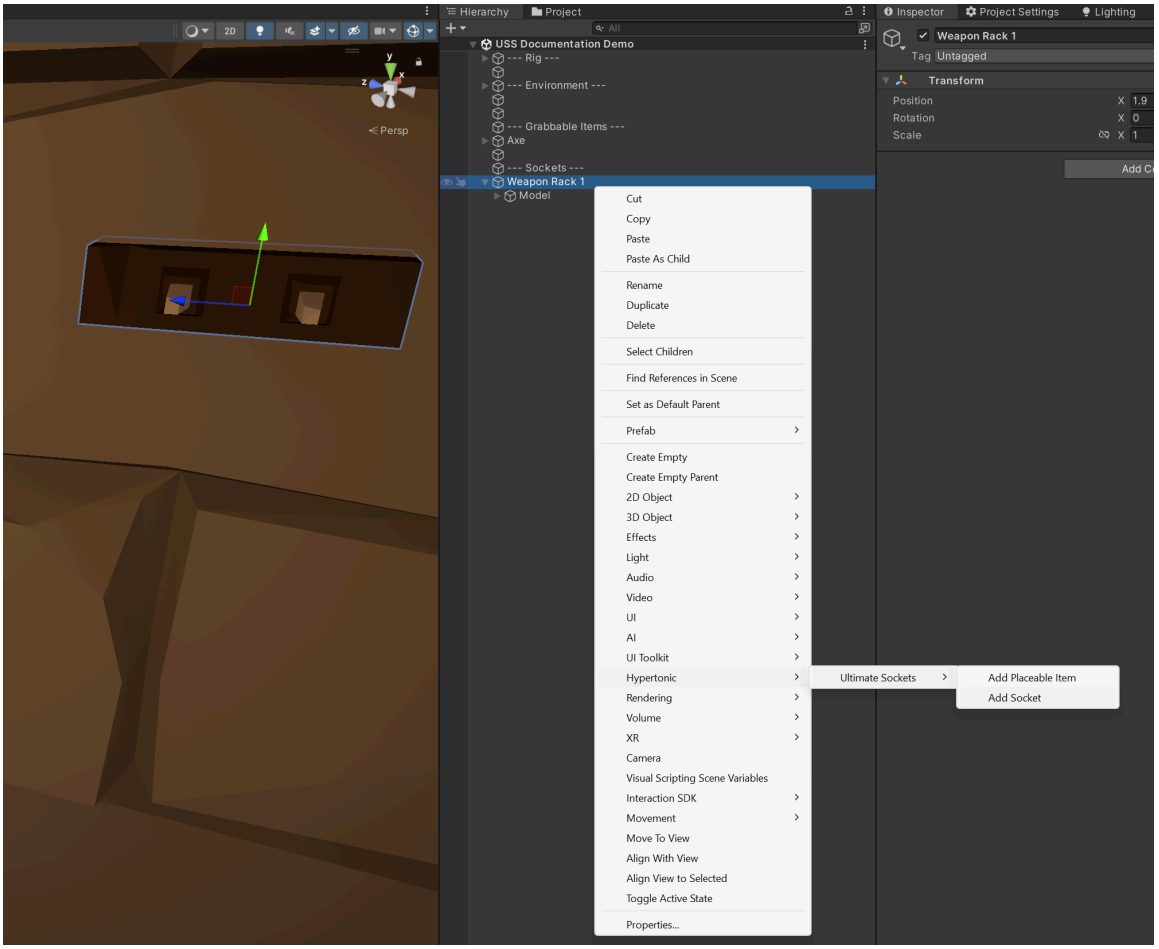


9. Click on the Tags tab and populate the list with the names of any items that will be placeable. The Posing Prefabs tab is out of scope for the quick state guide but is

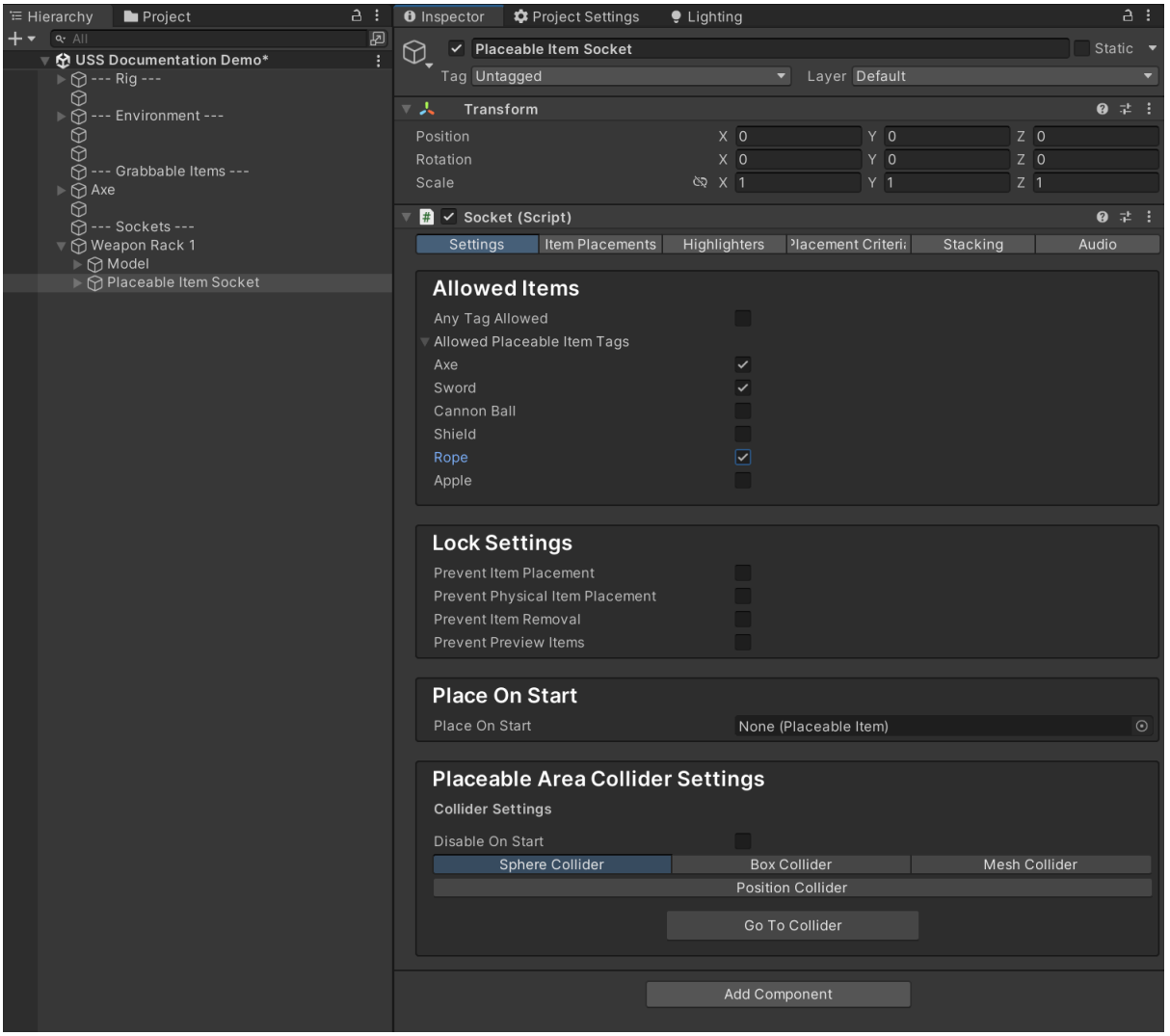
documented in the Posing Prefabs section.



10. You're now ready to create a socket. Select the gameobject which you'd like to add socket functionality to. Then right click it and use the context menu to select: Hypertonic > Ultimate Sockets > Add Socket

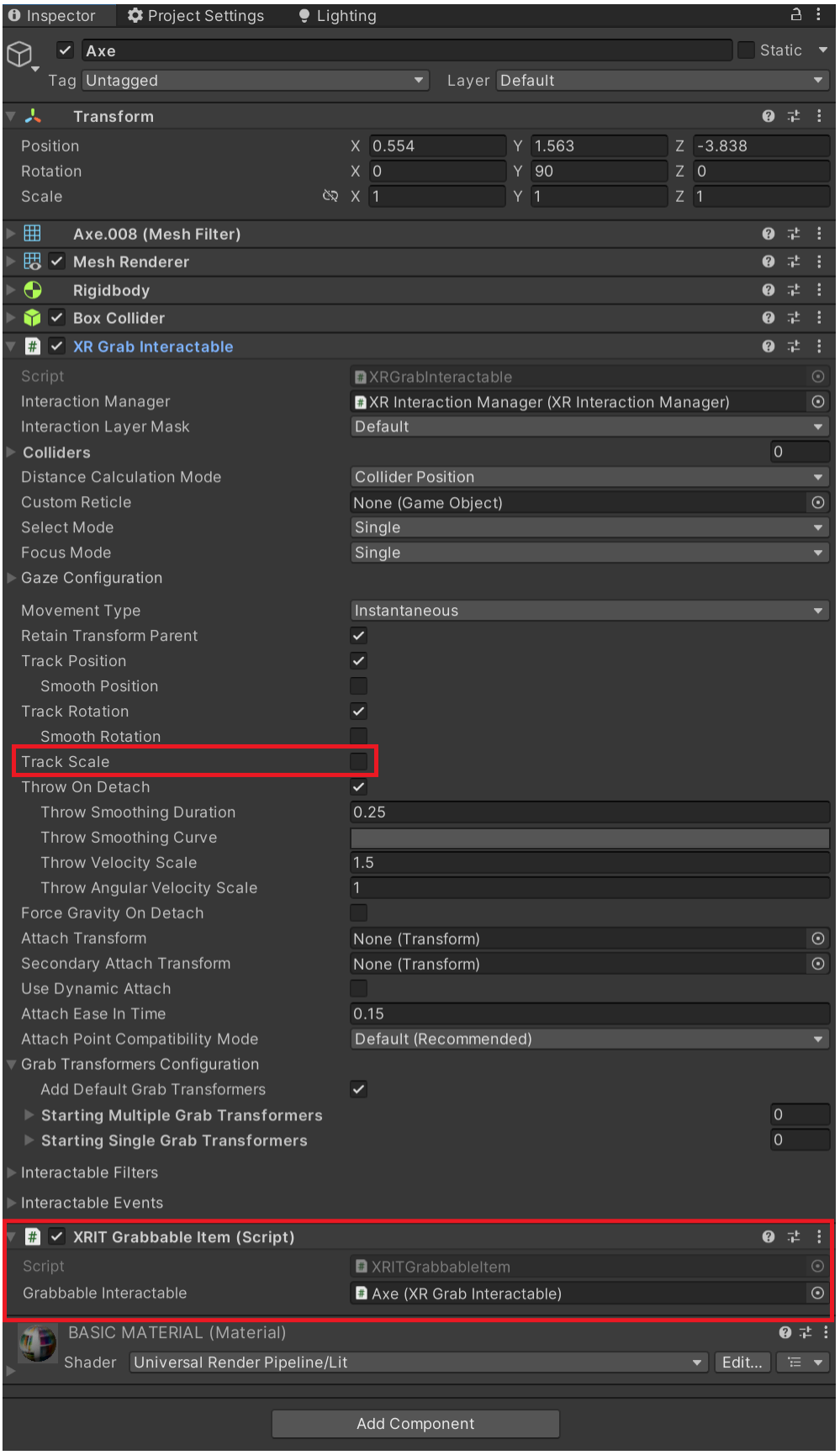


11. A new Placeable Item Socket gameobject has been created as a child of the selected object. From the settings menu choose the items this socket can accept.

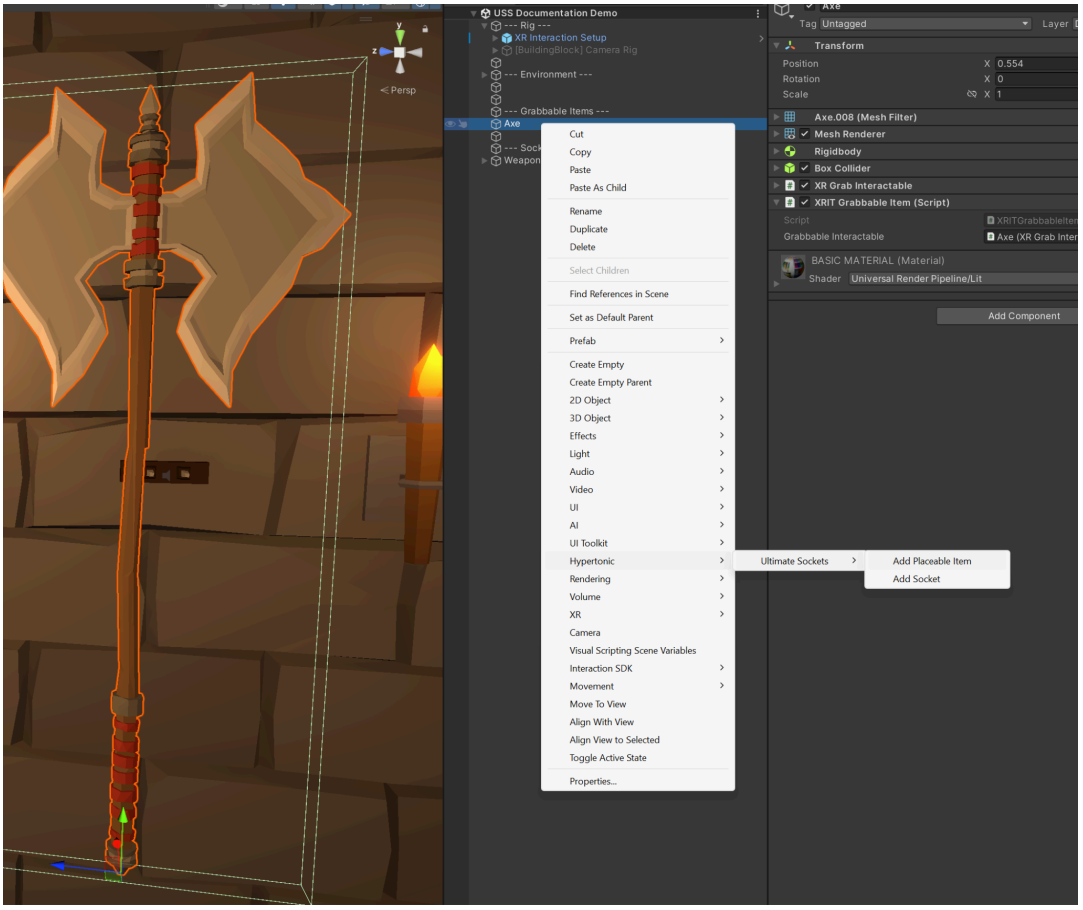


12. Now select the grabbable item you want to become placeable. Add the integration script that you imported via the integration package. As this example is using the XR Interaction Toolkit, the grabbable item needs the XRIT Grabbable Item component added. Then add a reference to the XR Grab Interactable. If you are going to be placing items at different scales to the items original scale you should disable Track

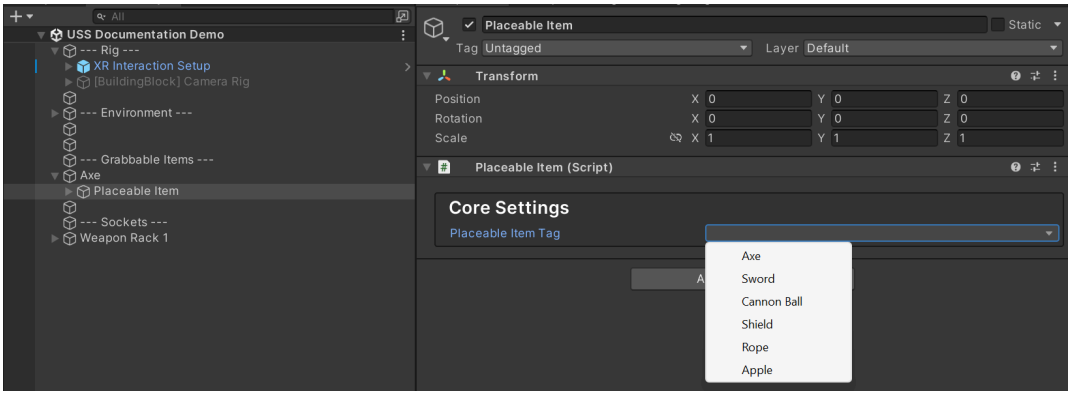
Scale as it can have undesired effects.



13. Right-click on your grabbable item and select: Hypertonic > Ultimate Sockets > Add Placeable Item from the context menu.

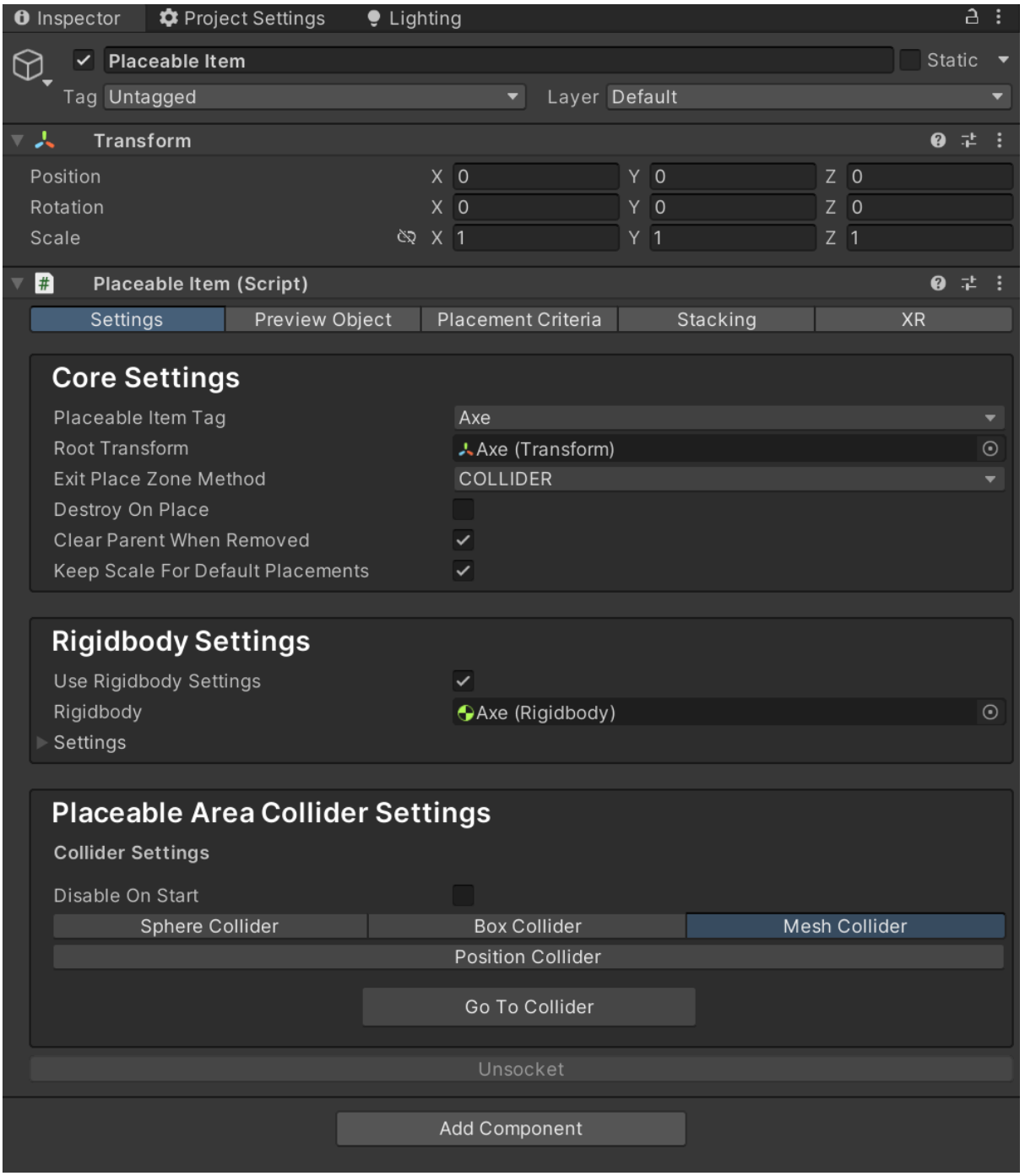


14. Once added you'll need to define the items tag. Using the list you setup earlier.



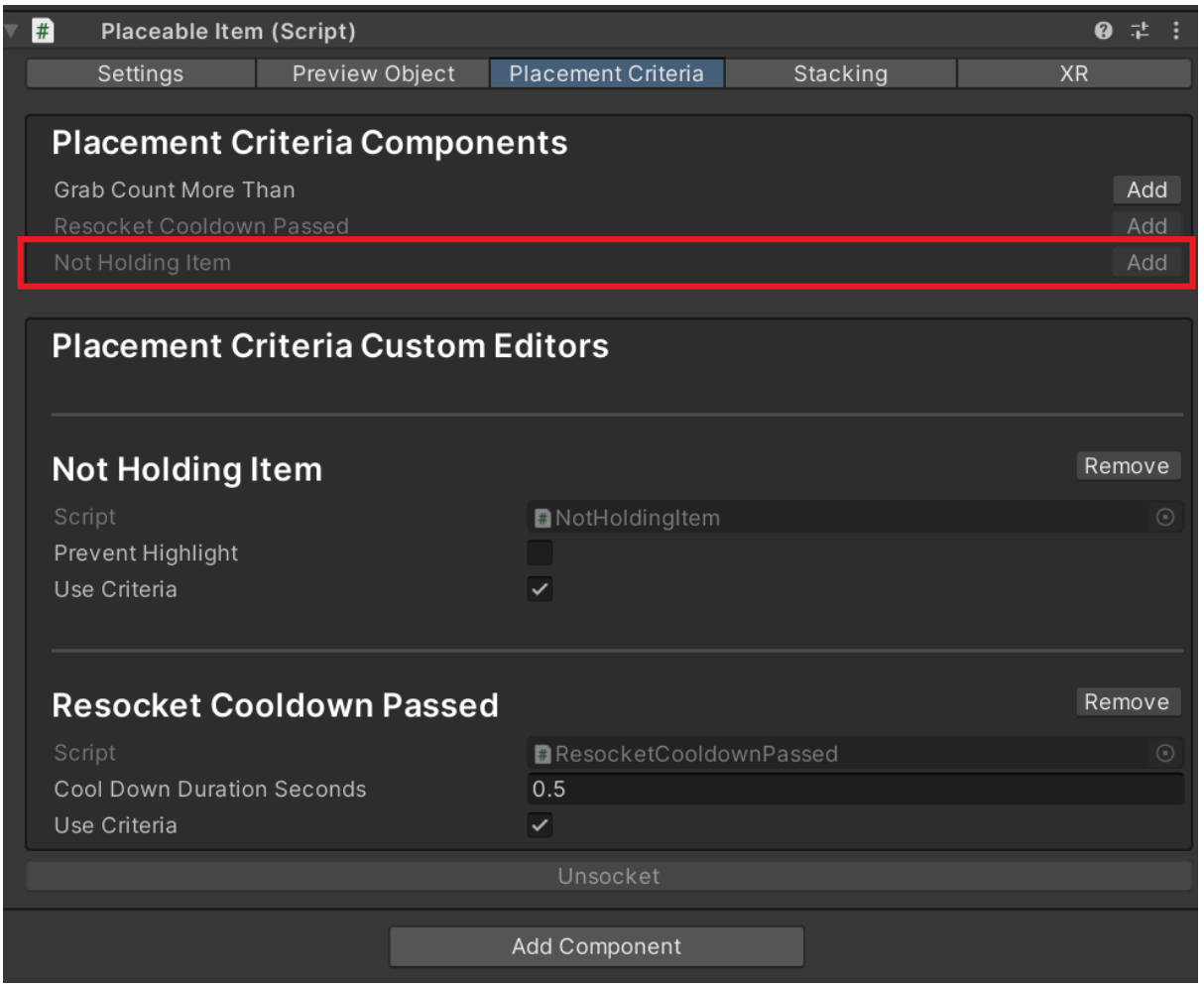
15. You should now see the custom editor for the Placeable Item. On the main settings page, you should see the Root Transform and Rigidbody have been set

automatically.



16. If you navigate to the XR tab you should also notice that the IGrabbable reference has been set to the Root gameobject where the XRIT Grabbable Item integrations script is located.
17. If any of these references have not automatically been set as they are not located on the parent object you'll need to set these references manually.
18. Finally, It's recommended to navigate to the Placement Criteria tab on either the Socket or the Placeable Item and add the Not Holding Item criteria. This prevents the

item from being placed until the user releases the object (when in placeable range).

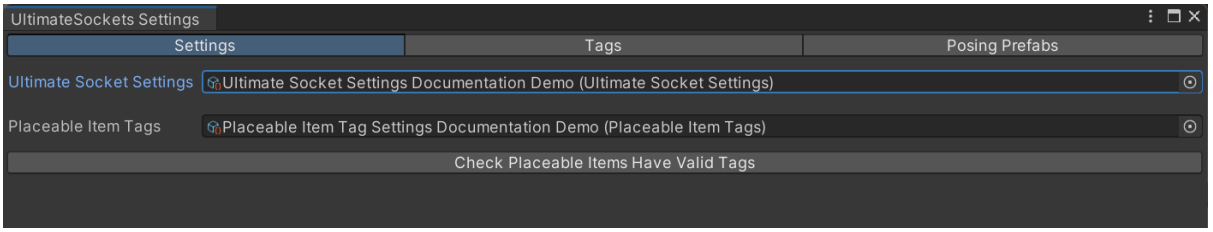


General Settings

Ultimate Socket Settings Window

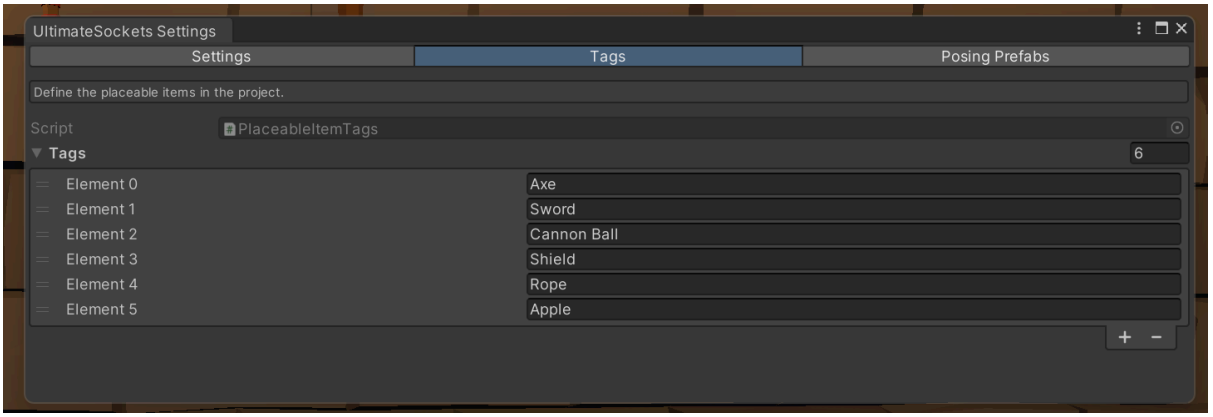
Open the settings window by navigating to Tools > Hypertonic > Ultimate Socket Settings

Settings



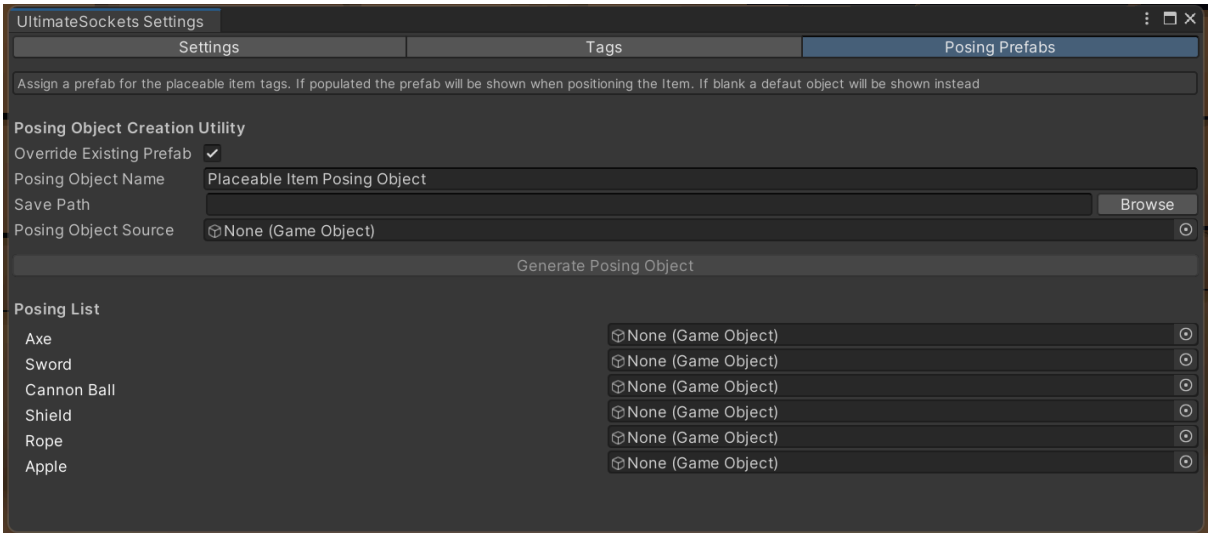
This is where you assign the settings you want to drive all socket-related behaviour. You can swap between different settings configurations if needed. If you don't have any settings assigned a button will display to automatically create them for you.

