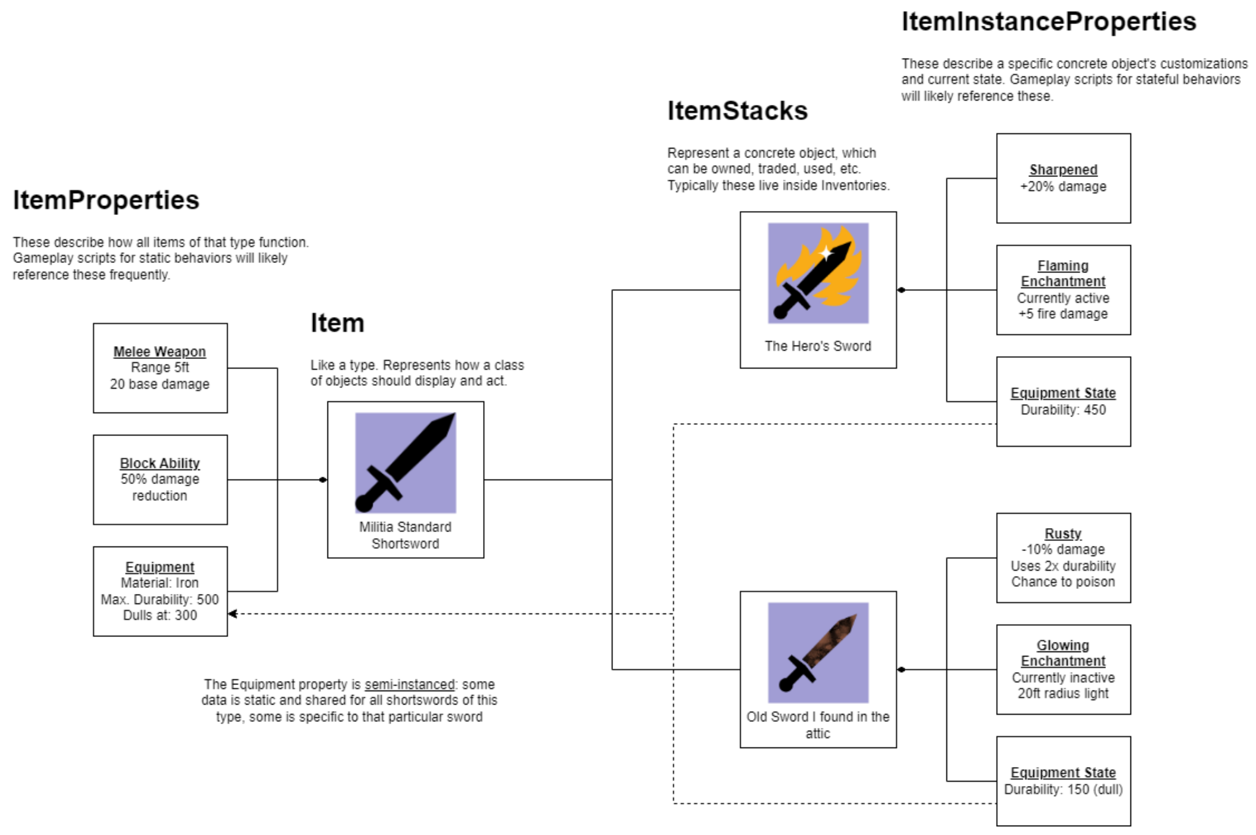
Item Anvil Manual

For v0.5.0 - Last updated 2023-06-18

Overview

Item Anvil centers around breaking down items and describing them in terms of their traits and properties. These work similar to MonoBehaviours and Components: giving an AI a vision cone might mean adding a Sight Sensor component, and giving a sword a bleed effect might mean adding an Inflicts Status or Inflicts Bleed property.


