*By FidelTFG*

*Skill Level Required:* ***INTERMEDIATE***

*Coding Required:* ***None/Little***

The Player Inventory System is designed to make implementing an inventory system in your game simple. The Player Inventory System was developed to replicate yet improve upon the functionality of other inventory systems without the bloat. The Player Inventory System will allow you to add this functionality to you game with ease.

This document explains how to set up the Player Inventory System in its default configuration. Plus an overview of how the system functions and some handy tips on ways you can use it in your games.

**[Full documentation and tutorials can be found on line here.](https://fideltfgsoftware.000webhostapp.com/projects/inventory/index.php)**

**Before we start...**

This is not a complete game that you can simply modify and release. The player Inventory System is design as a drop in solution to the most common of inventory problems. This system does not deal with player movement or interaction with your world. It provides methods to pass items and properties in and out of the system. Although simple to implement, just like many other assets it still requires that you put some effort in to making the most of it. Please read this and other documentation found on line. Just about every question is answered here somewhere.

**Setting up the Player Inventory System in Your Game**

The Player Inventory system is designed to work with best in conjunction with Unity's First or third person Character controllers.

To add The Player Inventory system to you game follow these steps;

1. Add the *InventorySystem* prefab to your scene. This can be found in the prefabs folder. There can be only one Inventory System per scene.
2. Add a *FirstPersonController* or *ThirdPersonUserControl* component to your player object if you don't already have one.
3. On your player object add a *InventoryPlayerController* component. This provides functionality to pick up items and is only required if your player needs to do so. Depending on how your player is set up you may need to change the settings on this component for it to function correctly within your game. The default setting work well with Unity’s standard character controllers.
4. Drag your player game object in to the Player value of the *InventoryController* component on the *InventorySystem* Object. If your player object is tagged 'Player' you can skip this step. The *InventoryController* will find and include your player at start up.

**The Demo Scene**

Included in with this package is a Demo scene and a collection of items for you to mess about and test with. The demo is not intended to represent a completed game, but instead to demonstrate how the system works and how you as the developer can build your item catalog and incorporate it in to your games.

**How it Works..**

**Inventory Items**

An Inventory Item is any object that the inventory can hold. Each item has its own listing in the Item Catalog with a number of settings associated with it.

Items can represent just about anything you can think of. The Player Inventory system comes prepackaged with a number of items for you to test with. You should however create your own models and item listings in the Item Catalog. Read more on creating items in the Item Catalog section below.

**Moving Items**

Items can be picked up and moved between slots on any panel by clicking on the item in the slot and then clicking on the destination slot. If the destination slot currently has an item in it then that item is swapped with the item the player is currently holding. Provided the held item is compatible with the slot. For more on moving items see the Slot Interactions section below.

**Dropping Items**

There are a three of ways a player can drop an item. These are listed below;

* Pick up an item from any slot and LEFT click out side any panel to drop the item or stack.
* Pick up a stack of items from any slot and RIGHT click out side any panel to drop one item from the stack.
* Select a slot on the Item Bar and press the drop key. Q by default.

*Note: No matter which method is used the Player Inventory System will spawn a prefab to represent the item in the game world. There are three world prefab values that can be set on the time in the Item Catalog. See the* ***Inventory Items*** *section above for details on these*

**Picking Up Items**

Providing the prefab for the item to be picked up has a *DroppedItem* component and the player object has the *InventoryPlayerController* component, The player simply walks over the item to pick it up. The *InventoryPlayerController* will try and place the item in the Item Bar first. If there is no space there it will try the players inventory. If both the item bar and inventory are full the item will not be picked up.

**Placing Items**

Although placing Items/Blocks in to your game world is beyond the scope of The Player Inventory System, the *InventoryPlayerController* provides the *PlaceItemInWorld* method for you to modify to call your games method for doing so. The *PlaceItemInWorld* method will only be passed the item bar’s selected item. Only items with a valid world prefab will be passed to the *PlaceItemInWorld* method.

*PlaceItemInWorld* is called when a player right clicks a game object within 5 (default) units that is not a chest. If the player has an item selected on the item bar and that item has a world prefab set in the item catalog, the prefab will be placed in the world a the point clicked.

*Note: If you require more functionality then this then you should modify the* PlaceItemInWorld *method to meet your requirements.*

**Giving Items**

The Player can pick up items that are dropped by a NPC or harvested from your game world. There are two methods that you could employ to move these items to the players inventory;

1. Spawn a ‘dropped’ item prefab for the player to walk over to collect. Use the *SpawnDroppedItem* method on the *InventoryController* component.
2. Place the item directly in to the inventory using the *GivePlayerItem* method on the *InventoryController component.*

**Equipping Items**

To equip an item the player can either drag an item to the appropriate slot on the Character Panel or Shift click the item to automatically move it to the correct slot.

