Item System Plugin

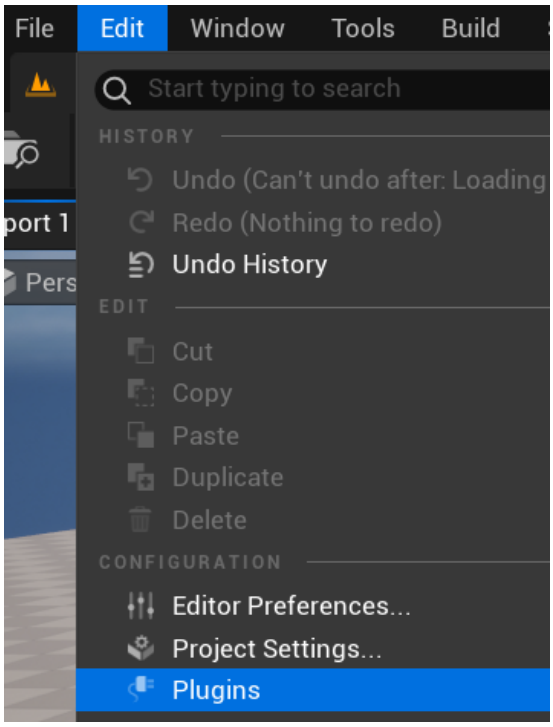
The Item System Plugin is a data-driven system that allows you to define items, create both static and dynamically generated item lists, and compare them. It manages item and list tracking, including availability and discovery, offering full control over item progression. Ideal for games involving crafting, recipes, or simulation-based mechanics like customer order.

Help: louisgirard.games@gmail.com

Disclaimer: This is not a full inventory system. Instead, it's a modular foundation for managing and manipulating items and item lists. However, it can be extended into an inventory system by leveraging custom item definitions and their runtime instances.

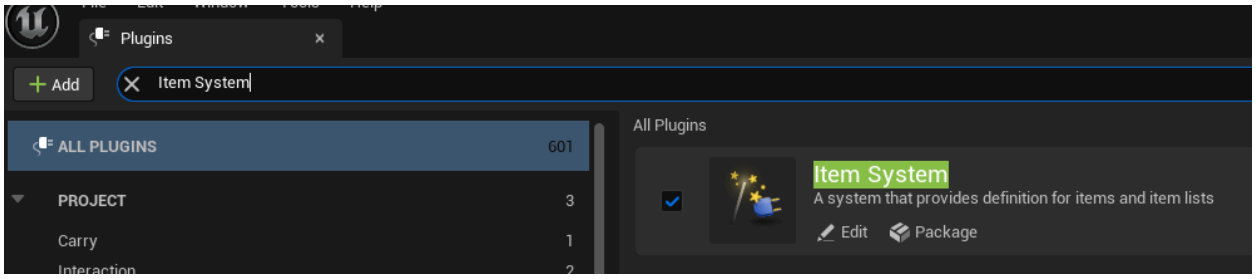
Activating the Plugin

Go to plugins



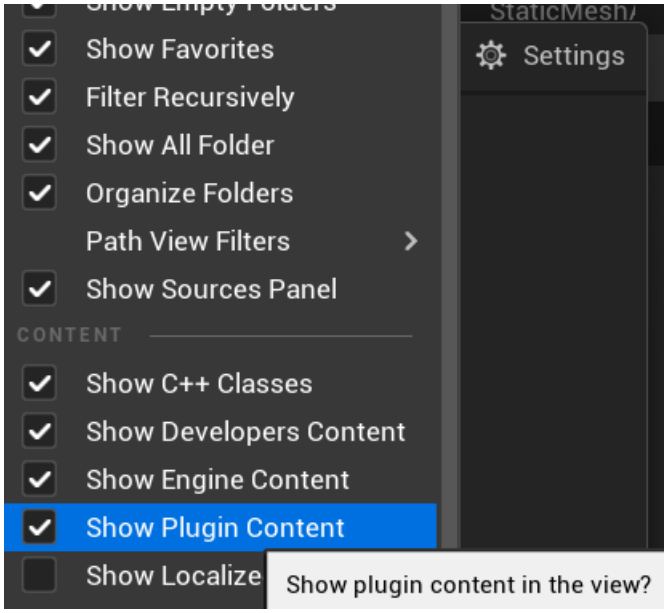
Enable the Item System plugin

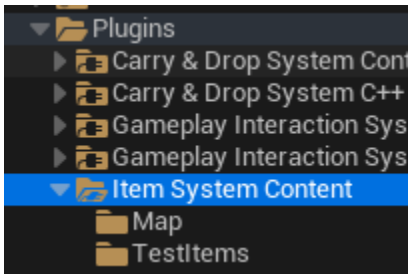
Search for Item System and tick the box to enable it.



Sample Content

Once the plugins are enabled you can enable the Plugin Content by going in the Content Browser and tick “Show Plugin Content”.





You can start exploring on your own the assets, or follow the documentation on how to set up your items and item lists.

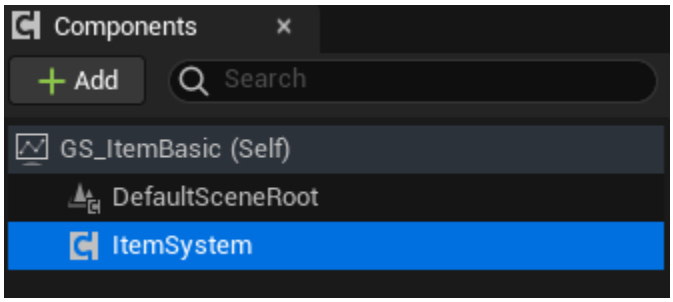
Item System Component

This is the component that will have the information about which items and lists are available or locked, which ones are discovered, and the lists that are generated at runtime.
Some examples of usages:

- You could have discovered items like Rock and Wood, and discovered a list Axe Crafting Recipe, but it could be locked with a required level to craft it.
- You want to buy some items like Furniture, some could be locked with a required level to buy them.
- You have unlocked items Bun, Steak and Cheese, and now you can generate lists that correspond to random orders from clients.

Depending on if these are some information that are **separated** between players (Eg. Each player discovers item separately, or unlock items separately), this component would go on the player (**Player State** or the **Character** itself). If they are **shared** between players (Eg. If one player discovers an item or a list, every player discovers it), then this component could go on the **Game State**.

So first add the **Item System Component** on the Game State or the Player State.



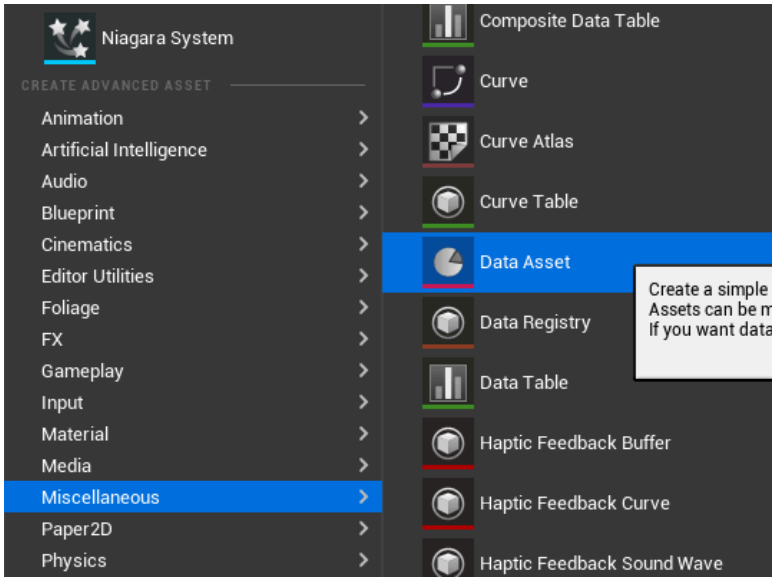
That's all the setup needed for the component, we'll take a look at the different functionalities as we go through the different item and lists available.

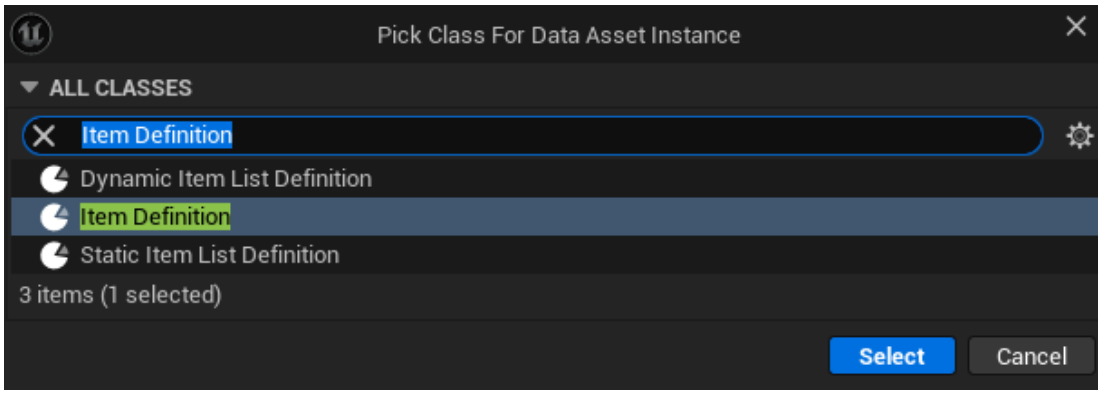
Item Definition

An item definition is a data asset that will represent an item.

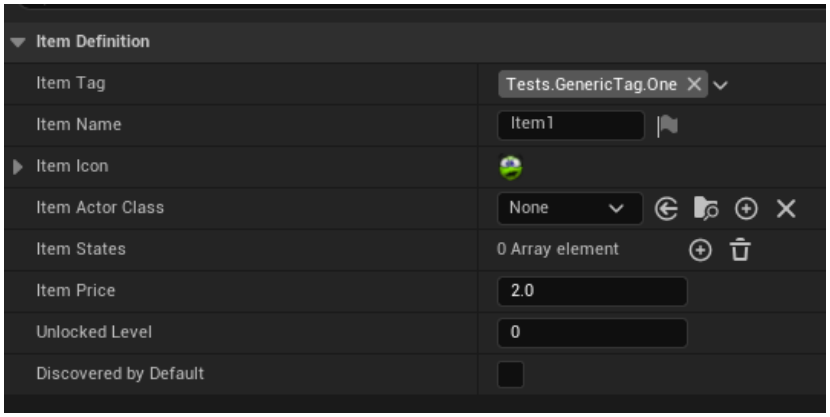
Creation

To create a new item definition, create a new data asset, and choose Item Definition.



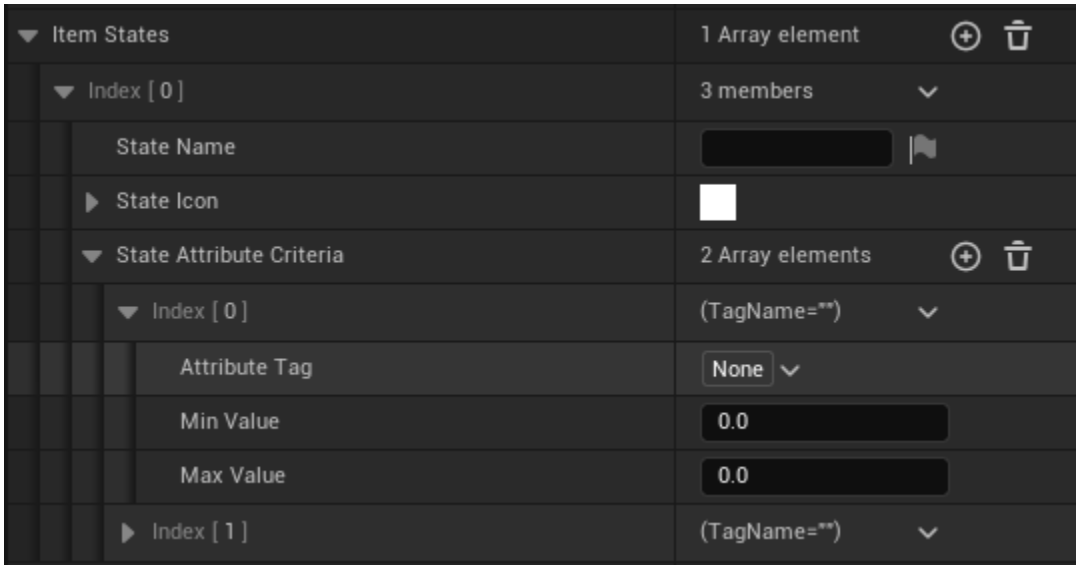


Configuration



You can give your item a Gameplay Tag to identify it, a Name, a Description, an Icon, a Price or even an Actor Class that would be the physical representation of the item.

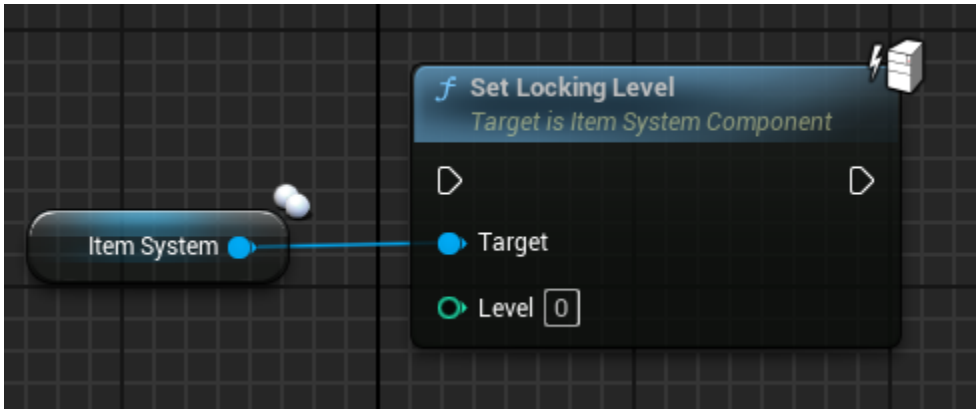
States



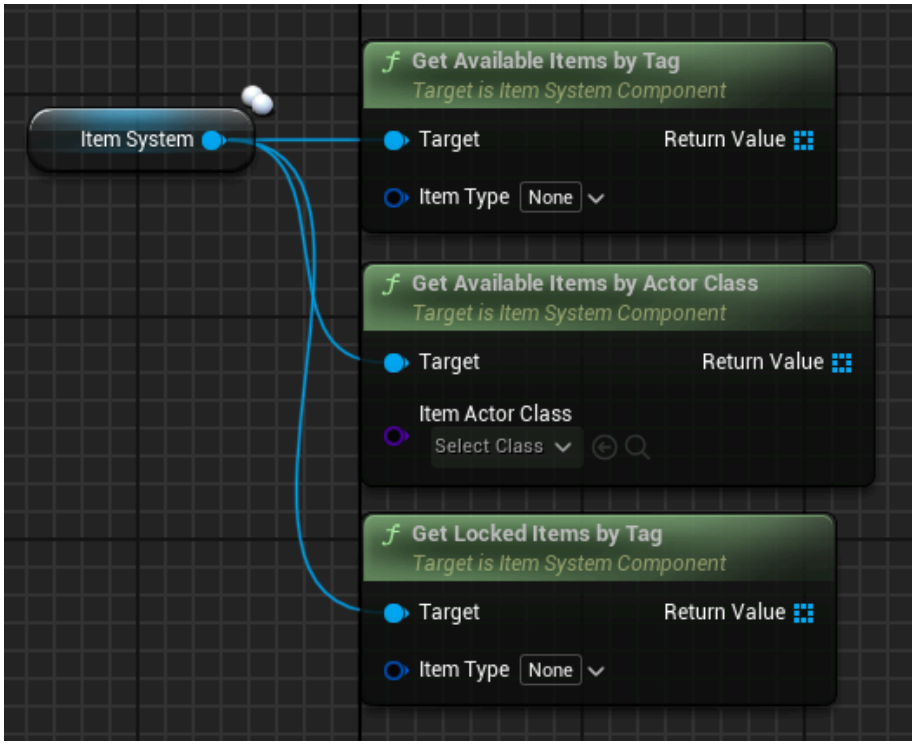
An item can have multiple states it can be in, based on its Attribute Criteria. You can configure the different attributes needed and in which range to be for a certain state.
For example, a Shield could have multiple states: New / Damaged / Broken, each with their name and visual, and an attribute “Damage Level” would influence the state. You can tag your attributes to have any combination you want for a state.