ItemInstanceProperties
These describe a specific concrete object's customizations and current state. Gameplay scripts for stateful behaviors will likely reference these.
ItemStacks
Represent a concrete object, which can be owned, traded, used, etc. Typically these live inside Inventories.
ItemProperties
These describe how all items of that type function. Gameplay scripts for static behaviors will likely reference these frequently.
Item
Like a type. Represents how a class of objects should display and act.
Melee Weapon
Range 5ft
20 base damage
Block Ability
50% damage reduction
Equipment
Material: Iron
Max. Durability: 500
Dulls at: 300
Militia Standard Shortsword
The Hero's Sword
Old Sword I found in the attic
Sharpened
+20% damage
Flaming Enchantment
Currently active
+5 fire damage
Equipment State
Durability: 450
Rusty
-10% damage
Uses 2x durability
Chance to poison
Glowing Enchantment
Currently inactive
20ft radius light
Equipment State
Durability: 150 (dull)
The Equipment property is semi-instanced: some data is static and shared for all shortswords of this type, some is specific to that particular sword

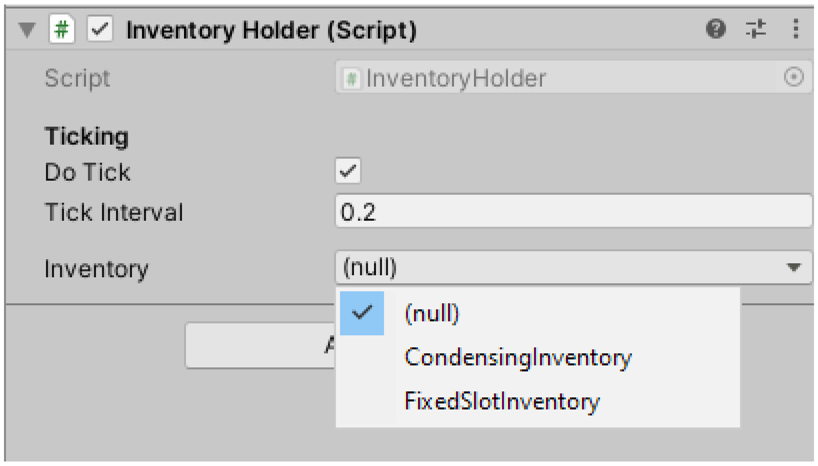
Item Anvil is data oriented. If you are a designer looking to create items, most of your time will likely be spent in the Unity Editor. New items can be created by right clicking in the Project browser and going to Create > Item Anvil > Item.

If you need to define new property types or script behavior for those new properties, expect to open a code editor. Note that unlike MonoBehaviour, almost all behavior of properties is scripted outside the property–it only holds the data.

Getting Started

Creating an Inventory

Item Anvil provides the Inventory Holder component for most inventory needs, which may be set to expand or condense as needed, or set to have a fixed slot count. Switching the inventory type will attempt to preserve its contents.

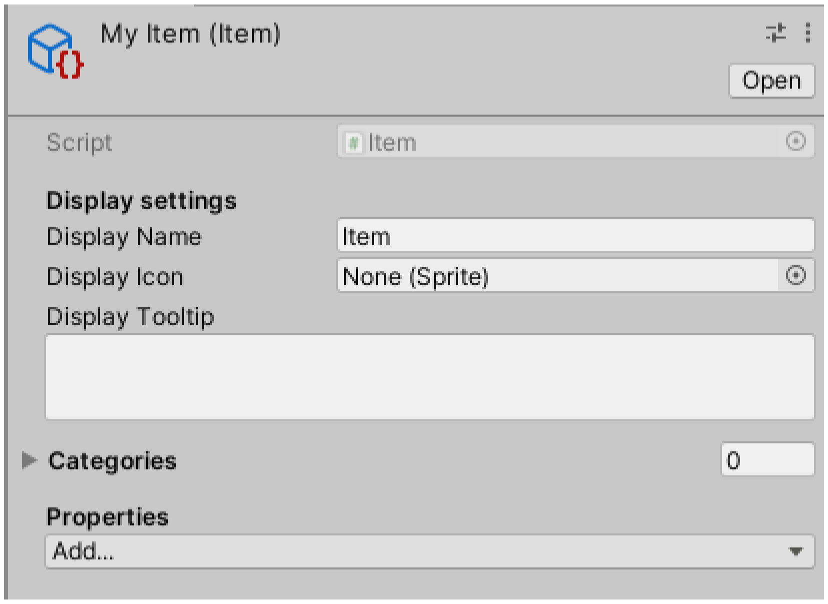


Inventory Holder assumes that you will only need one inventory per object, so most UI and other gameplay scripts are built to work with it. If you need a more complex setup, you can define inventories with the following and actively feed the UI components your inventory data:

```
[SerializeReference, TypeSwitcher] Inventory myInventory;
```

Creating Items

Items can be created by right clicking in the Project browser, then navigating to Create > Item Anvil > Item. They can be given an icon and a description, and a display name independent from the file name. Any characters are valid. You can even create a “mimic” item that pretends to be a different item.



