GRASBOCK’s VR Inventory

Intro

Thank you for purchasing my VR Inventory. It will provide you with a very simple but easily expandable Inventory for Virtual Reality using Unity’s built in XR Support. This should provide a future proof environment almost independent of the SDKs used.
This Asset was created with a lot of attention to ease of use, flexibility and stability. In the following you should find anything you need regarding information about this asset. The code is commented and is in the GVRI namespace so it won’t interfere with any existing code. I tested this using an HTC Vive. I do not have any experience with oculus, but it shouldn’t be too different. I will do my best to explain how everything works and links together.

If something is difficult to understand, write me an email and I will try my best to provide you with a better explanation. After all, the documentation improves with feedback.

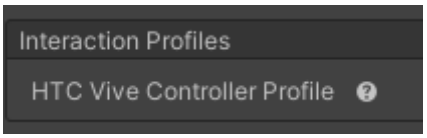
Quick Start

1. Enable Unity XR Support
   If XR-Plugin-Managment is not installed go to **Edit > Project Settings > XR-Plugin-Managment** and there check **Install XR Plugin management**.
   In the XR-Plugin Management check whichever Plugin Provider you are using.

   In case of OpenXR you need to set up some additional settings.
   XR-Plugin-Managment > OpenXR

   Here you need to set an interaction profile (basically which controllers you will use).
   In my case they were the HTC Vive Controllers

2. (This might not be needed in the future) Download the “XR Legacy Input Helpers” Package from the Package Manager. **Window > Package Manager**
   I read that one could implement the XR Input Helpers fairly easily, but this way is still easier and more robust.
3. Open the SampleScene from the Scenes Folder and test if everything works

This should be the basic startup procedure. Surely you would want to create your own Scenes though. For the start everything you need is contained in the Prefabs Folder. Keep reading. For a Custom Scene:

4. Create the environment
5. Add the “XR Rig” Prefab(this will represent the center of your playspace). To it are attached the camera and hands, as well as the “Tracked Pose Driver” which you imported with the “XR Legacy Input Helpers” package. **The only important part to this package is the Hand**
6. (Optional) Add a QuickAccessInventory Prefab to one of the Hands (preferably those that have the Tracked Pose Driver Attached).
7. Populate your Scene with Items and Inventory Slots. The Hands recognize the Item Script attached to Items allowing you to grab them. The Inventory Slots are the containers you can put Items into and pick Items from.
   That’s all. For more details visit the Definitions.

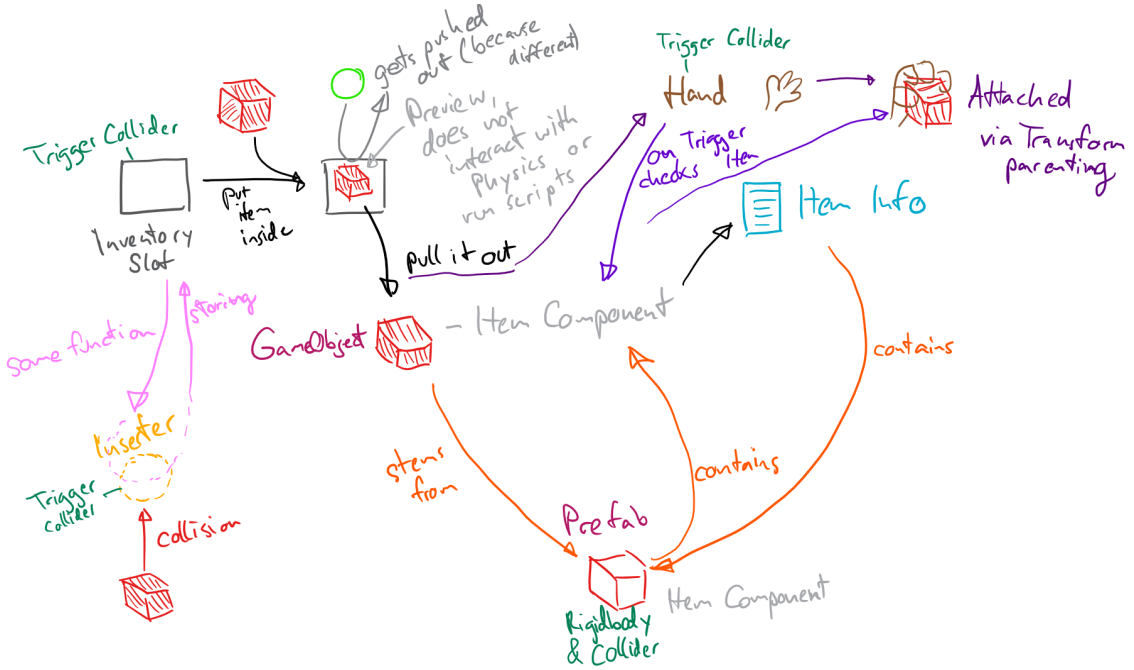
Also check out the LittleRoomba Scene!

