# Item System V0.1

## Table of Content

# Getting Started

## Setup

If you are not going to use the default scene (so in most cases!), add the _Inventory prefab from the Prefabs folder to the hierarchy. Then you are ready to start.

## Creating Items

First you should load in your object and make it a prefab, and store it in the Prefabs folder. Now load in the inventory icon for the object and store it in Sprites/Items as .png files. Open them in the Unity Editor Inspector and set Texture Type to 'Sprite (2D and UI)' and they can be used as items.

To make the inventory know about what model is related to what item, it needs a new entry in the database. Insert the fol owing structure in the Assets/StreamingAssets/Items.json folder

```
,{
     "id" : a_unique_number,
     "name" : "a_name",
     "handle" : "the_image_name_without_png_extention",
     "prefab" : "the_prefab_name",
     "descr" : "an_ingame_description_of_the_object"
}
```

The entry must be inserted before the last closing ] right at the bottom of the file. Also note the comma in front of the {.

## Hovering Description

To prevent the player from being lost, you can make a descriptive text hover over any object.

First add the ItemController/QuestItem script. Insert the Display Text in the Inspector. Below, the script prompts for Hover Text.

See the ItemUtils prefabs and drag and drop the _HoverText prefab onto the target object. And drag and drop the fresh instance into the Hover Text slot.

## Interact With The Items

To store items from the ground in your inventory, attach the StoreItem script

(Assets/scripts/ItemController/StoreItem) onto the prefab. After, change the ID in the inspector under Store Item to the unique one from the database. Also check the Collider's (e.g. Box Collider) "Is Trigger" option.

To move around items, attach the MoveItem script from the same directory to the prefab. Also check "Is Trigger".

You can now pickup stuff with E.

## Contain Items

To make a lootable box, dresser, ... make all contained item prefabs children to your container and throw the ItemContainer script on there. Check "Is Trigger". Finished.

## Making Things Solid

Al objects we created so far are walk-through only. To change that uncheck the objects "Is Trigger". In the menu bar choose GameObject>Create Empty and make this object a child to the item. In the inspector, right click on Transform and click on Reset.

Now Add Component>Physics>Sphere Collider (or whatever shape fits best), adjust the Collider settings in the Inspector, and check "Is Trigger". Drop the TriggerItem script on the Empty, and you are set up.

You can use this procedure for items (movable and storable) as well, as item containers.

## Interact With The Environment

There is a little bit of coding involved if you want to trigger certain behavior, but bear with me.

Throw the ItemUtils/_Trigger prefab on the object, you want to interact with. Now go to the inspector and insert the Trigger Item id (see database). If the player collides with the _Trigger and they have this item currently in their inventory, the _Trigger will, ... well trigger a certain behavior.

To define such behavior, define it in a separate script and let the component implement the TriggerObject.Action interface. Define the to be triggered behavior by implementing the pure virtual execute() function.

# The Long Version

For the rest of the documentation, there is an in-depth description for fine tuning stuff. If you do not want (or have) to alter the appearance and behavior of the presets, there is not much to see here.

One last thing: you might want to read the section on The Item Database, if you are going to insert new elements into the Json file.

## The Inventory

The inventory is quite easy to set up. Just create an empty GameObject in the scene, name it

whatever you feel like. Only two more things left to do. First go to 'Add Component' in the empty's Inspector and select scripts→Item Database. You now have all the item information available, for the whole game. Great!

But you want to use it, right? So we want a system to actual y do something with the data. With the empty GameObject still selected, click 'Add Component' and scripts→Inventory.

Look again in the Inspector. There is an options available. Stack Size, so the number of items of a kind – sharing the same ID – allowed to be stacked, i.e. allowed in the inventory.

## The Item Database

It gets a little more technical, unfortunately. But not too much. The information, that are required for now are the fol owing:

```
id:     a unique identifier. Only items sharing the same id
        will be stacked.
name:   that is the identifier, the user will end up seeing.
handle: all data have a handle, textures, images, icons,
        sounds will use this name, in the related file.
prefab: the big exception. This is the name of an attached
        prefab. Objects may or may not share the same handle
        or id. They are independent.
descr:  the description, that may end up in an info box,
        displayed to the user. Get creative ;)
```

Why is that important to you? Well for a new item you are creating, you will need to provide at least some information. If you don’t know what the handle or the prefab will be, or miss a

description, fear not, you will get an error message, but the system will work with the data provided. Only id is real y necessary to introduce an item.

Unfortunately I don’t have a script, for adding stuff to the database right now. However it is stored in Assets/StreamingAssets/Items.json. The data format[2] is easy to grasp, so for now you may want to add items manual y. If there is a mistake (missing comma or quotes), the item system should warn you. If you are entering prefabs, that do not exist yet, or handles, the system will also warn you, but most things will work. Messages in the Unity console are mostly friendly reminder to add a missing component.

## The Items

That’s where the fun begins. Just create / import / draw / - I don’t know, you guys are the artists :P - your object and click ‘Add Component’ scripts→StoreItem. Fair enough. The downside, the object must have a Collider attached, and the option “Is Trigger” must be checked. Or else, you won’t be able to interact with the items altogether.

And a second look in the Inspector reveals the Item ID box for the StoreItem script. 0 is an invalid id, so you must get one from the database.

Action Key is irrelevant to you and will be removed soon.

In fact, you can use the system right now. Run to your freshly created item, press the action key (E) and the item goes straight to the inventory. Great!