My Item (Item)

Open

Script: Item

Display settings

Display Name: Item

Display Icon: None (Sprite)

Display Tooltip

Categories: 0

Properties

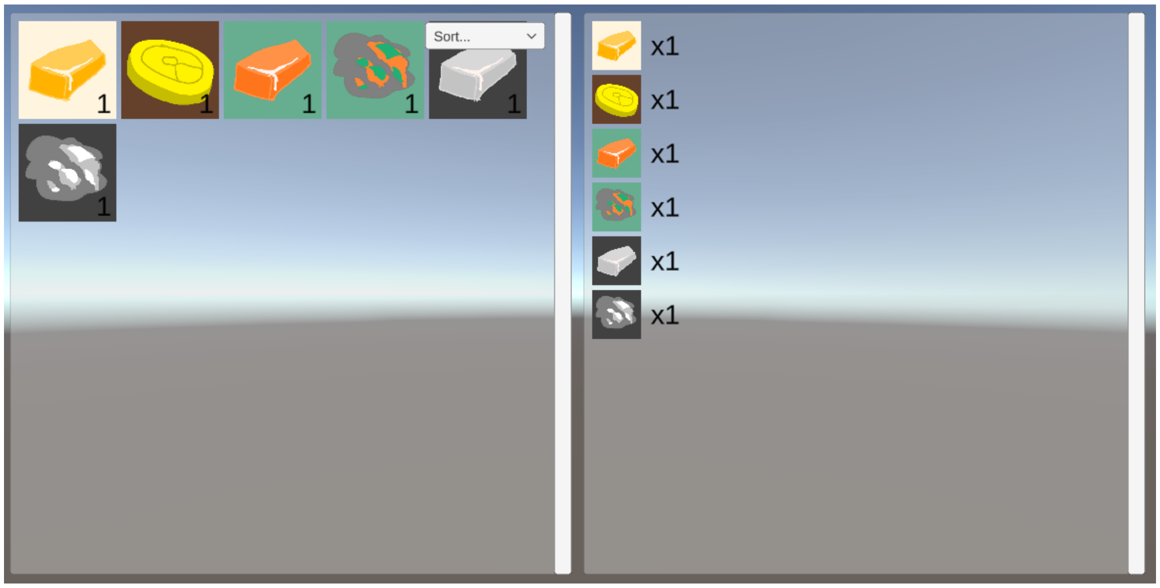
Add...

Items can also be organized into categories, which can be used to create a tabbed view, fuzzy crafting recipes, or quests with open-ended requirements. Categories can be created like Items, under Create > Item Anvil.

Setting up UI

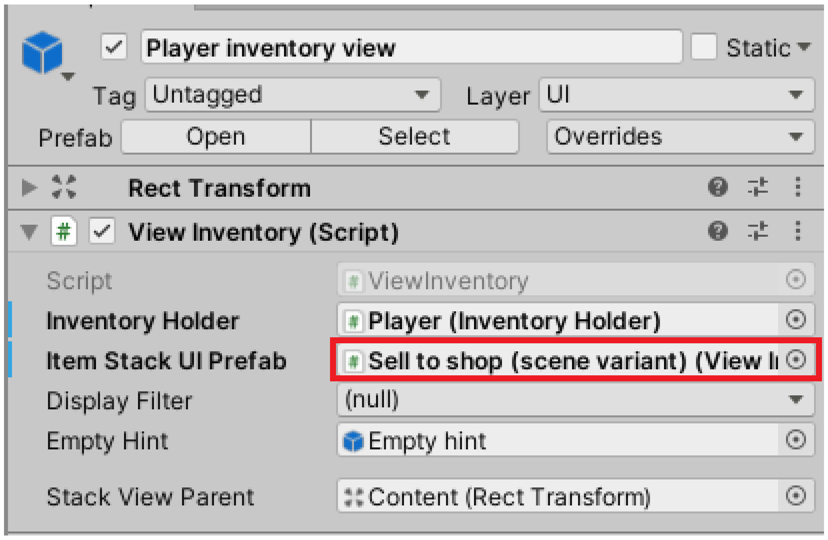
Item Anvil includes some basic prebuilt UI setups for use in any canvas. This includes a grid layout, a list layout, and a shop. Each of these also has their own prefabs for how each slot displays. Both slots and layouts can be customized.

