Table of Contents

- Inventory Asset Documentation
  - Inventory System
    - Data
      - Inventory
      - InventoryDictionary
      - InventoryItem
      - InventoryType
      - SortByEnum
    - Editor
      - InventoryWindow
    - Manager
      - GameManager
      - GameManager.LoadEvent<T>
      - GameManager.Restock
      - GameManager.SaveEvent<T>
      - InputManager
      - InventorySystemManager
      - InventorySystemManager.UIChanged
      - MouseManager
      - MouseManager.KeyPressed
      - MouseManager.ScreenClick
      - UIControllerManager
      - UIControllerManager.ControllerUpdate
      - UIModelManager
      - UIViewManager
    - Script
      - Characters

      - Enemy
      - Enemy.DeathEvent
      - Player
      - Player2D
      - Player3D
    - Static
      - GenerateData
    - UI
      - InventoryInteraction
      - LootInteraction
      - StorageInteraction
      - TraderInteraction
    - World
      - ConsumableHotbarItem
      - DroppedItem
      - GenerateInventory
      - Hotbar
      - HotbarItem
      - WeaponHotbarItem
      - WorldItem
      - WorldItem.UseItem
- Item System
  - Data
    - ItemData
    - ItemType
    - WeaponState
  - Editor
    - CreateItem
    - CreateItem.GiveItem
    - EditItem

    - ItemIDScrubber
  - Script.Interface
    - IPickUp
- Utility System
  - Extension
    - FloatExtension
    - IntExtension
    - TransformExtension
  - Tool
    - CustomLogger
    - Timer
  - UI
    - Icon
    - Icon.UxmlFactory
    - Slot
    - Slot.UxmlFactory
    - SpriteElement
    - SpriteElement.UxmlFactory

Namespace InventorySystem

Data

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Editor

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Manager

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Script

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Namespace Data

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Classes

Inventory

This Class is a Scriptable Object that stores the values from an inventory.

InventoryDictionary

This Class is located in the model class and is the brain of that operates and stores the inventory at runtime.

InventoryItem

This Class is the Item for the Inventory.

Enums

InventoryType

This Enum is used to determine how an Inventory should be Interacted with. To use this enum in UI Builder InventorySystem.Data.InventoryType, Assembly-CSharp

SortByEnum

This Enum is used to determine how an Inventory will be sorted. To use this enum in UI Builder InventorySystem.Data.SortByEnum, Assembly-CSharp

Class Inventory

This Class is a Scriptable Object that stores the values from an inventory.

Inheritance

System.Object
Inventory

Namespace: InventorySystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class Inventory : ScriptableObject
```

Fields

inventoryGold

This Variables is the amount of gold that is held.

Declaration

```
public int inventoryGold
```

Field Value

TYPE	DESCRIPTION
System.Int32	

inventoryID

This Variables is the Unique id for the inventory.

Declaration

```
public string inventoryID
```

Field Value

TYPE	DESCRIPTION
System.String	

inventoryName

This Variables is the name that is used in te UI title.

Declaration

```
public string inventoryName
```

Field Value

TYPE	DESCRIPTION
System.String	

inventorySize

This Variables is the size of the inventory.

Declaration

```
public int inventorySize
```

Field Value

TYPE	DESCRIPTION
System.Int32	

inventoryType

This Variables is the type of inventory this is.

Declaration

```
public InventoryType inventoryType
```

Field Value

TYPE	DESCRIPTION
InventoryType	

Properties

InventoryStorage

This property is for the inventoryStorage Variables when it is set it will Sort the list.

Declaration

```
public List<string> InventoryStorage { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List<System.String>	

Methods

AddMoney(Int32)

This Method increases the inventoryGold variable with the param

Declaration

```
public void AddMoney(int amount)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	amount	amount to be added to the inventory.

SaveStorage(List<String>)

This Method receives a param list of strings The Property InventoryStorage = the param list

Declaration