Tags



The Tags tab is where you define all the types of different placeable items in your app.

Posing Prefabs



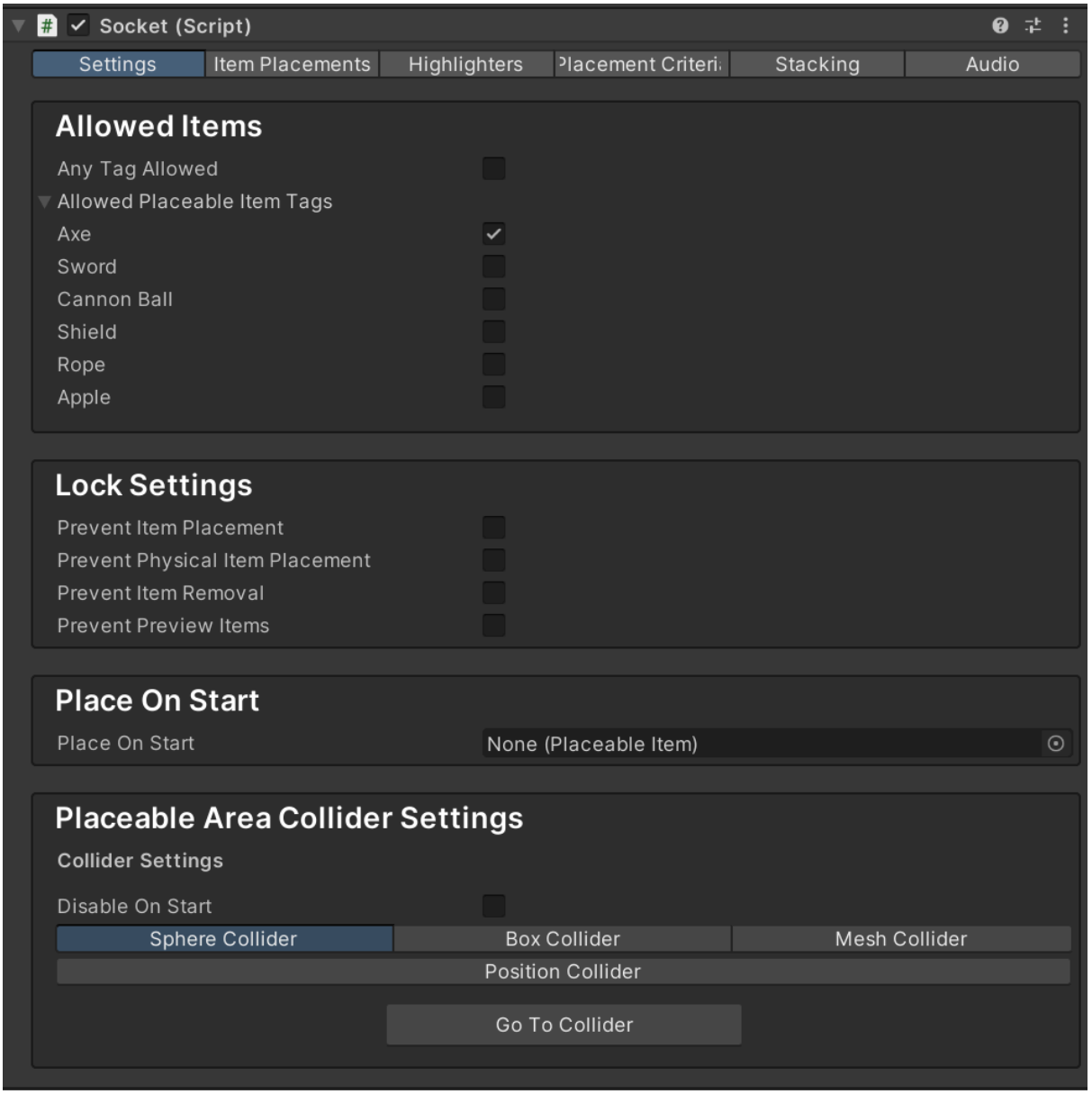
Posing Prefabs are the objects displayed when you are configuring the placement of an item into a socket. As these objects are purely for posing the item only the visuals are required so it's recommended no scripts are attached to the posing prefabs.

If you assign the prefab the tag is for (E.G Axe) then any scripts attached to that object will fire run the Unity lifetime hooks such as Awake, OnEnable, and Start. This may lead to undesirable behaviour.

To avoid this there is a utility tool that will essentially clone any object and recursively strip off any non visual components, handling any component dependencies you may have.

To use the utility too, assign your posing prefab a name, provide a location for the prefab. (To stay organised it's advised to save them under a "Posing Prefabs" folder). Then drag in a reference to the object you want to clone. Finally, hit Generate Posing Object and drop the reference into the field that corresponds to the item Tag.

Sockets



Allowed Items

Allowed Tags define which items can be placed into the socket.

To have more fine-grained control to determine if an instance of an item type is placeable, then you can use the Placement Criteria feature (see below). For example, if you had a socket that only accepted axes with 100% durability, then you'd need to set the allowed tags to be set to Axe. Then in the Placement Criteria section check the durability.

Lock Settings

Prevent Item Placement - Doesn't allow any items to be placed.

Prevent Physical Item Placement - Only allows placement programmatically.
Prevent Item Removal - Doesn't allow an item to be removed from the socket.
Prevent Preview Items - Prevent Preview Items being displayed (If activated on the placeable Item).

Place On Start

Place On Start - Places the item into the socket when the Start function on the socket is called.

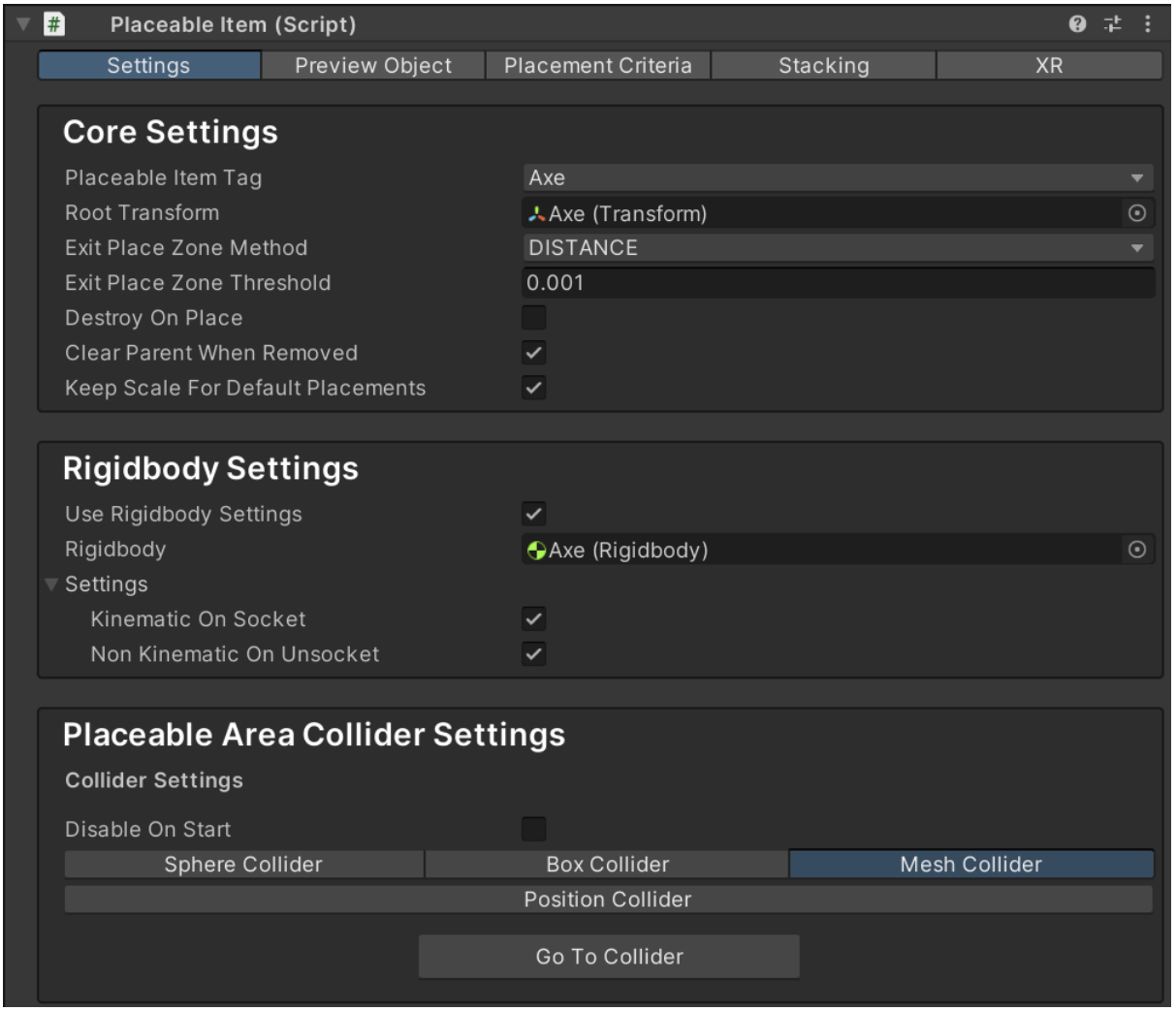
Placeable Area Collider Settings

This collider is responsible for detecting when a placeable item is within placeable range. If a placeable item comes within range and placement criteria are met, it'll be placed.

Disable On Start - Disables the placement detection collider when the Start function is called
Collider Type - Choose which type of collider you wish for the socket to use to detect the placeable item.

A future enhancement to have a custom editor to display the collider properties within this section is planned. Until then, you'll need to edit the collider size on the actual collider. To locate the collider just click the "Go To Collider" button.

Placeable Item



Core Settings

Placeable Item Tag - Define the tag for the placeable Item

Root Transform - The root game object that holds all of the placeable item functionality. The Root Transform is the game object that is actually placed into the socket. When a placeable item is created it'll assume the parent is the root transform, if it's not you'll need to set this to the correct root object.

Exit Place Zone Method (Collider is recommended)

- **Collider** - When an item leaved the sockets place detection collider
- **Collider Threshold** - When an item is further away from the socket then the radius of the place collider + the Exit Place Zone Threshold
- **Distance** - When an item has moved a certain distance away from the socket.

Destroy On Place - Destroys the placeable item's root transform (and nested objects) when placed.

Clear Parent When Removed - Sets the parent to null when removed. Most XR Frameworks automatically set this to the hand anyway.

Keep Scale For Default Placements - When true, if the item is placed and is using the socket's default placement config as it's not had a specific one setup, it'll keep it's scale.

Rigidbody Settings

Use Rigidbody Settings - When active the placeable item will interact with the rigidbody. E.G making it kinematic when placed

Kinematic On Socket - Makes the rigid body assigned kinematic when socketed.

Non-Kinematic On Unsocket - Makes the rigid-body non-kinematic when removed from a socket.

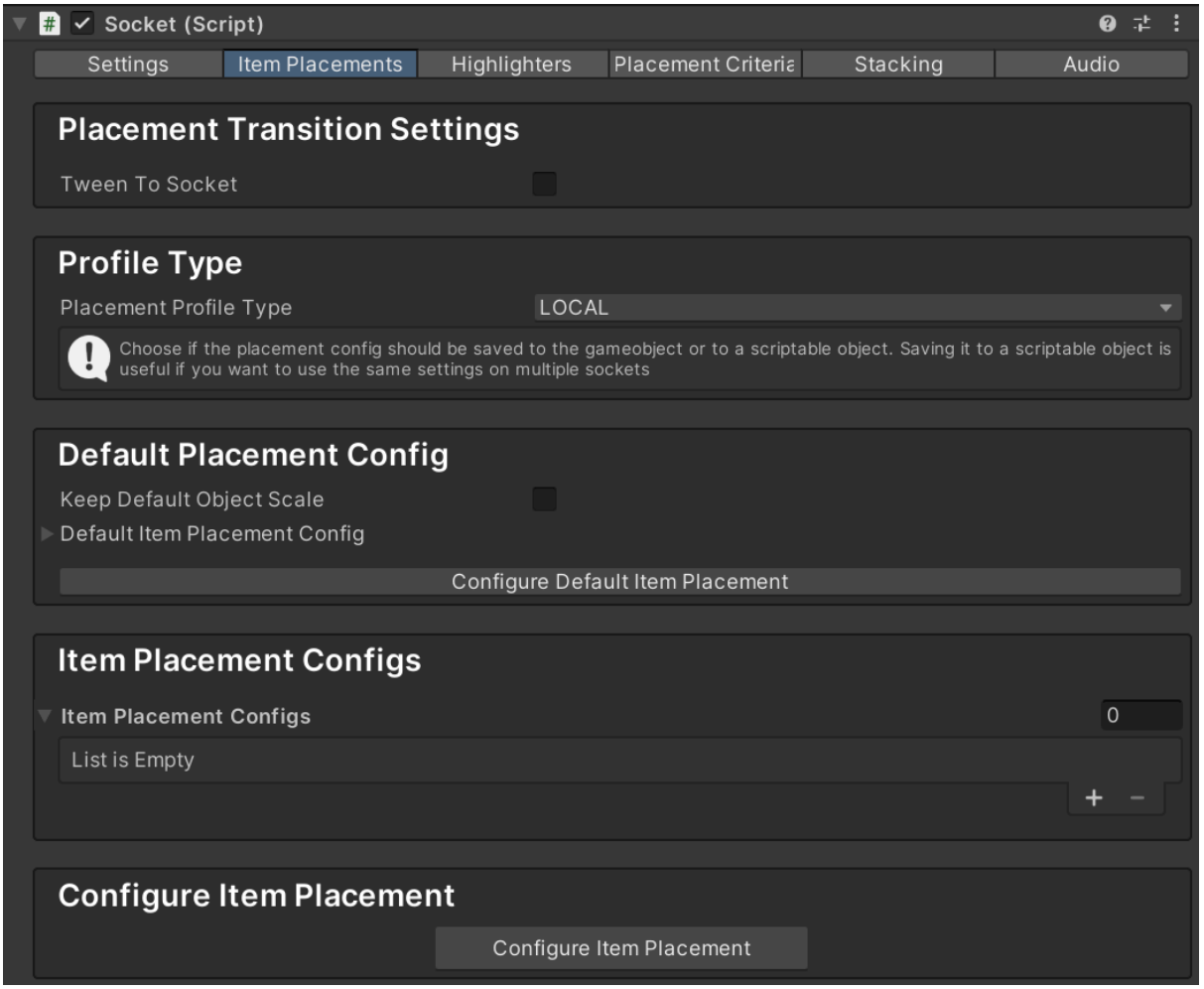
Placeable Area Collider Settings

This collider is responsible for triggering the socket's detection collider. When this collider enters the socket's placeable area collider, it'll trigger the socketing checks.

Disable On Start - Disables the placement detection collider when the Start function is called
Collider Type - Choose which type of collider you wish for the placeable item to use to trigger the socket's detection collider.

A future enhancement to have a custom editor to display the collider properties within this section is planned. Until then, you'll need to edit the collider size on the actual collider. To locate the collider just click the "Go To Collider" button.

Item Placements



Item Placements are a feature of the Socket that allows you to configure the position, rotation, and scale of items when placed in a socket.

Placement Transition Settings

Tween To Socket -

- **On**: As the item is placed it'll tween the position, rotation and scale to the configured placement.
- **Off**: As the item is placed it'll snap straight to the configured placement's position, rotation, and scale.

Profile Type -

- **Local:** The configured item placements are saved locally to the game object instance.
- Scriptabl_Object: The configured item placements are saved to a scriptable object. This allows you to reuse the configured placements across multiple sockets.

Default Placement Config

If a placeable item does not have an item placement configured for a specific socket, it'll fall back and use the default placement config. Therefore it dictates the placement for any objects that don't have a specific placement configured. It's recommended to always configure the placement of each placeable item. However for some simple use cases, you may want to set up a default configuration.

A use-case for using the default placement config is when you want the same offset for all placeable items. E.G you could modify the Y Position to be 1, resulting in all objects being placed slightly higher than the socket origin.

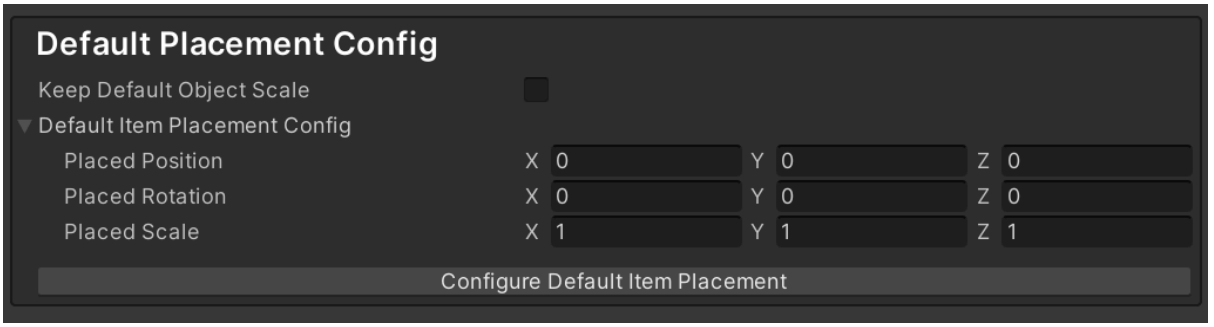
Keep Default Object Scale -

- **On:** When an item is placed and uses the default placement config, it keeps the current object's scale.
- Off: When an item is placed and uses the default placement config, its scale is changed to the "Placed Scale" of the default config.

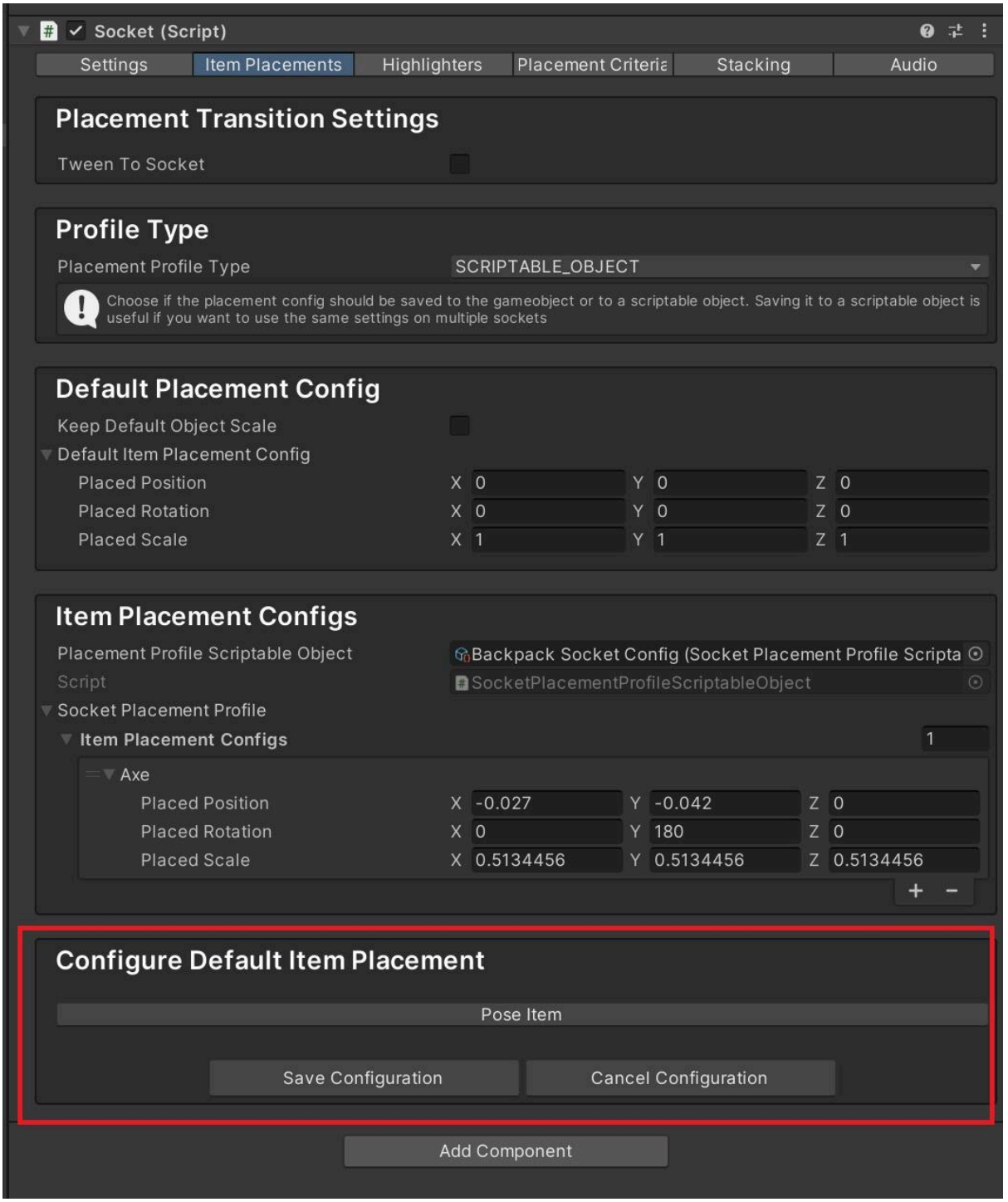
Default Placement Config - A model that contains the position, rotation, and scale of the placed items in the socket.

Instead of manually inputting the values you can use editor tools to configure the placement with the game object gizmos.

1. Click the **Configure Default Item Placement** button

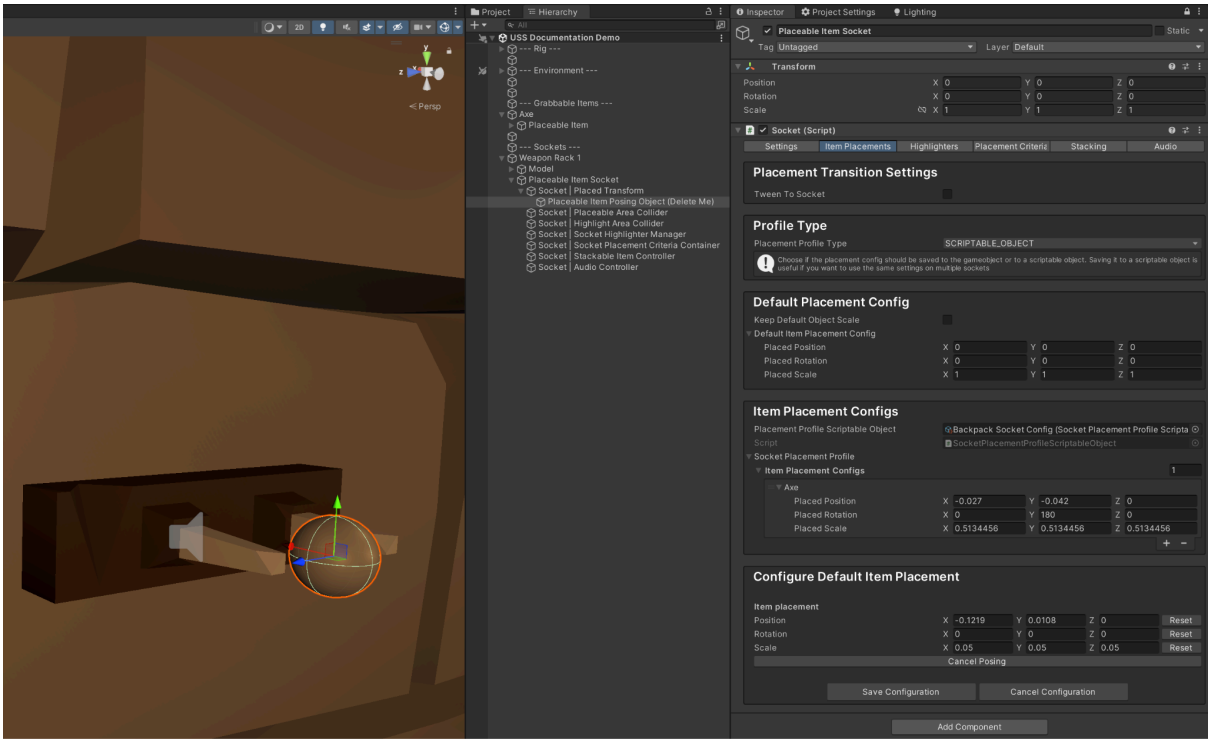


2. You'll now see that the Configure Default Item placement section is visible at the bottom of the Item Placement tab



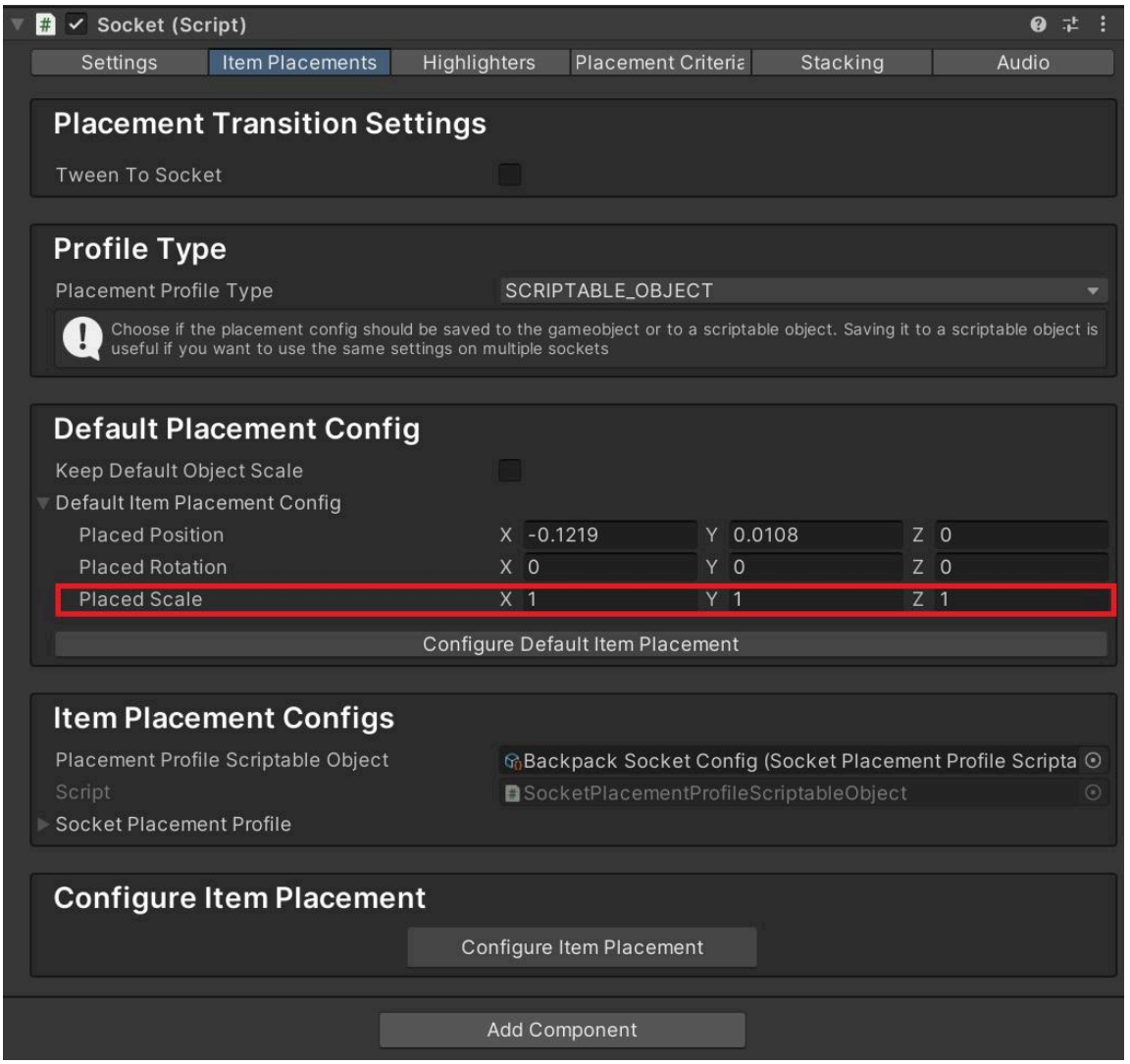
3. Click the "Pose Item" button. Because we're configuring the item placement of a default item, we don't get a specific posing model. Instead, you'll see a primitive sphere that can be moved and rotated.

4. Position the sphere to where you want the default items to be placed. When you've positioned the sphere to where you want it to be, click the Save Configuration button.