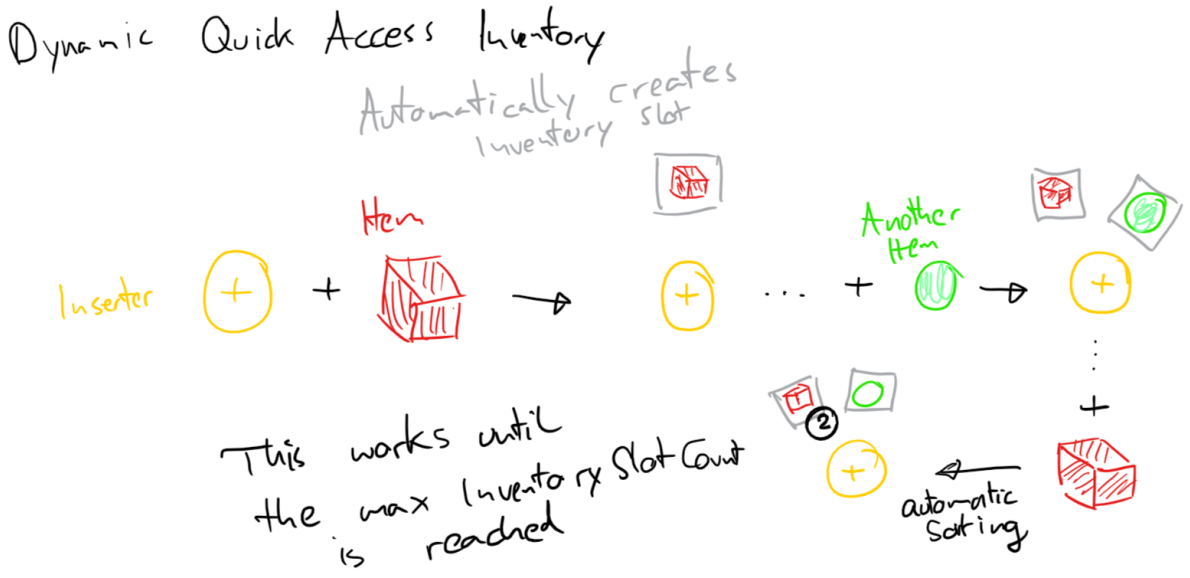
Definitions (what everything does)

- **ItemInfo**
  The ItemInfo is what represents an Item. However it's only information, not a GameObject. It contains the prefab that is to be Instantiated if someone for example picks an Item (in the gaming sense) from a Slot. It is meant to be extended by You. For example, one might want to inherit from the ItemInfo class and create a Weapon ItemInfo which contains information such as weight, durability, and damage. So it shall serve you as a base you can build upon.
- **Item**
  An item is the Component that the other Scripts need to identify an Object as an Item. It also stores the ItemInfo. The only functionality it has besides that is the storing of its Rigidbody Component so it gets pushed out of Inventory Slots that are full.
- **Hand**
  The Hand is a Component attached to a GameObject which has a Trigger Collider attached to it. You need a Rigidbody Component to interact with Inventory Slots. It allows sensing of Items and Inventory Slots. It records the UserInput and allows you to collect and pick Items out of Inventory Slots.
- **Slot**
  Handles the storage of Items by storing their ItemInfo. Whenever the item count changes, it sends a notification to all the delegate functions that have been given (optional and have to be set using scripts).
- **Inserter**
  This one simply tries to collect Items and using a delegate function communicates with another Script that decides which Slot the Inserter should insert the Item into. If the Slot returned by the delegate function is not null the Inserter attempts storage in the Slot.
- **Inventory**
  An Inventory is a Component that manages multiple Inventory Slots. It's more of a base building block to make more complex Inventories. Imagine it like a locker room where each locker is one inventory.
- **DynamicInventory**
  This a bit more advanced Inventory sorts the Items you put into an Inserter automatically and creates new InventorySlots accordingly.
- **QuickAccessInventory**
  This Inventory checks for User Input to open the Inventory and close it again. It's meant to be attached to a Hand.
- **DynamicQuickAccessInventory**
  Same as QuickAccessInventory but inherits from DynamicInventory
- **ItemCountUI**
  This component is for displaying the item count in an InventorySlot and adjusting the size of the Text so that it never clips over a certain radius. It tracks a target (normally the Camera), based on its Origin.