Unlock

An item can be locked or unlocked based on the Locking Level of the Item System Component. You can define at which level an item becomes unlocked. To change the Locking Level you can use

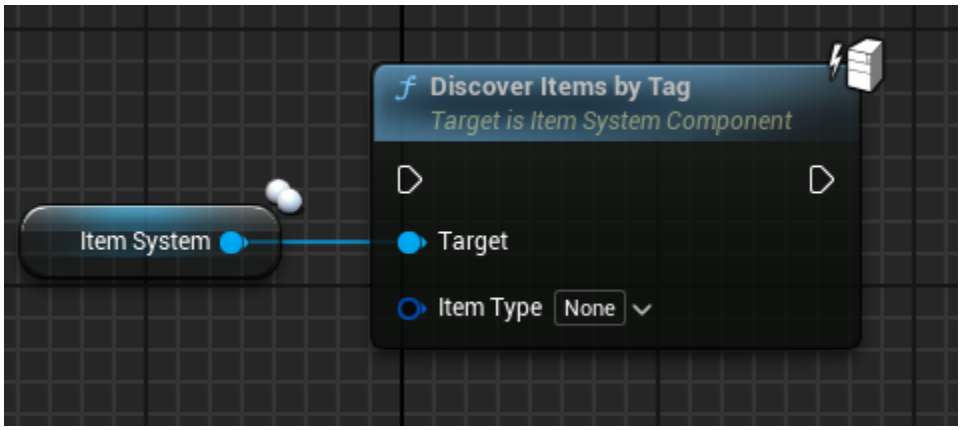


This will change the Level and unlock any items that are below this level. The level that you put can depend on any gameplay feature of your game, like the level of the player that allows to buy certain items.
You can easily get the items that are available or locked.

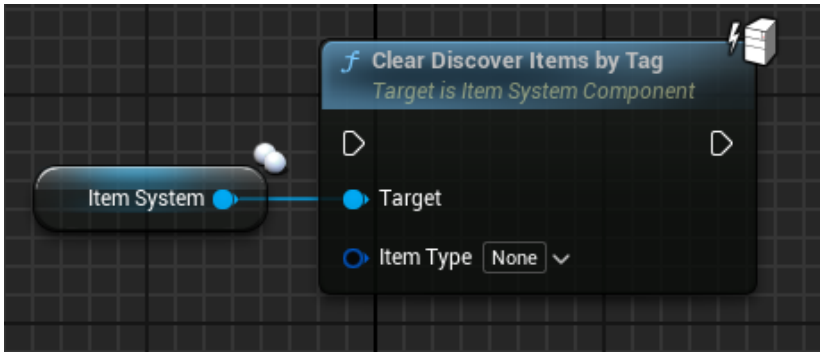


Discover

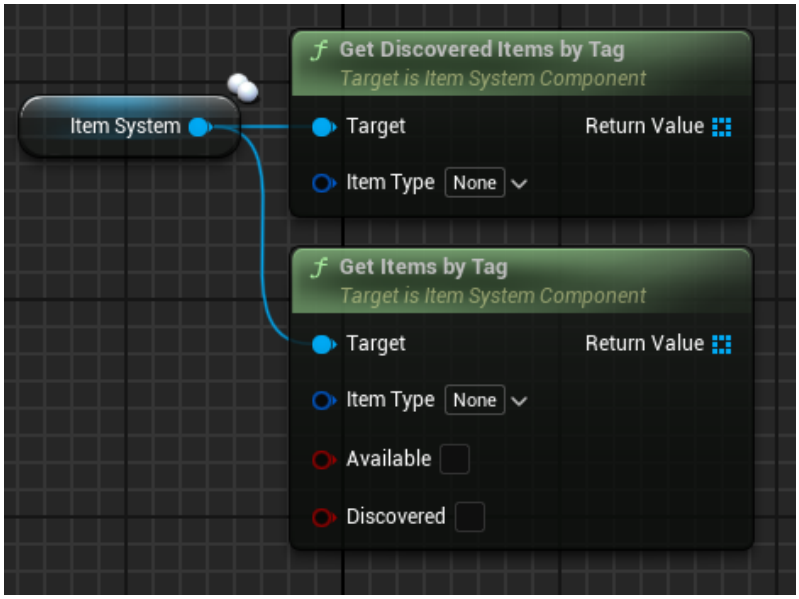
You can mark items as discovered or not. You have the option to discover them by default, or use



You can also clear any items that were discovered.

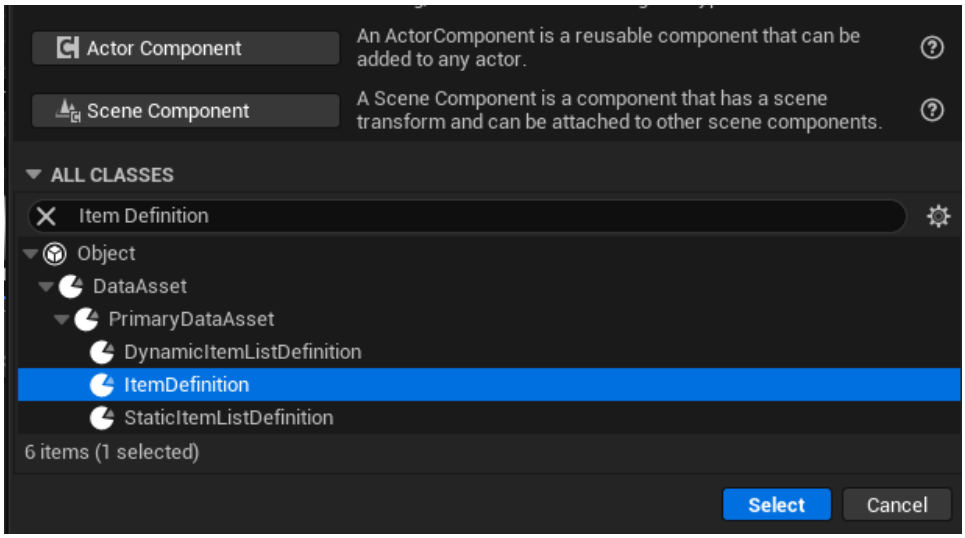


Same as for available and locked items you can easily get which items are discovered.

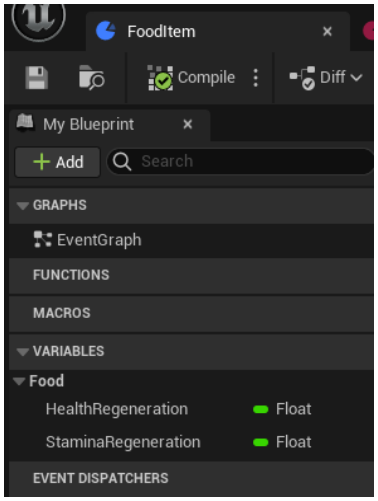


Custom Item Definition

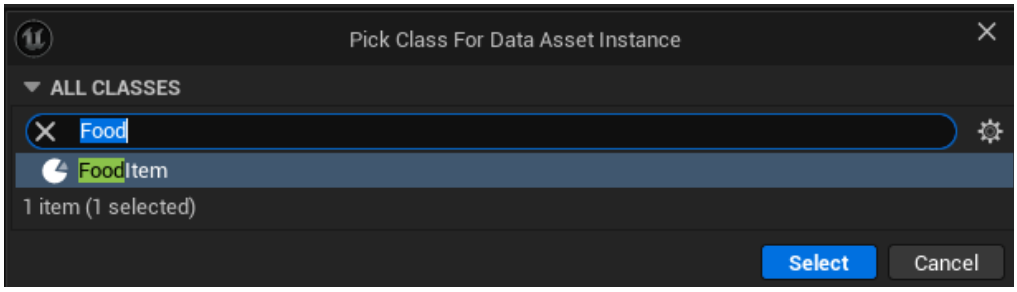
If you want to expand the Item Definition provided to have a custom one, you can create a new BP that derives from Item Definition.

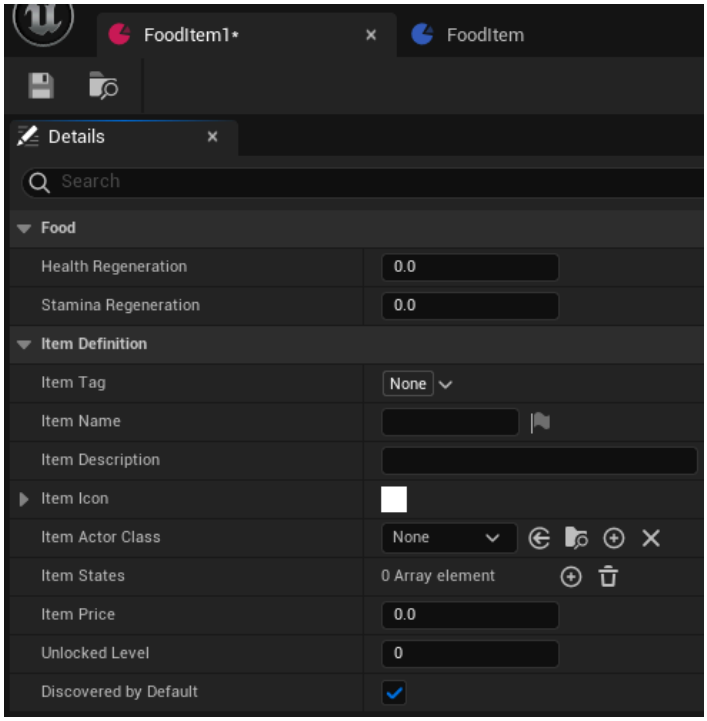


You can then add any variables that you want.



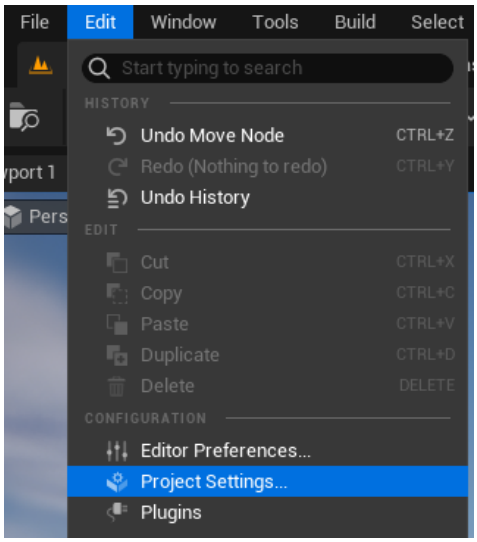
And then to create definitions from it, just create a Data Asset that uses this custom Item Definition you just created.

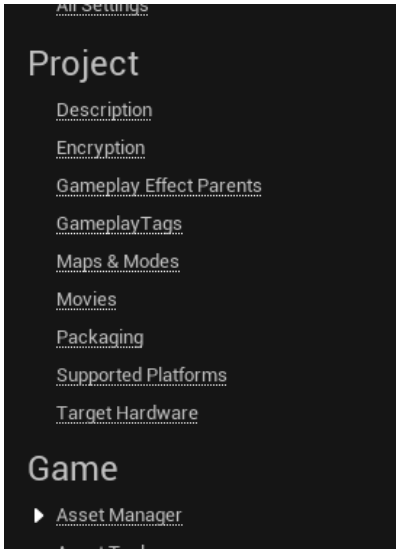




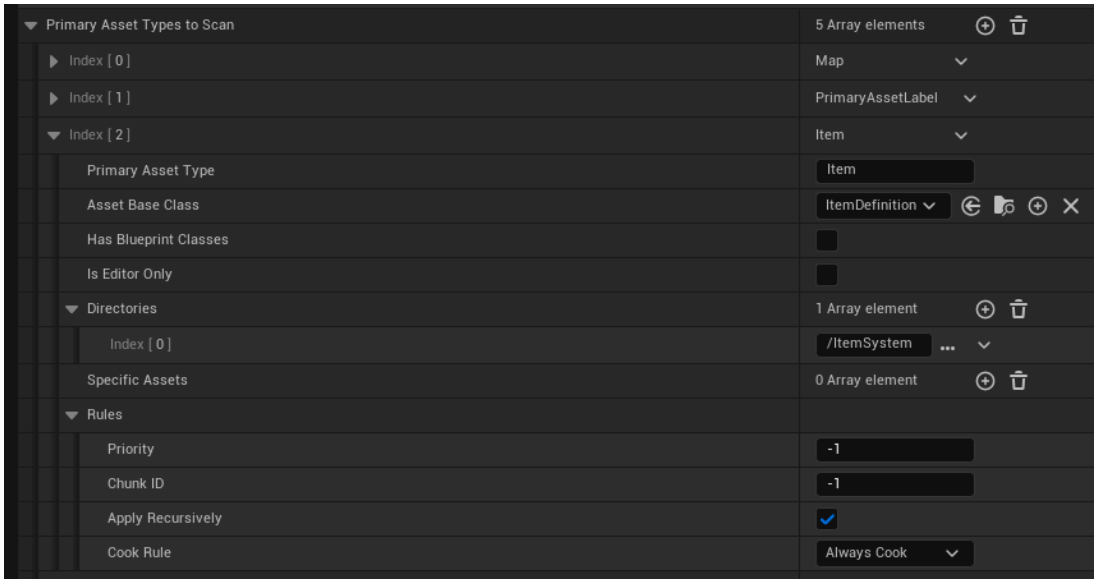
Asset Manager

You have to specify which Item Definitions will be used by the Item System. To do so go into Project Settings and Asset Manager

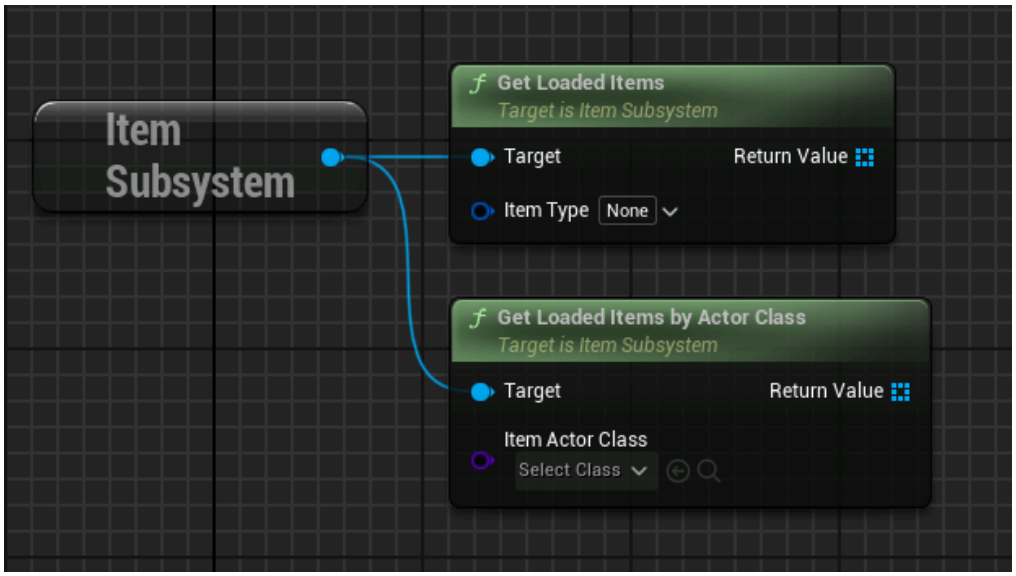




Then in the Primary Asset Types to Scan, add a new entry and fill it like the following. The part you will want to change is the Directories, where you should specify paths where you will put your Item Definitions.