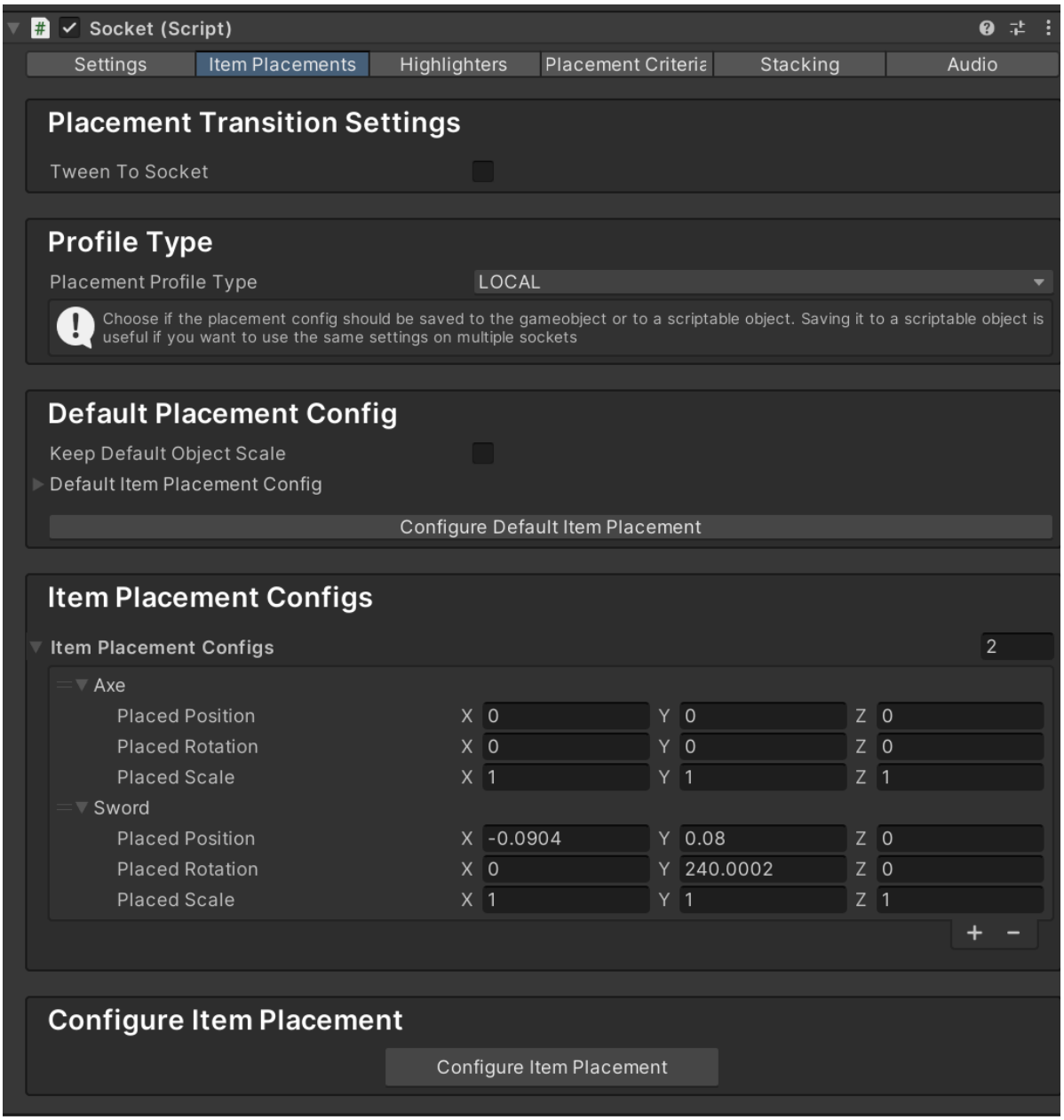
5. For default placements, ignore the scale property when posing the item. This can be changed in the "Placed Scale" property of the Default Placement Config section once

saved.


Socket (Script)
Settings
Item Placements
Highlighters
Placement Criteria
Stacking
Audio
Placement Transition Settings
Tween To Socket
Profile Type
Placement Profile Type
SCRIPTABLE_OBJECT
Choose if the placement config should be saved to the gameobject or to a scriptable object. Saving it to a scriptable object is useful if you want to use the same settings on multiple sockets
Default Placement Config
Keep Default Object Scale
Default Item Placement Config
Placed Position
X -0.1219
Y 0.0108
Z 0
Placed Rotation
X 0
Y 0
Z 0
Placed Scale
X 1
Y 1
Z 1
Configure Default Item Placement
Item Placement Configs
Placement Profile Scriptable Object
Backpack Socket Config (Socket Placement Profile Scripta
Script
SocketPlacementProfileScriptableObject
Socket Placement Profile
Configure Item Placement
Configure Item Placement
Add Component

Item Placement Configs

This will display the list of the configured placements for the selected socket.



Configure Item Placement

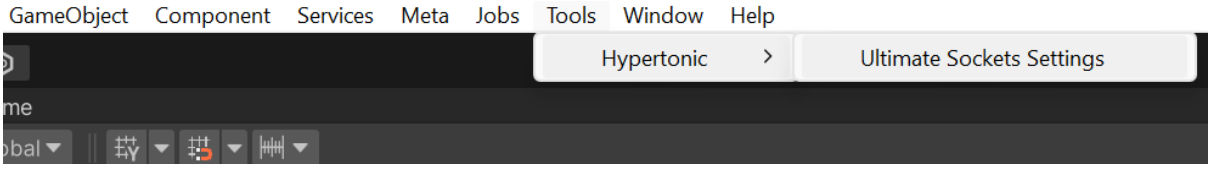
The **Configure Item Placement** section is where you set up each item placement for a socket. Before you can pose an item in the socket to configure its placed transform you'll need to ensure you have set up a posing prefab for that placeable item.

How to set up a posing prefab

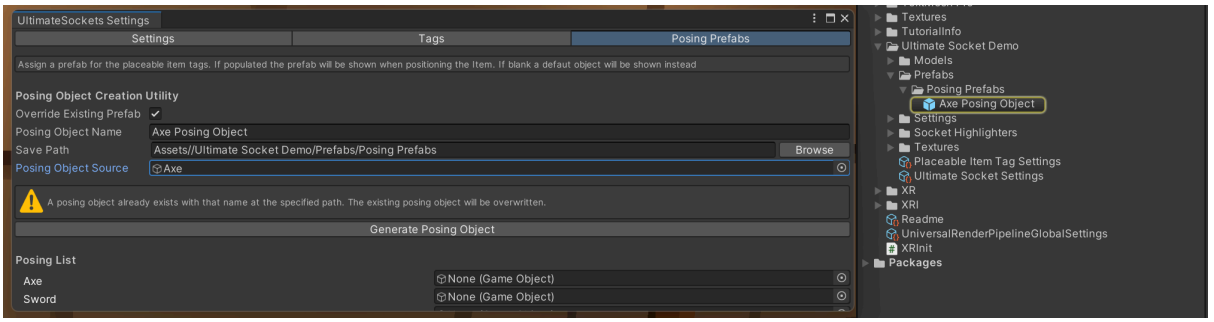
A posing prefab is a designated prefab that is used to visualise how the item will look when placed in the socket. It's recommended **not** to use the actual placeable item prefab itself as this will most likely have numerous components on that may not run correctly if instantiated in the editor. Because of this, there is a built-in utility for generating a posing prefab from a normal game object or prefab. The process essentially duplicates the object and strips off any components that don't relate to the visuals of the object, leaving you wish a visual posing prefab.

Once you have a posing prefab you'll need to assign it to the corresponding Item Tag in the list. See below for a step-by-step guide.

1. Open the Ultimate Socket Settings window from Tools > Hypertonic > Ultimate Sockets Settings.

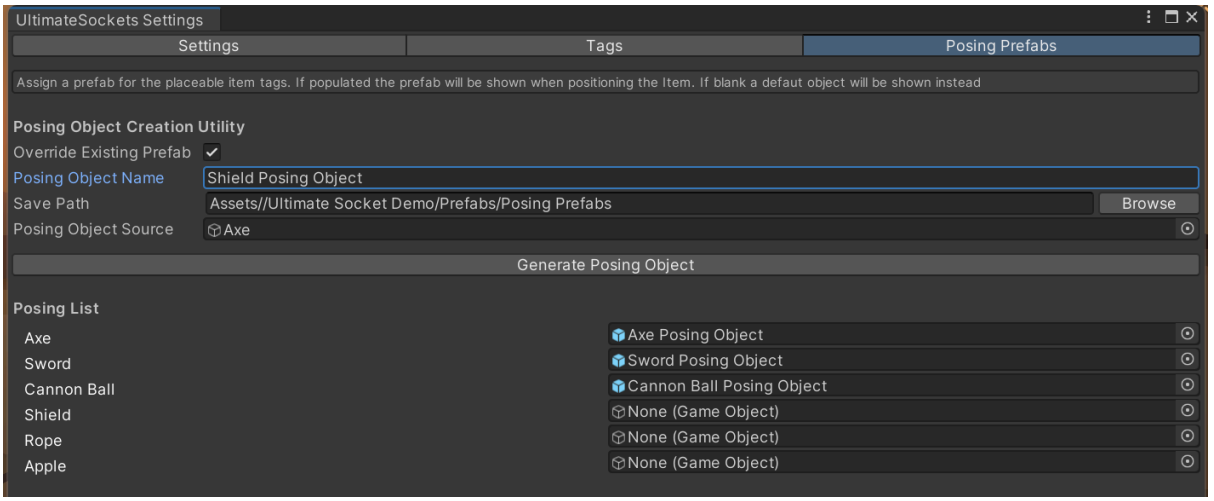


2. Navigate to the Posing Prefabs tab.
3. Use the utility section at the top to generate a posing prefab. You'll need to fill out the name of the posing prefab (what it'll be saved as), where the prefab will be saved, a reference to the source game object or prefab. Finally, Click the generate Posing Object button and you should see your new posing prefab in the Project window in the location specified.



4. Generate posing prefabs for each placeable item in your game. Then drag the references into the Posing List. Remember you can add or remove tags from the

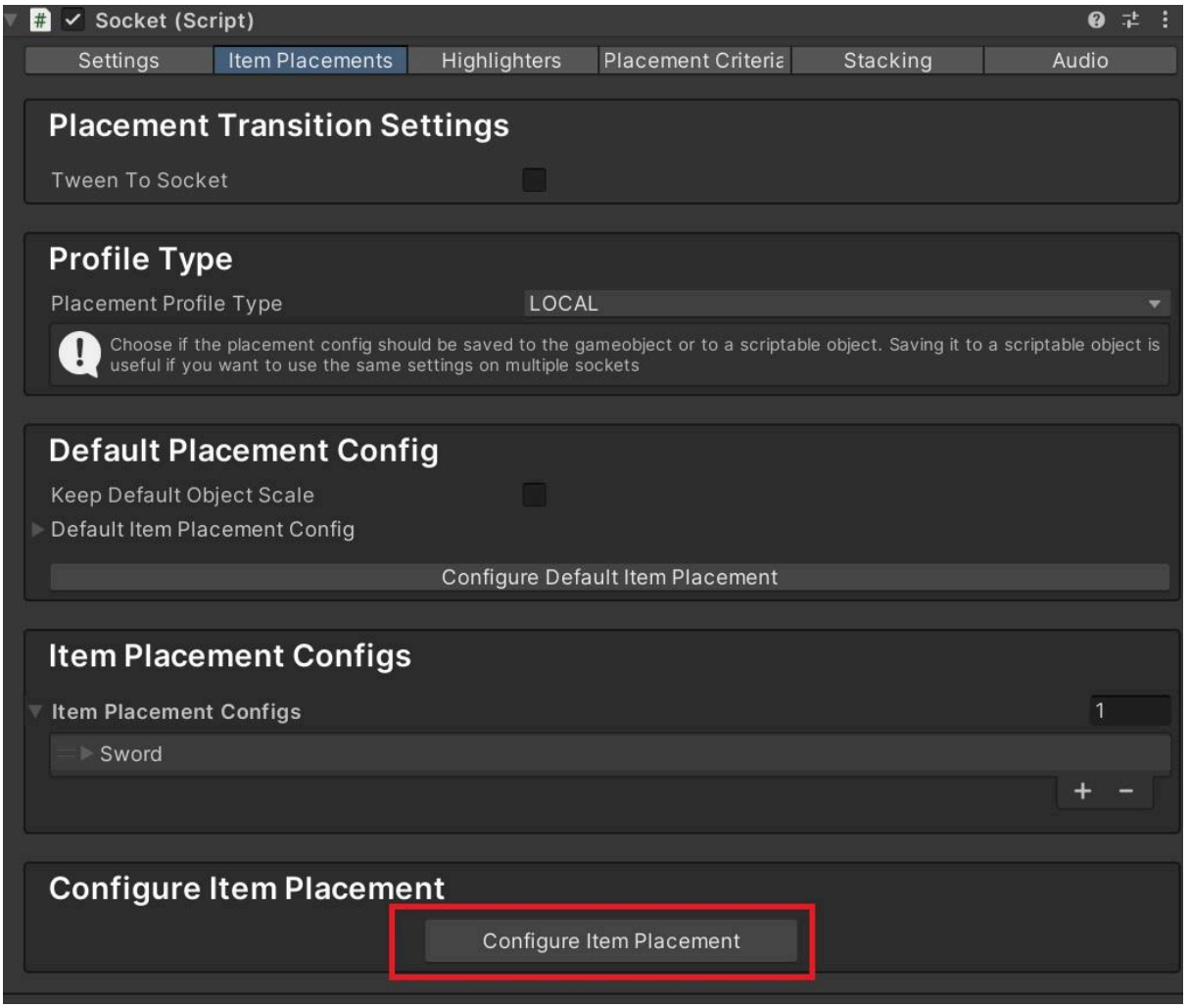
Tags tab on the Ultimate Socket Settings window.



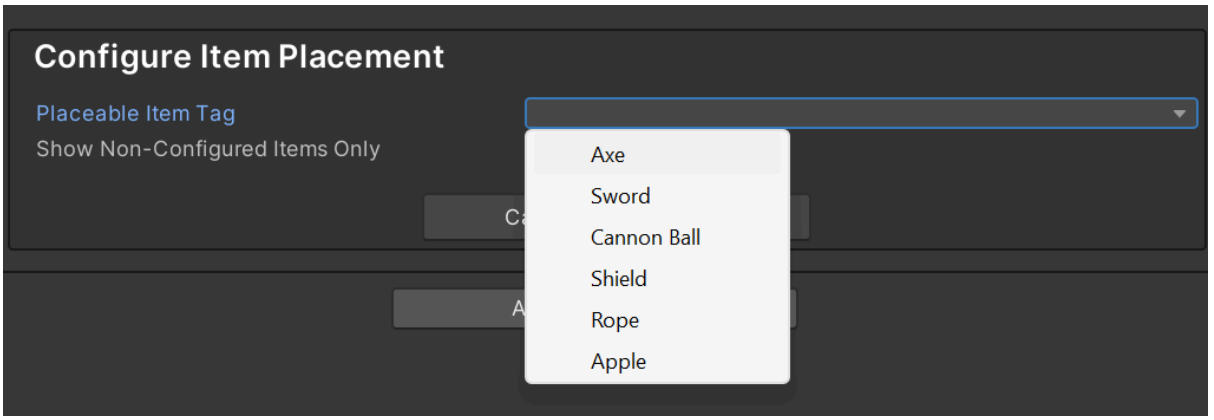
Item Placement Configuration

Following the above section you will now have posing prefabs assigned for 1 or more items. To configure the placement of an object follow the below steps.

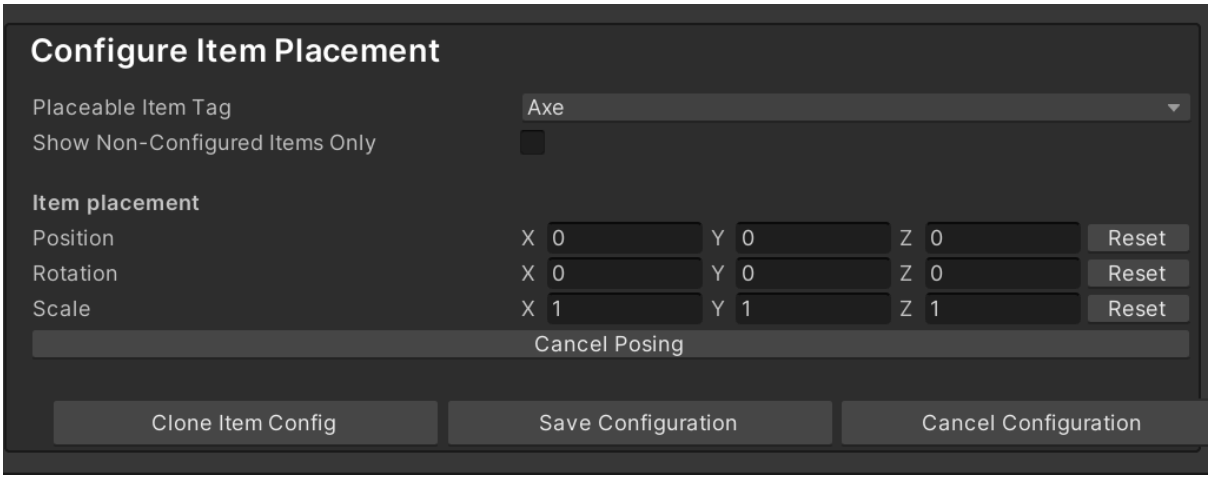
1. Click the Configure Item Placement



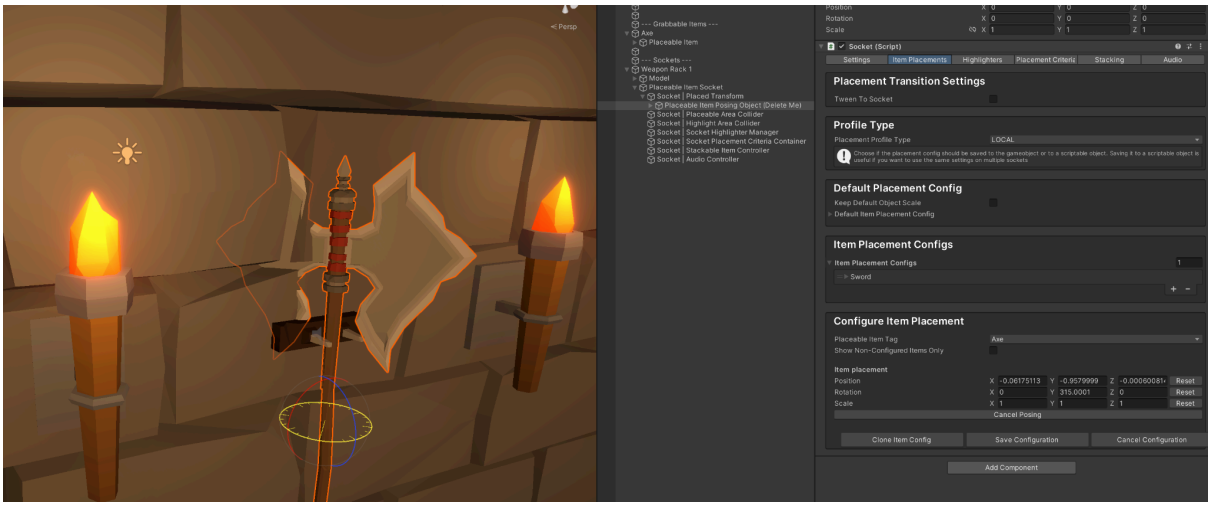
2. Select the item you want to configure from the drop-down



3. Once clicked the posing object will be present in the scene and the Configure Item Placement section should look similar to this

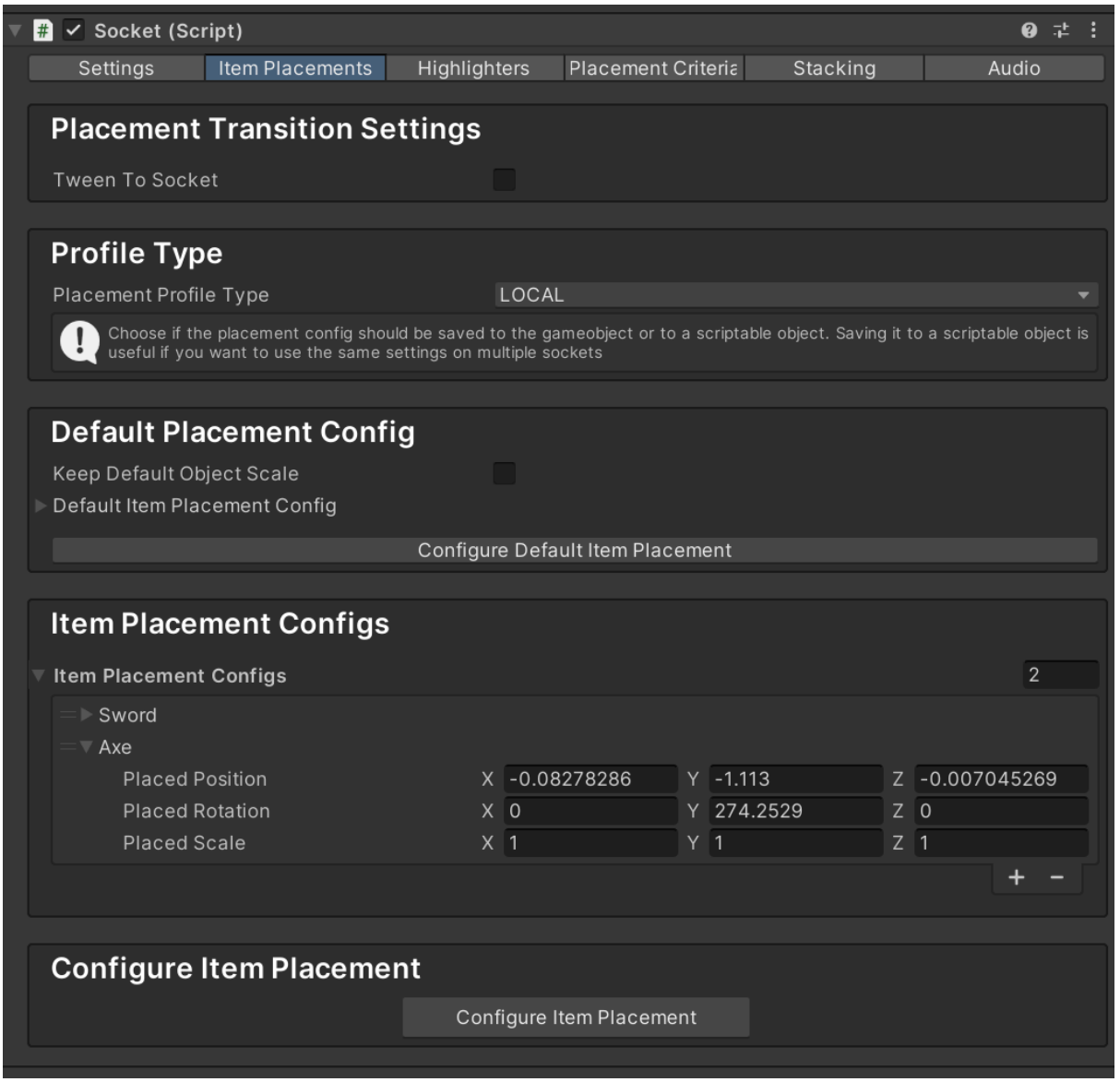


4. Use the gizmos to move, rotate and scale the posing prefab so it fits the socket correctly

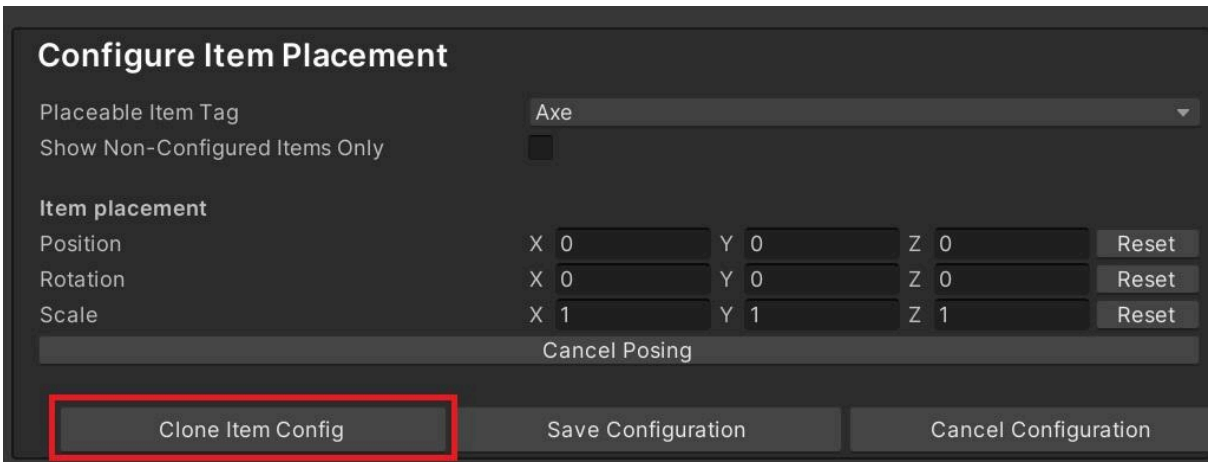


5. When happy with the placement, click the **Save Configuration** button.

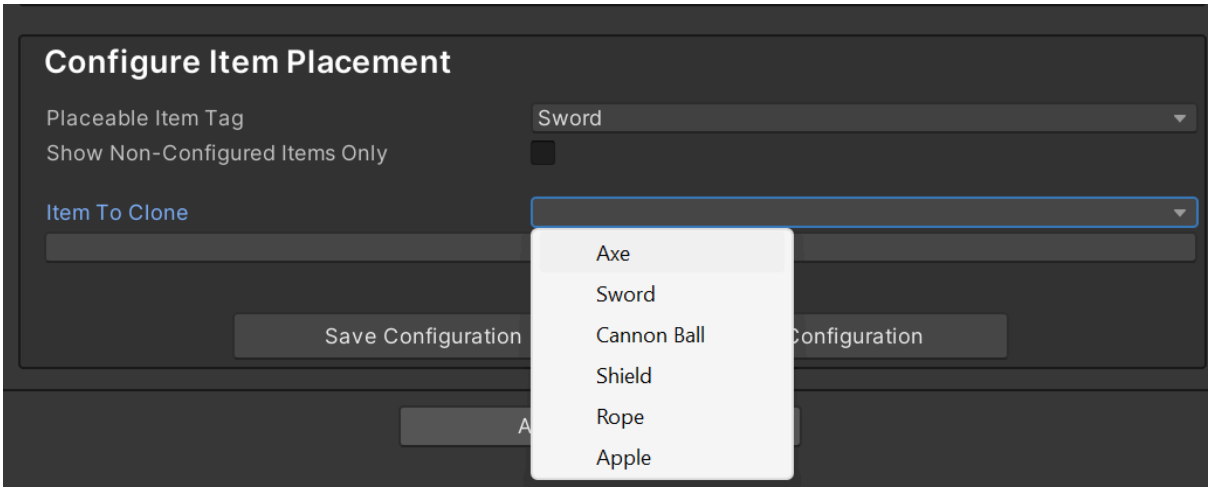
6. Once saved, you will see the configuration in the Item Placement Configs section



7. If the items have similar placements it can be helpful to clone an existing placement from the Item Placement Condfig list. To do this repeat steps 1 - 3. Then click the Clone Item Config button.