And stays there. You can test it only, if you use the test build and are using id 1, by pressing Q.

That will dump the item to the ground. This ‘feature’ will be removed.

Also note, that the item can only ever be restored, if prefab to the ID is set to the right name of the objects prefab. That has performance reasons.

## The Item Container

*[Major Change:]*

Only picking up items from the floor. They are dirty, and we are no animals, right? So you can store items on desks, dressers, in front, on top, under, left, right, inside, well everywhere to be honest. If the item itself is unreachable to the character, because of one of these reasons, make the item ‘blocking’ access a container, by adding the script ItemContainer. Al the items that are to be contained must have StoreItem script attached and must be children to the container object.

And you may notice, that it does not work like this. We are left with two options.
One: check the “Is Trigger” of the Collider component. If you do this, you can walk through the furniture and that is probably not, what you want.
Two: store an Empty Object inside the container and Add Component>[Sphere/Box/...] Collider, check “Is Trigger”. Now add the TriggerItem script to this Empty.
To adapt the bounding box, you may want to reset the position and size of the Empty by right clicking on Transform>Reset.

Pick up the items from a container: Great fun!

## The Trigger Item Script

I liked this feature so much (see the section above), that I made it available to all physical components: StoreItem, MoveItem, ItemContainer. An object with this script attached will call its parents public OnTriggerStay(Collider) function. (MoveItem.OnTriggerStay, StoreItem.OnTriggerStay or ItemContainer.OnTriggerStay in that order)

Create a children without a renderer attached and use it's box collider as a trigger, by checking "Is Trigger" and adding the TriggerItem script.

To not confuse the feed update too much, make sure that the very item, has it's "Is Trigger" unchecked.

## The Quest Item Script

A player might be overwhelmed with the sheer number of object in a scene. They might not be able to tell the importance of an item, or if they can even interact with it. This is the purpose of the QuestItem script. It snaps a 3D text to an object and keeps it relative to the isometric camera angle.

To create your own text – other than the prefabricated ItemUtils/_HoverText – make sure a Text Mesh component is attached to your text object. Add whatever components you need. To keep things organized, make your new object a child to the target item.

Now add the ItemController/QuestItem script to the target item (not the text object!) and alter the Display Text inside the Inspector view.

Note that the script prompts for a Hover Text instance. This is where you pass the instance to your custom text object. Prefabs have to be an instance in the scene first.

If the hovering text should be disabled from a script, call QuestItem.HideText().

## The Item Feed

Okay! You will have seen, … well nothing, cause there is nothing to be seen. What a shame.

Go ahead to the inventory object in your hierarchy. Right click→UI→Canvas. Here we will draw everything required. Add the script ItemFeed, we will inspect it’s settings later. For now, add a Panel to this canvas (Right click→UI→Panel). This is the background graphic for the panel, be nice, attach a Layout (Add Component→Layout→ *choose one*). It would work without I guess, but that is pure madness.

The great advantage comes with the next step, the actual display slots. A slot is just another panel and child to the feed panel, we just created. And the slot must have one UI→Image and UI→Text as children.

If you let the Layout handle position and scale of this slot, you can now simply copy paste your blueprint slot and they will be nicely arranged.

Now for a dirty hack. To mark a new update on the feed, we highlight the top element in the list. So select the first slot and Add Component→UI→Effects→Outline. The settings are handled through the script, so no need to adjust anything, right now.

Almost finished, you can now revisit the canvas, we created in the first step and drag and drop it's child – the feed panel – from the Hierarchy view right inside the Feed Label box in the Inspector under Item Feed. But wait, there is more.

Fade Away Speed sets the time it takes, for the whole feed to vanish. The animation is a sinoid interpolation. Fade Away Slot sets the time for the "slot updated" marker to disappear. The effect is only noticeable, if this value is small er than Fade Away Speed.

Last but not least, you can control the color of this very marker via Selection Slot. But beware, the script will override the alpha channel (thus it's opacity). If you are playing with the colors, have the alpha at 255. To match the in game color.

That's all there is to it. If you defined all prefabs' and handles' data to the database, you now have an inventory.

## The Trigger Object Script

A game would not be any fun, if there is no interaction with the environment whatsoever. The TriggerObject script is one possibility to implement an interaction.

First create an Empty object, and give it a collider. The collider will later establish the area, from where it can be triggered. Once that is done, attach the ItemController/TriggerObject script. In the inspector, you can select the items id, that should be possessed by the player in order to trigger the object. A related prefab would be ItemUtils/_Trigger.

Now make your custom trigger a child to the object you want to trigger. In the script of the – now – parent, you want to target, make the class implement the TriggerObject.Action interface. If you are not sure how to do that, it works like an inheritance:

```
public class Door : MonoBehaviour, TriggerObject.Action {...}
```

Your script must now implement the pure virtual, public function execute(). This is the function that will be called by the trigger object.

## Fine Tuning the Prefabs

In my great generosity I provided the prefab to the inventory. If you are lazy, and you should be, just replace the graphics, the size, color, opacity, and you save yourself some time.

## What Is About to Come

It is time to step up the game. With the next major update, there will be ways for the items to interact with the environment inside the Editor and not only the scripts, as is now.

There will be an inventory overview in the pause menu. Getting all fancy here.

## Bugs, Ideas, Comments

If you see flaws in the implementation, the design, or this documentation; if you wish for a convenience to be implemented; or if you just have THE idea on a killer feature, hit me up @vinzent or DM in Discord.

Thanks for your time!