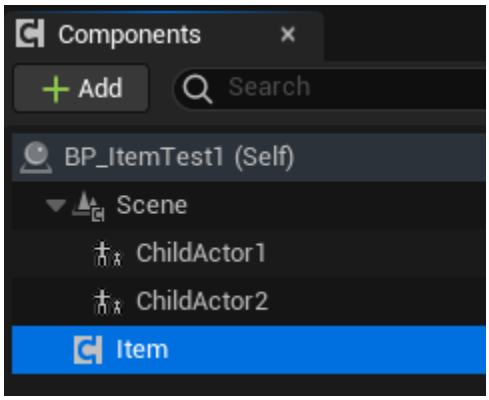


This will make it so that every Item Definition will be loaded and ready for you to use and be marked as locked or discovered. You can even access them at all times in the **Item Subsystem** with

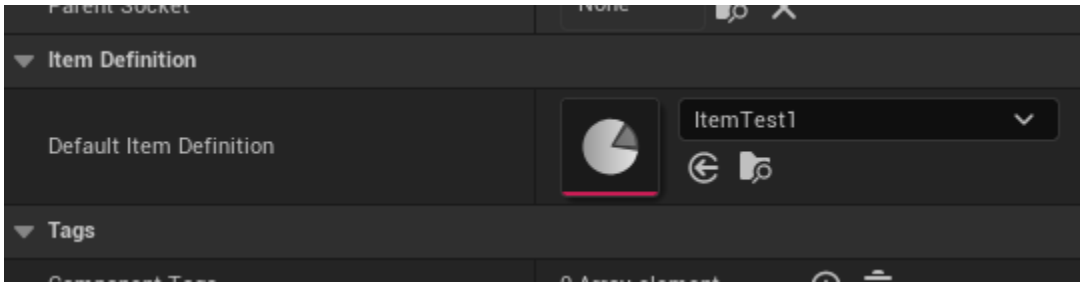


Item Component

An Item Component is a component that you place on an actor to physically represent an Item Instance.

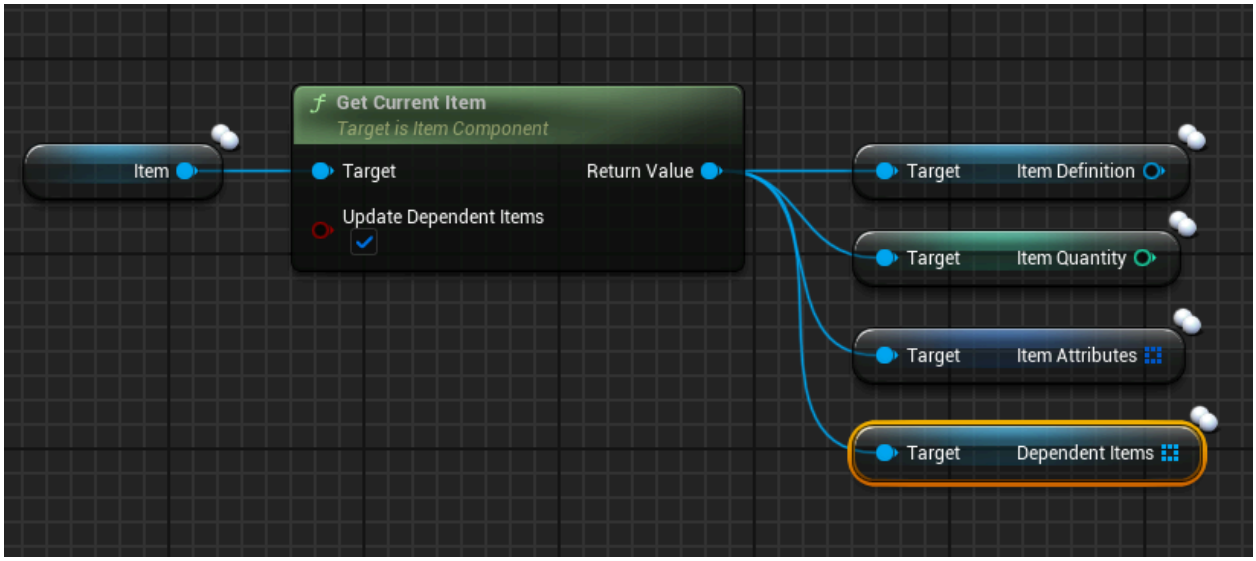


To define which item it is, you just have to specify the **Default Item Definition**.



Item Instance

An Item Instance will be created from the Item Definition, and be accessible from the Item Component.



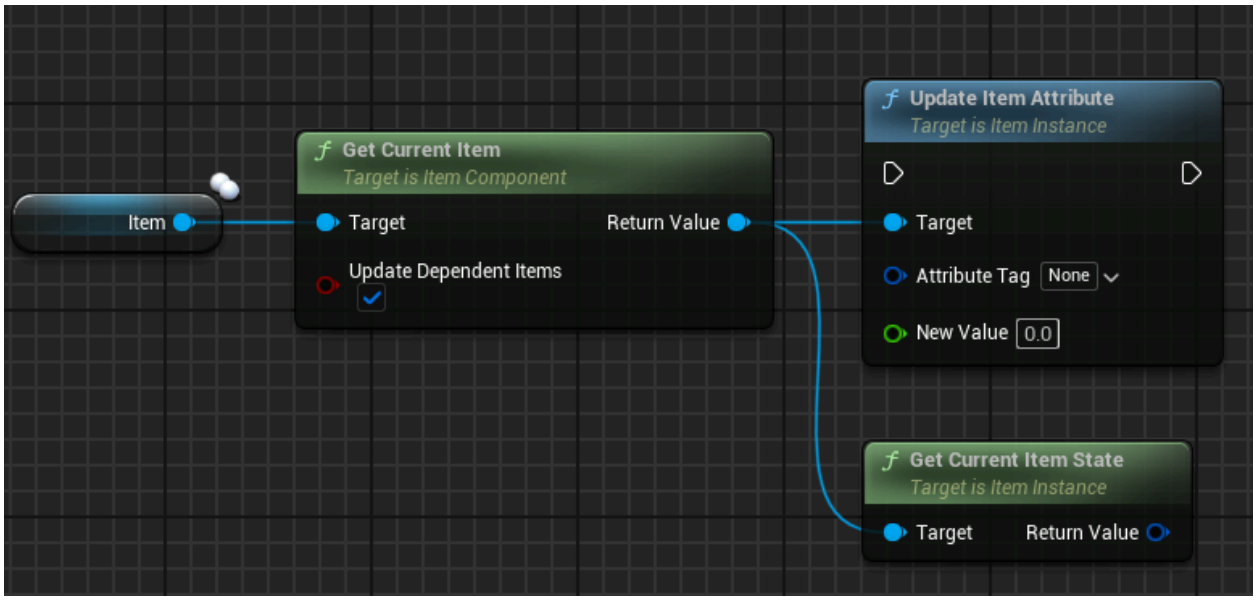
An Item Instance contains:

- A reference to the Item Definition
- A quantity of items there is
- The attributes of the item

- A list of dependent items

Item Attributes

You can update the attributes of the current item, which based on its Item Definition, will influence which state it is in. An attribute has a tag to identify it and a value.



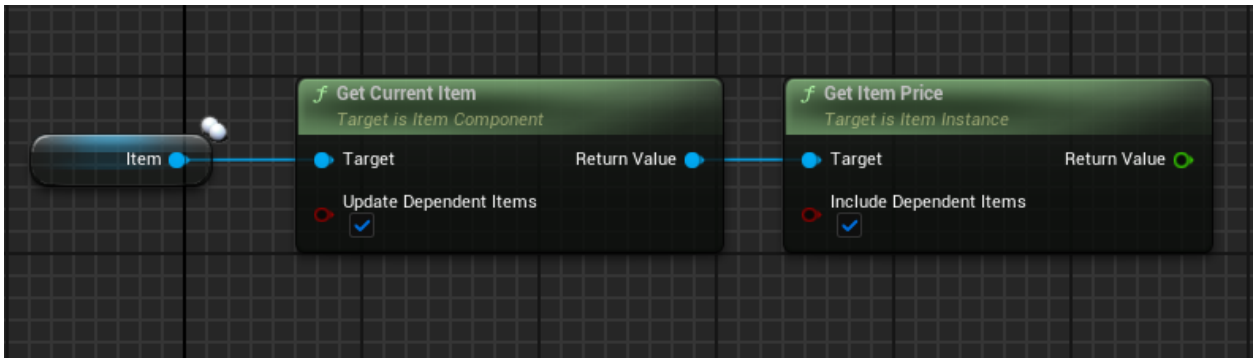
Dependent Items

An item could have child items. These dependent items will be filled based on the Child Actors of the actor that owns the Item Component.
So if for example there are 2 child actors with an Item Component, the current item will have 2 dependent items:

- The first one will be the current item of the first child actor
- The second one will be the current item of the second child actor. Just see it as a faster way to access the child items.

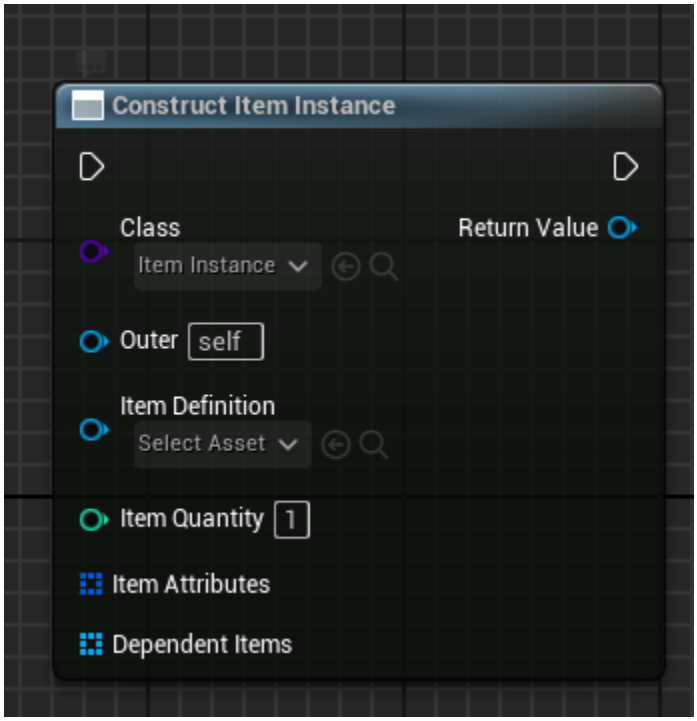
Item Price

You can easily access the price of the Item and the combined price of its dependent items



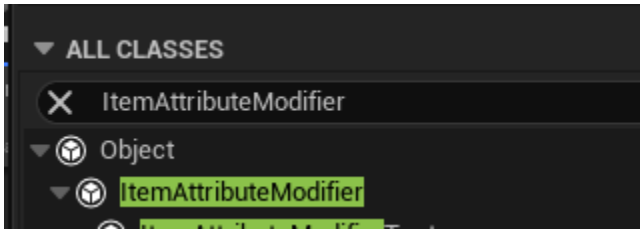
Construct an Item Instance

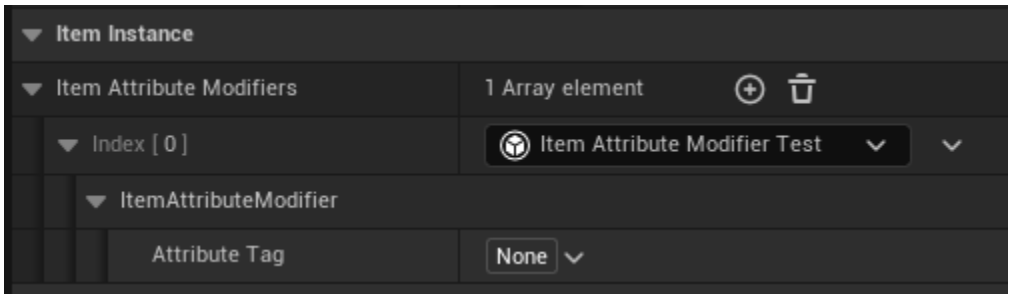
The Item Instance is not restricted to the Item Component, you could use it in any other component (like an Inventory for example). You can construct an new one like any other object, and fill the default values



Item Attribute Modifiers

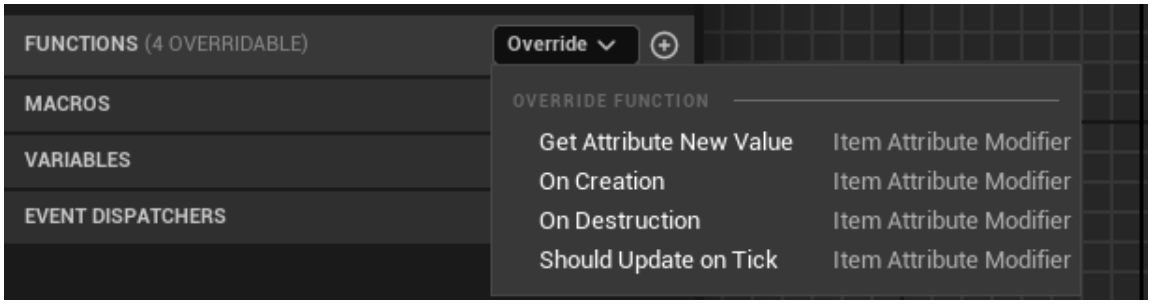
If you want reusable ways of modifying the item attributes, you can create Item Attribute Modifiers. Let's create one and set it in your Item Component.





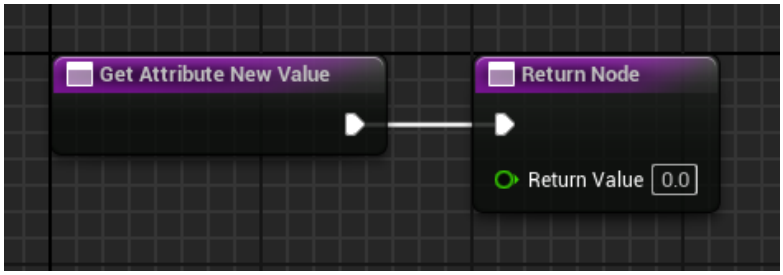
You can set the **Attribute Tag** to indicate which Item Attribute will be modified.

There are different functions you can override to make it work.



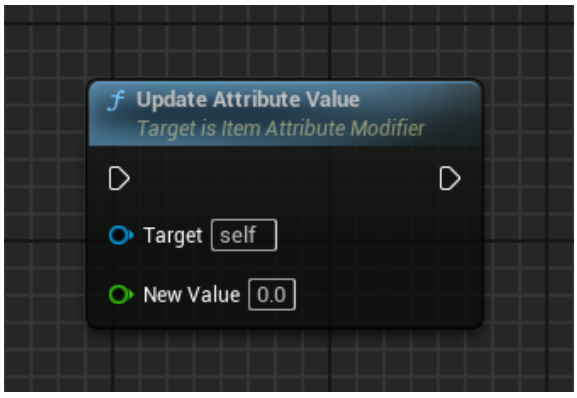
On Creation and **On Destruction** act like a Begin and End play, if you have variables to initialize.

You should override **Get Attribute New Value** and make it return the value you want for your Item Attribute.

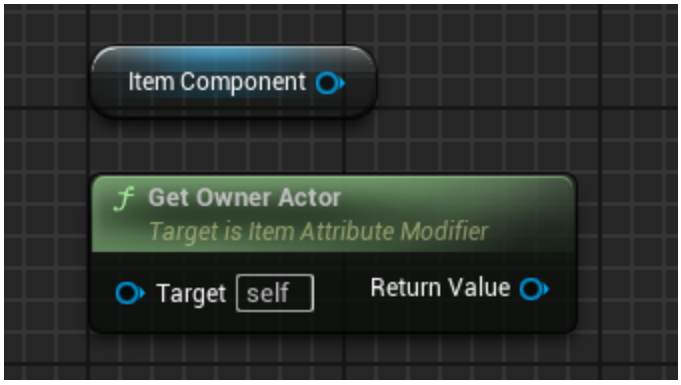


By default **Should Update on Tick** returns true, which means that the Item Attribute will be updated every frame. If you do not need an update every frame you can override it to return false.

If it is set to false then it is your responsibility to manually update internally the attribute using

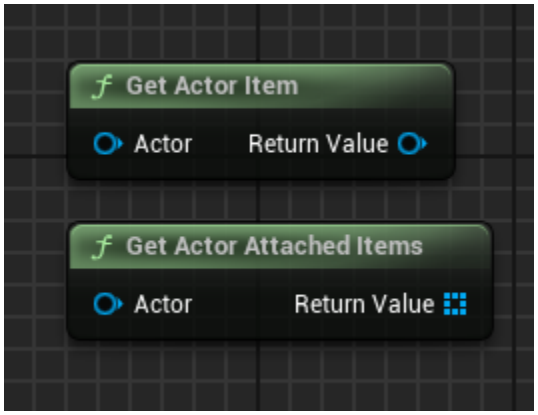


You also have access to the Item Component and the owning actor.