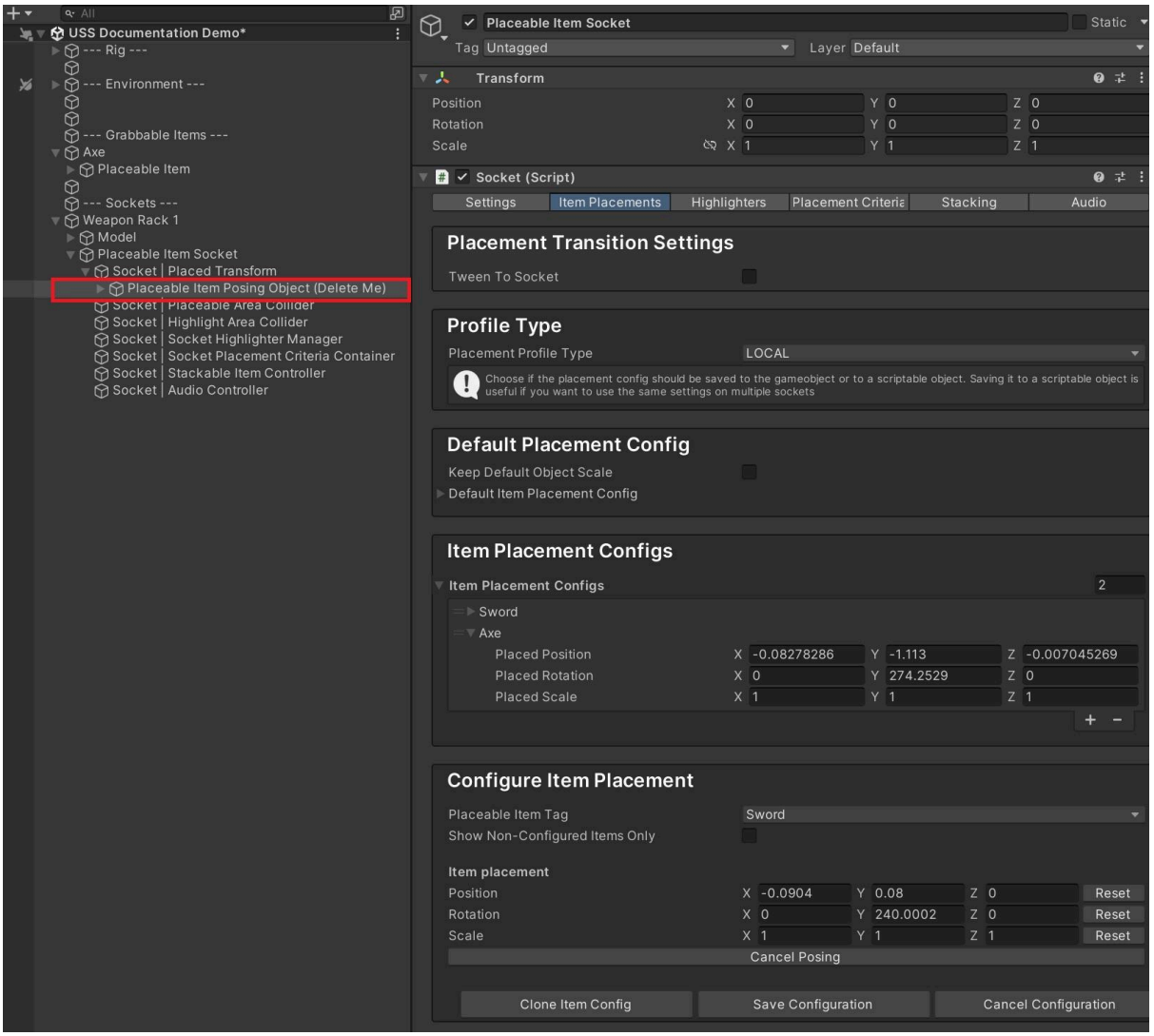


8. This will present the clone option. Select the placement you wish to clone from the list. This will set the posing prefab to have the same properties as the config you've just cloned. You can either save the configuration or continue to tweak the placement, using the cloned config as a starting point.

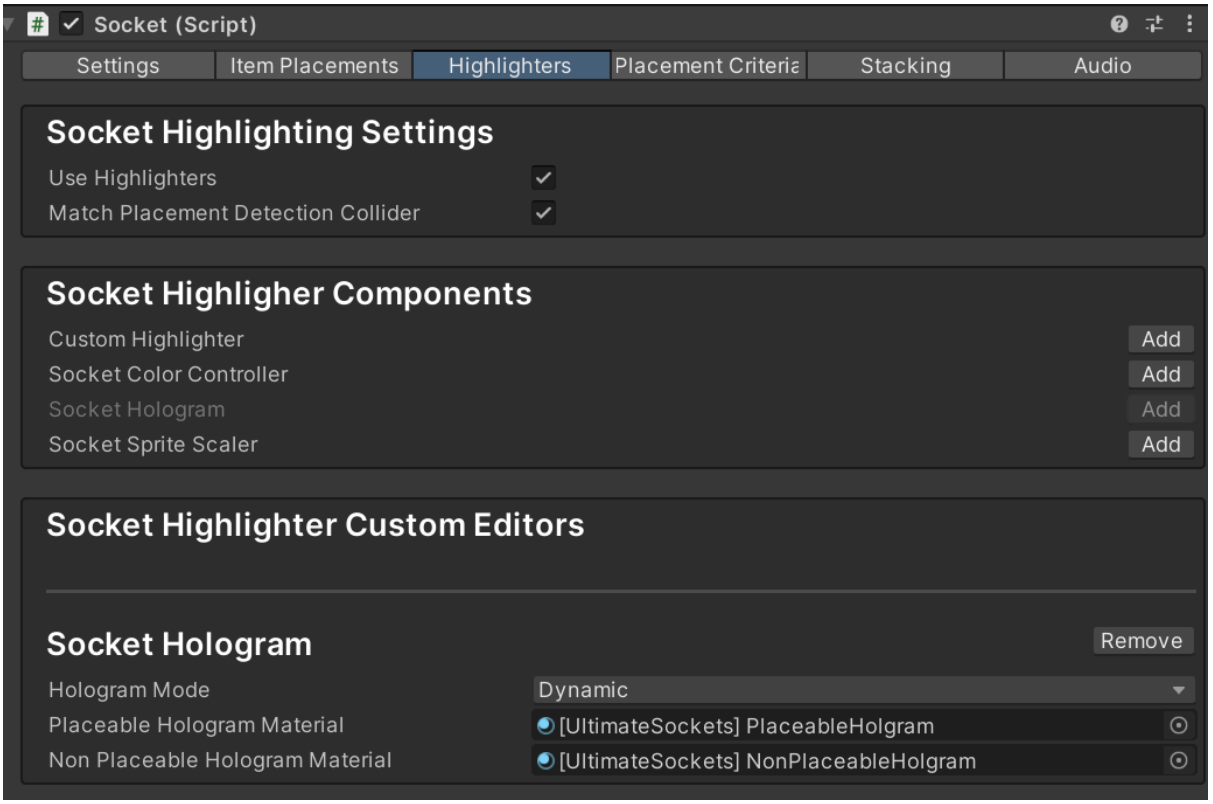


Something to note about the posing functionality is that the system will instantiate the posing prefab underneath the "Socket | Placed Transform" game object and lock the inspector to the Socket. (So you can remain viewing the item placement tab). When you click either Save Configuration or Cancel Configuration the posing prefab instance is automatically destroyed. It's not recommended but, if you happen to unlock the inspector window and navigate away from the socket while posing an item, you'll need to remember to clean up the posing prefab manually by deleting it. It's named "Placeable Item Posing Object (Delete Me)" so it's easy to

spot and search for if for whatever reason you don’t complete the posing process.



Highlighters



Highlighters are a feature of the socket, which displays when a placeable item enters the highlighting range.

Socket Highlighting Settings

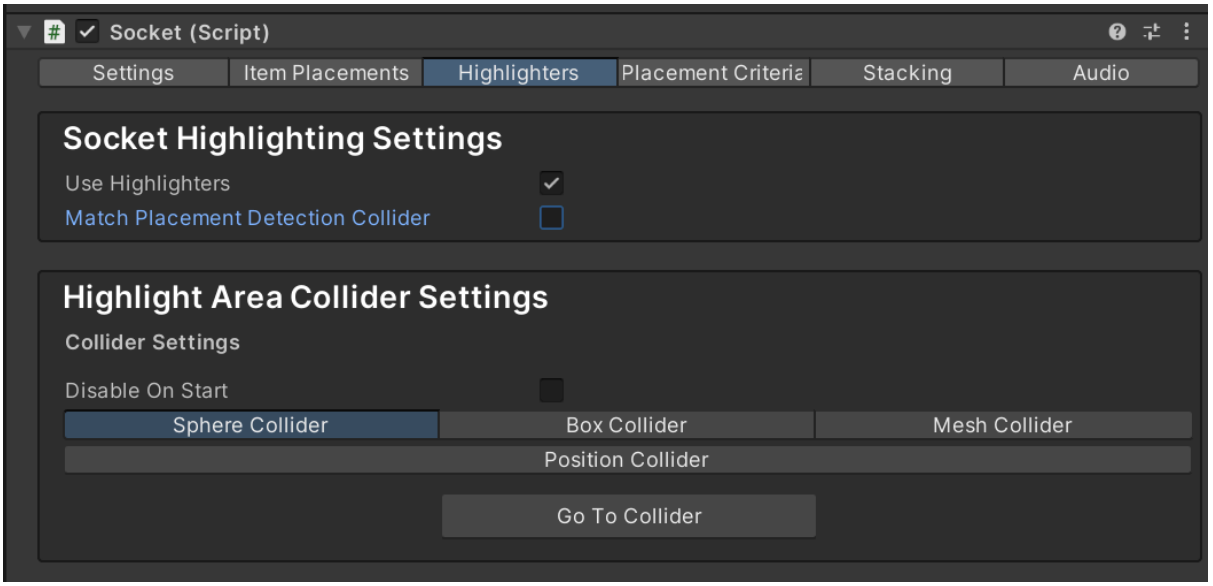
Use Highlighters - Determines if highlighter functionality is enabled or not on the socket
Match Placement Detection Collider - Each socket has a dedicated collider to detect if the item is in the placeable area and a dedicated collider to detect if the placeable item is within the highlight area. For most cases, you will probably want the socket to highlight when the item is within the placeable area. Therefore you can tick the Match Placement Detection Collider, which will set the highlight collider settings (radius, center, etc) to the placeable area collider. However, if you need to have a different range for highlighting and placing, you can untick the box and configure the collider settings.

Highlight Area Collider Settings

As mentioned above, if you choose to have a different highlight radius to the placeable area, then you can untick the “Match Placement Detection Collider” box which will display the standard collider settings section. The collider will always be active if “Use Highlighters” is ticked, however, you will only see the settings for the collider if “Match Placement Detection Collider” is not ticked, as it shouldn’t be editable if it is matching the placement collider.

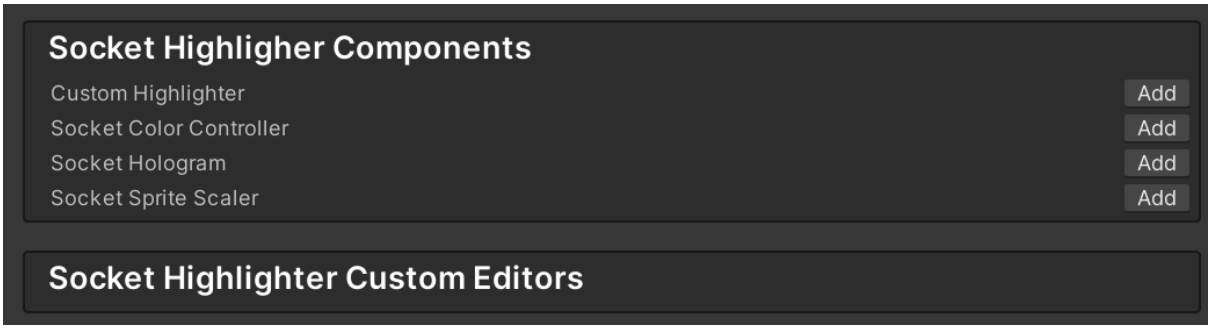
Disable On Start - Disables the placement detection collider when the Start function is called
Collider Type - Choose which type of collider you wish for the placeable item to use to trigger the socket's detection collider.

A future enhancement to have a custom editor to display the collider properties within this section is planned. Until then, you'll need to edit the collider size on the actual collider. To locate the collider just click the "Go To Collider" button.



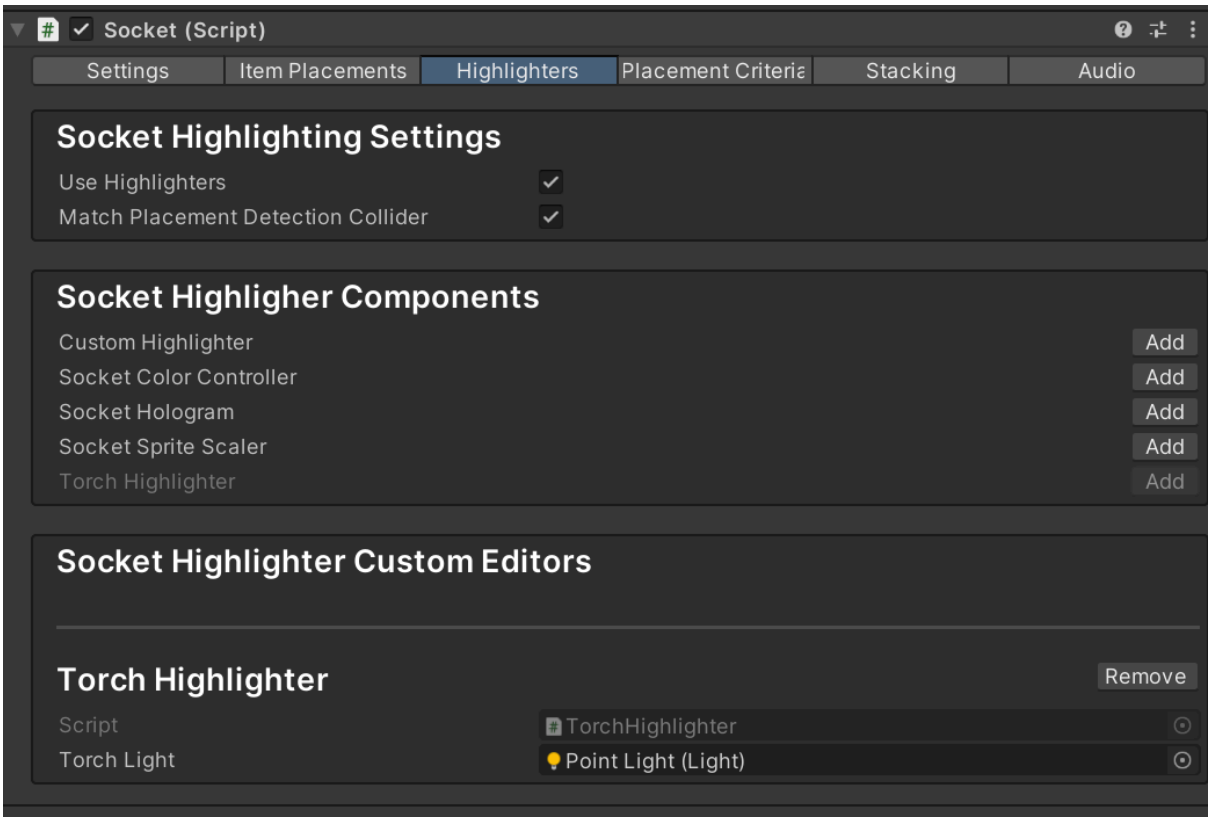
Socket Highlighter Components

This section displays all of the socket highlighters in the project. The asset comes with a few built-in highlighters but is designed for you to create your own custom highlights.

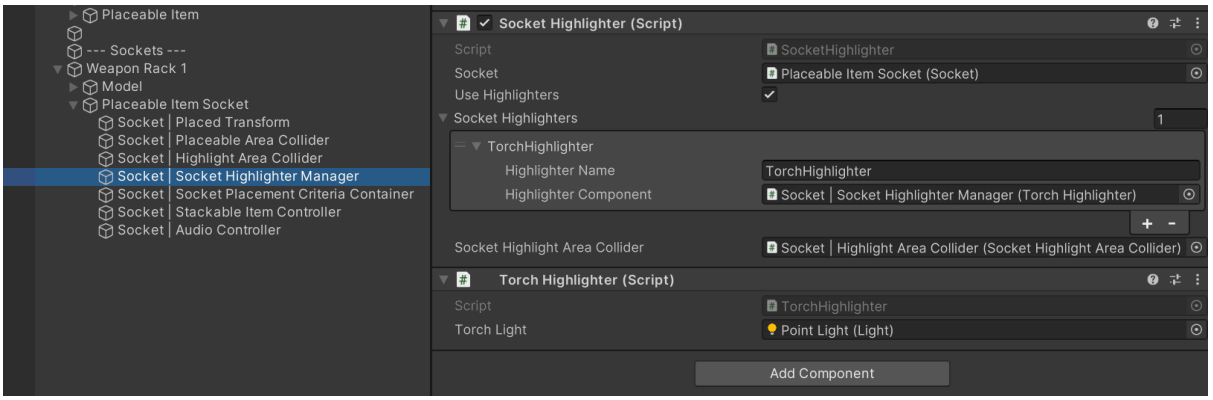


Socket Highlighter Custom Editors

The Socket Highlighter custom editor section displays all of the highlighter components that have been added to the socket. Once added you can click the **remove** button to delete the highlight component.



The custom editors are displayed here to keep all of the socket highlighting functionality in a single place. However, it is worth mentioning that the highlighter components are added to the "**Socket | Socket Highlighter Manager**" game object. You shouldn't ever need to click on this object to view the highlighter components and it's recommended you only add, remove, and edit properties of highlighter components via the highlighting tab on the socket. However, if something isn't working as expected then it may be helpful to see what's going on under the hood and check out this game object to view the actual components.

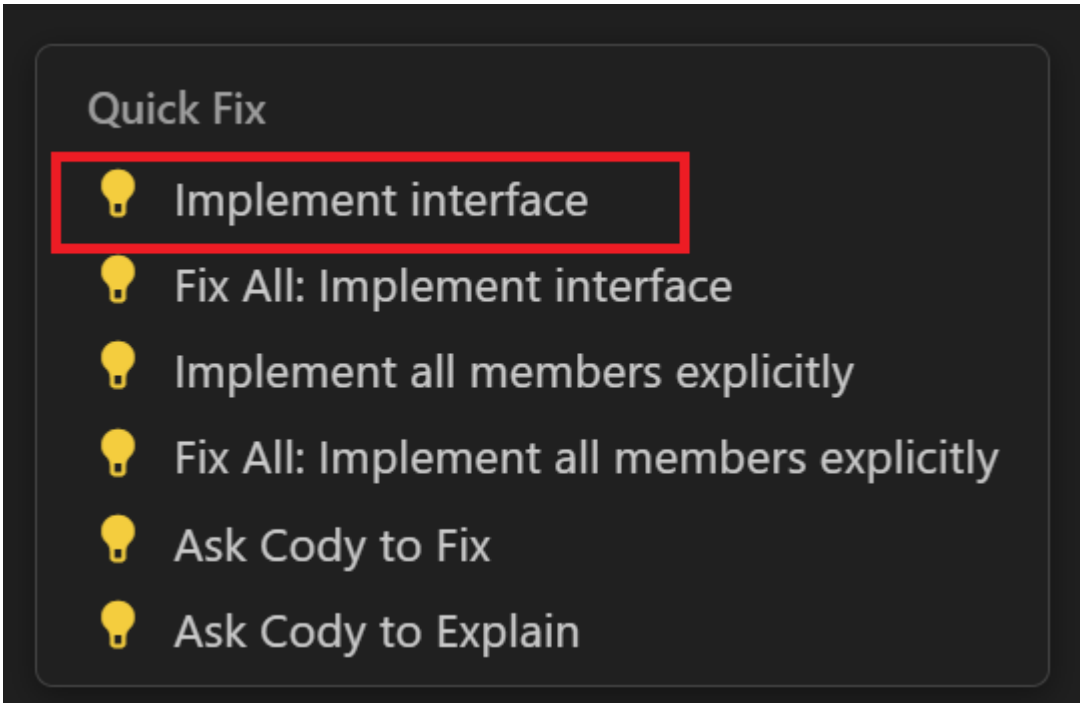


How to add your own socket highlighter.

To create your own socket highlighter component, you just need to implement the **ISocketHighlighter** interface onto a Monobehaviour component. The cool thing here is that as soon as you return to Unity and the scripts re-complie, your custom socket highlighter just shows up in the list of highlighter components.

This guide will walk you through creating a simple example to highlight a torch when a placeable item becomes in range. You can also view the code for the built-in highlighters to see how other highlighters work (such as displaying a different color highlight if the item is within range but not placeable - more on that in the placement criteria section.)

1. Create a new script and add the following namespaces:
    - using Hypertonic.Modules.UltimateSockets.Highlighters;
    - using Hypertonic.Modules.UltimateSockets.PlaceableItems;
2. Add the ISocketHighlighter interface after MonoBehaviour.
3. Implement the ISocketHighlighter functions - most IDEs will suggest adding the required functions. If not you can view the ISocketHighlighter script itself to view the functions.

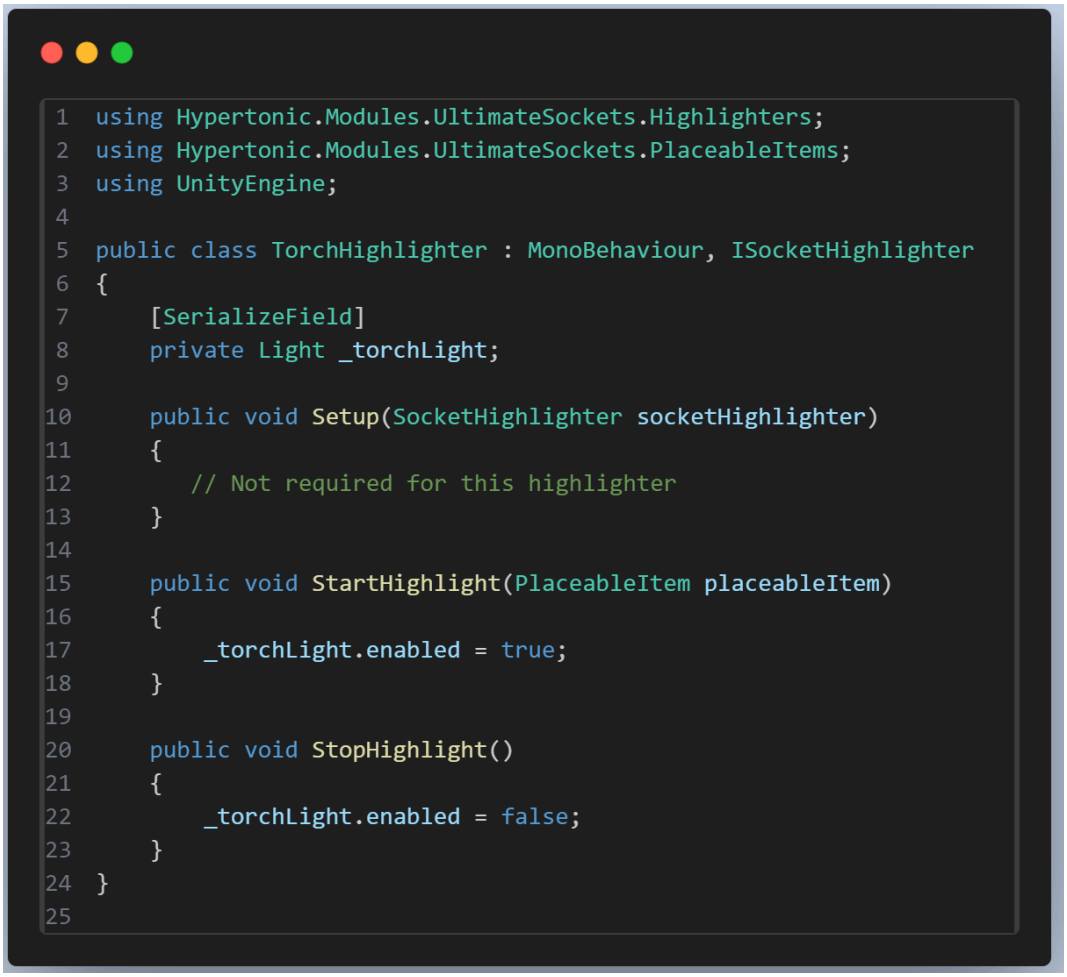


4. You should now have a class that looks similar to this:

```
using Hypertonic.Modules.UltimateSockets.Highlighters;
using Hypertonic.Modules.UltimateSockets.PlaceableItems;
using UnityEngine;

public class TorchHighlighter : MonoBehaviour, ISocketHighlighter
{
    public void Setup(SocketHighlighter socketHighlighter)
    {
        throw new System.NotImplementedException();
    }

    public void StartHighlight(PlaceableItem placeableItem)
    {
        throw new System.NotImplementedException();
    }

    public void StopHighlight()
    {
        throw new System.NotImplementedException();
    }
}
```

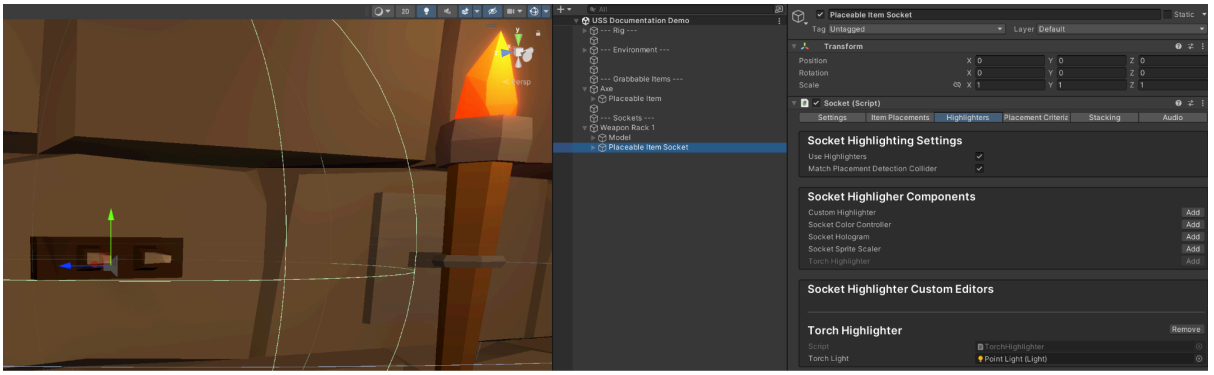
5. You won't always need to do anything in the Setup function. Although it can be handy for caching a reference to the socketHighlighter component, which in turn has a reference to the Socket component, which has references to all socket-related properties - useful when implementing more complex highlighters.

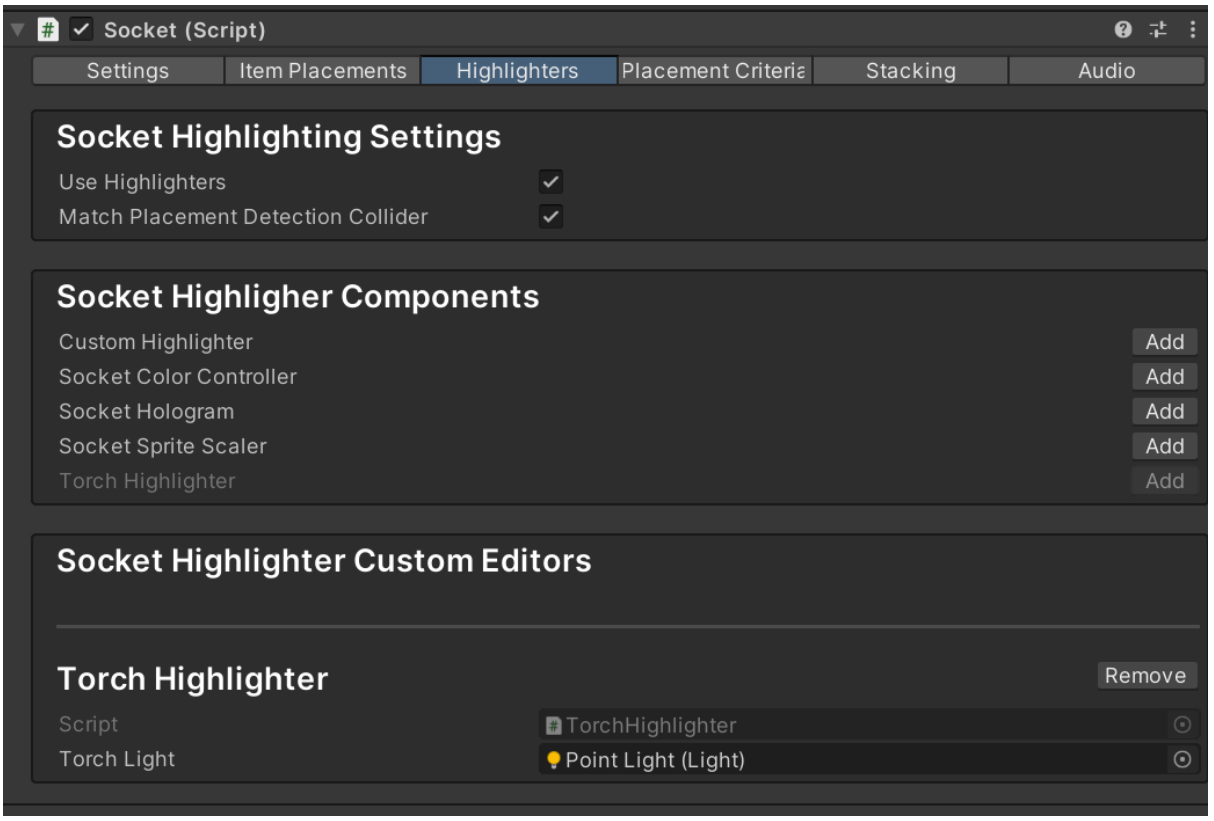
6. In this example, we'll add a light property that belongs to the torch and turn it on when we're in the highlight area and off when not in the highlight area.



```
using Hypertonic.Modules.UltimateSockets.Highlighters;
using Hypertonic.Modules.UltimateSockets.PlaceableItems;
using UnityEngine;

public class TorchHighlighter : MonoBehaviour, ISocketHighlighter
{
    [SerializeField]
    private Light _torchLight;

    public void Setup(SocketHighlighter socketHighlighter)
    {
       // Not required for this highlighter
    }

    public void StartHighlight(PlaceableItem placeableItem)
    {
        _torchLight.enabled = true;
    }

    public void StopHighlight()
    {
        _torchLight.enabled = false;
    }
}

```

7. Now when you navigate to the highlighters tab on the socket you'll see your highlighter in the list. You can use the "Add" button to add any highlighter components to the socket. Once added, they'll be added to the "Socket | Socket Highlighter Manager" game object nested below the socket game object. However, to keep the process streamlined, the highlighter components are displayed in the "Socket Highlighter Custom Editors" section as a custom editor. Providing the ability to set property values to any exposed fields.