At this point due to it being impossible to predict how you have developed your character models, it is up to you to generate any necessary game objects to represent equipped items in the game world. The same goes for held items selected on the item bar.

**Item Catalog**

To simplify creation of your game items a script-able object is provided to hold all item details. The Item Catalog is located by default in the root folder of the Player Inventory System. Selecting the Item Catalog in the Unity Editor will display the full list of prepackaged items in the inspector.

You can either edit the provided item catalog’s entries create you own from scratch. As well as being a good way to keep all your items is order, the item catalogs allow you to set up different sets of items for different parts of your game if that what you require.

To create a new Item Catalog, in the editor right click *InventorySystem* folder in the project view and select *Create > SO\_ItemList*. Then rename the list to whatever you want. You must then drag this new list on the to the Item Catalog property of the *InventoryController* component.

Once the item catalog is in place you can start adding your items to the list.

*NOTE: The first entry in any item catalog list MUST be named NULL and have an ID of 0 (zero). This is a place holder and represents the blank slots on a crafting recipe. Failing to set this correctly will prevent crafting from working correctly.*

Each item listing provides settings that represent the item in the game. The list below details what each item is and what its used for;

* **Name**: The name of the item. This will be used in game to refer to this item. This should be unique.
* **ID**: This is the item id and must be unique. This is used in code to reference this item. It is also used in creating crafting recipes.
* **Description**: A short description of the item.
* **Sprite**: The sprite that will be used to display the item in slots.
* **World Prefab Single**: This is the prefab that will represent a single instance of the item in your game world.
* **World Prefab Multiple**: The is the prefab the will represent a stack of the item in your game world. If the Item is not stack-able (stack count = 1) then you can ignore this.
* **World Prefab**: The prefab that will represent the item when equipped on the player or placed in the world.
* **Item Type**: This defines what type of item this is. Item Types and how they perform are listed below.
* **Slot Type**: This defines what slots this item can be placed in. Inventory allows the Item to be placed in any slot matched this slot type or that is an Inventory slot type.
* **Max Stack Size**: This defines how many of this item can be stacked in a single slot. Set this > 1 to make the item stack-able.
* **Recipe**: The crafting recipe for this item. See Crafting section below for more information.
* **Craft Count:** The number of items that the item’s recipe will create. See Crafting section below for more information.

The following (buffs) are used mainly for items that the player can equip or use as a tool. If the item is not meant to be equipped or is not a tool then you can ignore these. For more information on buffs see the character panel section below.

* **Max Durability**: The max durability if this item.
* **Damage**: The damage increase/decrease this item adds to the players stats.
* **Speed**: The speed increase/decrease this item adds to the players stats.
* **Health**: The health increase/decrease this item adds to the players stats.
* **Stamina**: The stamina increase/decrease this item adds to the players stats.
* **Mana**: The mana increase/decrease this item adds to the players stats.
* **Armor**: The armor increase/decrease this item adds to the players stats.
* **Intelligence**: The intelligence increase/decrease this item adds to the players stats.
* **Dexterity**: The dexterity increase/decrease this item adds to the players stats.
* **Luck**: The luck increase/decrease this item adds to the players stats.

**Panels**

There are five panels provided with the Player Inventory System. These are;

* Inventory Panel <E or Tab>
* Character Panel <C>
* Crafting Panel <V>
* Chest Panel <Mouse 1>
* Item Bar

With the exception of the Item Bar these panels can be toggled by the player using the keys noted in the list above. By default the Item Bar can not be toggled. Though a method to do so is provided in the *InventoryController* class. The Chest Panel can only be displayed when the player moves a chest to the center his/her view and clicks the right mouse button.

Panel toggle keys can be set via the *InventoryController* component on the Inventory System object.

Opening any panel will disable player movement and display the cursor.

**ESC** will close all panels that are open, hide the cursor and enable player movement.

**Inventory Panel**

The Inventory Panel is the primary storage for the player. Any item may be placed in to the inventory. By default the players inventory has 24 slots. The number of slots can be changed via the *PlayerInventoryCapacity* property on the *InventoryController*. The panel will resize automatically when the size changes.