Getting Item on Actor

There are helpers to easily get an item or attached items on an Actor.



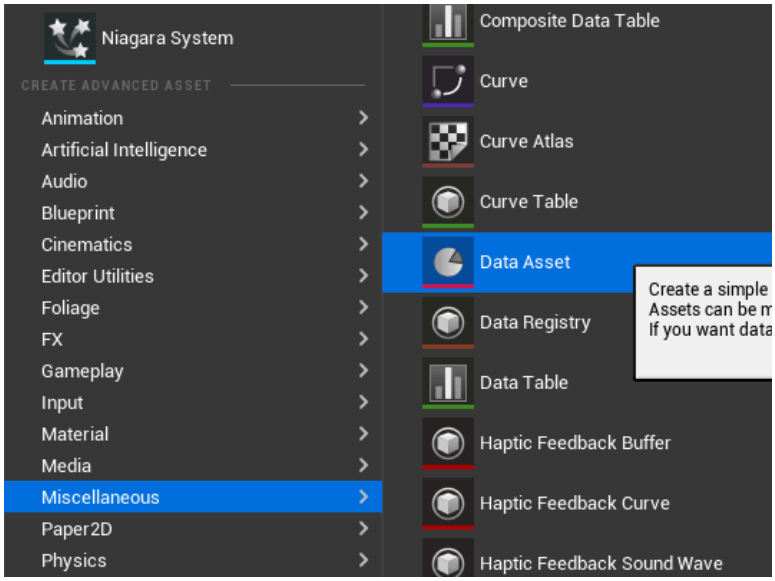
GetActorItem will return the Current Item of the Item Component on the Actor, if any.
GetActorAttachedItems will return the Current Item of any attached actors on the Actor. Use this if you know the Actor does not have any Item Component.

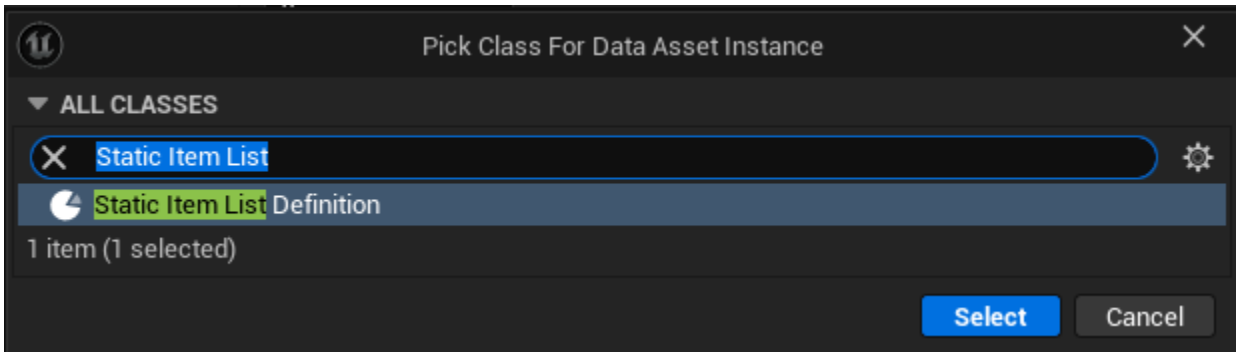
Static Item List

A Static Item List is a Data Asset that represents a list of Item Instances. It won't be generated or modified at runtime. It could be used for any recipe lists for example.

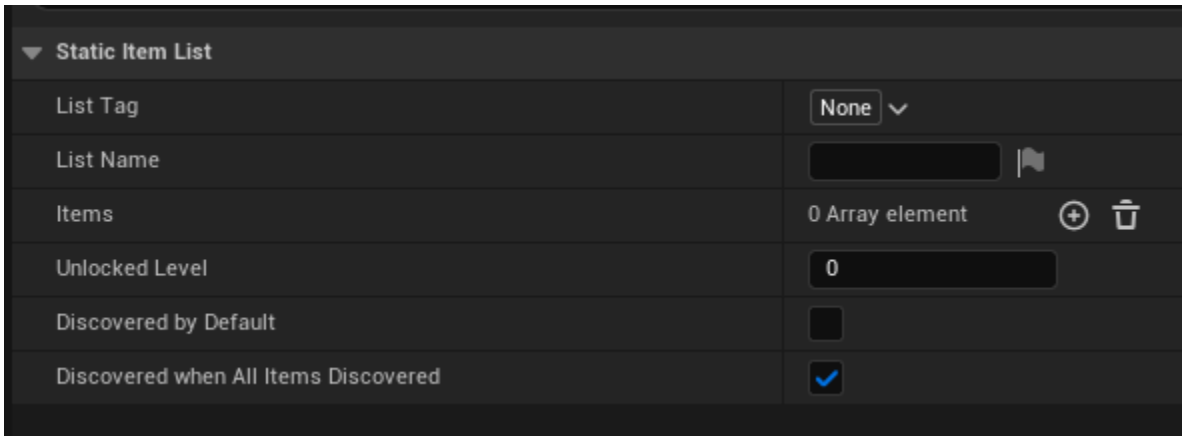
Creation

To create a new static item list, create a new data asset, and choose Static Item List Definition.





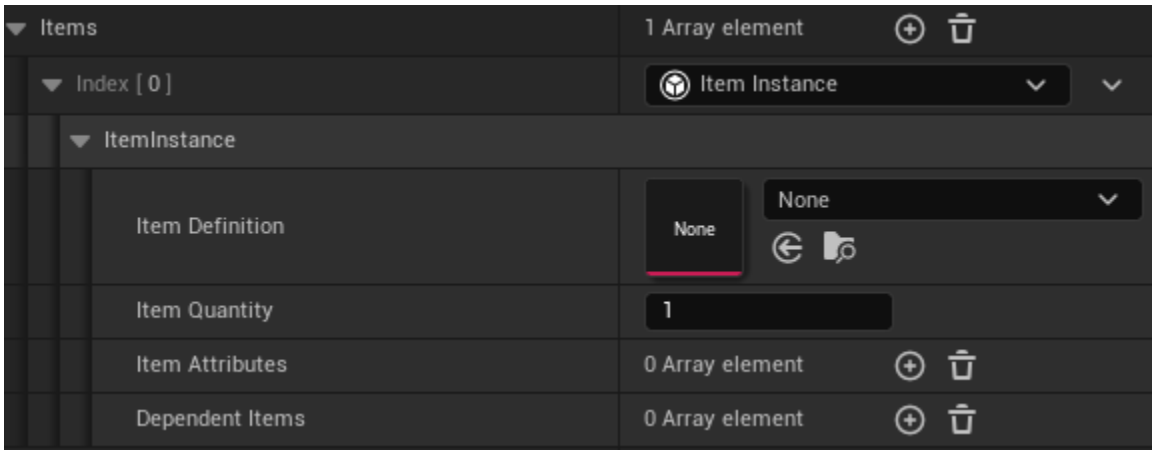
Configuration



You can give your list a Gameplay Tag to identify it, a Name and a Description.

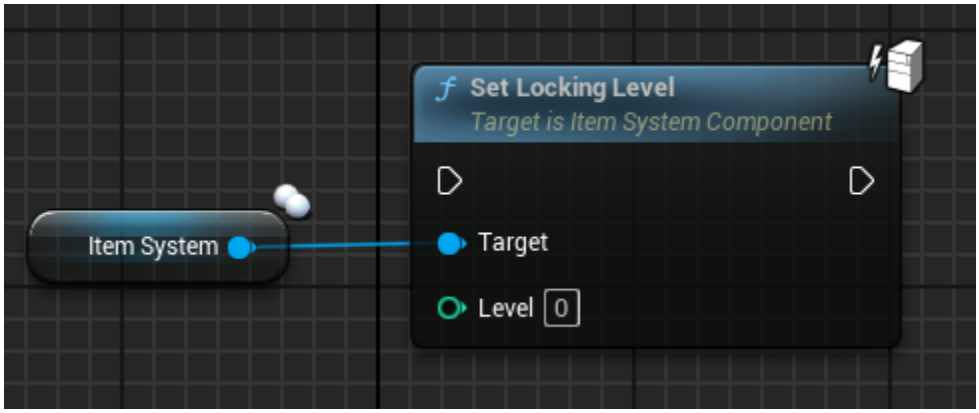
Items

You can specify a list of Item Instance that will compose this list.



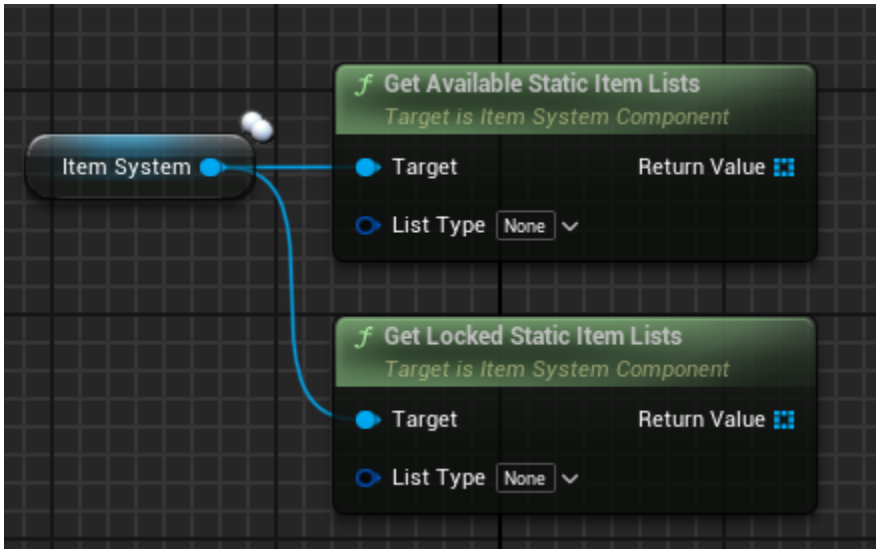
Unlock

A list can be locked or unlocked based on the Locking Level of the Item System Component. You can define at which level a list becomes unlocked. To change the Locking Level you can use

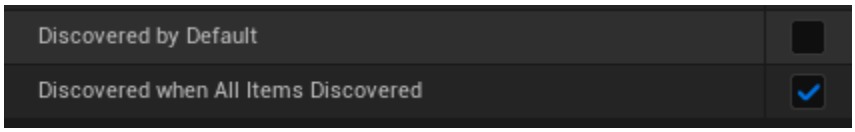


This will change the Level and unlock any static item lists that are below this level. The level that you put in can depend on any gameplay feature of your game, like the level to craft certain recipes.

You can easily get the lists that are available or locked.

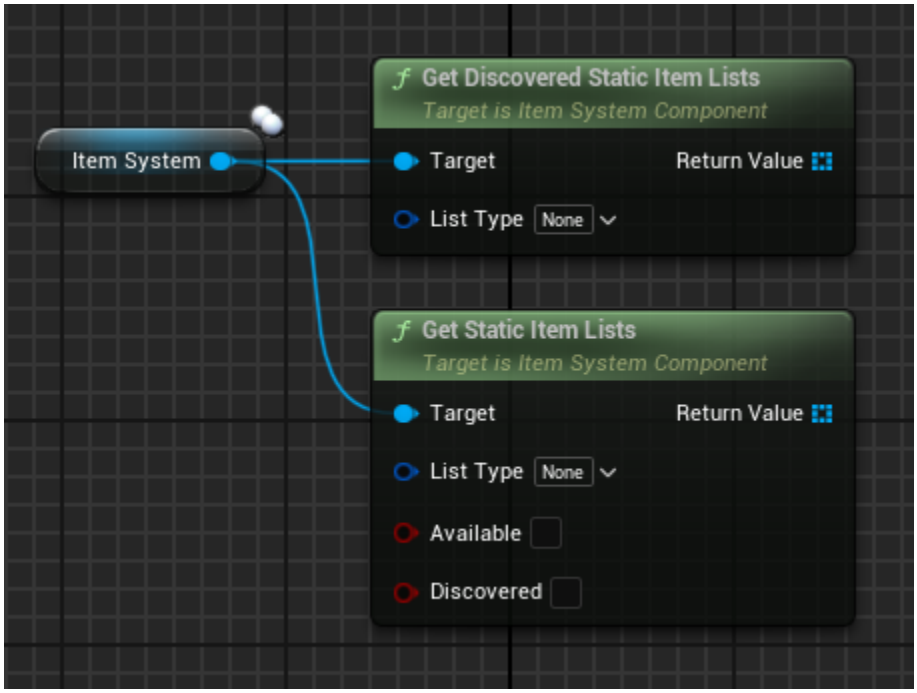


Discover



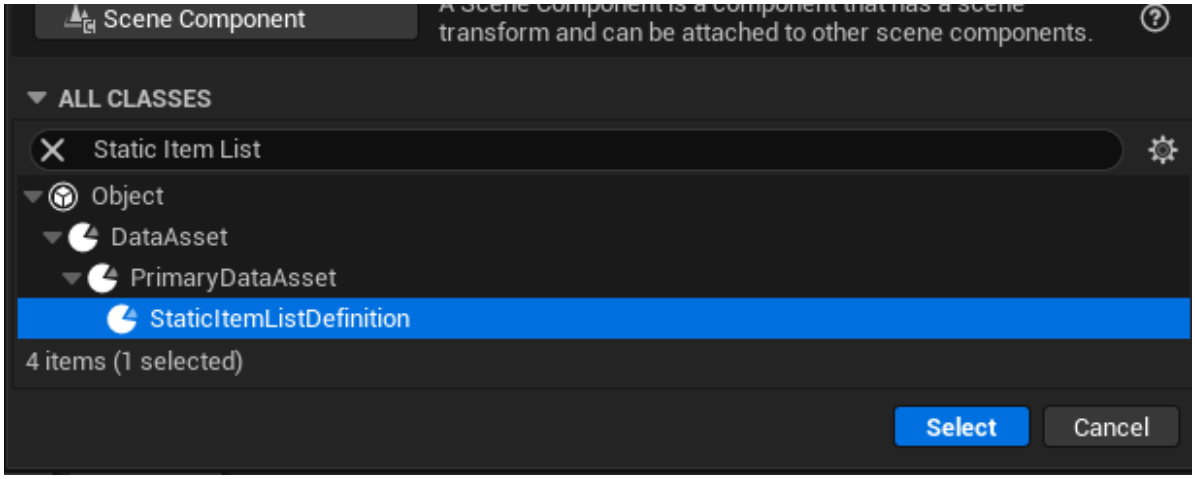
You can mark static lists as discovered or not. You have the option to discover them by default, or discover them when all the items they use have been discovered.

Same as for available and locked lists you can easily get which lists are discovered.

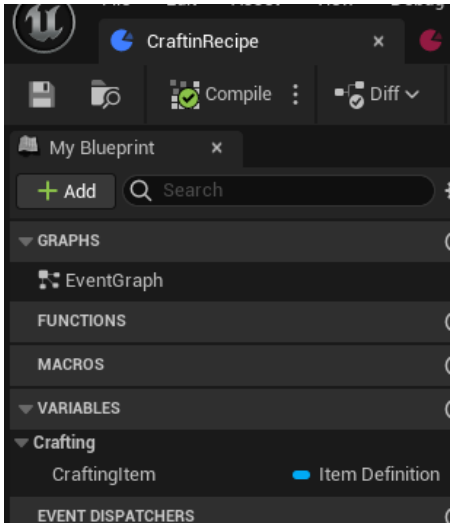


Custom Static Item List Definition

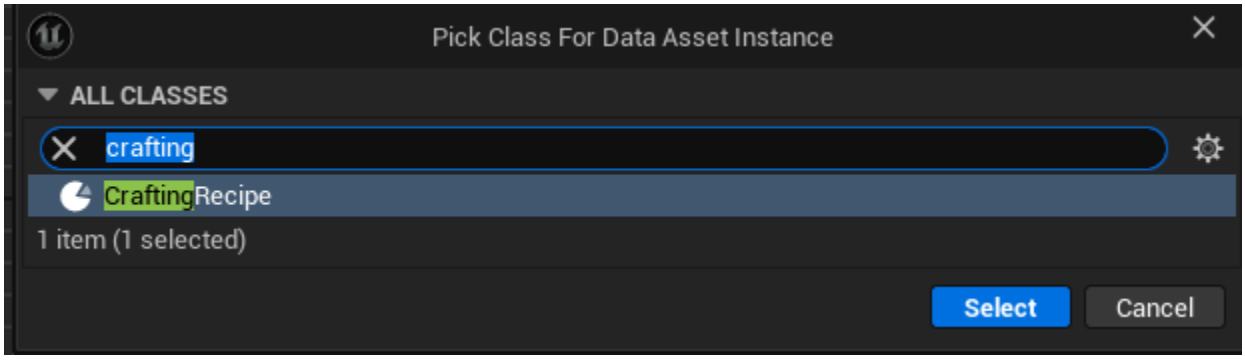
If you want to expand the Static Item List Definition provided to have a custom one, you can create a new BP that derives from it, similar to the items.