As mentioned above, the setup method isn't always required. However, to achieve more complex implementations you should cache the socketHighlighter variable like so:

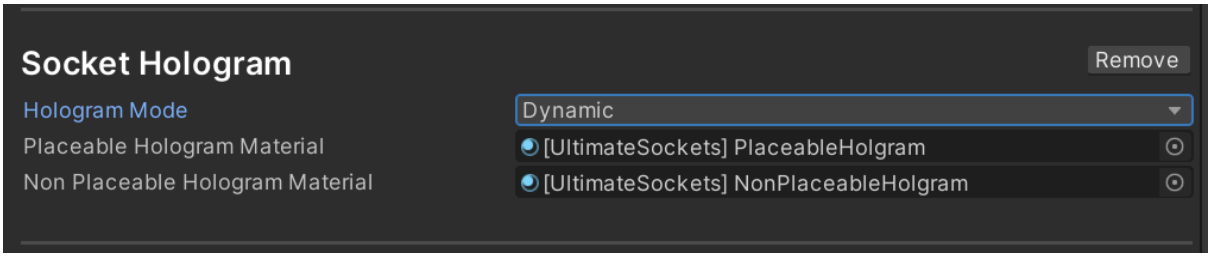
```
1        private SocketHighlighter _socketHighlighter;
2
3        public void Setup(SocketHighlighter socketHighlighter)
4        {
5            _socketHighlighter = socketHighlighter;
6        }
```

This will allow you to access the socket class and therefore determine if the socket is placeable or not - allowing you to adjust the highlighter based on the result.

```
bool itemIsPlaceable = _socketHighlighter.Socket.CanPlace(placeableItem, placementCriteriaNamesToIgnore);
```

Built-in highlighters

Socket Hologram



The socket hologram highlighter has a few options to choose from. The default mode is dynamic.

Dynamic

This will display a hologram of the placeable item that is within the highlight area. You can set the hologram material for when the item is within the highlight area and is placeable, or when it's in the highlight area but is not placeable.

Item Specific

Allows you to use a config to map a placeable item tag to a specific prefab to display.

Game Object

This will instantiate a specified game object when a placeable item enters the highlight area.

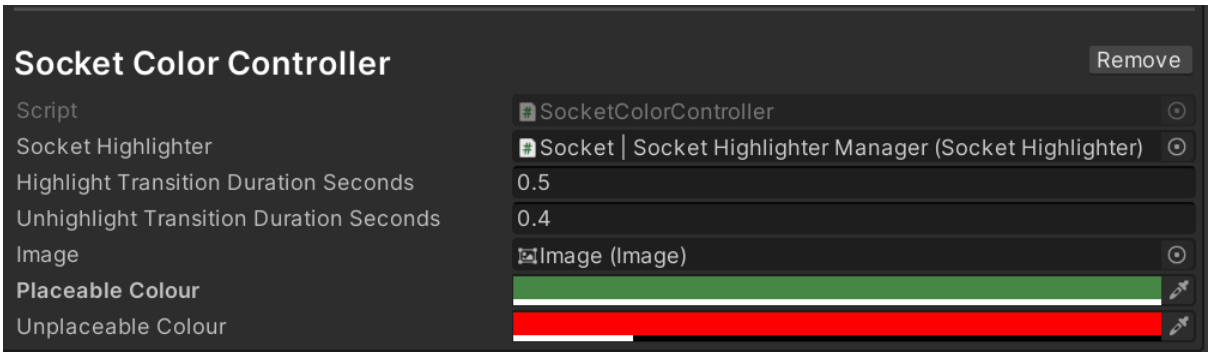
Single Prefab

This will instantiate a specified prefab when a placeable item enters the highlight area.

None

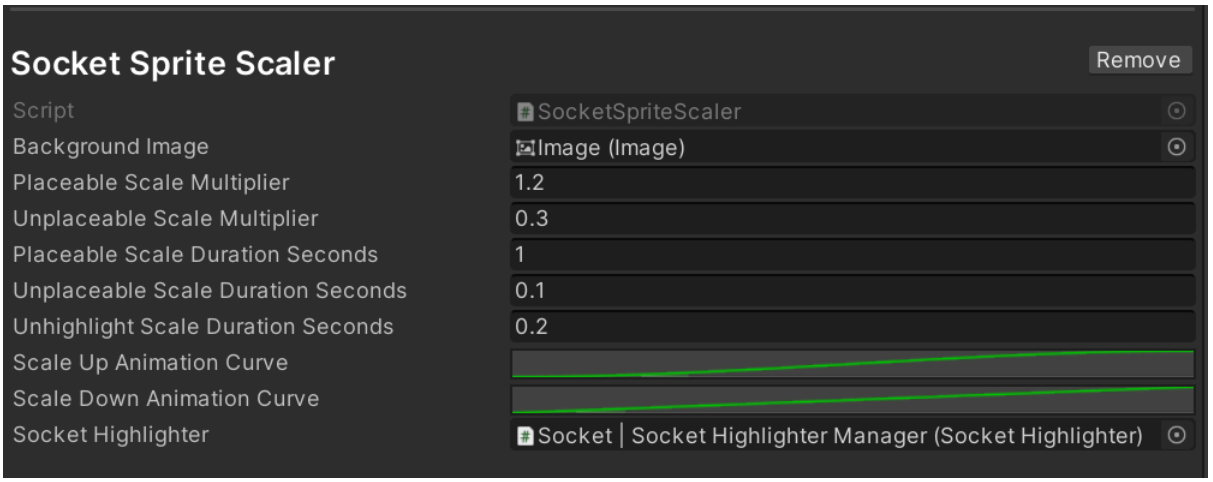
No highlighting functionality will activate

Socket Color Controller



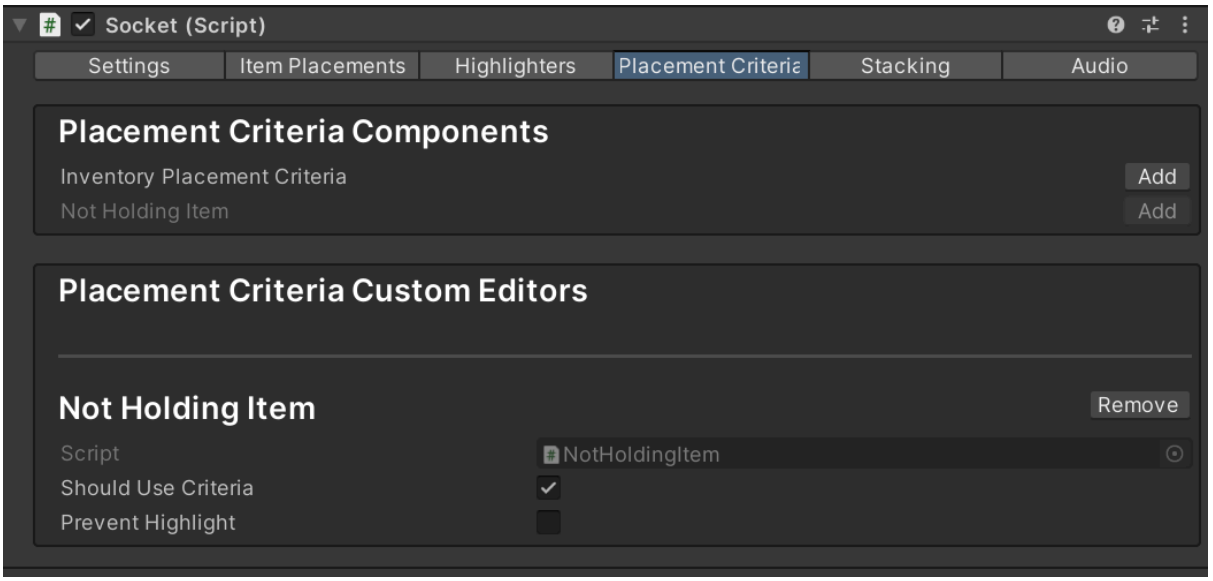
The Socket Color Controller is designed to change the colour of an image when a placeable item enters the highlight zone. It has a designated colour for if the item is currently placeable or not. An example of where this works well is inventory slots on a backpack.

Socket Sprite Scaler



This highlighter will tween the scale of an image when an item enters the highlight area. There are set scales for when the item is currently placeable or not. This can work well in combination with the socket sprite scaler highlighter.

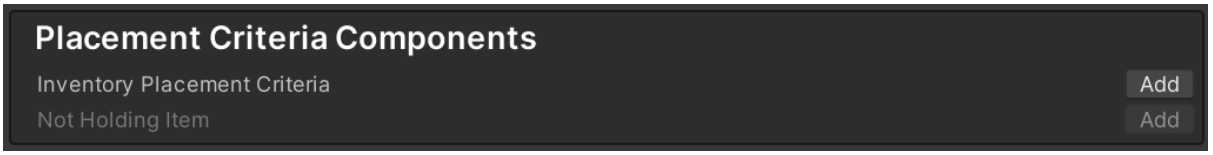
Placement Criteria



Placement criteria are a set of customisable rules that allow you to determine if an item can be placed or not. You can add placement criteria to either the socket or placeable item. When added to the socket, the criteria will be applied to all items that enter the placeable area of the socket. When added to the placeable item the criteria will be checked by any sockets trying to place that item.

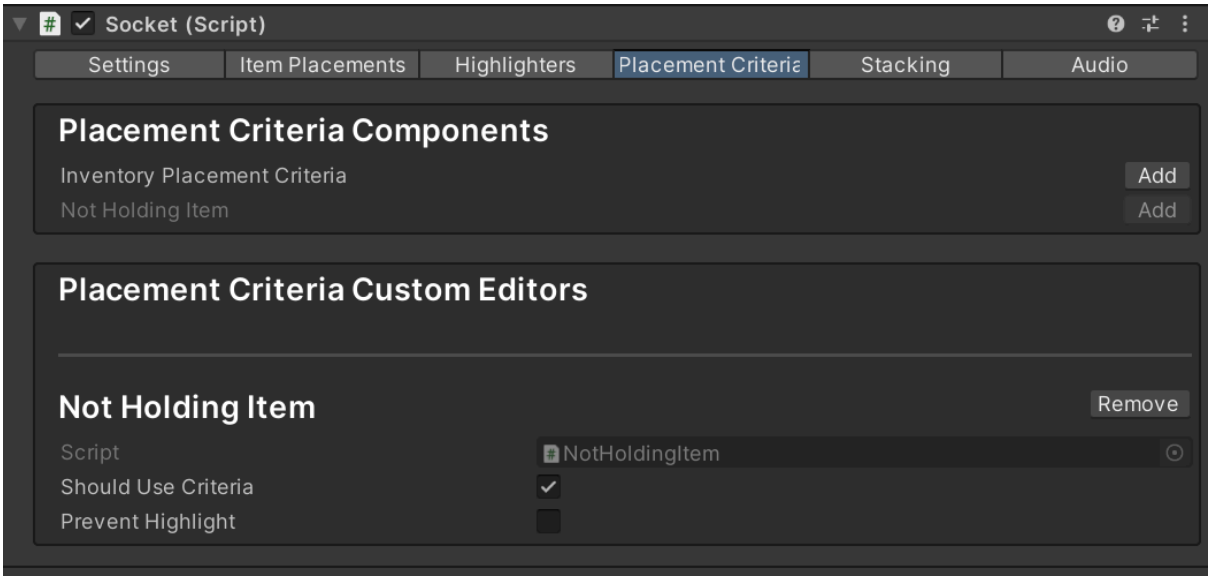
Placement Criteria Components

This section will display a list of all placement criteria scripts in the project. The asset comes with a few built-in placement criteria but is designed for you to create your own custom highlights. You can click the add button to add the criteria to the selected socket or placeable item.

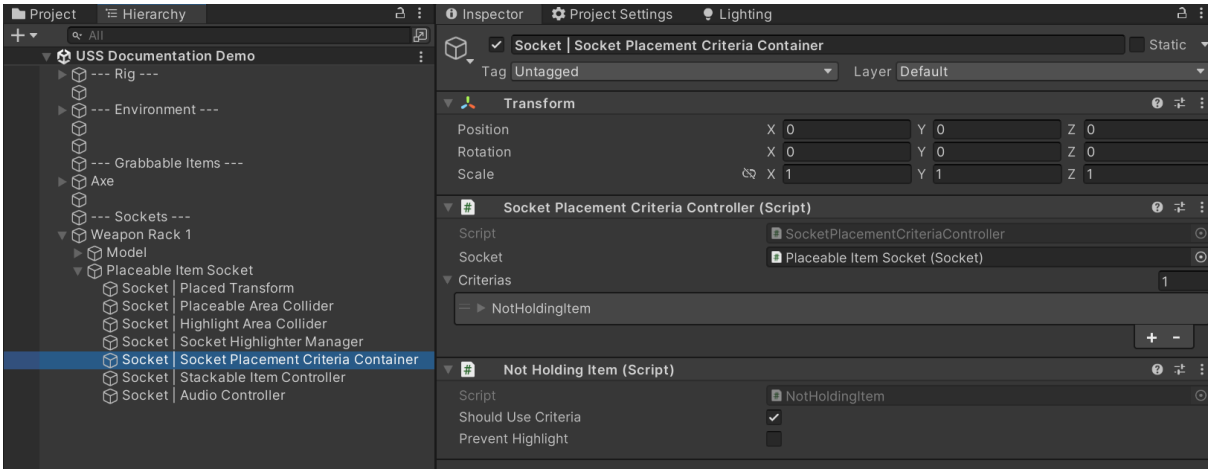


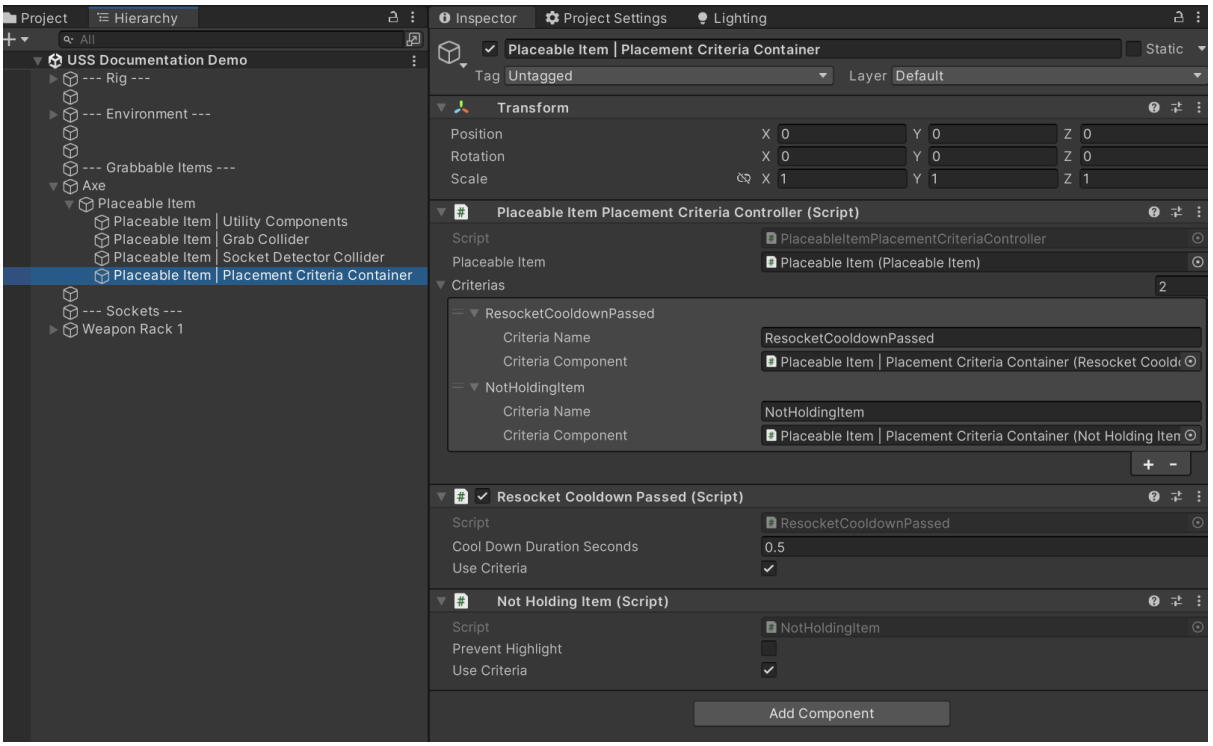
Placement Criteria Custom Editors

The Highlighter custom editor section displays all of the highlighter components that have been added to the socket or placeable item. Once added you can click the **remove** button to delete the criteria component.



The custom editors are displayed here to keep all of the criteria functionality in a single place. However, it is worth mentioning that the criteria components are added to the "**Socket | Socket Placement Criteria Container**" game object on sockets, and "**Placeable Item | Placement Criteria Container**" on placeable items. You shouldn't ever need to click on these objects to view the criteria components and it's recommended you only add, remove, and edit properties of criteria components via the criteria tab. However, if something isn't working as expected then it may be helpful to see what's going on under the hood and check out this game object to view the actual components.


Project
Hierarchy
Inspector
Project Settings
Lighting
USS Documentation Demo
--- Rig ---
--- Environment ---
--- Grabbable Items ---
Axe
--- Sockets ---
Weapon Rack 1
Model
Placeable Item Socket
Socket | Placed Transform
Socket | Placeable Area Collider
Socket | Highlight Area Collider
Socket | Socket Highlighter Manager
Socket | Socket Placement Criteria Container
Socket | Stackable Item Controller
Socket | Audio Controller
Socket | Socket Placement Criteria Container
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Socket Placement Criteria Controller (Script)
Script SocketPlacementCriteriaController
Socket Placeable Item Socket (Socket)
Criterias 1
NotHoldingItem
Not Holding Item (Script)
Script NotHoldingItem
Should Use Criteria
Prevent Highlight


Project
Hierarchy
Inspector
Project Settings
Lighting
USS Documentation Demo
--- Rig ---
--- Environment ---
--- Grabbable Items ---
Axe
Placeable Item
Placeable Item | Utility Components
Placeable Item | Grab Collider
Placeable Item | Socket Detector Collider
Placeable Item | Placement Criteria Container
--- Sockets ---
Weapon Rack 1
Placeable Item | Placement Criteria Container
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Placeable Item Placement Criteria Controller (Script)
Script PlaceableItemPlacementCriteriaController
Placeable Item Placeable Item (Placeable Item)
Criterias 2
ResocketCooldownPassed
Criteria Name ResocketCooldownPassed
Criteria Component Placeable Item | Placement Criteria Container (Resocket Coold
NotHoldingItem
Criteria Name NotHoldingItem
Criteria Component Placeable Item | Placement Criteria Container (Not Holding Iten
Resocket Cooldown Passed (Script)
Script ResocketCooldownPassed
Cool Down Duration Seconds 0.5
Use Criteria
Not Holding Item (Script)
Script NotHoldingItem
Prevent Highlight
Use Criteria
Add Component

How to add your own Placement Criteria

There are two types of placement criteria, socket placement criteria and placeable item criteria. Firstly you'll need to decide which type you need to create based on your use-case. Socket placement criterias allow you to add criteria once to a socket which will be used on any item that enters the placeable range. Whereas the placeable item criteria will apply that criteria to any socket that is within the placeable range.

Here is the interface for the **ISocketPlacementCriteria** interface.

```
public interface ISocketPlacementCriteria
{
    public void Setup(Socket socket);
    public bool CanPlace(PlaceableItem placeableItem);
    public bool UseCriteria();
    public bool PreventHighlight();
}
```

And here is the interface for the IPlaceableItemPlacementCriteria interface.

```
public interface IPlaceableItemPlacementCriteria
{
    public void Setup(PlaceableItem placeableItem);
    public bool CanPlace();
    public bool UseCriteria();
    public bool PreventHighlight();
}
```

You may notice that they are very similar and have the same 4 required functions, however, the parameters in the Setup and CanPlace functions differ slightly.

The **Setup** function is called when the criteria are added to the socket or placeable item. In which, the ISocketPlacementCriteria is provided a reference to the socket it's been added to and a reference to the placeable item in the CanPlace function. Allowing the socket placement criteria to utilise references across both core components to perform the placement checks. The IPlaceableItemPlacementCriteria component is only provided with a reference to the placeable item. Meaning it can only use the placeable item to determine if the item is placeable.

The **CanPlace** function is responsible for telling the socket or placeable item if it can be placed and is where the main logic will happen.

The **UseCriteria** function is designed to enable users to dynamically turn criteria on or off programmatically during runtime. This option should be used instead of simply adding or removing the criteria itself as it may have exposed references that are set in the editor and would be lost if removed at runtime.

The **PreventHighlight** function allows more complex highlighter behaviour. Highlighters activate when an item enters the placeable area, they don't necessarily care if the item is actually placeable or not. Therefore they can highlight differently based on whether the item is placeable. E.G. show a blue hologram if the item can be placed, or show a red hologram if an item cannot be placed. If the highlighter components are simple and are not concerned with if the item is placeable or not then you can simply return true.

If you inspect the built-in hologram highlighter script, you'll see that when the item enters placeable range it will start to highlight, it then checks to see if the item is placeable or not. If placeable, a placeable hologram material is displayed, if it's not placeable then a non placeable hologram material is displayed.

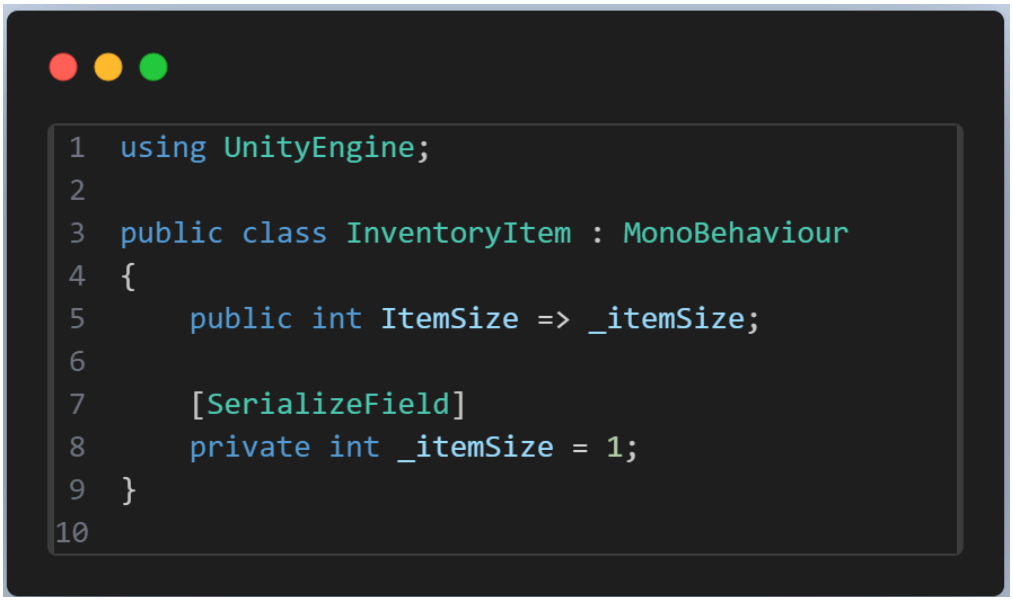
So, in order for more complex highlighters that highlight differently based on the item being placeable (such as the built in hologram) to work, the **Prevent Highlight** implementation needs to return false on any criteria being used on that socket or placeable item.

Example Guide

To create a new placement criteria, simply implement the ISocketPlacementCriteria or IPlaceableItemPlacementCriteria on your Monobehaviour class and implement the required functionality, once Unity recomplies it'll be displayed in the list of placement criteria or either the socket or placeable item.

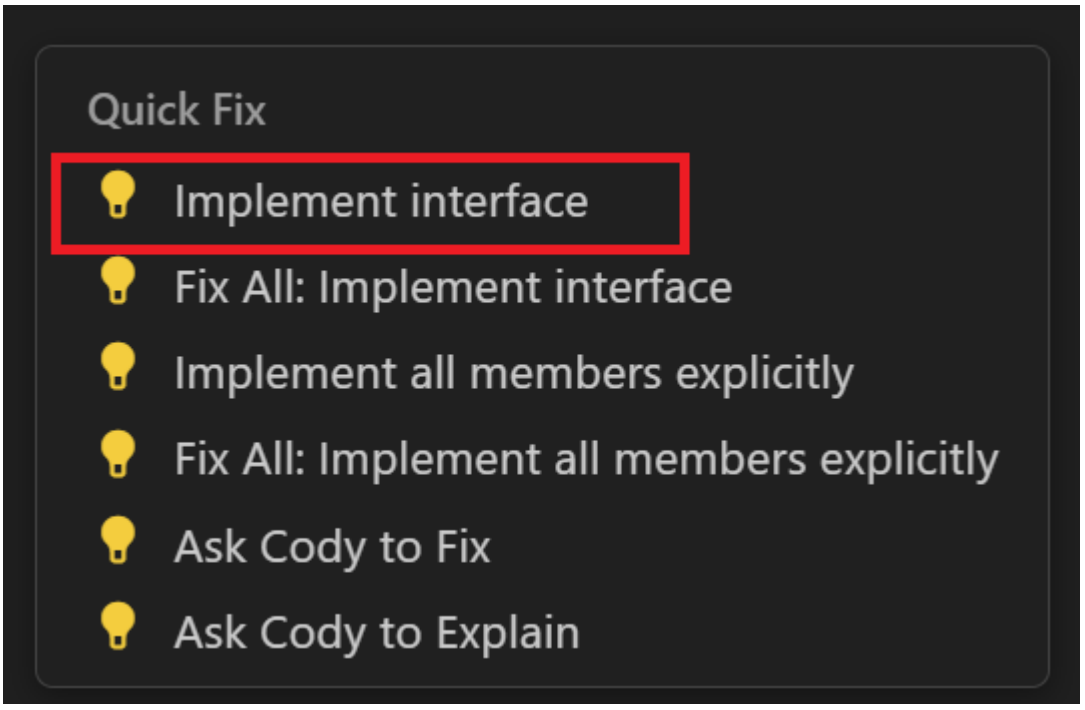
This guide will walk you through an example of creating a simple placement criteria component for a socket. In this example, we'll assume that there is an inventory system which placeable items can be placed into if they have an **InventoryItem** component attached.

1. We'll quickly make a simple InventoryItem component that holds the item size. This will go onto any placeable item that can be stored in the player's inventory.



```
using UnityEngine;

public class InventoryItem : MonoBehaviour
{
    public int ItemSize => _itemSize;

    [SerializeField]
    private int _itemSize = 1;
}

```

2. Next, we'll create a new script for the placement criteria and add the following namespaces:
    - Using Hypertonic.Modules.UltimateSockets.Interfaces;
    - Using Hypertonic.Modules.UltimateSockets.PlaceableItems;
    - Using Hypertonic.Modules.UltimateSockets.Sockets;
3. Add the **ISocketPlacementCriteria** interface after the Monobehaviour.
4. Implement the **ISocketPlacementCriteria** functions - most IDEs will suggest adding the required functions. If not you can view the **ISocketPlacementCriteria** script itself to view the functions.



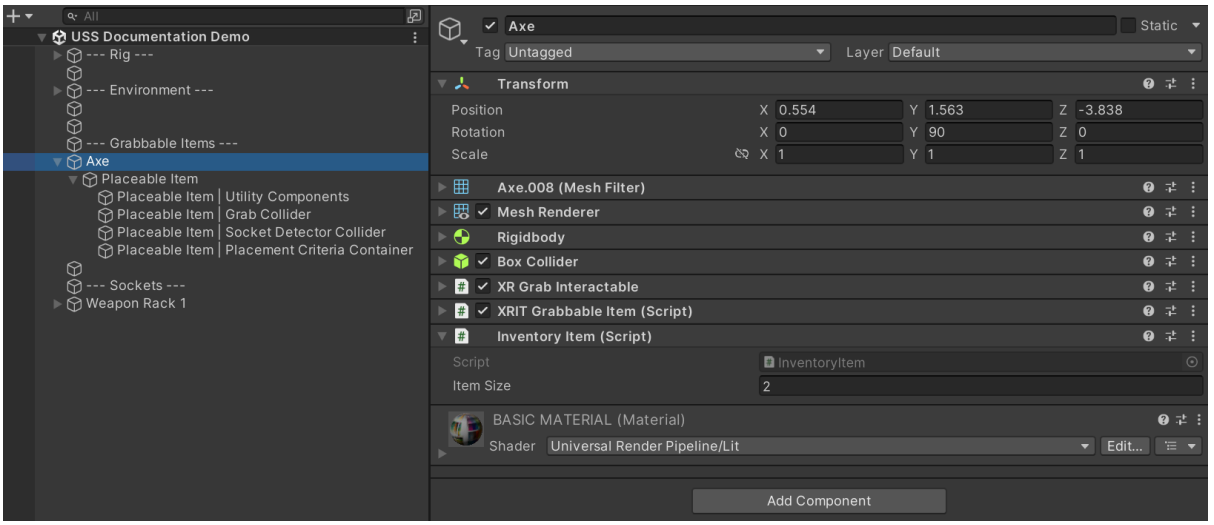
5. We now have a class that looks similar to the below

```
public class InventoryPlacementCriteria : MonoBehaviour, ISocketPlacementCriteria
{
    public bool CanPlace(PlaceableItem placeableItem)
    {
        throw new System.NotImplementedException();
    }

    public bool PreventHighlight()
    {
        throw new System.NotImplementedException();
    }

    public void Setup(Socket socket)
    {
        throw new System.NotImplementedException();
    }

    public bool UseCriteria()
    {
        throw new System.NotImplementedException();
    }
}
```

6. Let's add a slot size variable that can be set on the placement criteria.

```
public class InventoryPlacementCriteria : MonoBehaviour, ISocketPlacementCriteria
{
    [SerializeField]
    private int _slotSize = 1;
```

7. Then we'll implement the **ISocketPlacementCriteria** functions. In our example, we place the InventoryItem component on the root game object of the placecable item.