```
public void SaveStorage(List<string> storage)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<System.String>	storage	The Property InventoryStorage = This param

SetData(Inventory)

This Method Populates the Inventory File with data from another Inventory File

Declaration

```
public void SetData(Inventory inventory)
```

Parameters

TYPE	NAME	DESCRIPTION
Inventory	inventory	This param Inventory will add it's contents to this Inventory

SubMoney()

This Method sets the variable inventoryGold to 0;

Declaration

```
public void SubMoney()
```

SubMoney(Int32)

This Method decreases the inventoryGold variable with the param

Declaration

```
public void SubMoney(int amount)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	amount	amount to be subtracted to the inventory.

Class InventoryDictionary

This Class is located in the model class and is the brain of that operates and stores the inventory at runtime.

Inheritance

System.Object
InventoryDictionary

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: InventorySystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
public class InventoryDictionary
```

Fields

dictionary

This Variable is the Dictionary and it hold the int Keys & InventoryItem Values.

Declaration

```
public Dictionary<int, InventoryItem> dictionary
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.Dictionary<System.Int32, InventoryItem>	

Methods

AddItem(out List<Int32>, InventoryItem, Int32, Int32, Boolean)

This Method will try and add an item to the Dictionary.

Declaration

```
public bool AddItem(out List<int> keys, InventoryItem item, int size, int index = -1, bool isMerge = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<System.Int32>	keys	out the list of int keys that were changed.
InventoryItem	item	the item to be added to the Dictionary.
System.Int32	size	size determines how big the inventory is so not add beyond the set size of the inventory.
System.Int32	index	if index isn't -1 it will try adding the item in that dictionary key entry. else try the first available key entry.
System.Boolean	isMerge	if isMerge will check if the item will merge with any other of the item located in the Dictionary.

Returns

TYPE	DESCRIPTION
System.Boolean	return bool if item is added or merged.

ConsolidateInventory(Int32)

This Method will Consolidate the items in the Inventory to the smallest number of stacks. return true when completed.

Declaration

```
public void ConsolidateInventory(int size)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	size	This Integer is the size of the inventory.

ConvertDictionaryToStringList()

This Method will take all entries from the Dictionary and convert them to a string list. Format: "Key":"Item ID":"Item Current Stack". return list of sting for each of the entries in Dictionary.

Declaration

```
public List<string> ConvertDictionaryToStringList()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List<System.String>	The return is a list of strings from the dictionary.

GenerateDictionary(List<String>)

This Method will Generate the dictionary from a list of strings.

Declaration

```
public void GenerateDictionary(List<string> itemData)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<System.String>	itemData	This is a List of Strings.

GetValue(String, Int32)

This Method will create a new InventoryItem from a string ID & string Count.

Declaration

```
public InventoryItem GetValue(string id, int count = 1)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	id	the Item ID.
System.Int32	count	the amount if this item.

Returns

TYPE	DESCRIPTION
InventoryItem	return InventoryItem that is created.

MergeIntoInventory(InventoryItem, out List<Int32>)

This Method will merge the item param into the same items located in the Dictionary's values.

Declaration

```
public bool MergeIntoInventory(InventoryItem item, out List<int> keys)
```

Parameters

TYPE	NAME	DESCRIPTION
InventoryItem	item	the item to be Merged into the Dictionary's entries.
System.Collections.Generic.List<System.Int32>	keys	out the list of int keys that were changed.

Returns

TYPE	DESCRIPTION
System.Boolean	Return a bool if any items were affected.

RemoveItem(Int32)

This Method will remove the the Item from dictionary key entry.

Declaration

```
public void RemoveItem(int key)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	the key for the dictionary.

ReorderDictionary()

This Method will reorder the Dictionary after it has been sorted. and Change it's keys.

Declaration

```
public void ReorderDictionary()
```

SortByAmount(Boolean)

This Method will sort the Dictionary by the amount of items it has. if isDescending is true it will sort by descending order. else by ascending order.

Declaration

```
public void SortByAmount(bool isDescending = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	isDescending	This is a Boolean that Determine whether to sort ascending or descending.