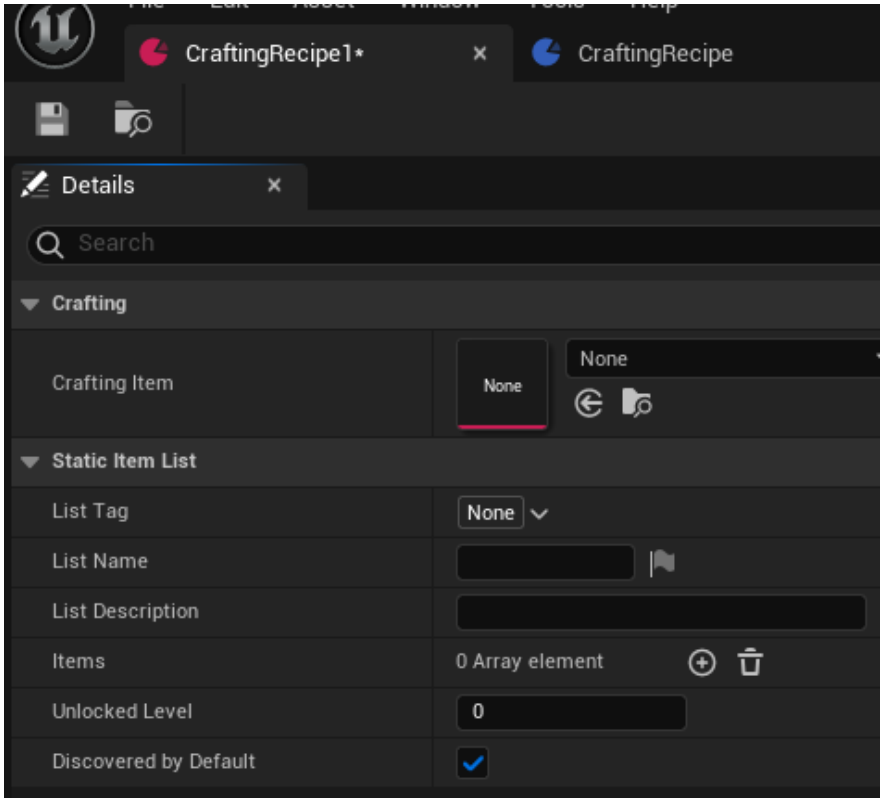


You can then add any variables that you want.



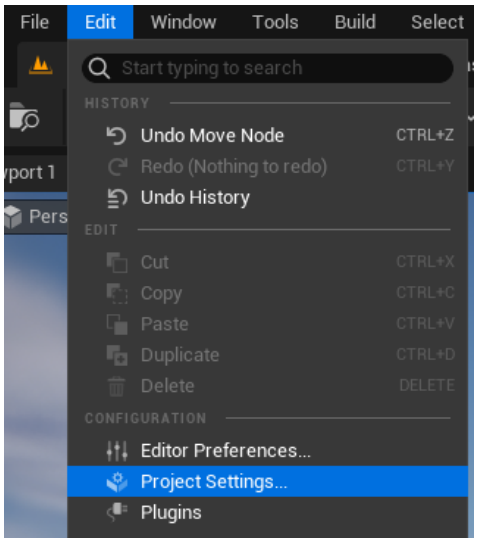
And then to create definitions from it, just create a Data Asset that uses this custom Static Item List Definition you just created.

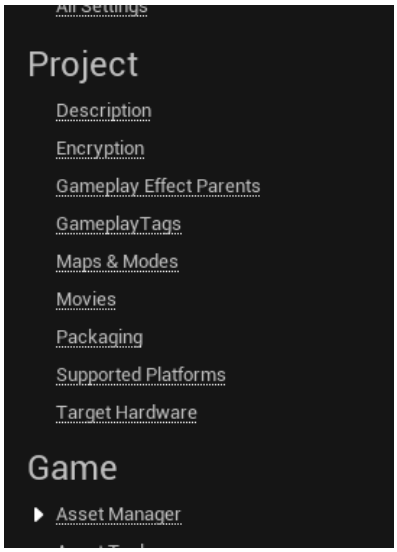




Asset Manager

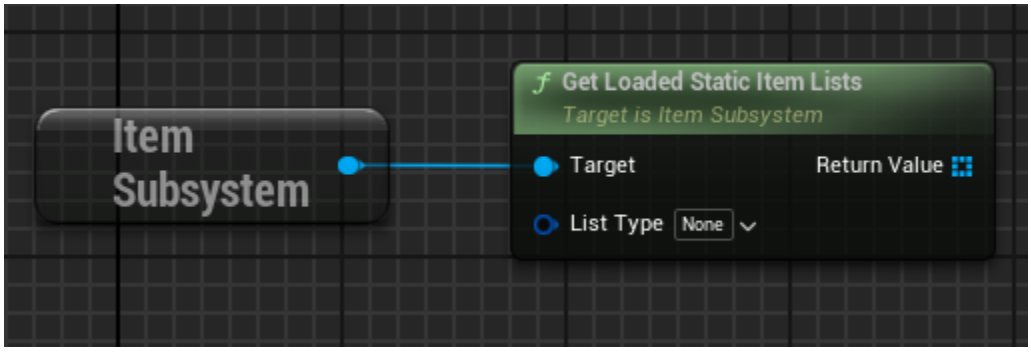
You have to specify which Static Item List Definitions will be used by the Item System. To do so go into Project Settings and Asset Manager





Then in the Primary Asset Types to Scan, add a new entry and fill it like the following. The part you will want to change is the Directories, where you should specify paths where you will put your Static Item List Definitions.

This will make it so that every Static Item List Definition will be loaded and ready for you to use and be marked as locked or discovered. You can even access them at all times in the **Item Subsystem** with

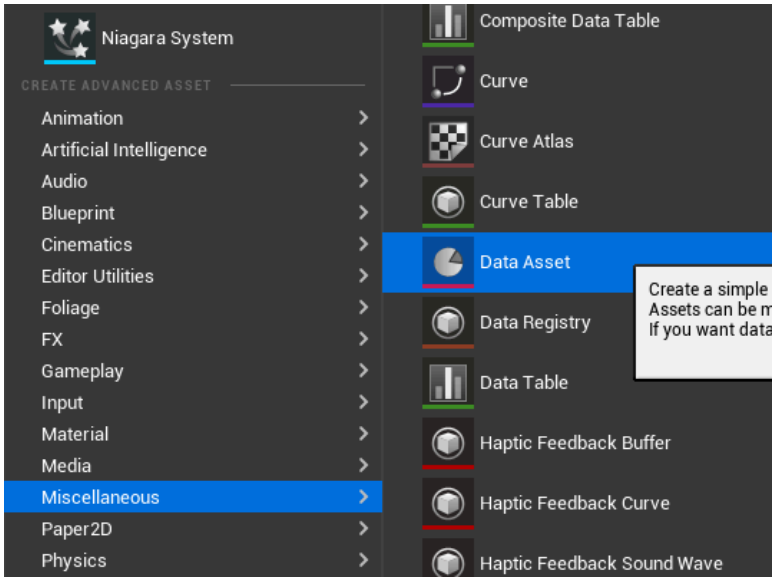


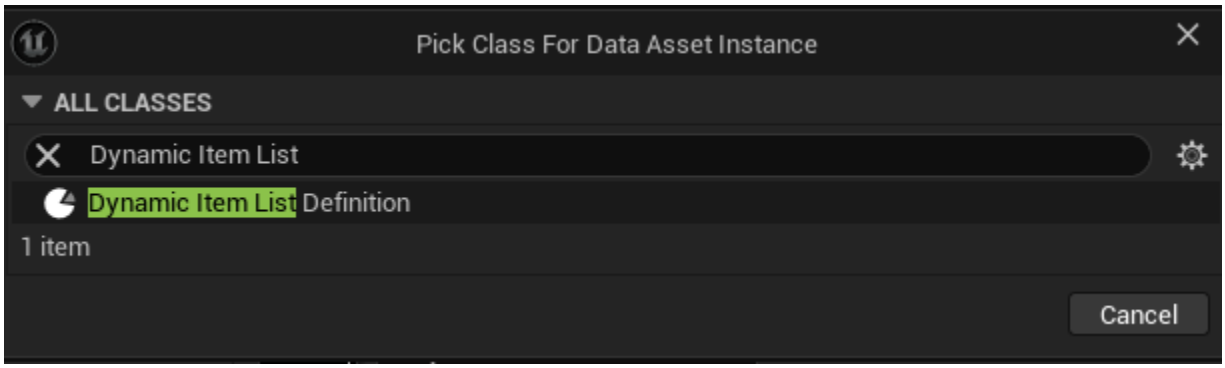
Dynamic Item List

A Dynamic Item List is a Data Asset that represents a list that will be generated at runtime based on parameters. It could be used for orders that would be generated from certain items.

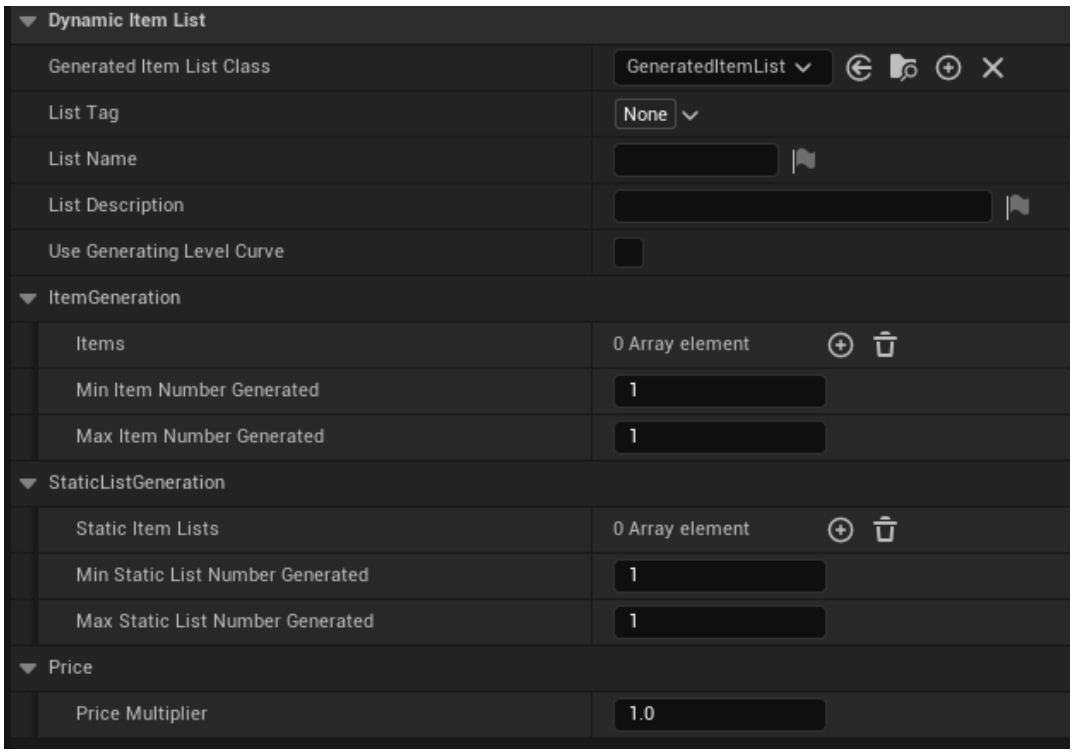
Creation

To create a new Dynamic Item List, create a new data asset, and choose Dynamic Item List Definition.



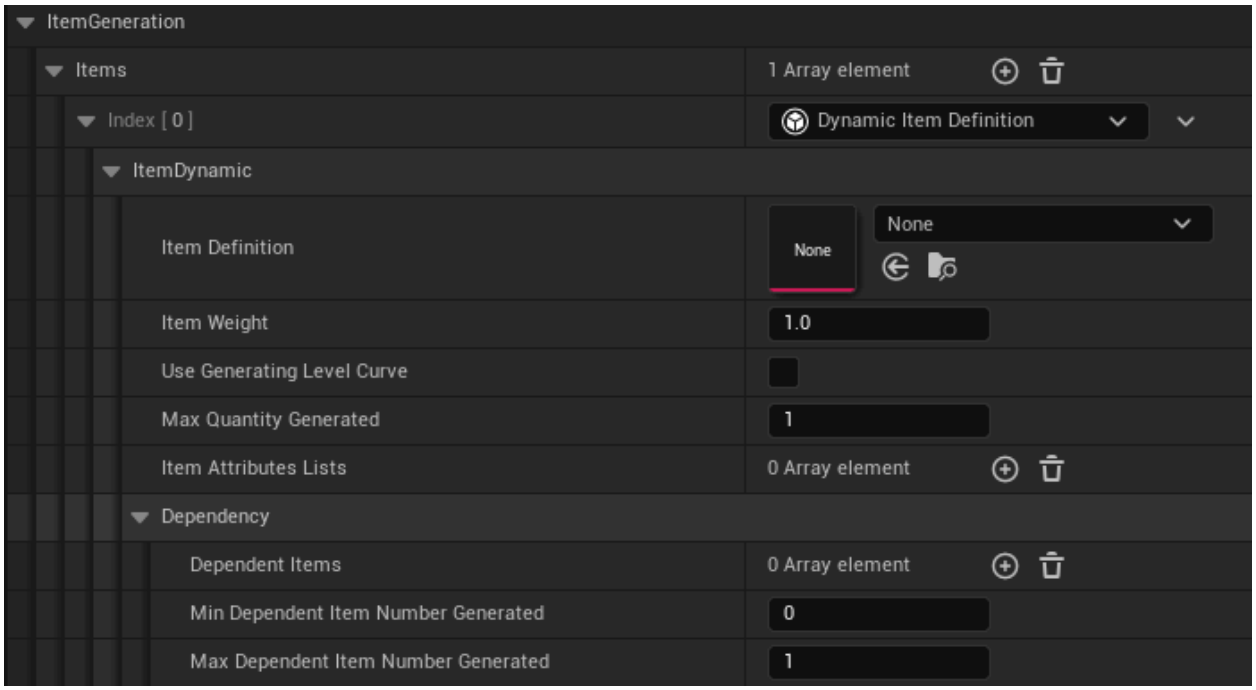


Configuration



You can give your Dynamic List a Gameplay Tag to identify it and a Name.
You can specify a custom class for the Generated Item List.