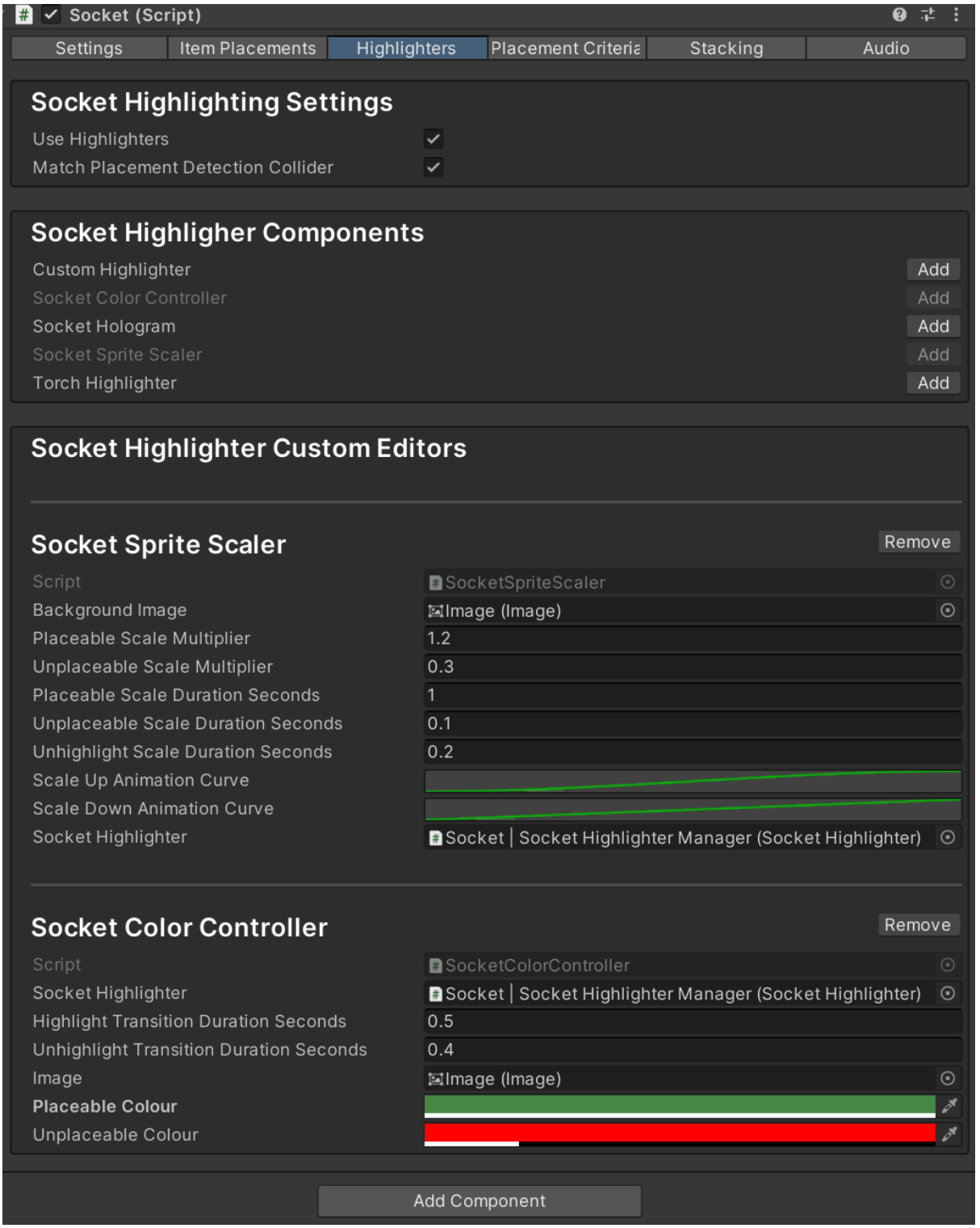
8. Now in the **CanPlace** function, we'll obtain a reference to the InventoryItem by accessing the it from the root transform of the placeable item that we're checking is

placeable.

```
public bool CanPlace(PlaceableItem placeableItem)
{
    InventoryItem inventoryItem = placeableItem.RootTransform.GetComponent<InventoryItem>();

    if (inventoryItem == null)
    {
        return false;
    }

    return inventoryItem.ItemSize <= _slotSize;
}
```

9. We don’t need to cache a reference to the socket component that’s provided in the **setup** function.
10. We don’t want to **prevent highlighters** if the item is too big for the inventory slot. This way we can display a visual que to the user that the socket has detected the placeable item but it isn’t placeable (because the inventory item is too big in this case). For this example we’ll add the built in highlighters: Socket Spirte Scaler & Socket Color controller. These highlighters have functionality to change the sprite on the socket to be green if placeable and red if not placeable. Which is why we don’t want to prevent highlighters, otherwise this functionality wouldn’t run and you’d see

nothing.

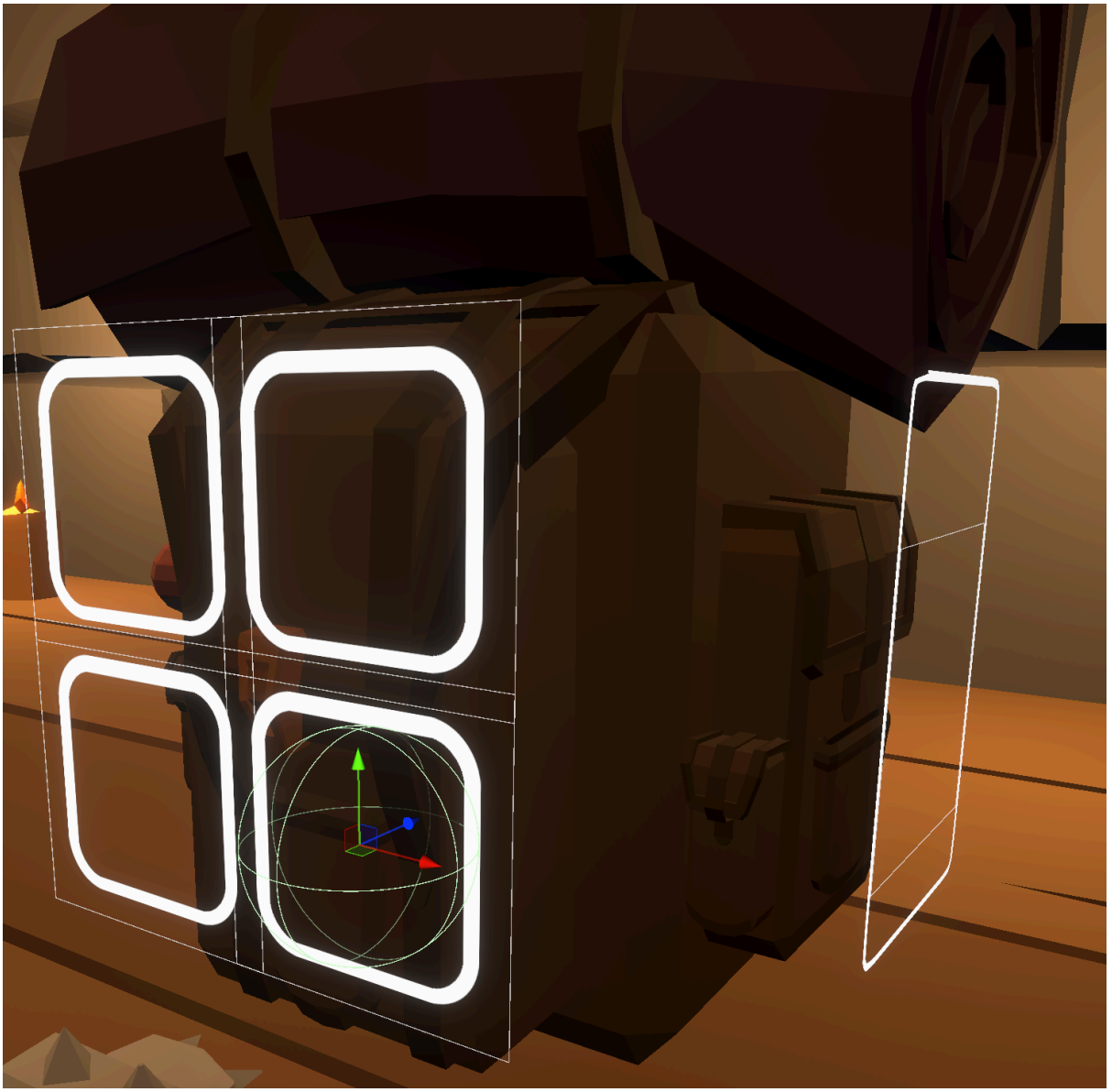


11. For the UseCriteria function, simply return true as we always want to use this criteria.

12. We now have a complete class that looks like this:

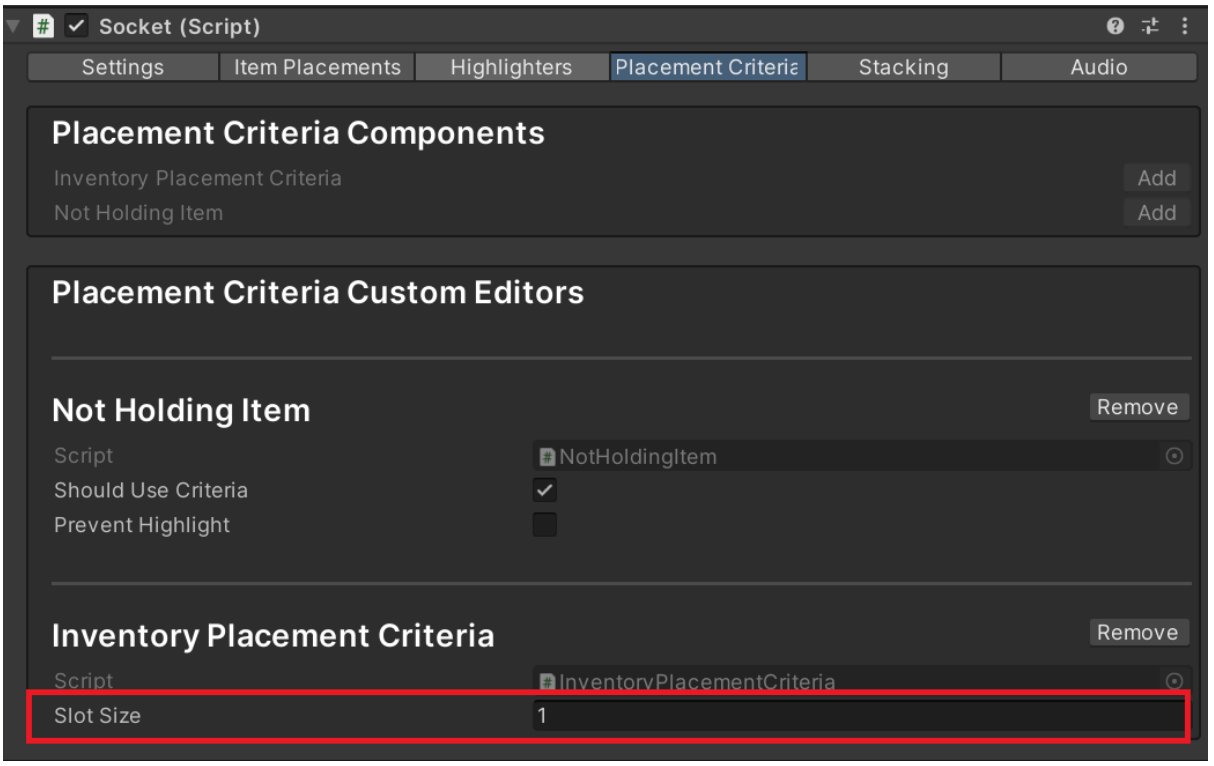
```
public class InventoryPlacementCriteria : MonoBehaviour, ISocketPlacementCriteria
{
    [SerializeField]
    private int _slotSize = 1;

    public bool CanPlace(PlaceableItem placeableItem)
    {
        InventoryItem inventoryItem = placeableItem.RootTransform.GetComponent<InventoryItem>();

        if (inventoryItem == null)
        {
            return false;
        }

        return inventoryItem.ItemSize <= _slotSize;
    }

    public void Setup(Socket socket) { } // Not required for this placement criteria

    public bool PreventHighlight() => false;

    public bool UseCriteria() => true;
}

```

13. Now we can add this criteria to our inventory backpack. That looks something like this



14. For each 1x1 slot we'll give it a size of 1. Then for the 1x2 slot on the side we'll give it a size of 2.
15. From the Placement Criteria tab on the socket we can add our new Inventory Placement Criteria (It'll show up in the list automatically). Once added we can adjust

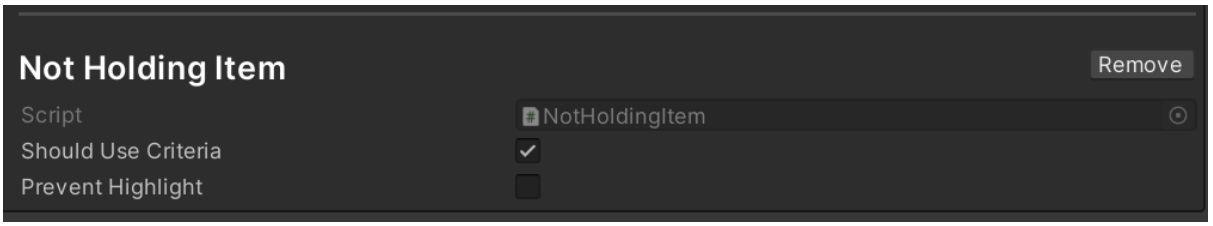
the slot size using the exposed parameter.



Now we have our Socket Placement criteria setup and working on our inventory backpack. To sum up, we created a custom placement criteria component that can be added to any inventory slot (which has a socket component) that we add to the backpack. We can then change the slot size dynamically on each slot, preventing users from placing an Axe that has an item size of 2 in the front 1x1 slots. Finally, because we didn't prevent highlighting, then a user attempts to place an item that is too big for a slot into the socket, the socket highlighters will cause the white sprite to turn red to signify it cannot be placed.

Built-in Placement Criteria

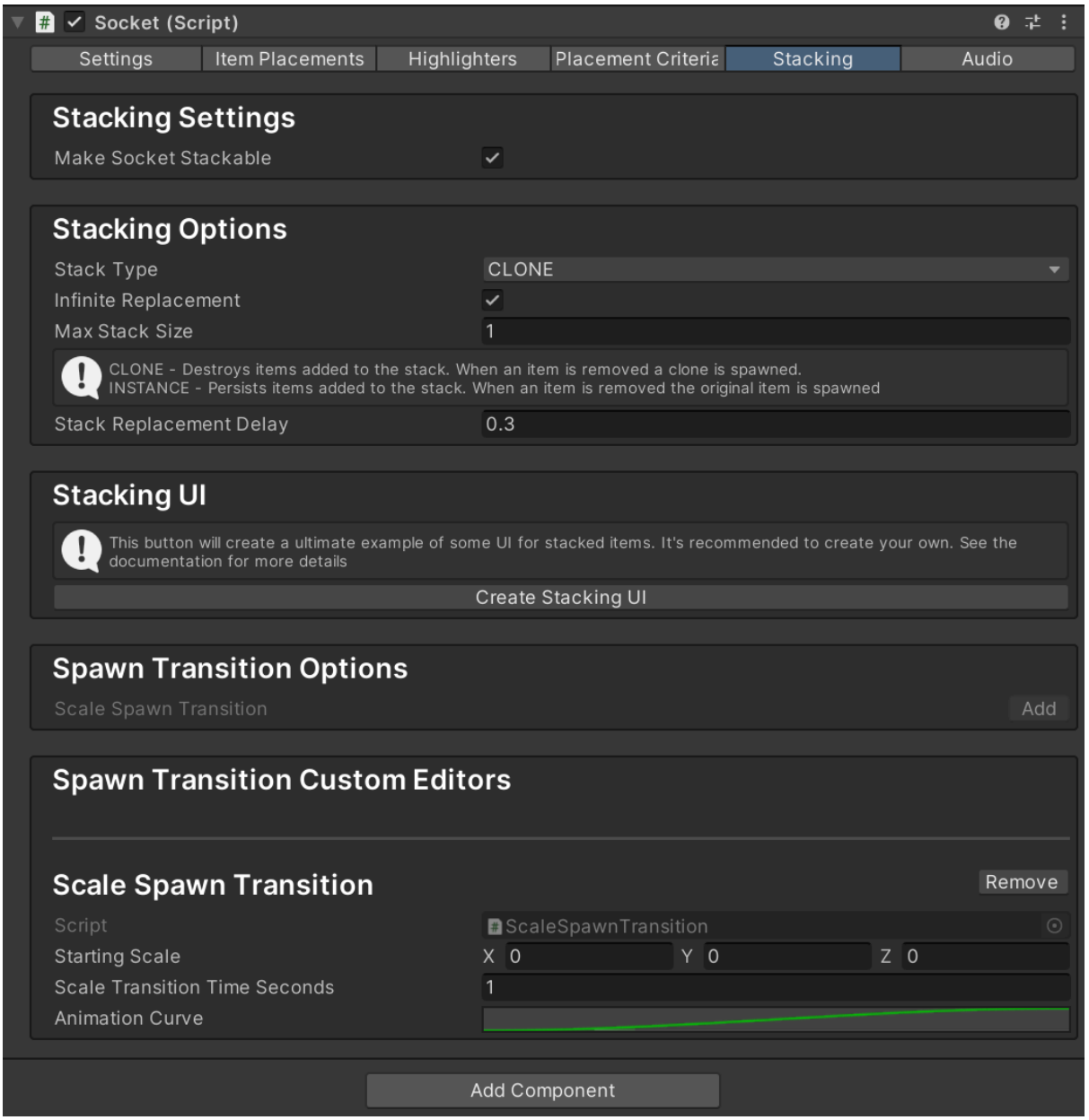
Not Holding Item



This placement criteria can be added to the socket or the placeable item. This criteria will prevent placement if the user is holding the item. Therefore it requires the user to release the placeable item from their hand to confirm the placement.

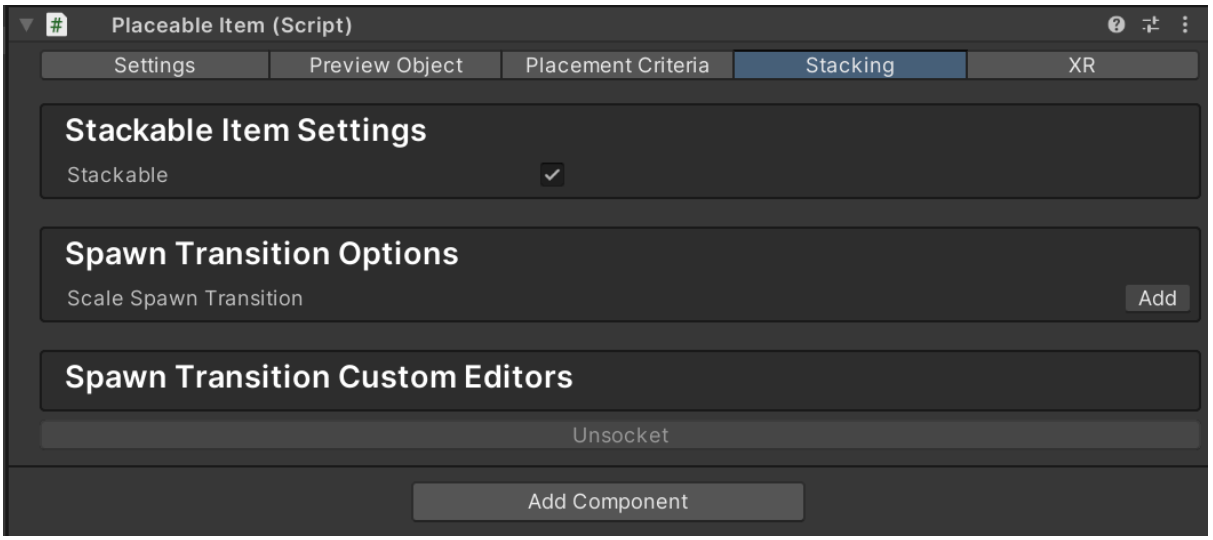
If this placement criteria isn't present on either the socket or the placeable item then it will be placed as soon as it enters the placeable area (assuming it meets the other placement requirements).

Stacking



Stacking is a feature that allows you to place multiple objects into a socket. There are many different options within the stacking tab to provide a broad range of functionality out the box.

To enable stacking functionality it needs to be enabled on the socket and the placeable item.



Stacking Settings

Make Socket Stackable - Enables stacking functionality on the socket.

Stacking Options

Stack Type - There are two options for the stack type, Clone or Instance.

Clone: The first item socketed will be used as the clone source. Any subsequent items added to the socket are destroyed and the stack count is incremented. When an item is removed from the socket a replacement item is cloned based on the source object and the stack count is decremented. When an item is removed from the socket and the stack count is 0 no more replacements will spawn (unless infinite replacement is ticked).

Instance: Any object placed in the socket is stored in a list of placed objects. All non-active items in the stack will be disabled.

Infinite Replacement - Will keep spawning replacement items even if the stack is empty. Only available for Clone stack types.

Max Stack Size - Sets a limit to how many items can be placed in the stack.

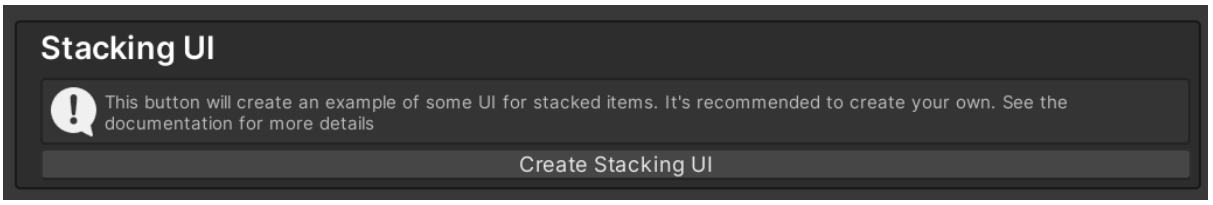
Stack Replacement Delay - Time in seconds before the replacement stack item is spawned back into the socket.

Instance Stack Type - When the stack type is set to instance you can choose between two ways of ordering the items that are placed into the socket.

FILO (First In Last Out): Meaning when you place an item into a stack it'll go to the back of the queue and you won't be able to retrieve it until all other items that were placed before it have been removed.

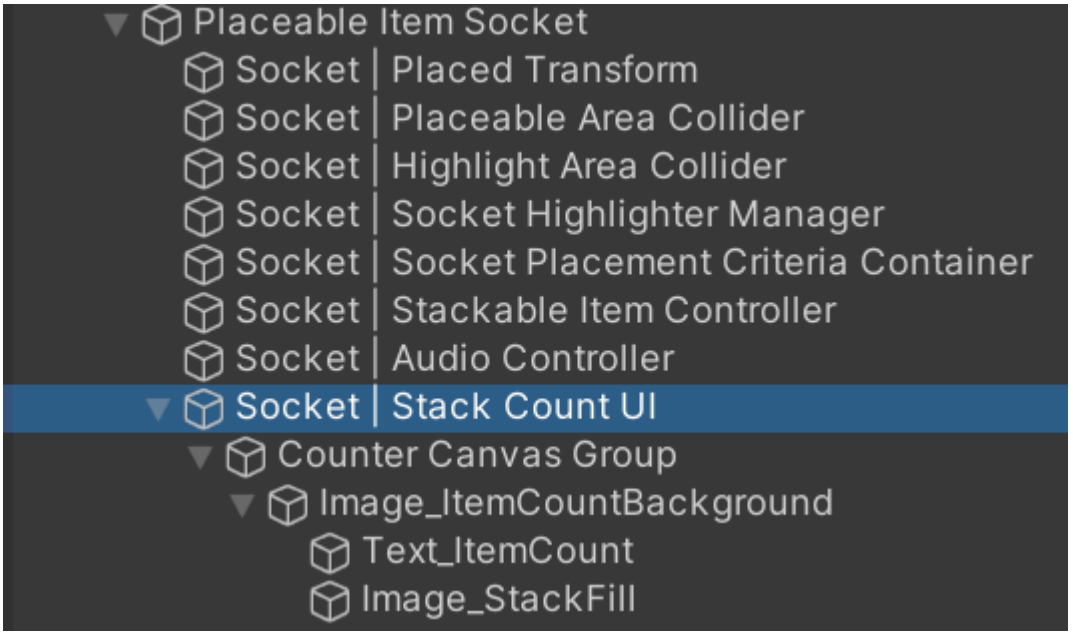
FIFO (First In First Out): Meaning when you place an item into a stack it'll go to the front of the queue. It'll push the currently displayed item back in the queue and will instantly be able to be retrieved from the stack.

Stacking UI

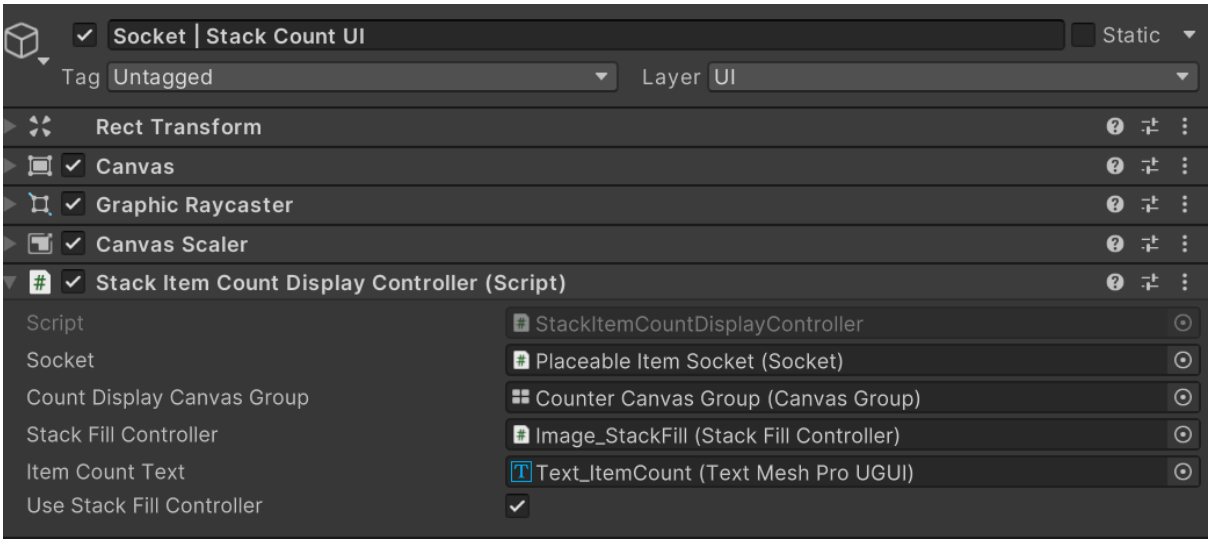


This section allows you to quickly add a piece of UI that displays the stack count. This UI is designed to provide you with a quick starting point on adding your own stack count display, if you require one in your project.

When you click the Create Stacking UI button a new game object called “**Socket | Stack Count UI**” will be created and nested under the Placeable Item Socket game object.



The key here is that on that object is a script called “**Stack Item Count Display Controller**” which will provide an example of how you can hook into the stacking controller allowing you to trigger updates when the stack size changes.