SortByCost(Boolean)

This Method will sort the Dictionary by Item Cost. if isDescending is true it will sort by descending order. else by ascending order.

Declaration

```
public void SortByCost(bool isDescending = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	isDescending	This is a Boolean that Determine whether to sort ascending or descending.

SortByName(Boolean)

This Method will sort the Dictionary by Item Name. if isDescending is true it will sort by descending order. else by ascending order.

Declaration

```
public void SortByName(bool isDescending = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	isDescending	This is a Boolean that Determine whether to sort ascending or descending.

SortByType(Boolean)

This Method will sort the Dictionary by Item Type. if isDescending is true it will sort by descending order. else by ascending order.

Declaration

```
public void SortByType(bool isDescending = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	isDescending	This is a Boolean that Determine whether to sort ascending or descending.

SortByWeight(Boolean)

This Method will sort the Dictionary by Item Weight. if isDescending is true it will sort by descending order. else by ascending order.

Declaration

```
public void SortByWeight(bool isDescending = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	isDescending	This is a Boolean that Determine whether to sort ascending or descending.

TryRemoveItem(Int32, out InventoryItem)

This Method will try and remove an item from the dictionary key entry.

Declaration

```
public bool TryRemoveItem(int key, out InventoryItem value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	the key for the dictionary.
InventoryItem	value	an out param of type InventoryItem.

Returns

TYPE	DESCRIPTION
System.Boolean	return true if the entry is remove or false if not.

Class InventoryItem

This Class is the Item for the Inventory.

Inheritance

System.Object
InventoryItem

Namespace: InventorySystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
public class InventoryItem : ScriptableObject
```

Fields

canStack

This Variables determines if the item can be stacked.

Declaration

```
public bool canStack
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

itemCost

This Variables This cost of the item;

Declaration

```
public int itemCost
```

Field Value

TYPE	DESCRIPTION
System.Int32	

itemCurrentStack

This Variables how many items are in this stack.

Declaration

```
public int itemCurrentStack
```

Field Value

TYPE	DESCRIPTION
System.Int32	

itemDescription

This Variables The description of the item.

Declaration

```
public string itemDescription
```

Field Value

TYPE	DESCRIPTION
System.String	

itemID

This Variables the ItemData's id.

Declaration

```
public string itemID
```

Field Value

TYPE	DESCRIPTION
System.String	

itemImage

This Variables the image used by the UI Icon.

Declaration

```
public Sprite itemImage
```

Field Value

TYPE	DESCRIPTION
Sprite	

itemMaxStack

This Variables determines how may items can be placed into a stack.

Declaration

```
public int itemMaxStack
```

Field Value

TYPE	DESCRIPTION
System.Int32	

itemName

This Variables the name of the item.

Declaration

```
public string itemName
```

Field Value

TYPE	DESCRIPTION
System.String	

itemType

This Variables The ItemData's Type.

Declaration

```
public ItemType itemType
```

Field Value

TYPE	DESCRIPTION
ItemType	

itemWeight

This Variables how Heavy the item is.

Declaration

```
public float itemWeight
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

SetData(InventoryItem)

This Method Populates the InventoryItem File with data from another InventoryItem File

Declaration

```
public void SetData(InventoryItem itemInventory)
```

Parameters

TYPE	NAME	DESCRIPTION
InventoryItem	itemInventory	

SetData(ItemData, Int32)

This Method Populates the InventoryItem File with data from an ItemData File.