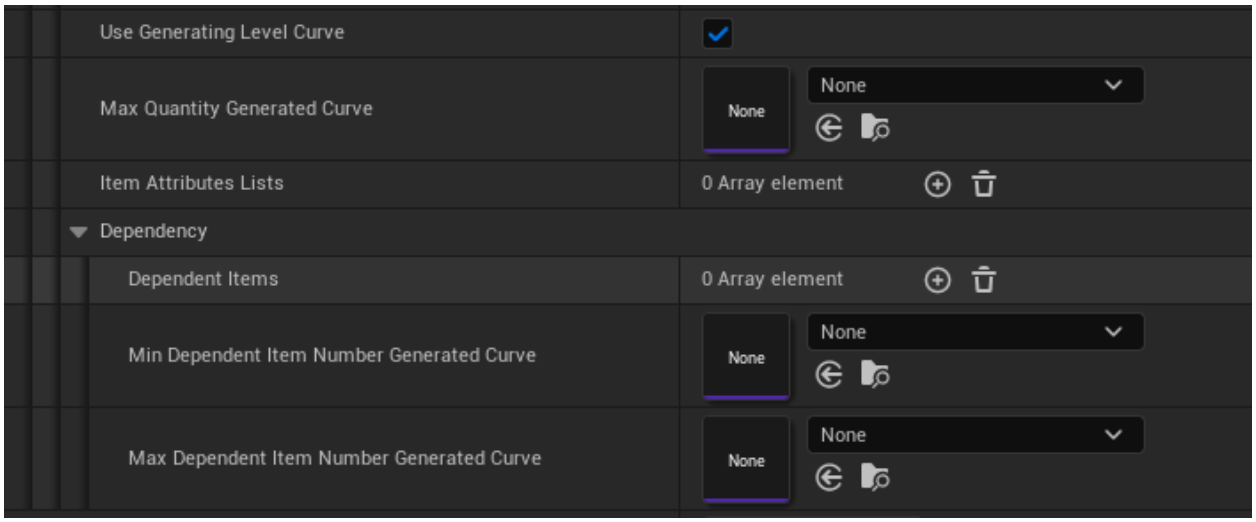
Item Generation



You can define which Items will be generated and their properties. But keep in mind that **only Unlocked Items will be considered** for the generation. For each entry, you can specify:

- Item Definition: Definition of the Item to generate
- Item Weight: Weight of the item in the list, the higher, the bigger the chances for it to be generated
- Max Quantity Generated: Maximum amount of this item will be generated
- Item Attributes Lists: A random list of attributes will be assigned to the generated item
- Dependency: Configuration for dependent items and the maximum number of dependent items that can be generated

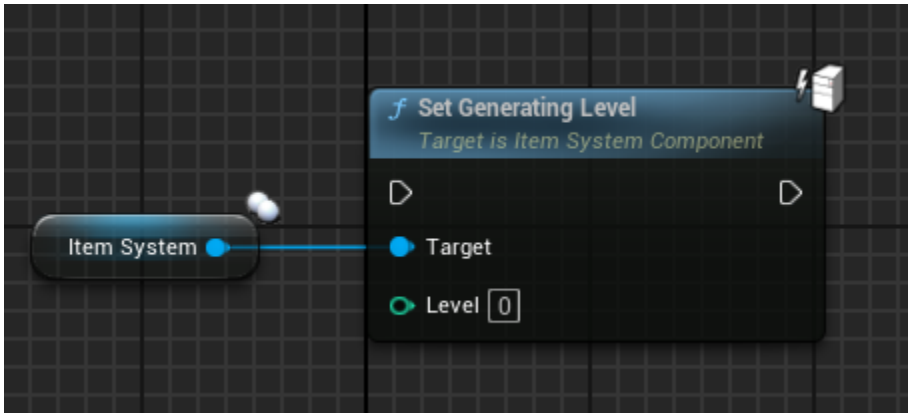
If you toggle **Use Generating Level Curve** you can have a curve to define the maximum number of items. This can be useful if you have difficulty levels, and you want to generate more items based on difficulty.



Once you have configured the items that can be generated, you can specify a minimum and maximum number to generate. These values can also take a curve to scale them with the Generating Level.

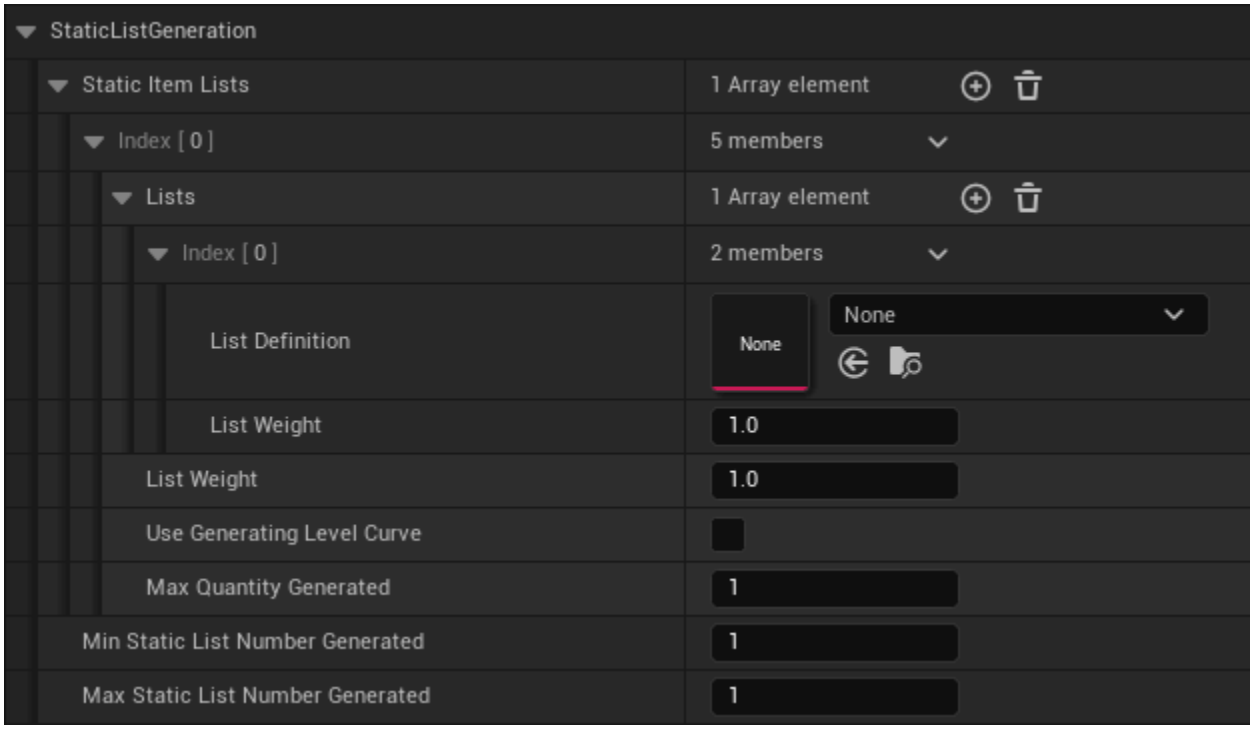
To modify the Generating Level that will influence the number of minimum / maximum items to generate, it's done through the Item System Component. This will make each curve take this level as the

value on the horizontal axis to look at.



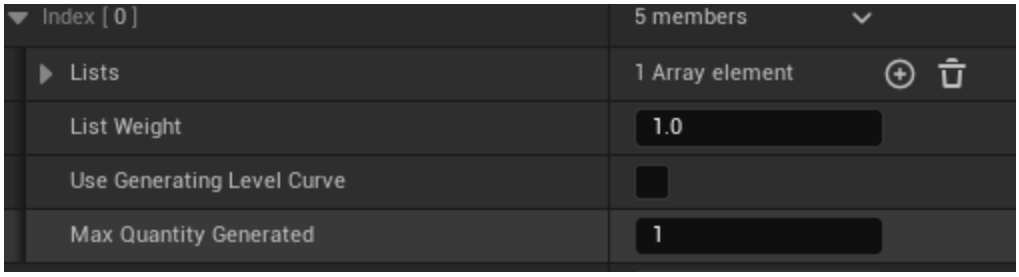
Static Item List Generation

This is very similar to Item Generation, but instead of generating individual items you can generate entire predefined static lists.



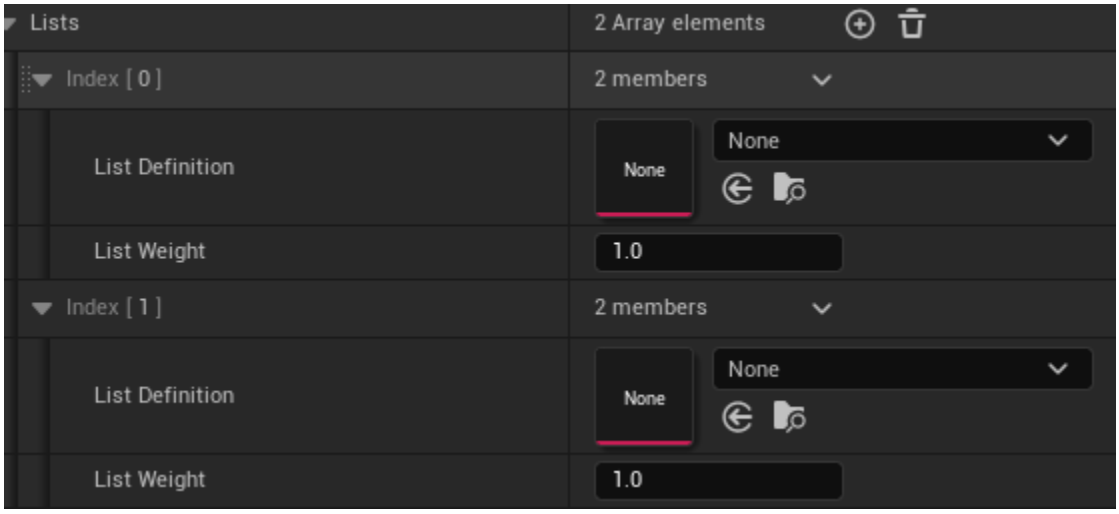
You can define a minimum and maximum number of lists that will be generated. Keep in mind that **only Unlocked Static Lists will be considered** for the generation.

You can define **Lists** of Static Item Lists. This can be useful if you want to group your Static Item Lists of the same type.
Similar to Items you have List Weight, Max Quantity and you can use the Generating Level Curve.



This weight will give more importance to certain List of Static Item List over others.

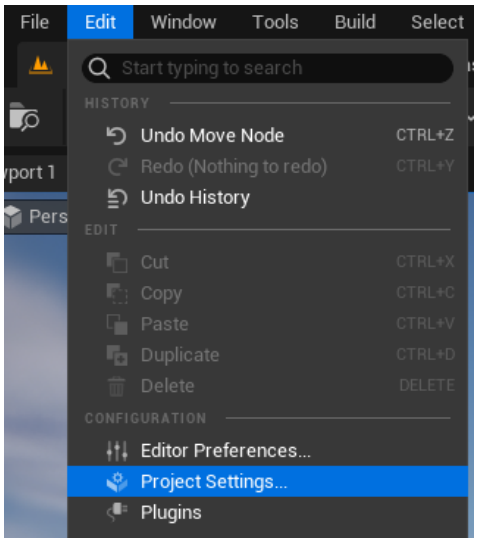
And then within the List you can define Weight for Static Item Lists that will be generated.

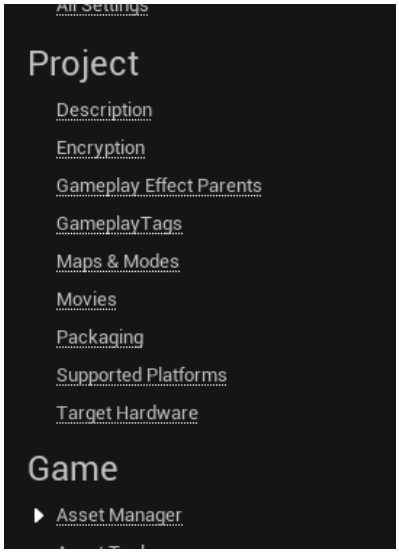


In the end it's a **Static Item List that will be chosen**, and the Items that compose it will be generated.

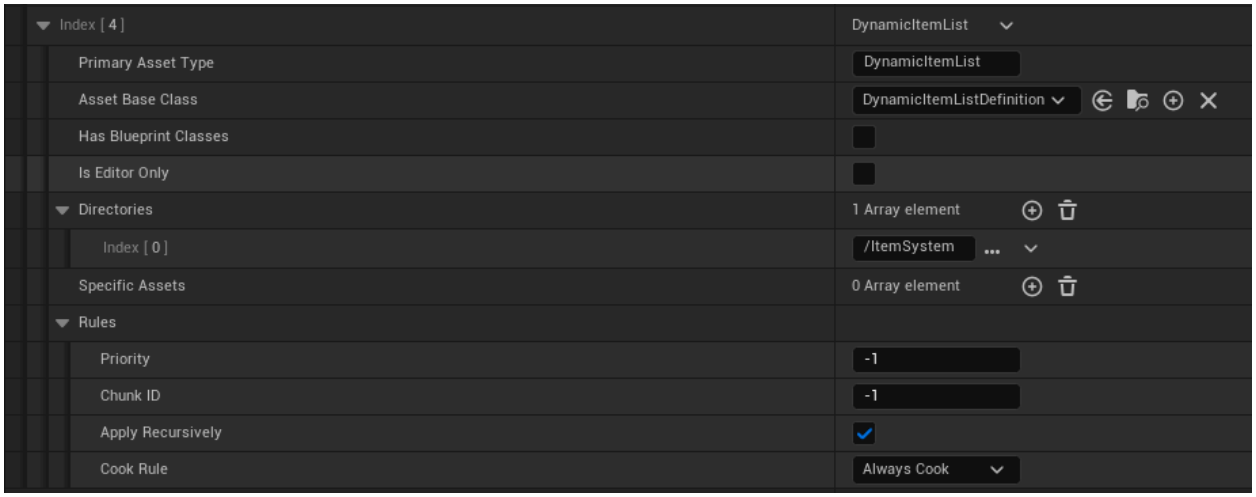
Asset Manager

You have to specify which Dynamic Item List Definitions will be used by the Item System. To do so go into Project Settings and Asset Manager

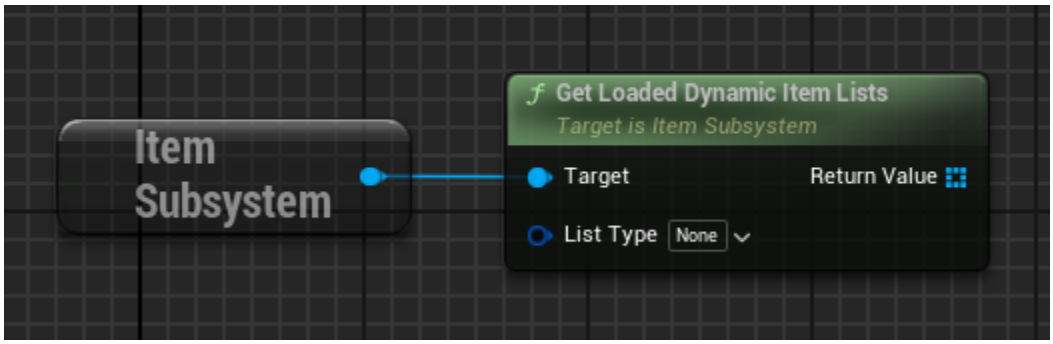




Then in the Primary Asset Types to Scan, add a new entry and fill it like the following. The part you will want to change is the Directories, where you should specify paths where you will put your Dynamic Item List Definitions.

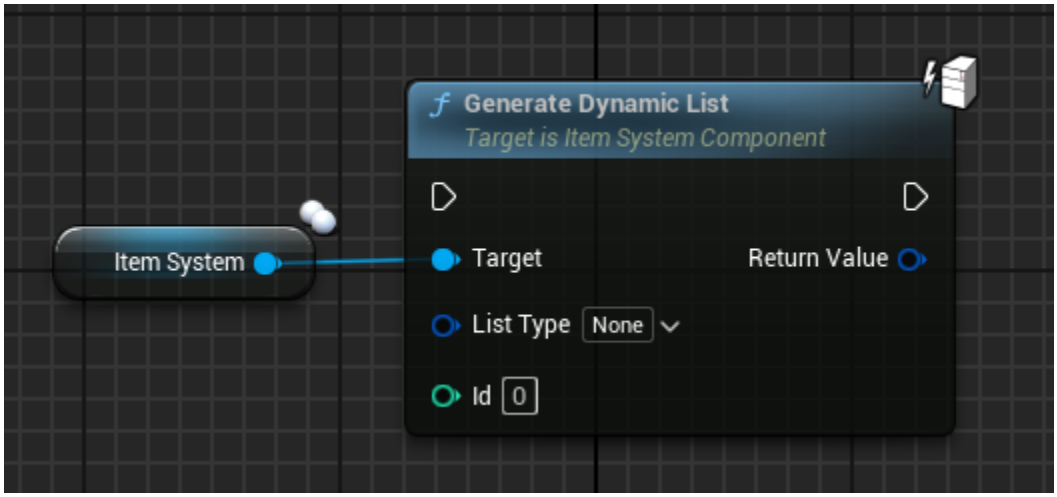


This will make it so that every Dynamic Item List Definition will be loaded and ready for you to use and be marked as locked or discovered. You can even access them at all times in the **Item Subsystem** with



Generation

To generate a new list, you can do it from the [Item System Component].