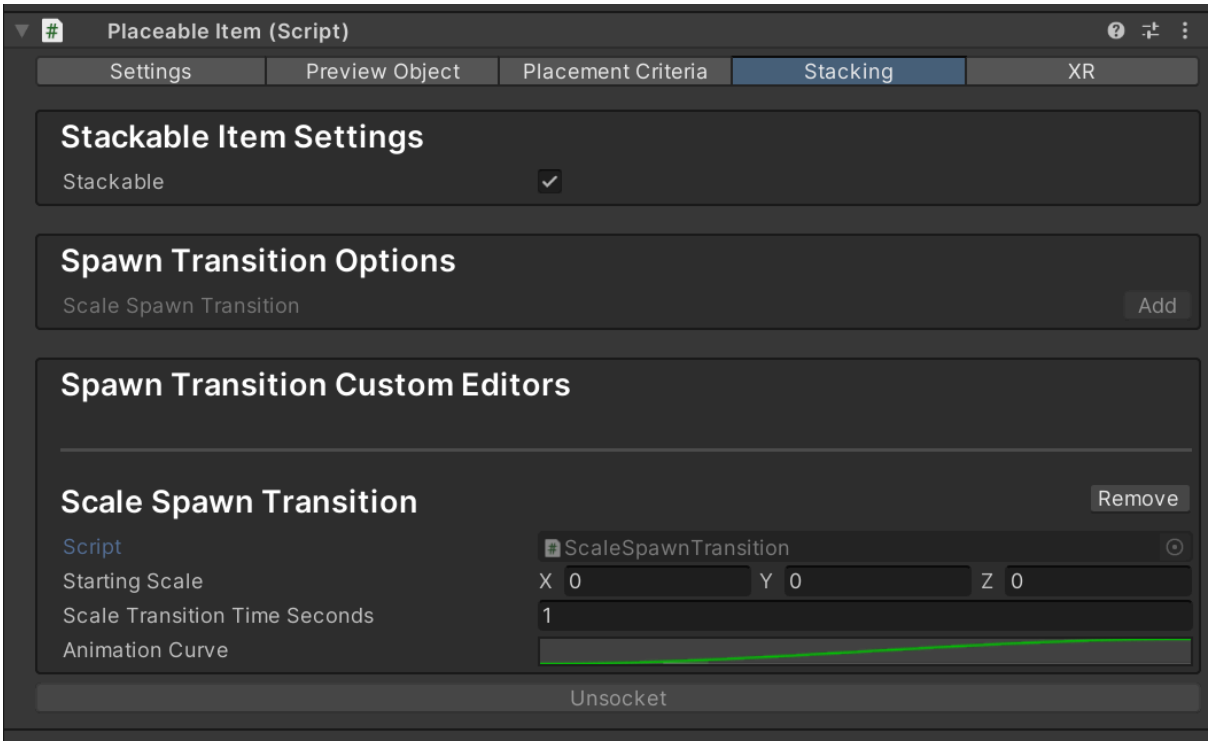


Spawn Transition Options

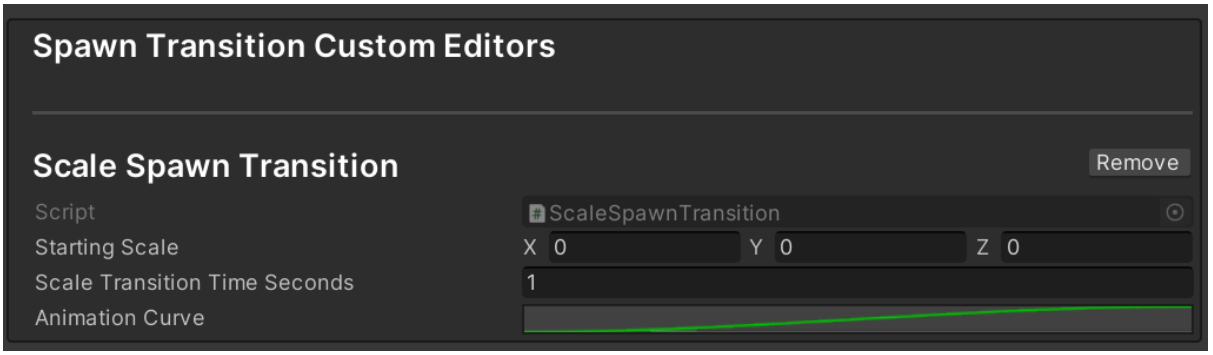
This section displays all of the spawn transitions in the project. The asset comes with a Scale Spawn Transition component but is designed for you to create your own custom spawn transitions.

A spawn transition is activated when a replacement item in the stack is displayed. This extendable system allows you to add multiple spawn transitions together to create your own bespoke replacement effect.

Spawn transitions can be added to the socket or the placeable item, allowing you to have more control over how individual items are spawned in the socket.



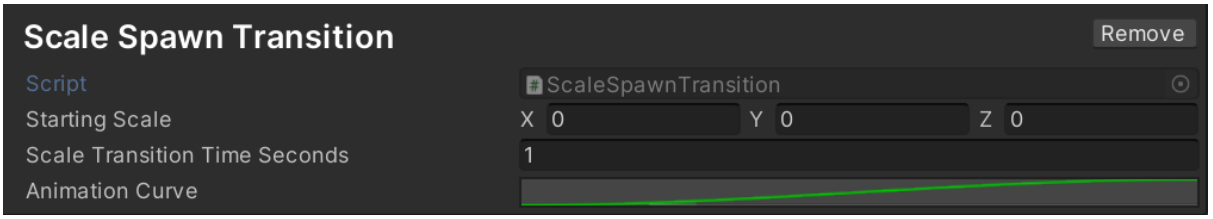
Spawn Transition Custom Editors



This section displays any spawn transition components you've added to the socket or the placeable item.

Built-in Spawn Transitions

Scale Spawn Transition



The Scale Spawn Transition is a simple component that will tween the replacement item's scale from the starting scale property to the original size. You can adjust the transition time and the animation curve to add some more juice to the tween.

How to add your own spawn transitions

To create your own spawn transition component, you just need to implement the **IStackSpawnTransition** interface onto a Monobehaviour component. As soon as you return to Unity and the scripts re-compile, your custom spawn transition just shows up in the list of spawn transition components.

```
1  public interface IStackSpawnTransition
2  {
3      public void Spawn(Socket socket, PlaceableItem placeableItem);
4      public bool IsSpawning();
5  }
```

The **IStackSpawnTransition** requires you to implement 2 functions.

Spawn - Called at the start of the spawn transition. It passes in the socket and placeable item so you can manipulate them however you need for your use case.

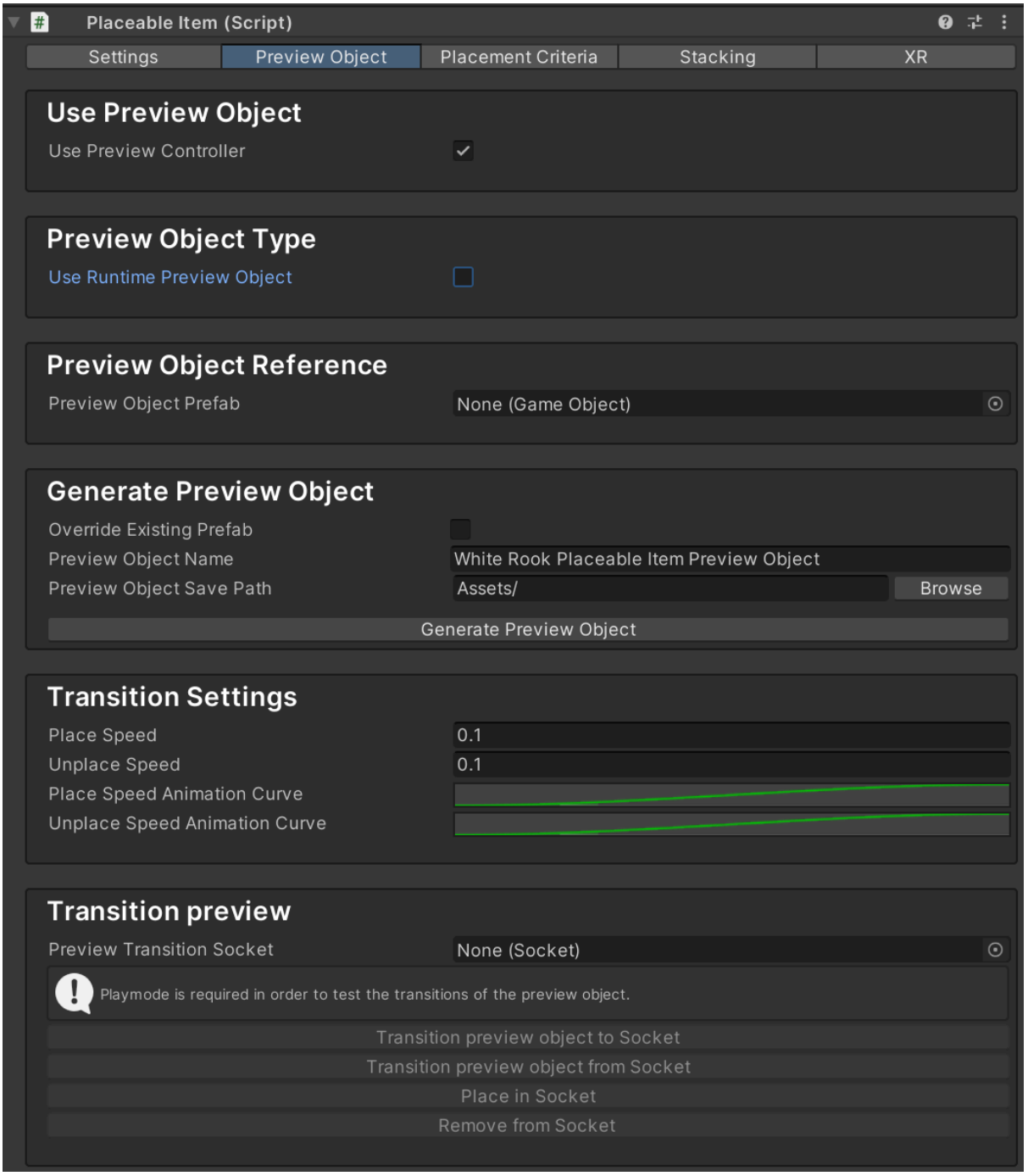
IsSpawning - This function is used to inform the stack controller if the item is performing the spawn transition.

The Scale Spawn Transition is a great simple example of how to implement the IStackSpawnTransition interface.

```
public class ScaleSpawnTransition : MonoBehaviour, IStackSpawnTransition
{
    [SerializeField]
    private Vector3 _startingScale;

    [SerializeField]
    private float _scaleTransitionTimeSeconds = 1f;

    [SerializeField]
    private AnimationCurve _animationCurve = AnimationCurve.EaseInOut(0, 0, 1, 1);

    private bool _isSpawning = false;

    public bool IsSpawning() => _isSpawning;

    private void OnEnable()
    {
        _isSpawning = false;
    }

    public void Spawn(Socket socket, PlaceableItem placeableItem)
    {
        _isSpawning = true;

        ScaleUp(placeableItem);
    }

    private void ScaleUp(PlaceableItem placeableItem)
    {
        Vector3 targetScale = placeableItem.RootTransform.localScale;

        Tweener.TweenVector3(_startingScale, targetScale, _scaleTransitionTimeSeconds, (scale) =>
        {
            if (placeableItem == null)
            {
                _isSpawning = false;
                return;
            }

            placeableItem.RootTransform.localScale = scale;
        },
        _animationCurve,
        onComplete: () => { _isSpawning = false; }
        );
    }
```

Notice how the two IStackSpawnTransition functions (Spawn and IsSpawning) have been implemented. There is a boolean called _isSpawning which tracks the state of the scaling tween.

Preview Objects



Preview Objects are a feature of the placeable item. When enabled Preview objects display a visual copy of the placeable item and tween it to the item's socketed placement. This is useful when you wish to see how an item may look in a socket without actually socketing it and triggering the socketing behaviour that would come with that.

When a preview item is displayed the original placement item is hidden which creates a smooth illusion that it's the placeable item moving to the socket.

Use Preview Object

Use Preview Controller - Determines if the preview object functionality is enabled or not.

Preview Object Type

Use Runtime Preview Object -

- **On**: The system will automatically generate a visual copy of the placeable item.
- **Off**: You'll be able to provide a specific prefab to be the preview object

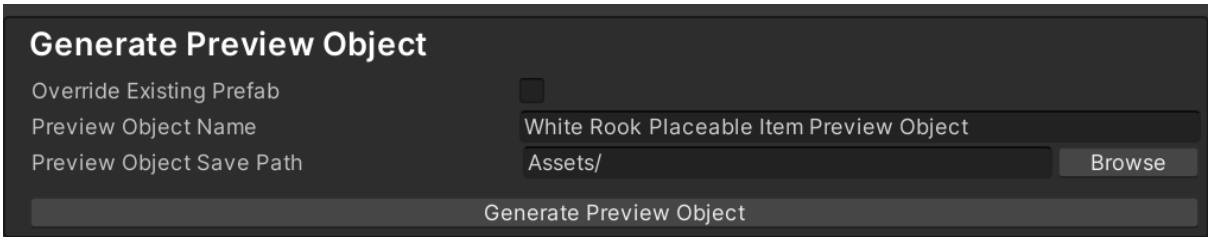
Preview Object Reference

This section becomes visible if the **Use Runtime Preview Object** is unticked. Preview objects are used as a visual, which is why the prefab you use (If not using the runtime preview object) shouldn't contain any components that contain logic that may fire the Monobehvaiour lifecycle hooks and cause undesired behaviour.

Preview Object Prefab - The prefab to use as the preview object

Generate Preview Object

This section becomes visible if the **Use Runtime Preview Object** is unticked. It's a utility section that will create a visual copy of a given object and stip off all components except the visual components (Mesh Renderer, Mesh Filter etc). Leaving you with a simple visual prefab to use as the preview object.

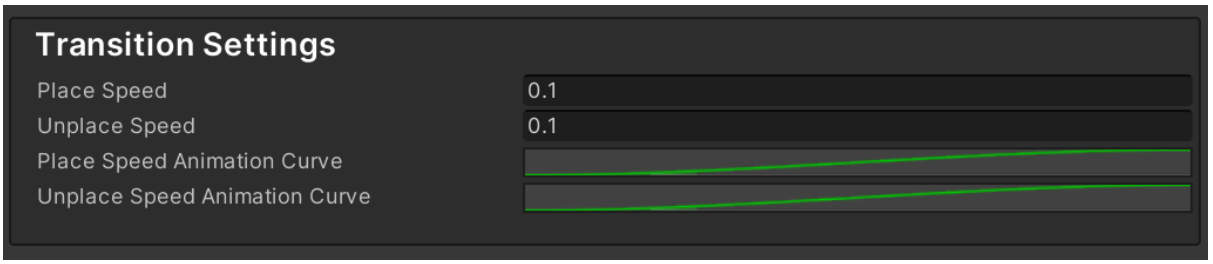


This section contains a few basic properties that are used to generate the preview object.

Override Existing Prefab: Determines if the new asset should override and existing one If there is already an asset in with the same name in the file path provided.
Preview Object Name: The name the prefab asset will be saved as.
Preview Object Save: Path: The file location where the prefab will be saved.

When you have filled in the above fields you can click the **Generate Preview Object** to create you new prefab.

Transition Settings



Defines how the preview object transitions to the socket.

Place Speed: The time in seconds for the preview object to transition from the placeable object's position to the socket position.
Unplace Speed: The time in seconds for the preview object to transition from the socket's position back to the placeable item's position.
Place Speed Animation Curve: The curve applied to the place tween.
Unplace Speed Animation Curve: The curve applied to the unplace tween.

Transition Preview

This section is an editor utility that allows you to test the transition of the placeable item's preview object to a specific socket. Useful for quickly testing how items will look going into a socket.

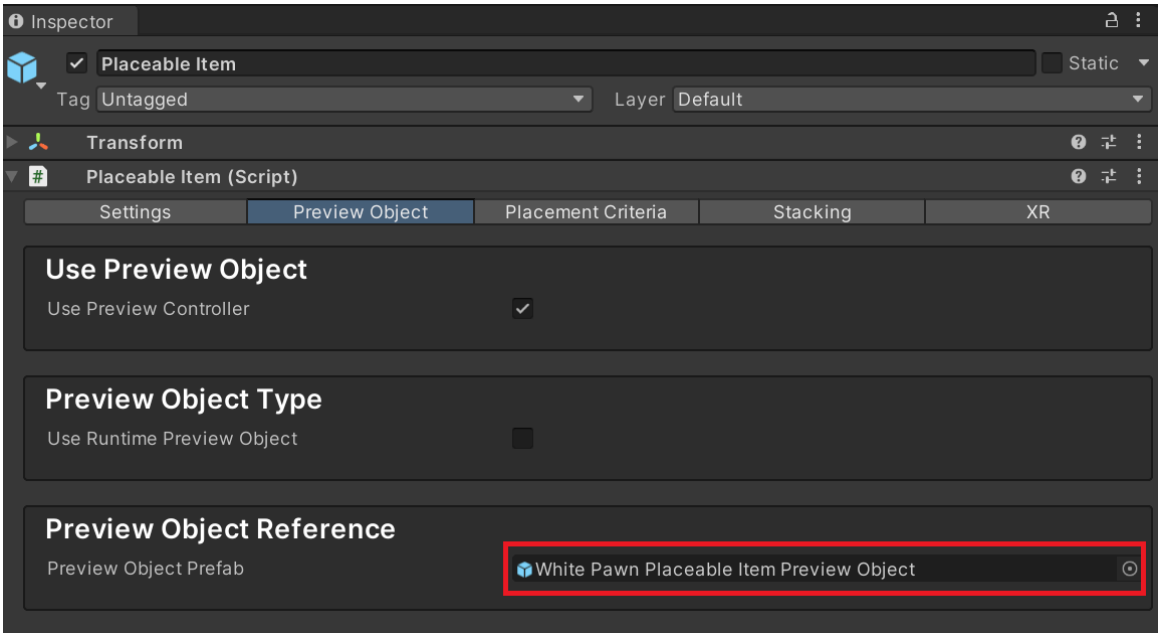
You must be in play mode for this utility to work.

Spawn Handlers

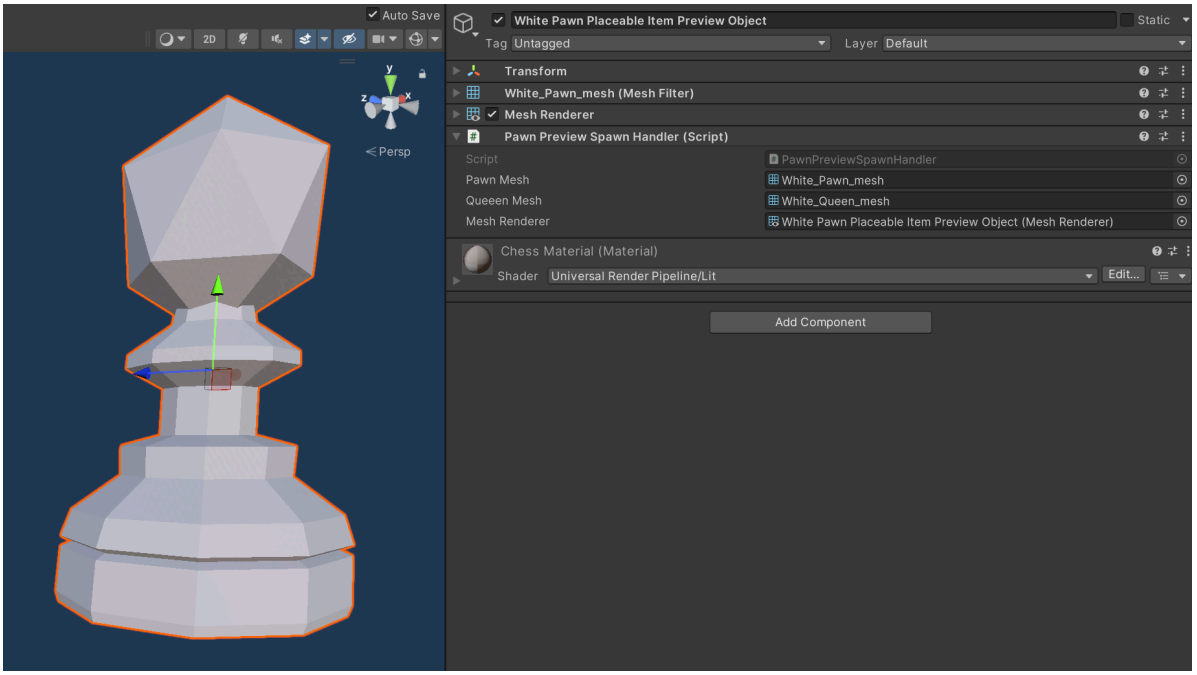
You can perform custom logic on a preview item when it is spawned. When the prefab is instantiated the system will look for any components on the item that implement the IPreviewObjectSpawnHandler interface. If a component is found then the HandlePreviewObjectSpawned function is called. The preview object, placeable item and socket are passed to the function, providing context for the function logic.

```
/// <summary>
/// Designed as an optional interface that allows components to apply specific functionality
/// </summary>
public interface IPreviewObjectSpawnHandler
{
    public void HandlePreviewObjectSpawned(GameObject previewObject, PlaceableItem placeableItem, Socket socket);
}
```

A an example of this is the chess board (shown in the promotional material). As the user moves the pawn piece up the board a preview snaps into position onto each square it nears. However at the end of the board the pawn will be promoted to a queen. To achieve this functionality you can create a dedicated prefab for the preview object.



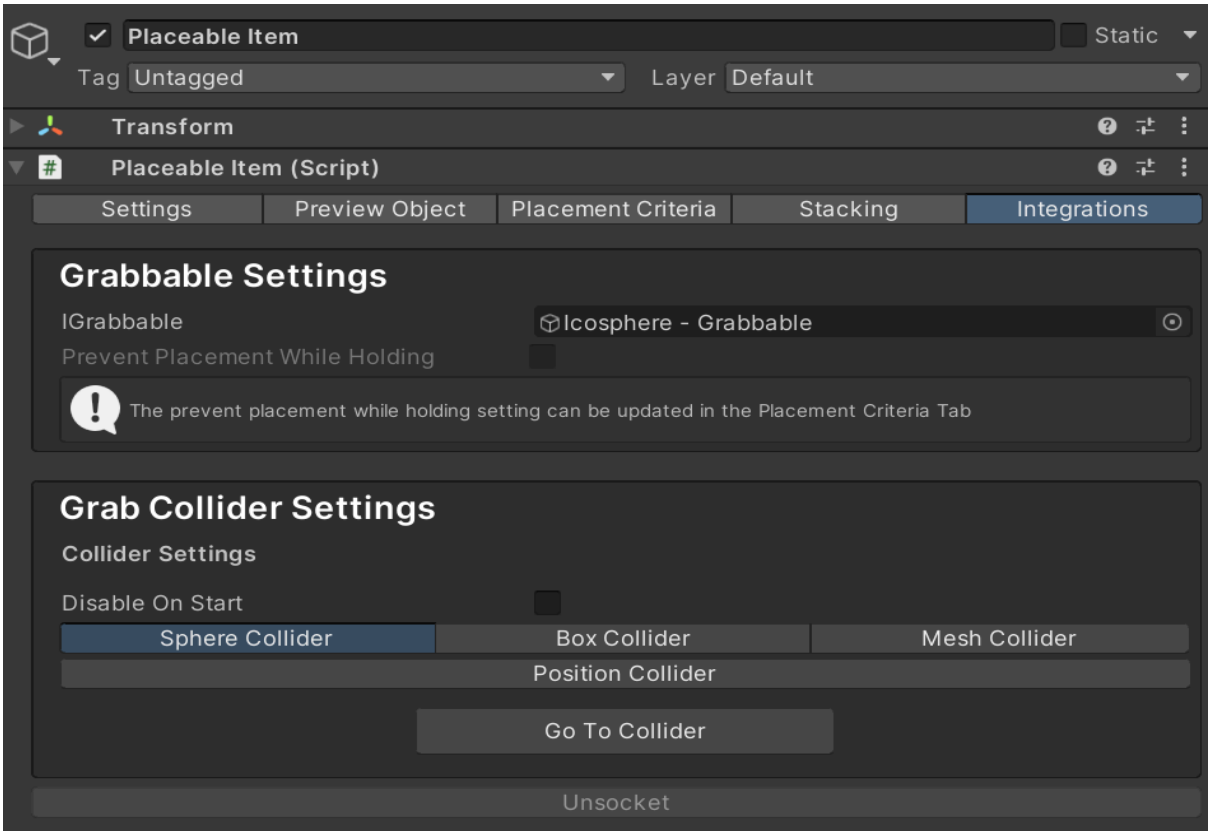
Then on the item preview prefab add a component that implements the IPreviewObjectSpawnHandler interface.




```
public class PawnPreviewSpawnHandler : MonoBehaviour, IPreviewObjectSpawnHandler
{
    [SerializeField]
    private Mesh _pawnMesh;

    [SerializeField]
    private Mesh _queeenMesh;

    [SerializeField]
    public MeshRenderer _meshRenderer;


    public void HandlePreviewObjectSpawned(GameObject previewObject, PlaceableItem placeableItem, Socket socket)
    {
        BoardSquare boardSquare = socket.GetComponent<BoardSquare>();

        if (boardSquare == null) return;

        if (boardSquare.IsEndSquare)
        {
            previewObject.GetComponent<MeshFilter>().mesh = _queeenMesh;
        }
        else
        {
            previewObject.GetComponent<MeshFilter>().mesh = _pawnMesh;
        }
    }
}

```

Using the socket reference, the preview object can find out which square is displaying the item and therefore change it’s own mesh accordingly.

Integrations Tab



The Integrations tab on the placeable item is used to configure the integration between the placeable item and the interaction framework the current project is using.

Grabbable Settings

IGrabbable - This field has to contain a reference to the game object that holds the IGrabbable integration component. (See Intergrations for more details). When you use the context menu to add the placeable item functionality to a gameobject, the system assumes that the IGrabbable implementation is already on the root game object. If this is the case it'll set the reference automatically. If it located on a different object or hasn't been added yet, you'll need to set the reference manually.

Because an exposed field cannot accept an interface as a reference, the system expects you to provide a reference to a game object which has a component that implements the IGrabbable interface on.

Prevent Placement While Holding - This is a readonly field that displays if placement is prevented while the item is being grabbed. This is based on the **Not Holding Item** placement criteria

Grab Collider Settings

The grab collider is used to detect when the user grabs the placeable item from the socket. Which, triggers the remove from socket functionality. It defaults to a small sphere collider but you may need to adjust it based on your use case.
Disable On Start - Disables the placement detection collider when the Start function is called
Collider Type - Choose which type of collider you wish for the placeable item to use to trigger the socket's detection collider.

A future enhancement to have a custom editor to display the collider properties within this section is planned. Until then, you'll need to edit the collider size on the actual collider. To locate the collider just click the "Go To Collider" button.

Audio

The socket comes with simple built-in functionality to place an audio clip when an item is placed or removed from the socket. This out the box audio feature is great for adding SFXs to sockets to see how they feel. However, each socket will create an AudioSource component on itself which isn't always desirable. As a result, you may wish to hook our own more sophisticated audio solution into the socket's placed and unplaced events using the Socket API. (See APIs for more details).

Audio Clips

Place Audio Clip - The audio clip to play when an item is placed.
Remove Audio Clip - The audio clip to play when an item is removed.

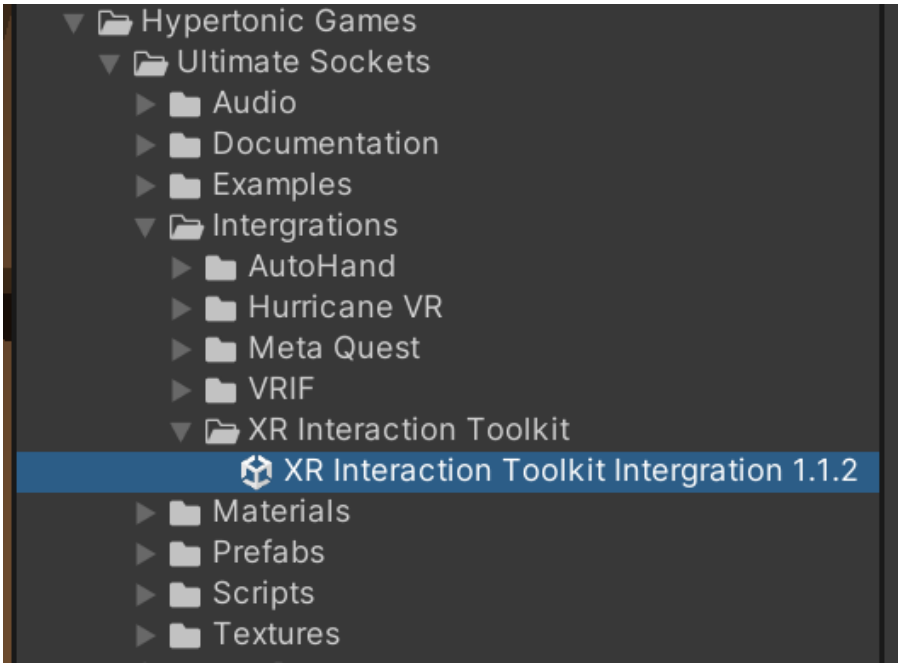
Integrations

Integrations allow the Ultimate Socket System to work with various XR Frameworks. The integration is purely to allow the Ultimate Socket System to interface with any item that is grabbable in your XR framework of choice while keeping it agnostic to any specific frameworks.