****

*Note: If you make the players inventory smaller while all slots are full, the system will repack the inventory. Any items left over will be dropped.*

**Item Bar**

The item bar represents the items the player has at hand at any given time. The selected slot is highlighter in yellow by default. You can change the highlight color on the slot prefab. To select an item either roll the mouse or press the corresponding number key 1 though 0.

****

**Character Panel**

The Character Panel provides a way for players to equip items. There are slots provided to equipping items to the head, body, arms/hands, legs and feet. Only items set with the correct slot type can be equipped on a character slot.



Blow the Character Slots there is the stats panel. This is a sub panel because it relies on the slot contents to display the current stats for the player.

Each item has a number of settings to represent the buff or stat increase the item will give to the player when equipped. These are listed above in the Item Catalog section. Buffs provide you with a way to enhance the players abilities through equipping items.

Equipped items provide the player with an increase on one or more of the starts indicated on the character panel. You can set the value of these stats in the item catalog.

The *InventoryPlayerController* component provides access to the stats shown on the character panel via the *BuffValues* property. You can use these to modify your players character stats outside of the inventory system. The BuffValues property is ready only, however you can set the ‘base’ values on *InventoryController.Instance.CharacterPanel*. See the class documentation online for more information on this.

You can access the character stats via *BuffValues* parameter on the *InventoryPlayerController* component. You can set base or starting values for your character via the base values on the *InventoryController.Instance.CharacterPanel*. For more details on accessing these values please consult the online documentation.

**Crafting Panel**

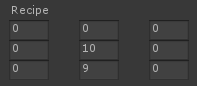
The Crafting Panel has 9 input slots and one output slot. The player can place items in to the input slots to create new items. For more information on crafting and how to create recipes please read the crafting section below.

****

**Crafting**

Each item in the Player Inventory System can be assigned a crafting recipe via the Item Catalog, that allows the player to craft them in game using other items. The nine boxes there represent the nine slots on the crafting panel.

To define a recipe enter the item ID of the ingredients in to the required box as shown in the image below.

This image shows the recipe for a torch.

In game the player can place the ingredients in to the crafting slots in any location. As long as the pattern the ingredients create matches the recipe defined. The images below show some possible arrangements of the ingredients to make a torch.



This method of crafting potentially allows for an infinite number of items to be crafted. Provided you set each item in the item catalog and define a unique recipe for it.

Items that are generated by a crafting recipe will be shown in the output slot on the crafting panel. Although the player can stack as many items as he/she wishes in the crafting Panel, the output slit will only show the number of items that a recipe will create as if all stacks contain only one item. For example the images above show the two stacks contain 12 items but the output slot will only display 4. As that is what would be generated if each item only had one in its stack.

Repeatedly clicking the output slot will load the created items in to the players hand. Allowing the player to then drop or place them wherever he/she wants.

Holding down shift and clicking the output slot will cause the crafting panel to generate as many items as the stacks will allow or the total number that the free space in both the players inventory and item bar will allow. Which ever is least.

Below the recipe in the item’s catalog entry, there is the Craft Count property. Set this to the number of items the recipe will create.

**Slots**

Each panel contains a number of slots that are capable of holding one item. Items can be stacked in a slot if they are the same. The number of items that can be stacked is defined in the Item Catalog. See the Inventory Items section above for more details.

By default the inventory is set up with 24 slots, the item bar and crafting panel with 10 and the character panel with 6. Chest panels differ depending on the number of slots assigned when the chest is created.

Slots display the sprite (as defined in the Item Catalog) of any item it contains. As well as the number of that item stacked there. Items that are single (or can not be stacked) will not display this number. If the item has a *MaxDurability* value greater than zero, the slot will display a bar to indicate the items current durability value.

A slot can only hold items that have a matching slot type set in the Item Catalog. A head slot can only hold items with slot type head. Most slots have the default slot type setting of inventory with the exception of those on the character panel and the crafting panel’s output slot.

Items can only be placed in slots that are marked as *SLOTTYPE.Inventory* or that match the items slot type in the Item Catalog. For instance a stack of sticks can not be placed on the head slot of the character panel as that slot has a slot type of head and the sticks have a slot type of inventory.

**Slot Interactions**

A slots reaction to interaction may differ depending on which panel it is on. Below are the details of what a slot will do by default.

* Left click any slot to pick up the item or stack of items it holds.
* Left click an empty slot to place the held item(s) in to the slot.
* If you left click on a slot that contains the same item as you are holding, the item(s) you are holding will be combined with those already on the slot. Any excess will remain in hand. However, If the stack being clicked is already full, the two will be swapped.
* Right click a stack of items to pick up half of the stack.
* Right click a slot to place one of the held item(s) in to the slot.