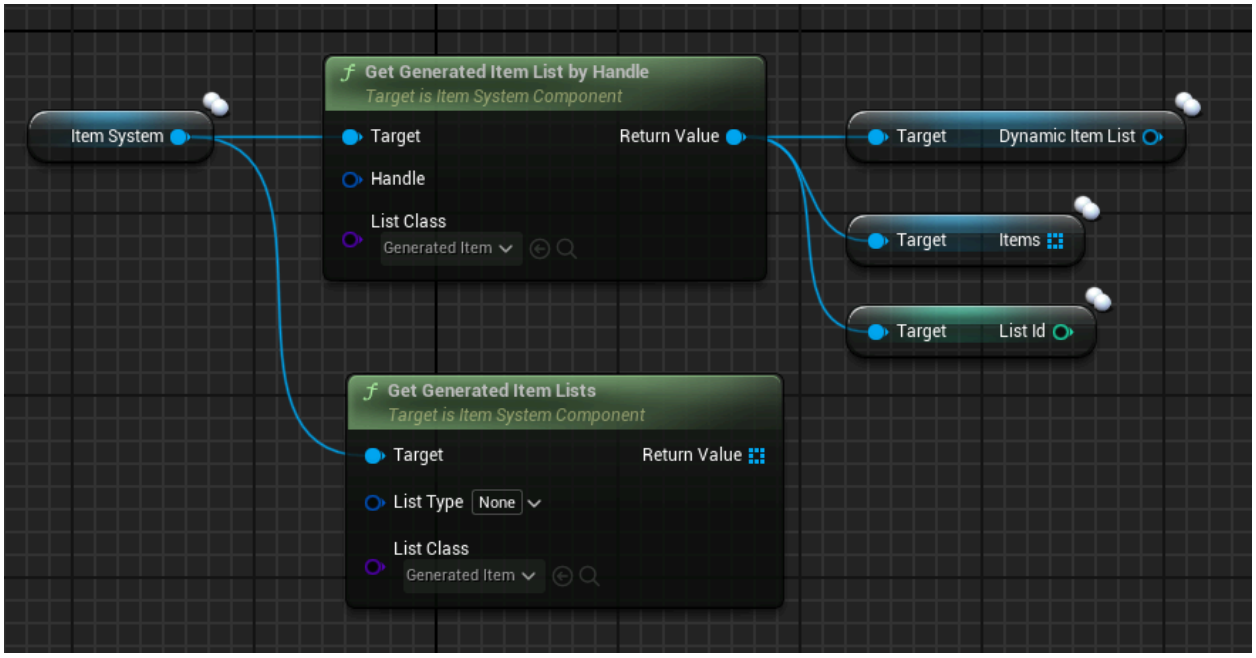


You have to specify which type of list with the Tag.
You can also specify an Id that will be a number associated with the list. If you leave it as is, a number will be generated and be incremented each time. This can be useful if you have orders, each new one will have the last Id + 1.

This function returns a **Generated Item List Handle** that will serve as an identifier to easily retrieve the list. This is a lightweight option to access the list without holding it at all times.

Generated Item List

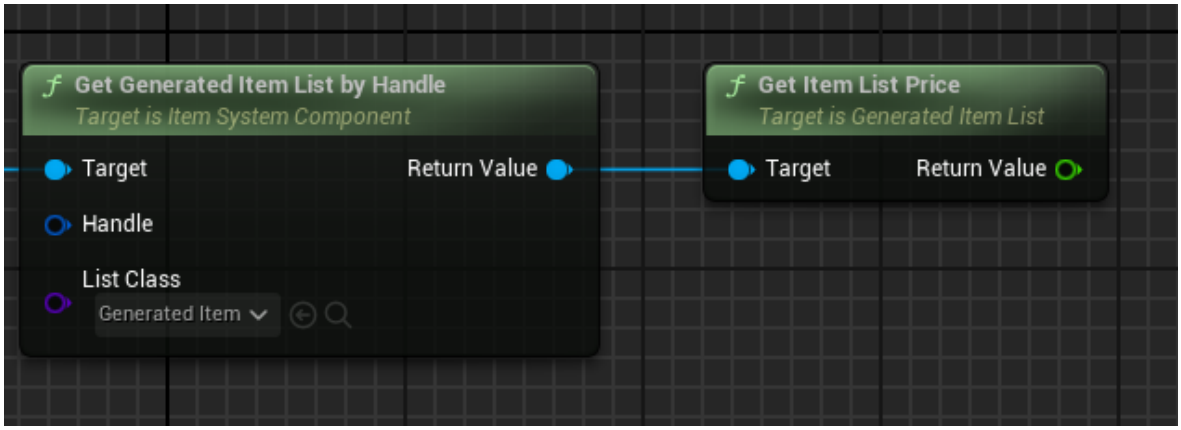
You can access a generated list with its handle using



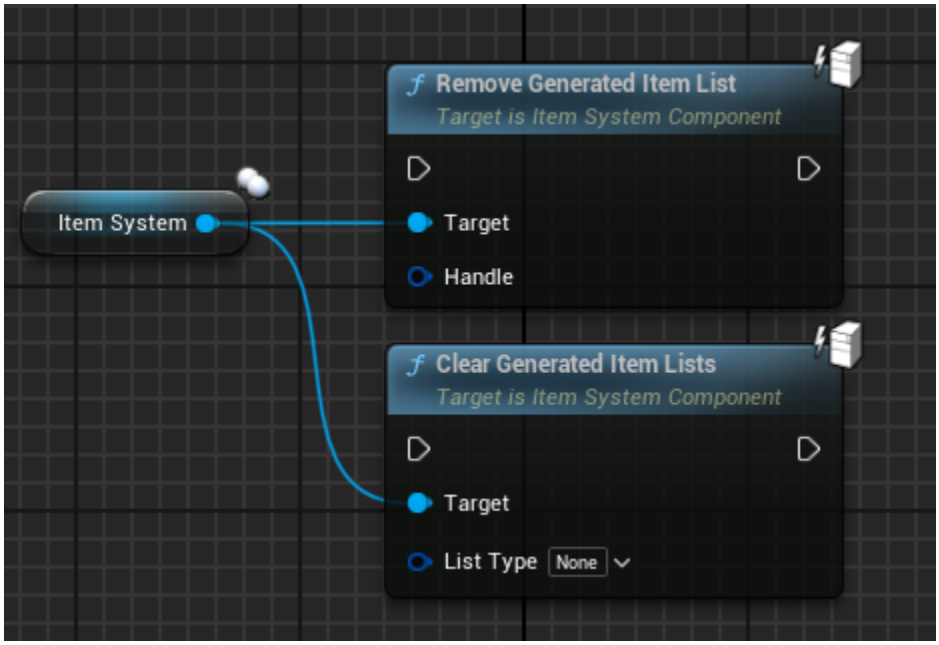
By specifying the **List Class** it will automatically return the Generated List casted to the proper type.

It has a reference to the original **Dynamic Item List Definition**.
It has a list of **Item Instances** that were generated and now form the list.
The **ListId** is the one that was created when generating the List.

If you want to get the price of the list you can use **GetItemListPrice.** It will return the combined price of all items, multiplied by the Price Multiplier of the **Dynamic Item List Definition**.

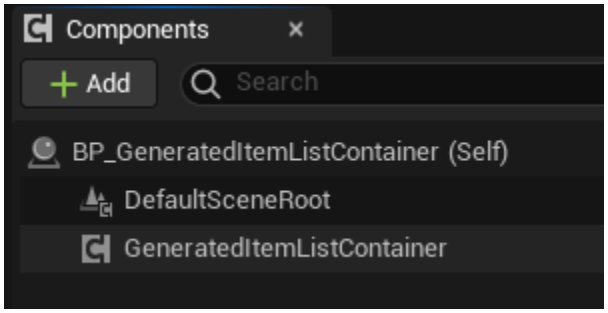


If you want to remove a list you can use

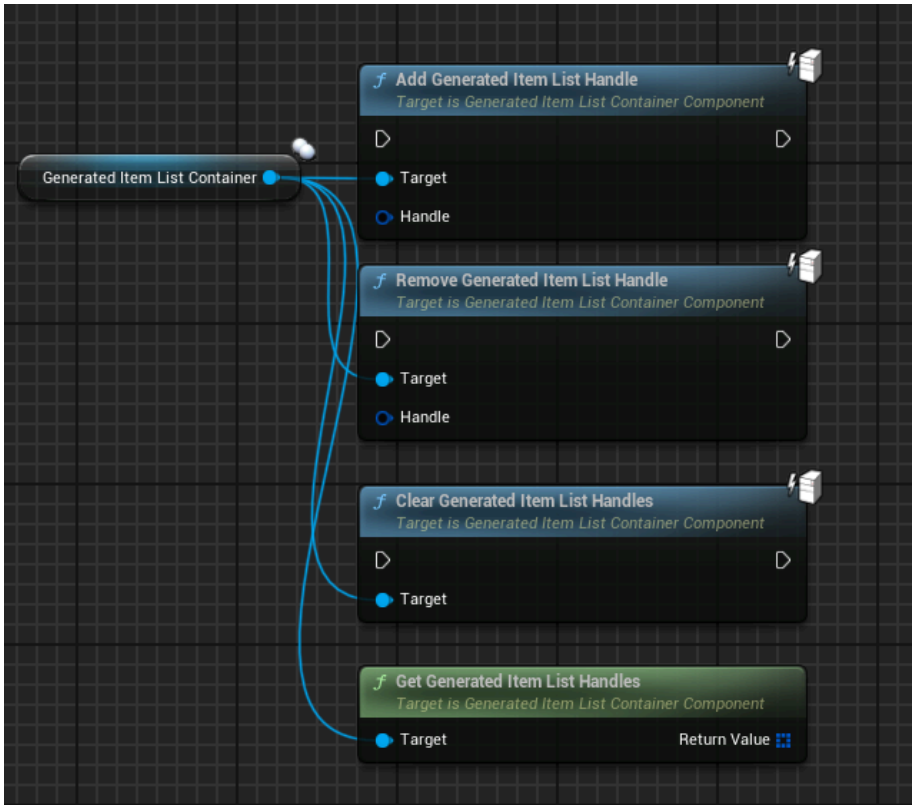


Generated Item List Container Component

This component allows you to hold a list of **Generated Item List Handles** on it. It can be useful if you want to link an actor to some specific **Generated Item Lists**.

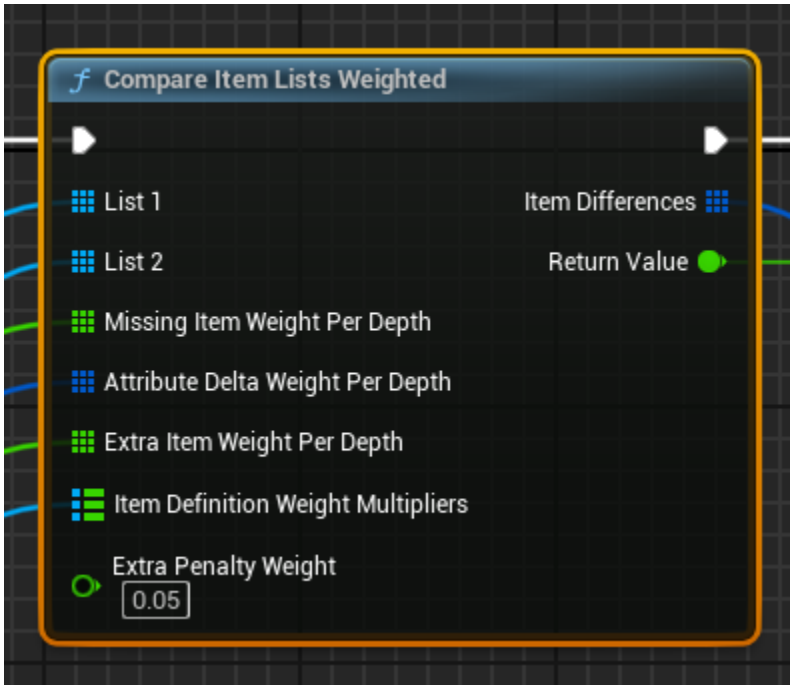


You can add, remove, or get the list handles that are contained in the component



List Comparison

Let's say you have generated a list and you want to compare it to a static list, to see if they match, if there are differences and which one.



You can use this function that will give you a comparison between 2 Lists of Item Instance, and return a score weighted by the parameters.

A score of 1 means that the lists are identical. The more the score goes towards 0, the more it means there are differences. If the score is lower than 0, it usually means that there are some extra or missing items in one of the lists.

Missing Item Weight Per Depth

This is the weight applied to the score given to any missing item, proportional to the number of items expected for this type.
Let's say there are 3 items missing in List 1 out of 4 in total, and the weight is 1. This means that from the final score we will be removing: 3 (missing items) * 1 (weight) / 4 (total items) = 0.75. So, the final score would be 0.25.

The depth is to indicate the dependent items. You might want a different weight based on the depth, so that dependent items have less influence in the final score. This weight will be used when comparing dependent items of items contained in each list.

In the case where every item in List 2 is missing from List 1, we will apply an Extra Penalty per missing item.
For example, if List 2 had 3 * Item1, we would apply a penalty of: 3 (missing items) * 1 (weight) * 0.05 (Extra Penalty Weight) = 0.15.
But if List 2 had 10 * Item1, we would apply a penalty of: 10 (missing items) * 1 (weight) * 0.05 (Extra

Penalty Weight) = 0.5.
The final score would already be at 0 because there are no items that are matching. It will be -0.15 or -0.5 based on the previous examples. You can see that missing more items will have a bigger impact on score.

Attribute Delta Weight Per Depth

This is the weight applied to the score given to the delta difference of attributes between 2 items, proportional to the number of items expected for this type.
Let's say List 1 has 2 * Item1 with an attribute "Doneness" of 1. List 2 has 2 * Item1 with an attribute "Doneness" of 2. The delta for "Doneness" is 2 – 1 = 1. Let's take a weight for "Doneness" of 0.5, less influence than missing items. This means that from the final score we will be removing: 1 (Delta) * 0.5 (Weight) / 2 (total items) = 0.25. So, the final score would be 0.75.

Similar to the missing items, the depth is to have a different weight for dependent items, so that their attribute delta can have less influence in the final score.

Extra Item Weight Per Depth

This is the weight applied to the score given to any extra item, proportional to the number of items in the list.
Let's say there are 3 items that are in List 1 but not in List 2, and List 2 contains 4 items. This means that from the final score we will be removing: 3 (extra items) * 0.5 (weight) / 4 (total items) = 0.375. So, the final score would be 0.625.

Similar to the missing items, the depth is to have a different weight for dependent items, so that they can have less influence in the final score if there are extra.