The Asset already contains integrations for:

- Unity's XR Interaction Framework (XRIT)
- Meta Quest
- Auto Hand
- HurrianceVR
- VR Interaction Framework
- Ultimate XR (coming soon)

To use an integration locate the integration folder within the asset and double click on the Unity Package for your chosen integration. It's recommended to ensure you have the XR Framework already in the project otherwise this integration will throw an error.



How to create your own XR Integration

If you use a framework that isn't yet supported of have your own custom framework, you can create your own integration by implementing the **IGrabbableItem** interface. The key to this is to identify your XR framework's grabbable component and get a reference to it in the integration script. Then you'll be able to hook into that grabbable component and interface with the socket system.

The XR's framework's grabbable component will typically be whatever component you add to a game object to allow you to pick it up. In the case of Unity's XR Interaction Toolkit, it's the **XRGrabInteractable**.

```
public interface IGrabbableItem
{
    public event IGrabbableItemEvent OnGrabbed;
    public event IGrabbableItemEvent OnReleased;

    public void Enable();
    public void Disable();
    public bool IsGrabbing();
    public void HandleRemovedFromSocket(Socket socket, PlaceableItem placeableItem);
}
```

Above is the **IGrabbaleItem** interface which you'll need to implement onto a Monobehaviour.

OnGrabbed - Invoke this event whenever the grabbale item is grabbed.
OnReleased - Invoke this event whenever the grabbable item is released.

Enable - Implement logic to enable the grabbable component / functionality.
Disable - Implement logic to disable the grabbable component / functionality.
IsGrabbing - Implement logic to return if the grabbable item is currently being grabbed.
HandleRemovedFromSocket - Allows you to hook into the removed from socket flow to perform any logic required when an item is removed. Some frameworks such as the XRIT don't need to implement this function at all. Others, such as Meta Quest use this function to re-initialise Meta's grabbable component.

```cs
public class XRITGrabbableItem : MonoBehaviour, IGrabbableItem
{
    [SerializeField]
    private XRGrabInteractable _grabbableInteractable;

    public event IGrabbableItemEvent OnGrabbed;
    public event IGrabbableItemEvent OnReleased;

    #region Unity Functions

    private void Awake()
    {
        if (_grabbableInteractable == null && !TryGetComponent(out _grabbableInteractable))
        {
            Debug.LogError("The XR Grab Interactable component could not be found on the game object", this);
            return;
        }
    }

    private void OnEnable()
    {
        _grabbableInteractable.selectEntered.AddListener(HandleGrabbed);
        _grabbableInteractable.selectExited.AddListener(HandleReleased);
    }

    private void OnDisable()
    {
        _grabbableInteractable.selectEntered.RemoveListener(HandleGrabbed);
        _grabbableInteractable.selectExited.RemoveListener(HandleReleased);
    }

    #endregion

    #region IGrabbableItem Functions

    public void Disable()
    {
        _grabbableInteractable.enabled = false;
    }

    public void Enable()
    {
        _grabbableInteractable.enabled = true;
    }

    public bool IsGrabbing()
    {
        return _grabbableInteractable.isSelected;
    }

    public void HandleRemovedFromSocket(Socket socket, PlaceableItem placeableItem) { }

    #endregion IGrabbableItem Functions

    #region Private Functions

    private void HandleGrabbed(SelectEnterEventArgs obj)
    {
        OnGrabbed?.Invoke();
    }

    private void HandleReleased(SelectExitEventArgs obj)
    {
        OnReleased?.Invoke();
    }

    #endregion Private Functions
}
```

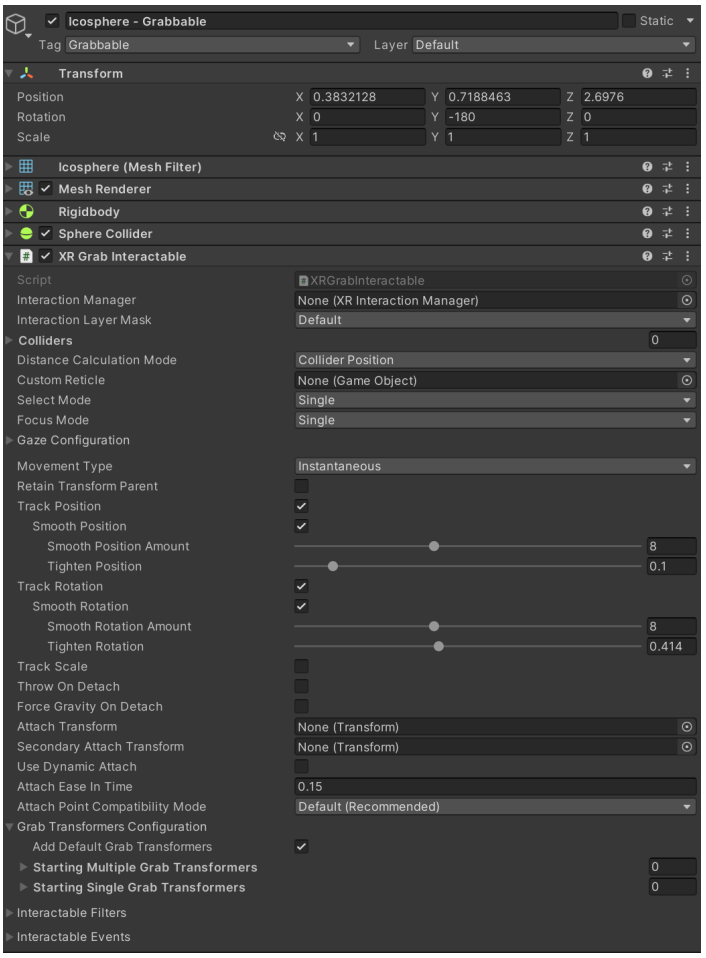
Here is the XRIT integration script. It will serve as a good example to demonstrate how you may wish to implement your own integration. The class implements the **IGrabbableItem** interface. It has a single property which is an exposed reference to the grabbable item, in this case, it's the XRIT's XRGrabInteractable component. It used Unity's OnEnable and OnDisable lifecycle hooks to subscribe to the grab and release events of the XRGrabInteractable (in this case via the selectEntererd and selectExited properties). Which in turn raises the OnGrabbed and OnRelease events for the **IGrabbableItem** implementation. It implements the Enable and Disable functions from the **IGrabbableItem** interface to allow the Unlimited Socket System to toggle grabbable functionality in a way that's best suited for the XR framework, in this case simply enabling and disabling the XRGrabInteractable is sufficient.

Finally, the nice thing about implementing the IGrabbableItem interface on a new script like the above is that it keeps the responsibility of the integration in a single component that can be easily added to any gameobject.

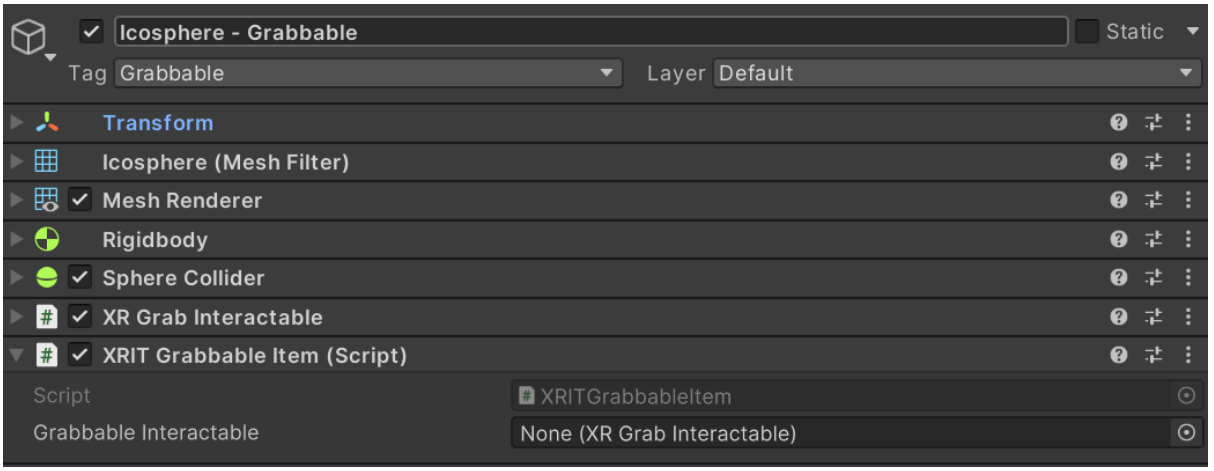
Where to hook up your integration component

For this example, we'll use Unity's XR Interaction Framework.

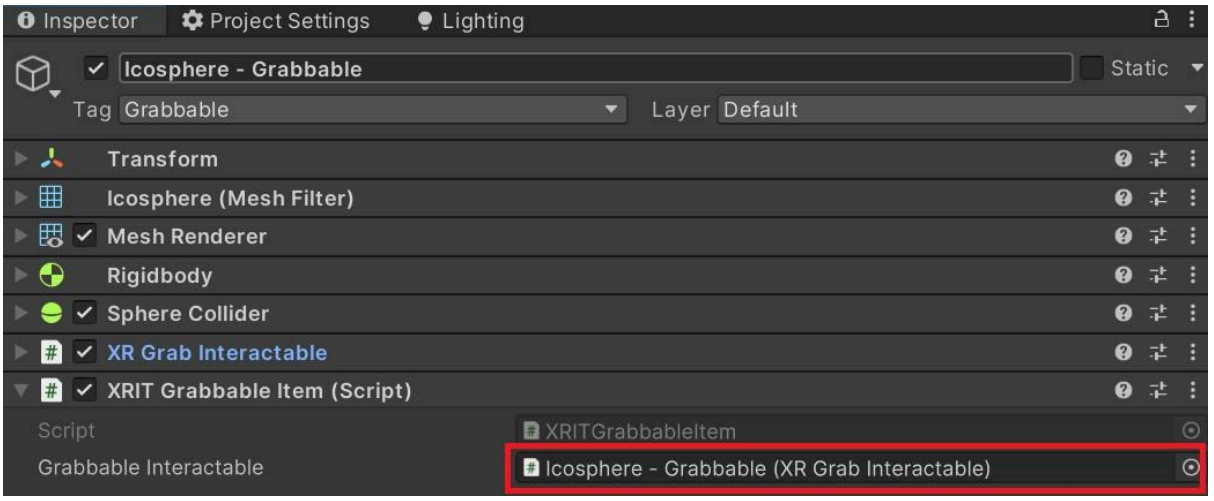
1. Firstly ensure you have an integration script for your XR framework of choice.
2. Locate your grabbable object in the scene.



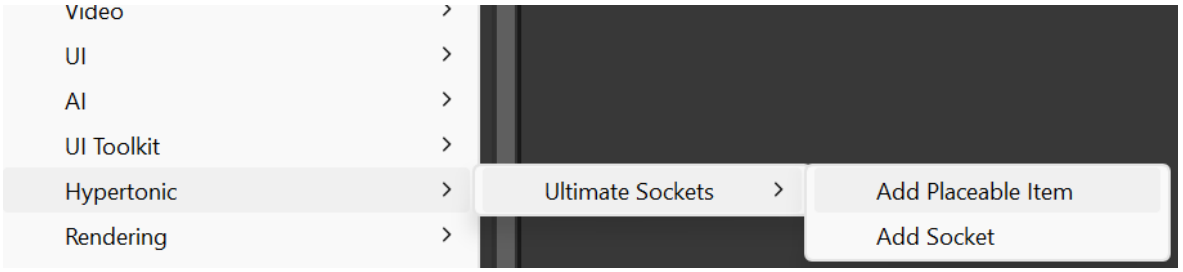
3. Next, add your integration script to the game object.



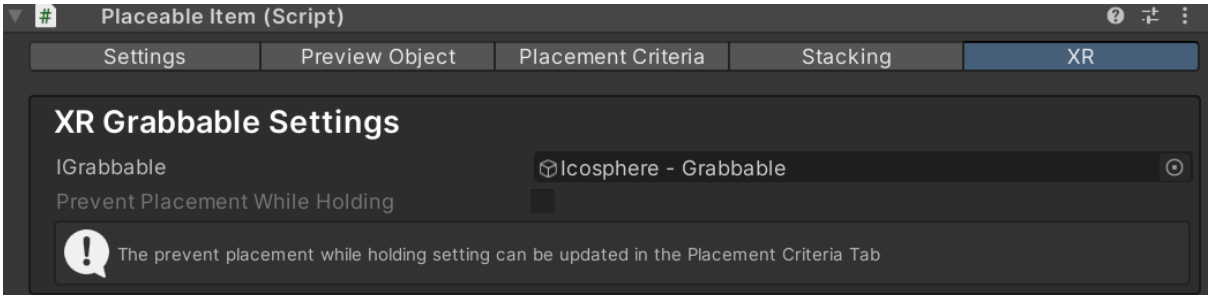
4. Then drag the reference to the grabbale item, in this case the XR Grab Interactable.



5. Next, add the placeable item to the grabbable object using the context menu.



6. The placeable item will automatically set a reference to the integration component which can be seen on the XR Tab of the placeable item.



7. If this field isn't automatically populated when you add the placeable item, or if you have added the placeable item before adding the integration; you can resolve this by dragging a reference to the game object where the integration script is attached.

Core Components and APIs

You'll be able to access the majority of functionality programmatically through the two core scripts within this asset; **Socket.cs** and **PlaceableItem.cs**.

Placeable Item

The placeableItem.cs script is the core entry point for anything related to the item. Via it's public properties you should be able to access all sub components of the placeable item.

Events

OnPlaced: Invoked when the item has been placed.
OnRemovedFromSocket: Invoked when the item has been removed from the socket.
OnEnteredPlaceableZone: Invoked when an item enters the placeable area.
OnExitedPlaceableZone: Invoked when the item has left the placeable area.

Public Properties

ItemTag (string): The designated tag of the placeable item.
Placed (bool): If the item is placed in a socket.
WithinPlaceableZone (bool): If the item is within the placeable area.
KeepScaleForDefaultPlacements (bool): If the item's current scale should be used when placed into a socket that doesn't have a configuration for the items tag setup.
RootTransform (Transform): Provides a reference to the root transform of the grabbable object.
NonSocketColliders (List<Collider>): Provides a list of all the colliders on the grabbable object that aren't used in the socketing process. They will be disabled when placed.
SocketGrabCollider (SocketGrabCollider): A reference to the SocketGrabCollider which is used to detect when the item is grabbed while in the socket.
PlaceableItemCollider (PlaceableItemCollider): A reference to the component that manages the collider that triggers the socket's placement functionality.
PlaceableItemPlacementCriteriaController: (PlaceableItemPlacementCriteriaController): A reference to the component that handles the placement criteria
PlaceableItemPreviewController: (PlaceableItemPreviewController): A reference to the component that handles the preview object functionality.
PlaceableItemSocketScaler (PlaceableItemSocketScaler): A reference to the component that handles the scaling of the placeable item.
PlaceableItemRigidbody (PlaceableItemRigidbody): A reference to the component that handles the rigid body of the grabbable item.

PlaceableItemPlacementController (PlaceableItemPlacementController): A reference to the component that handles the placement of the item.
StackableItemController (StackableItemController): A reference to the component that handles the stackable functionality of the placeable item.
PlaceableItemGrabbable (PlaceableItemGrabbable): A reference to the component that handles the XR framework integration.
DestroyOnPlace (bool): If the item will be destroyed when placed.
ClosestSocket (Socket): The closest socket to the placeable item. Will be null unless the placeable item is within a socket's placeable area.
CurrentSocketHighlighter (SocketHighlighter): The active socket highlighter component triggered by this placeable item.
PreventPlacement (bool): Prevent the item from being placed. This is used by the editor utility to test the preview transitions without placing the item. If you want to prevent an item placement at runtime it's recommended to use the Placement Criteria functionality.
UsePreviewController (bool): If the preview object functionality is active or not.

Public Functions

Enable (): Enables placeable functionality.
Disable (): Disables placeable functionality.
HandleEnteredPlaceableZone (): Called when the item enters the socket's placeable area.
HandleLeftPlaceableZone (): Called when the item leaves the socket's placeable area.
PlaceInSocket (): Places the item into the closest socket.
PlaceInSocket (Socket): Places the item into the specific socket.
HandlePlaced(Socket): Called when the item has been placed into a socket.
RemoveFromSocket (): Removes the item from the socket.
SetSocketHighlighter(SocketHighlighter highlighter): Stops any current highlighter and sets the current socket highlighter to the new one.
ClearSocketHighlighter(): Clears the current socket highlighter reference.
PreventItemRemoveal(): Turns off colliders to prevent the item from being grabbed out of the socket.
EnableItemRemoval(): Turns on colliders to allow for the item to be grabbed from the socket.

SocketGrabCollider

The component responsible for detecting when the user grabs the item out of the socket.

Public Properties

ColliderManager (ColliderManager): A reference to the collider manager component.
Collider (Collider): A reference to the socket grab collider.

Pubic Functions

EnableCollider(): Enables the grab collider.
DisableCollider(): Disables the grab collider.

PlaceableItemCollider

The component responsible for managing the collider that triggers the Socket's placement detection functionality.

Public Properties

PlaceableItem (PlaceableItem): A reference to the placeable item.
Collider (Collider): A reference to the socket detection collider.
Enabled (bool): The state of if placeable item collider.
ColliderManager (ColliderManager): A reference to the collider manager.

Public Functions

Enable(): Enables the functionality of the component.
Disable(): Disables the functionality of the component.
EnableCollider(): Enables the placeable item collider.
DisableCollider(): Disables the placeable item collider.

PlaceableItemPlacementCriteriaController

The component responsible for managing the placement criteria of the placeable item.

Public Properties

Criterias (List<CriteriaEntry>): A list of all the placement criteria.

Public Functions

CanPlace (List<string> ignoreCriteraNames): Returns if the criteria are met, ignoring any criteria in the ignoreCriteriaNames list.
CanHighlight(): Returns if the criteria allows highlighting.

RemovePlacementCriteria (string criteriaName): Removes the criteria with the provided name.
HasCriteria(string criteriaName): Returns true if the component contains the specific criteria.

PlaceableItemPreviewController

The component responsible for the preview object functionality.

Public Properties

PreviewObjectPrefab (GameObject): The prefab to be displayed as the preview object.
PreviewObjectSaveFilePath (string): Where the utility tool will save the asset.
UseRuntimePreviewObject(bool): If the component will create a preview object at runtime dynamically or uses the specified prefab.

Public Functions

HandleEnteredPlaceableZone(Socket socket): Called when the item enters the socket's placeable area.

PlaceableItemMeshController

The component responsible for handling the mesh visibility of the placeable item.

Public Properties

HasActiveRenderers (bool): Returns if the object has active renders.

Public Functions

EnableMeshRenderers(): Enables all the active renderers that were disabled.
DisableMeshRenderers(): Disables all renderers on the placeable item.

PlaceableItemSocketScaler

The component responsible for handling the scale of the placeable item when it's unsocked.

Public Events

OnReturnedToOriginalScale: Called when the placeable item has returned to it's original scale.

Public Properties

OriginalLocalScale (Vector3?): The local scale of the object before it was socketed.

PlaceableItemRigidbody

The component responsible for managing the placeable it’s rigidbody.

Public Functions

SetKinematic(bool state): Set the kinematic state of the rigidbody to the specified state.

PlacecableItemPlacementController

The component responsible for placing the item into the socket if all the conditions are met.

StackableItemController

The component responsible for stackable functionality on the placeable item.

Public Properties

Stackable (bool): If the placeable item is stackable.
Spawning (bool): If the placeable item is currently spawning.
SpawnTransitionCount (int): How many spawn transitions the controller has.
SpawnTransitionEntries (List<SpawnTransitionEntry>): A reference to the spawn transitions.

Public Functions

AddItemSpawnTransitionController(): Adds a spawn transition controller.
AddSpawnTransition(string transitionName, Type transitionType): Adds a new spawn transition.
RemoveSpawnTransition(string transitionName): Removes the specified spawn transition.
HasSpawnTransition(string transitionName): Returns true if the controller has the specified spawn transition.
ActivateSpawnTransitions(Socket socket, PlaceableItem placeableItem): Activates all spawn transitions on the item.
SetIsSpawning(bool IsSpawning): Sets a variable to store if the item is currently spawning.

PlaceableItemGrabbable

The component responsible for interfacing with the Grabbale component of the XR Framework.

Public Properties

GrabbableItem (IGrabbableItem): A reference to the IGrabbableItem interface attached to the integration component.
GrabbableItemGameObject (GameObject): The game object that has the integration component attached.

Public Functions

RemoveFromSocket(): Removes the item from the socket. Called when the placeable item is grabbed.
EnableGrabbable(): Tells the integration component to enable grabbable functionality on the grabbable component
DisableGrabbable(): Tells the integration component to disable grabbable functionality on the grabbable component.

Socket

The Socket script is the core entry point for all socketing behaviour. Via its public properties, you'll be able to reference all other socket sub-components.

Events

OnItemPlaced: Invoked when an item is placed into the socket.
OnItemRemoved: Invoked when an item is removed from the socket.
OnItemEnteredPlaceableAreaCollider: Invoked when an item triggers the placeable area collider.
OnItemLeftPlaceableAreaCollider: Invoked when an item leaves the placeable area collider.

OnPreventItemPlacementChanged: Invoked when the state of PreventItemPlacement changes.
OnPreventPhysicalItemPlacementChanged: Invoked when the state of PreventPhysicalItemPlacement changes.
OnPreventItemRemovalChanged: Invoked when the state of PreventItemRemoval changes.

OnStackSizeChanged: Invoked when the size of the stack changes.

Public Properties

IsSetup (bool): If the socket has been set up - set to true on Start.
PlacedItem (PlaceableItem): The item that's been placed in the socket.
IsHoldingItem (bool): If the socket currently has a placed Item.
SocketPlaceCollider (SocketPlaceColldier): A reference to the SocketPlaceCollider.
SocketPlaceTransform (SocketPlaceTransform): A reference to the SocketPlaceTransform.

SocketPlacementProfileScriptableObject (SocketPlacementProfileScriptableObject): A scriptable object that holds the placement profile data.
SocketPlacementProfile (SocketPlacementProfile): A reference to the placement profile, either the scriptable object version or the instance variable based on selected settings.
PlacementProfileType (PlacementProfileType): Which placement profile option has been selected.
SocketHighlighter (SocketHighlighter): A reference to the SocketHighlighterComponent.
SocketHighlightAreaCollider (SocketHighlightAreaCollider): A reference to the SocketHighlightAreaCollider component.
SocketPlacementCriteriaController (SocketPlacementCriteriaController): A reference to the SocketPlacementCriteriaController.
SocketStackableItemController (SocketStackableItemController): A reference to the SocketStackableItemController.
SocketAudioController (SocketAudioController): A reference to the SocketAudioController.
AnyTagAllowed (bool): If an item with any tag can be placed into the socket.
AllowedTags (List<string>): A list of item tags that the socket will accept.
PreventPreviewItems (bool): Prevents preview items from displaying from the placeable object that enters the placeable area.
PlaceRadius (float): Returns the radius of the placeable collider.
DefaultItemPlacementConfig (ItemPlacementConfig): The placement config to use when there isn't a config set up for a placeable item.
PlaceOnStart (PlaceableItem): The item that should be socketed when the Start lifecycle hook fires.
PreventItemPlacement (bool): If the socket should prevent items from being placed.
PreventPhysicalItemPlacement (bool): If the socket should only allow socketing programmatically.
PreventItemRemoval (bool): If the socket should prevent the placeable item from being removed.

Public Functions

Enable(): Enabled socketing functionality.
Disable(): Disabled socketing functionality.
PlaceItem(PlaceableItem placeableItem, bool invokeEvent): Places a placeable item into the socket.
PlaceSingleItem(PlaceableItem placeableItem, bool invokeEvent): Places a placeable item into the socket bypassing the stacking functionality.
RemoveItem(): Releases the placeable item from the socket.
RemoveItem(PlaceableItem placeableItem): Removes the placeable item from the socket.
RemoveSingleItem(PlaceableItem placeableItem): Removes the placeable item from the socket, bypassing the stacking functionality.
SetAllowedTag(string tag): Adds an allowed tag to the allowed tag list.
SetNotAllowedTag(string tag): Removes an allowed tag from the allowed tag list.
SetAllowedTags (List<string> tags): Sets the allowed tag list overwriting the original.

CanPlace(PlaceableItem placeableItem, List<string> placementCriteriaNamesToIgnore): Returns if the placeable item has met all the conditions to be placed into the socket.

SocketPlaceCollider

The component responsible for managing the collider that detects when a placeable item is near.

Events

OnItemWithinPlaceableArea: Invoked when a placeable item enters the place collider.
OnItemLeftPlaceableArea: Invoked when a placeable item leaves the place collider.

Public Properties

ColliderManager (ColliderManager): A reference to the ColliderManager component.
Collider (Collider): A reference to the place collider.
IsEnabledAndActiveInHieraracy (bool): Returns if the place collider is active and enabled.
IsEnabled (bool): If the detection collider is enabled.

Public Functions

Enable(): Enables collider functionality.
Disable(): Disables collider functionality.

SocketPlaceTransform

The component responsible for the placement of an item in a socket.

Public Functions

PlaceItem(PlaceableItem placeableItem): Places the item into the socket.
GetPlacementPositionForItem (TransformData): Returns the transform data of a placement for a specific item. Will return the default placement if no config is set for the specified item.
HasPlacementPositionForItem (string itemTag): Returns true if there is a placement config set for the specified item.
KeepDefaultScale(): Returns if the socket should keep the scale of the placeable item when socketed if using the default config.

SocketHighlighter

The component responsible for the highlighting functionality on the socket.

Public Properties

Socket (Socket): A reference to the socket.
UseHighlighter (bool): If the socket should use highlighter functionality.
SocketHighlighters (List<HighlighterEntry>): A list of each assigned highlighter.

SocketHighlightAreaCollider

Events

OnPlaceableItemNear: Invoked when a placeable item enteres the socket's highlight collider area.
OnPlaceableItemStay: Invoked while a placeable item is within the socket's highlight collider area.
OnPlaceableItemLeftArea: Invoked when a placeable item leaves the socket's highlight collider.

Public Properties

ColliderManager (ColliderManager): A reference to the ColliderManager component.
DetectionCollider (Collider): A reference to the collider that detects items in the highlight area.

Public Functions

Enable(): Enables collider functionality.
Disable(): Disables collider functionality.

SocketPlacementCriteriaController

The component responsible for handling the placement criteria functionality.

Public Properties

Criterias (List<CriteriaEntry>): A reference to the list of criterias assigned to the socket.

Public Functions

CanPlace (PlaceableItem placeableItem, List<string> placementCriteriasToIgnore): Returns true if all of the placement criteria are met. Ignoring any criteria specified in the placementCriteriasToIgnoreList.
CanHighlight(PlaceableItem placeableItem): Returns true if the conditions for highlighting are met.

AddPlacementCriteria(string criteriaName, Type criteriaType): Adds a placement criteria to the criteria list.
RemovePlacementCriteria (criteriaName): Removes the specified criteria name from the criteria list.
HasCriteria (string criteriaName): Returns if the criteria list contains the specified criteria.

SocketStackableItemController

The component responsible for stacking functionality on the socket.

Events

OnStackableItemAdded: Invoked when a placeable item is socketed and added to the stack.
OnStackableItemRemoved: Invoked when a placeable item is removed from the stack.
OnStackSizeChanged: Invoked when the stack size changes due to an item being added or removed from the stack.
OnItemAddedToCloneStack: Invoked when an item is added and the stack type is set to “Clone”.
OnItemAddedToInstance: Invoked when an item is added and the stack type is set to “Instance”.

Public Properties

SpawnTransitionEntries (List<SpawnTransitionEntry>): A reference to the list of spawn transitions assigned to the socket.
Settings: (SocketStackableItemControllerData): The settings that define how the stackable functionality behaves.
StackSize (int): The amount of items in the stack.
IsEmpty (bool): If the stack has no items in.
IsFull (bool): If the stack is at it’s limit and no more items can be added.
StackCounterUI (GameObject): A reference to the counter UI game object that’s added via the custom editor.

Public Functions

CanAddToStack(): Returns true if the conditions are met for an item to be added to the stack.
AddToStack (PlaceableItem placeableItem, bool invokeEvent): Adds a placeable item to the stack.
RemoveFromStack (PlaceableItem placeableItem): Removes an item from the stack.
AddItemSpawnTransitionController (): Adds a SpawnTransitionController if one does not already exist.
AddSpawnTransition (string transitionName, Type transitionType): Adds a spawn transition to the list.

RemoveSpawnTransition (string transitionName): Removes the specified spawn transition from the list.
HasSpawnTransition (string transitionName): Returns if the specified spawn transition is in the list.

SocketAudioController

The component responsible for handling audio on the socket.

Public Properties

UseAudio (bool): Returns if audio functionality is enabled on the socket.
AudioSource (AudioSource) A reference to the socket's AudioSource component.

Public Functions

HandleUseAudioChanged (): Updates audio functionality on the component when the UseAudio property changes in the editor.
SetSocket (Socket socket): Sets a reference to the Socket component.
AddAudioSource(): Adds an audio source component.