<Shift> Click will move the contents of the clicked slot depending on the following. *Note that 'space' in the following includes that left in other stacks of the same item. Items will fill these stacks first.*

* If the slot <Shift> clicked is on the Item Bar, Inventory Panel or Chest panel and the slot contains a wearable item (clothing/weapon) and the character panel is open then the item will be moved on to the character (swapping with exiting item if there is one)
* If the slot <Shift> clicked is on the Character panel and the slot contains an item and the Inventory panel is open the item will be moved to the inventory if there is space. Else it will be placed on the item bar if there is space. If there is no space on the item bar or inventory nothing will happen.
* If the slot <Shift> clicked is on the item bar and is NOT a wearable item the item will be moved to the inventory panel if there is space. If there is no space in the inventory nothing will happen.
* If the slot <Shift> clicked is on the inventory panel and is NOT a wearable item it will be moved to the item bar if there is space. If there is no space on the item bar nothing will happen.
* If the slot <Shift> clicked is on the chest panel and the inventory panel is open then items are move to the inventory panel if there is space. Else they are sent to the item bar if space there. If neither have space nothing will happen.
* If the slot <Shift> clicked is on the crafting panel, items will be placed on the item bar if there is space else in the inventory if there is space. If neither have space nothing will happen.

If the player moves the mouse over a slot, the slot will be highlighted. The color of the highlight will vary depending on the following;

* If the slot is empty and no item is being held: Yellow
* If the slot contains an item and no item is being held: Yellow
* If an item is being held and the held item can be placed in the slot: Green
* If an item is being held but the held item can NOT be placed in the slot: Red

The colors noted above are the default and can be changed on the Slot Prefab located in the Prefabs/Slots folder.

Hover the mouse over a slot and press 1 through 0 to move the item to the corresponding slot on the item bar. If the item bar slot already contains an item the two will swap places so long as the items are valid for the appropriate slot. If not this function will do nothing.

**Chests**

The Player Inventory System provides chests for players to store items. Chests can have any number of slots you wish. The slot count is set on the *ChestController* component of the items world prefab.

A chests *WorldPrefab* requires the *ChestController* component. If you do not add it to your prefab the Inventory system will add it. This is fine, but the chest will have the default capacity of 24 slots.

If you wish to set a size different from the default you must add the *ChestController* component to your chest item *WorldPrefab* object.

To remove a chest from your game world call the *DestroyChest* method on the *ChestController* component. This will ‘eject’ all the items from the chest, create an item from the chest for the player to pick up and remove the chest from the inventory system’s chest list.

*Note: If you simply remove the chest game object, the chest and whatever items it holds will remain in the inventory system but be unreachable by the player. To fix this you must recreate the chest game object and assign the correct chestID to its chest controller component.*

*Or call InventoryController.Instance.*ChestInventories*.Remove(chestID); Though in doing this you will lose all items in the chest.*

**Saving and Loading Inventory Data**

On first run the inventory data file is created and saved to the users computer/device.

By default inventory data is saved automatically when the Inventory System game object is destroyed or the application is closed. The data is then reloaded when the game starts.

What is saved?

* Player inventory size and contents
* Item bar contents
* Character panel contents and base values.
* Panel Locations
* Panel Slot Contents
* Chest locations, rotations, sizes and contents
* Dropped and spawned item locations.
* Placed Item locations and rotations.

You can load and save the inventory data whenever you choose by calling the static methods *Load()* and *Save()* respectively.

The save location depends on the value of the *UsePersistandDataPath* option on the *InventoryController* component;

* True will save it at: *Application.persistentDataPath + "/Data/data.dat"*
* False will save it at: pathToYourProject + "/Data/data.dat"

You should set this value to true only when you are ready to deploy your game as the value of *Application.persistentDataPath* is based on the platform you build for and other Unity Player settings being set prior to building.

There is also the *LoadInventory* option that when set false will cause the inventory not to be loaded. Saving still takes place effectively overwriting previous data. This will reset all panels as well.

As a security method and to prevent players from cheating, save data is encrypted. See comments in the code on how this works and how to set your own keys.

There are many more options and methods not covered in this introduction. Please review the online documentation for full details of all classes and methods.

**[Full documentation can be found on line here.](https://fideltfgsoftware.000webhostapp.com/projects/inventory/index.php)**

Thank you for purchasing and using my assets.

If you have any questions please email then to

**[fideltfg@gmail.com](mailto:fideltfg@gmail.com)**