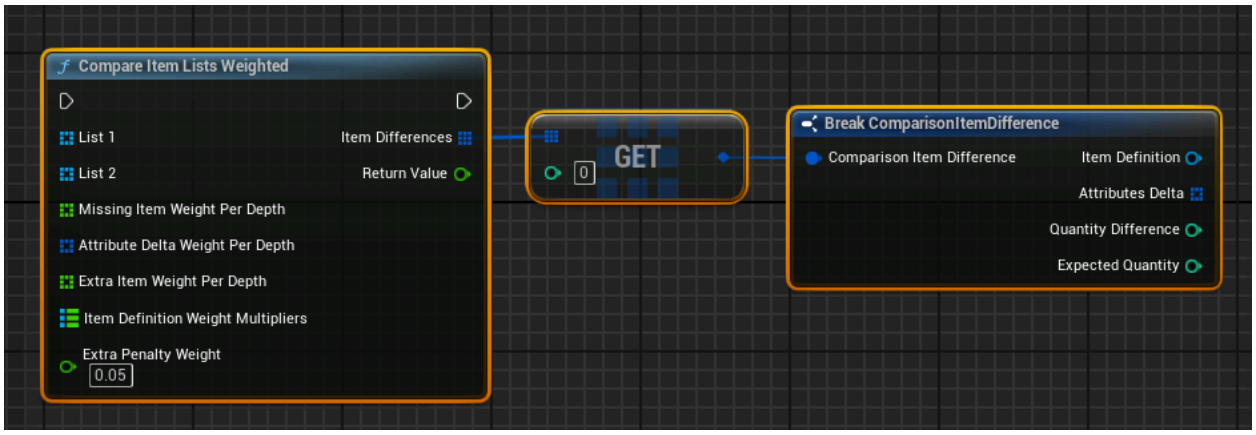
Take into consideration that all these are scored together. If there are missing items, some with attributes delta or some extra, the final score would reflect all that.

Item Definition Weight Multiplier

If you want to multiply one of the previous weights for a specific Item Definition you can define it in the map. This is to give more importance to some specific items over others.

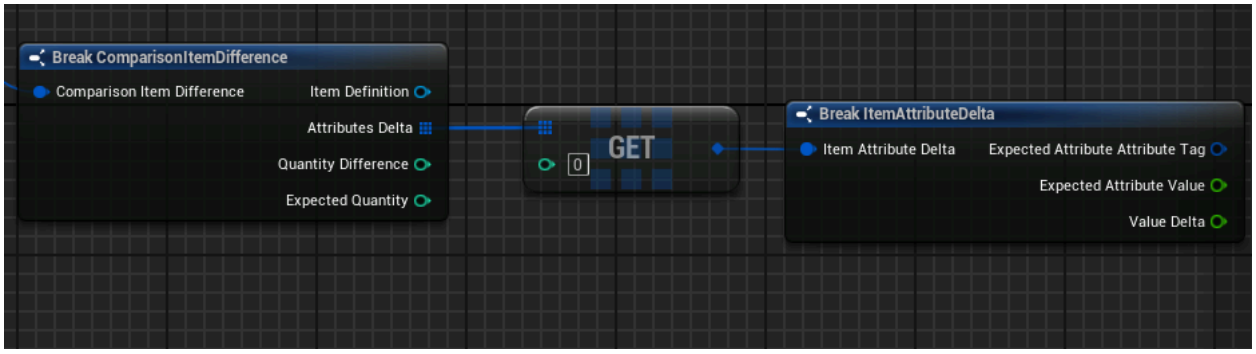
Item Differences

On top of the score, the function returns exactly which items are missing, with delta attributes, or extra.



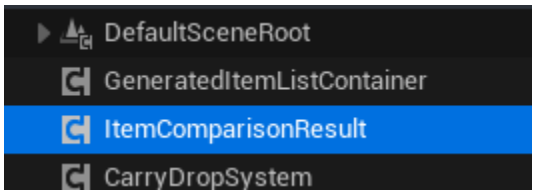
If the **Attributes Delta** is empty, the **Quantity Difference** will tell you the quantity of this item that is missing or extra (negative if missing, positive if extra).
If the **Attributes Delta** is not empty, the **Quantity Difference** will tell you the quantity of this item that has this delta.
The **Expected Quantity** will tell you the quantity of item that was expected (quantity from List 2).

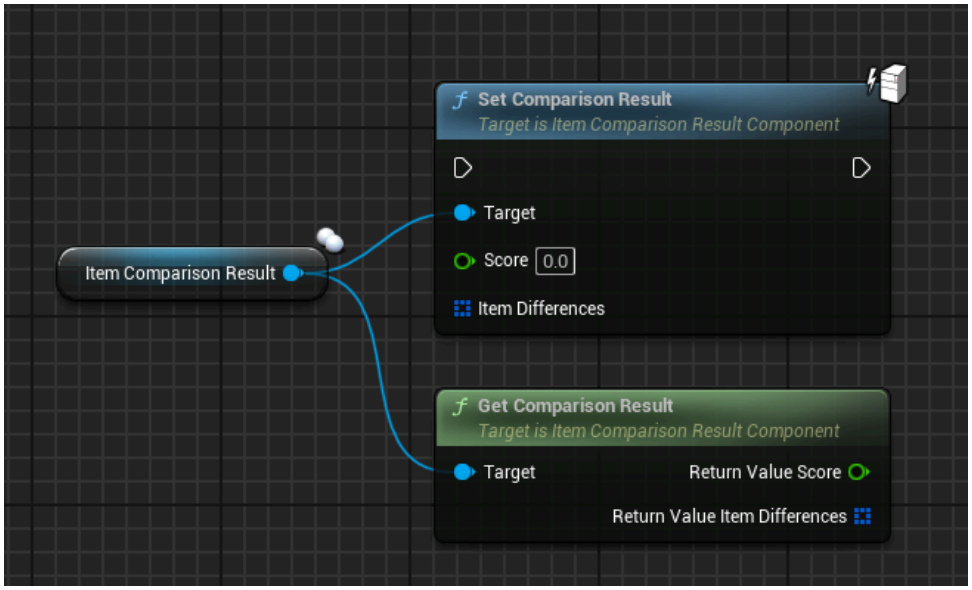
For the **Attributes Delta** you will have the **Expected Attribute** value (value from List 2) and the **Delta** (positive, negative or zero if extra, missing or equal value).



Item Comparison Result Component

As a utility component, you can add the Item Comparison Result Component that serves as a place to hold the result of the comparison (**Score and Item Difference**). Since the comparison can be expensive it is a good idea to store it somewhere and to benefit from the replication of the component if needed.

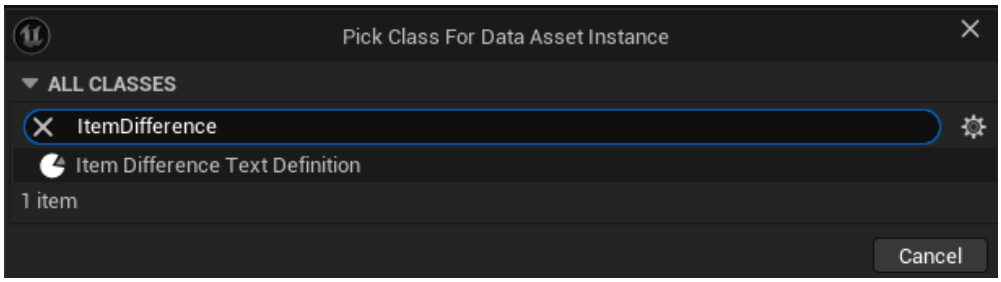




Item Difference Text Definition

In some cases you might want to use the result of the comparison to map it to some texts. For example we could imagine some customers ordering some items and reacting based on the **Item Difference** from what they expected and what they received.

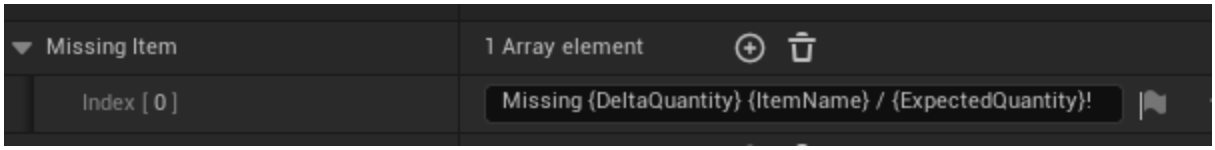
For that you can create an **Item Difference Text Definition** Data Asset.



Overview

Missing Item

You can define multiple texts for when an item is missing (QuantityDifference is negative).

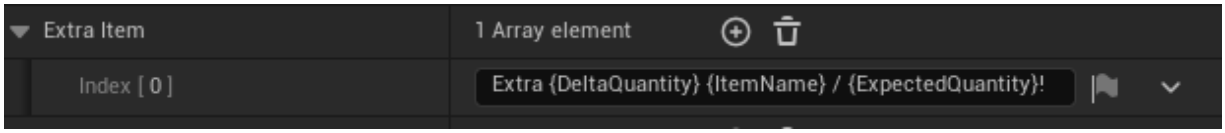


One of these texts will be chosen at random when getting a text for an Item Difference entry.

As you can see there are some parameters {} that you can use by default to customize your text per item: {ExpectedQuantity} / {CurrentQuantity} / {DeltaQuantity} / {ItemName}

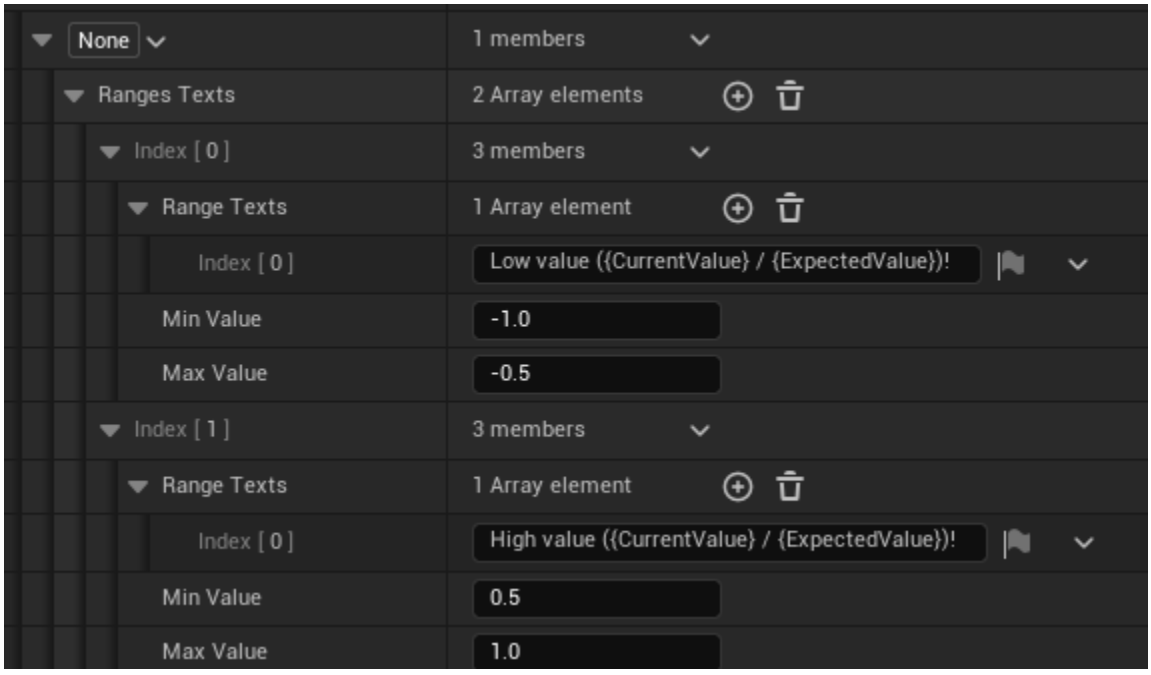
Extra Item

Similar to missing items, you can define some texts for when an item is extra (QuantityDifference is positive).



Attribute Ranges Texts

For any tagged item attribute, you can define some texts for different ranges of the Delta Value.

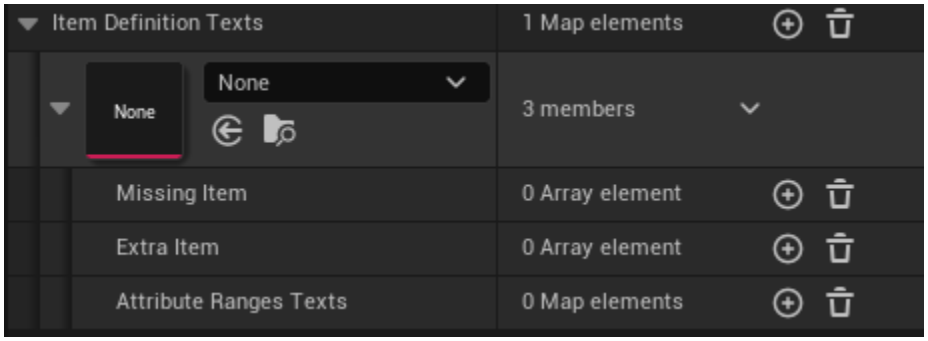


This example shows that for negative or positive Delta Value you could define different texts (a Delta of zero meaning that there is no difference between the value expected and the value compared).

Similar to missing and extra items there are some parameters you can use: {ExpectedValue} / {DeltaValue} / {CurrentValue} / {DeltaQuantity} / {ItemName}

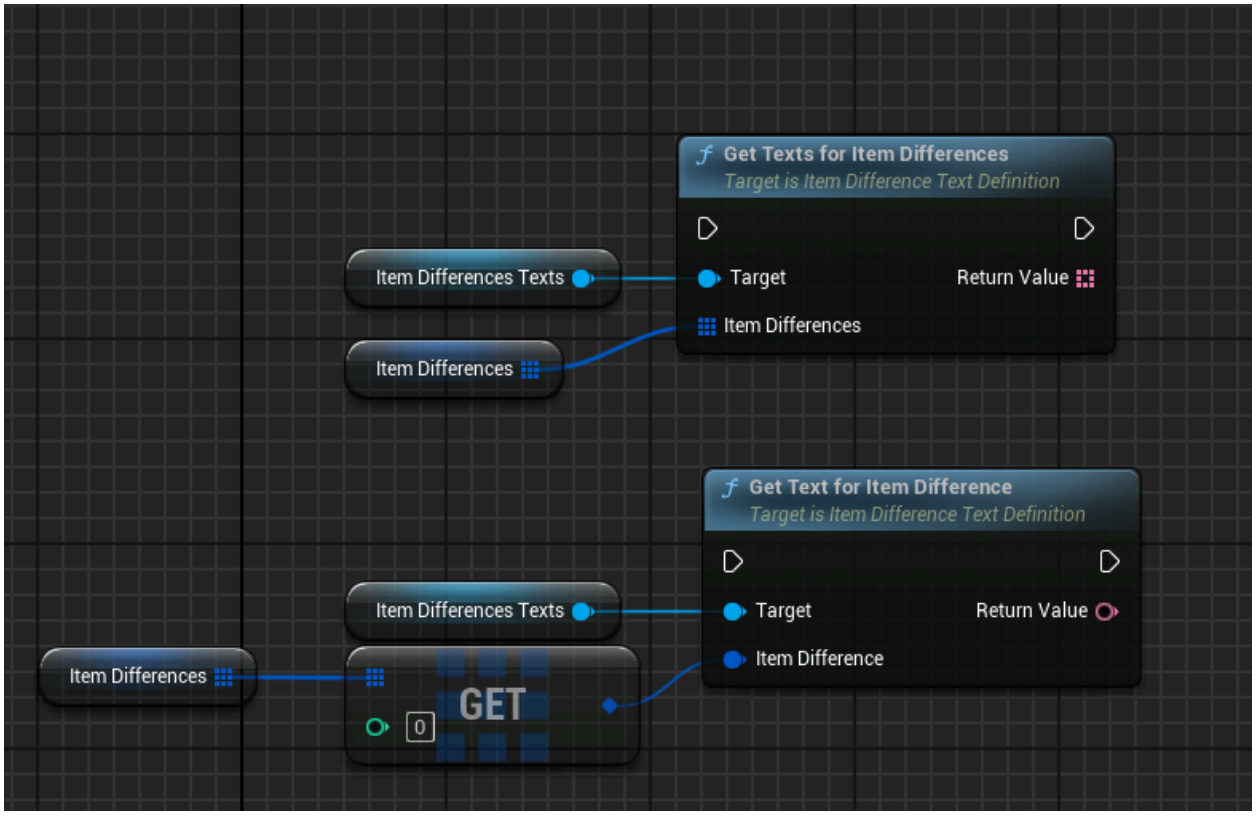
Item Definition Texts

This is exactly like the generic texts, but you can define them for specific **Item Definition.**



Get Text For Item Difference

Once your data asset is configured, anywhere you use it you can call Get Texts for Item Differences and it will return the appropriate texts based on the Item Differences you provide (from a previous comparison).



With all that you should have all the information needed to start using the plugin and customize it the way you want. If you feel there is some information missing or you don’t understand some part, feel free to contact me at louisgirard.games@gmail.com.