- **CoreInventory & CoreSlot** provide some functionality that the inventories share

System Overview





Suggested Setup (the DynamicQuickAccessInventory is optional):

The XR Rig is essentially the 0 position of the Player in their room. By moving the XR Rig, you can move the playspace of the player relative to the World.
Custom User Input can be realized by changing the CommonUsages variable in the Hand script.

GVRI Components can be added using Add Component > Scripts > GVRI;

How to Add new Items?

Short:

1. Create Item prefab
2. Add Item Component to the Item prefab
3. Create > Inventory > ItemInfo
4. Add ItemInfo to Item Component
5. Add Prefab to ItemInfo

Detailed:
From inside Unity, press inside your project folder, Create > Inventory > Item
A new ItemInfo will appear.

In the inspector give it a prefab that represents the GameObject of the Item.This GameObject will be instantiated as soon as it gets pulled out of an InventorySlot. The same prefab will be used to preview what's inside the InventorySlot.
You can now assign it to an InventorySlot Component in the Inspector of an InventorySlot GameObject and set the StoredItemCount.
However to make your Item pickupable, you have to add an Item Component to the Prefab. That Item component needs to contain a reference to an ItemInfo Object. That way, the Inventory Slot knows which ItemInfo is to be inserted when. This is useful for example, if you have multiple visual types of flowers, but don't want to differentiate between each one of them.
The Prefab needs to contain a collider, so it will be registered by the Hand, an InventorySlot or an Inserter. A Rigidbody is only needed for it to be registered by Inventory Slots and Inserters, because the Hand already has a Rigidbody.

Callback Functions

You have the Option to get notified whenever an Item is Added to an Inventory Slot or an Inserter. To get notified by the InventorySlot:
Create a function that takes the InventorySlot as a parameter.

```
void InventorySlotChanged(InventorySlot invSlot)
```

Then somewhere in your Script

```
someInventorySlot.subscribers.Add(InventorySlotChanged);
```

Whenever the stored ItemCount in the InventorySlot changes, the InventorySlot will then call all of its subscribers.

The Inserter is different, because it requires the function to return the InventorySlot it is meant to attempt storing in.

```
InventorySlot InsertionIntoInserter(Inserter inserter, Item i)
```

Then somewhere in your script

```
someinserter.target = InsertionIntoInserter;
```

Note, that the inserter has only one target, because the Inserter won't try to insert at multiple InventorySlots at once.

You can see implementations of these in the DynamicInventory.

Things to keep in mind when tinkering

- The Hand needs a RigidBody Component and a Trigger Collider to allow you to pick up Items from an InventorySlot.
- This applies to Unity in general, but use a better Collision Detection on the Items, otherwise they will fall through stuff if they are too fast (You can throw very fast in VR!).
- If you want to change the Input behaviour of the Hands or the Quick Access Inventories you need to change the CommonUsages variable (more Info here https://docs.unity3d.com/2019.1/Documentation/ScriptReference/XR.CommonUsages.html) and adjust the input() function accordingly

FAQ

- *I integrated the package into my project before testing the demo Scene. Help PLZ!*

Please make sure the demo scene works, otherwise I cannot narrow down the problem.

- *Why is my Hand not able to grab anything from a Slot?*

Make sure a RigidBody is attached to your Hand and “IsKinematic” is checked on it

- *Why is my Object falling through when I drop it into the Inventory Slot?*

Check the error messages in the console. Usually this is because a component on your Item Object was deleted in the wrong order. The first time you drop an Item into an Inventory Slot it generates a visual representation by removing all non visual components (like RigidBody and Colliders) on it.
Solution: In the Slot Monobehaviour Class there is a `CreatePreviewCopy` Method. Before the for loop where all components are removed, Destroy the offending component first.

- Why is my HeadSet not being tracked even in the Demo Scene?

See into Edit > Project Settings > XR Plugin Management. If not yet done install the XR Management System. There you have to check your XR Plugin providers like OpenXR/OculusSDK/other. If you selected OpenXR you need to add an Interaction Profile to the OpenXR settings.

Future Updates

Depending on how well this Asset is received (reviews, sales, suggestions and questions) I will continue to update this asset with new features such as

- Item Stacks which allow a user to hold more than one Item in their hand. For example in a small box.
- Depending on Unitys upcoming Updates; a new Input System
- More examples on how the systems can be used

Contact

If you experience any bugs, have some suggestions or questions, don’t shy away from writing me an email to grasbock@gmail.com
I normally answer within at least one day (depending on which side of the planet you write from) and I am known for doing my best to give you the support you need (and paid for).
I would be kind of you to test the demo scenes before asking though.