What is shown in each inventory view can be further customized with filters. This can be useful for displaying quest items in a different panel, for hiding currency items, or for inventories with tabs for each category. Multiple views can exist per Inventory, which also allows hotbars.



Left: Grid view. Right: List view. Both use the same Inventory Holder.

The prebuilt UI uses the Inventory View component, which is designed to work with the Inventory Holder component. Typically, the only necessary setup is dragging the Holder into the slot on the Inventory View. In some cases, you may also find it useful to create a hidden scene instance of the slot UI prefab if you need to reference scene objects.



Player inventory view ✓ | Static
Tag: Untagged | Layer: UI
Prefab: Open | Select | Overrides

Rect Transform

View Inventory (Script) ✓

Script	ViewInventory
Inventory Holder	Player (Inventory Holder)
Item Stack UI Prefab	Sell to shop (scene variant) (View I
Display Filter	(null)
Empty Hint	Empty hint
Stack View Parent	Content (Rect Transform)

In Metallurgy, the player's inventory view uses a scene instance so it can sell to the shop. A more elegant solution might assign it by script, but that would be game-specific logic.

If you don’t want to build your own layouts using the prefabs as reference, you can also extend them. When right clicking any prefab, Unity gives you the option to create a Prefab Variant under the Create menu, which lets you override certain properties while still being a prefab. This can be useful for quickly adding sprites. Note that if you create a variant for a slot view, you will also need to create an inventory view variant so you can assign the correct prefab. It is highly discouraged to unpack the UI prefabs, as UI feature updates will not be reflected.

Using Items in Scripts

Most of the time, you will probably want to script around properties. That way, if in future someone wants to add the same functionality to a different item, they need only add the property. This is the preferred method even if the item has effects that are planned to be unique.

However, if you need to reference a specific item, it should be passed in through the Inspector. This is useful when giving players an item, or for defining crafting recipes. You might also want to directly reference an item if you use them as keys in a puzzle.

Items can be created by right clicking in the Project browser, under Create > Item Anvil, and passed to scripts through the Inspector. They should not be located or created by code except by editor tools.

```
[SerializeField] Item myItem;
```

Scripting for Properties

Item Anvil currently supports two types of properties: those that apply to all items of a certain type, and those that are specific to an instance of an item. These are implemented using the PropertyBag class, which ensures no two properties have the same type.

Item Properties are per item type, and are intended for balancing, tuning, and other such changes that might be found in updates. They can be freely altered, added, and removed, and gameplay will reflect those changes even in Play Mode.

Item Instance Properties are owned by a specific item. These can help add customization and flavor to gear. They can also be used for active abilities with cooldowns, as they can receive ticks from inventories. These are not as data-oriented, and on their own, balance changes must happen in code.

Semi-instancing is a technique which helps alleviate this by defining some data as shared on the item property stored on the type, and instanced properties that reference it. This can be useful for cases such as durability or stats with random ranges (+2 to 8 Armor).

For semi-instancing with randomization, it is recommended to store a seed or a percent rather than a raw value. This makes it so no find-replace is needed when balancing. See the table below for an example.

Instanced data (rolled once per item)	Value range (balance changes!)	C# expression and raw value	Final value (round down)
23% (0.23)	-10 to 20	Mathf.Lerp(-10, 20+1, v) = **-2.87**	-3
	10 to 50	Mathf.Lerp(10, 50+1, v) = **19.43**	19
	0 to 10	Mathf.Lerp(0, 10+1, v) = **4.83**	4

Note that final values #1 and #2 are illegal in other ranges.

Crafting Recipes

Crafting recipes can be found in Create > Item Anvil, next to Item definitions. Simple Recipe attempts to match only the types of ingredients. Fuzzy Recipe searches using filters, and can match based on categories, properties, and even quantity. It can also optionally copy instance properties to the output item(s).

Moving Items Between Inventories

The Transaction class is a safe way to move items from one inventory to another. It will not transfer items if there are insufficient inputs on either side. If an error occurs, all items will be returned to their original inventories.

Filters

Item Anvil’s advanced features use a form of querying called Filters. Here are the current filters available:

- **Match Type(s):** Only match items with the same type(s).
- **Match Category:** Match items in the same category.
- **Match Reference:** Match certain parts of a “reference” ItemStack, including quantity and instance properties.
- **Match Any/All:** Allows combining individual search criteria.

You can make your own filters by extending the ItemFilter class, and it will automatically show up in the dropdown.