Declaration

```
public void SetData(ItemData itemData, int itemCurrentStack = 1)
```

Parameters

TYPE	NAME	DESCRIPTION
ItemData	itemData	
System.Int32	itemCurrentStack	

Enum InventoryType

This Enum is used to determine how an Inventory should be Interacted with. To use this enum in UI Builder InventorySystem.Data.InventoryType, Assembly-CSharp

Namespace: InventorySystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
public enum InventoryType
```

Fields

NAME	DESCRIPTION
Hotbar	
Loot	
None	
Player	
Storage	
Trader	

Enum SortByEnum

This Enum is used to determine how an Inventory will be sorted. To use this enum in UI Builder InventorySystem.Data.SortByEnum, Assembly-CSharp

Namespace: InventorySystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
public enum SortByEnum
```

Fields

NAME	DESCRIPTION
None	
SortByAmount	
SortByCost	
SortByName	
SortByType	
SortByWeight	

Namespace Editor

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Classes

InventoryWindow

This Class Is an Editor Window for the creation & editing of th inventory and it's content.

Class InventoryWindow

This Class Is an Editor Window for the creation & editing of th inventory and it's content.

Inheritance

System.Object
InventoryWindow

Namespace: InventorySystem.Editor

Assembly: cs.temp.dll.dll

Syntax

```
public class InventoryWindow : EditorWindow
```

Properties

SelectedInventory

This Property is connected to the selectedInventory Variable. When set if selectedInventory is different from the incoming value then set the value & refresh the UI.

Declaration

```
public Inventory SelectedInventory { get; }
```

Property Value

TYPE	DESCRIPTION
Inventory	

Methods

AddItemData(ItemData)

This Method will add a new ItemData File to items list Then Add that ItemData to the UI.

Declaration

```
public void AddItemData(ItemData item)
```

Parameters

TYPE	NAME	DESCRIPTION
ItemData	item	

AddToDropdown(String)

This Method will add the Inventory file to the dropdown field.

Declaration

```
public void AddToDropdown(string text)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	text	

ChangeItemData(ItemData)

This Method gets called from an event when the OpenItemUpdate clicks the update or Duplacate button

Declaration

```
public void ChangeItemData(ItemData newItem)
```

Parameters

TYPE	NAME	DESCRIPTION
ItemData	newItem	

CloseWindow()

This Static Method will Close the InventoryWindow. The window can be open either by double clicking an Inventory file, Though the UnityEditor Toolbar: Tools/DownUnder Studios/Inventory System/Tools/Close Inventory Editor Or by the shortcut key CTRl + SHIFT + ALT + O

Declaration

```
public static void CloseWindow()
```

CreateGUI()

This Method is a unity method that will be run when the window is created.

Declaration

```
public void CreateGUI()
```

GenerateItemDataFiles()

This Method will get the ItemData file from the Resources folder and add them to the items list. Then Create UI for each ItemData.

Declaration

```
public void GenerateItemDataFiles()
```

GetInventoryFiles()

This Method will get all Inventory files in the Resources.

Declaration

```
public void GetInventoryFiles()
```

NewInventory(String)

This Method will create a new Inventory File And add it to the *inventories* List & Dropdown Field

Declaration

```
public void NewInventory(string name)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	name	

OnOpenAsset(Int32, Int32)

This Static Method will open the InventoryWindow when an Inventory file is double clicked.

Declaration

```
public static bool OnOpenAsset(int instanceID, int line)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	instanceID	not used.
System.Int32	line	not used.

Returns

TYPE	DESCRIPTION
System.Boolean	not used.

OpenCreateItem()

This Method will open the CreateItem Window to create a new ItemData.

Declaration

```
public void OpenCreateItem()
```

OpenEditItem(ItemData)

This Method will open the EditItem Window to edit the selected ItemData.

Declaration

```
public void OpenEditItem(ItemData item)
```

Parameters

TYPE	NAME	DESCRIPTION
ItemData	item	

OpenWindow()

This Static Method will open the InventoryWindow. The window can be open either by double clicking an Inventory file, Though the UnityEditor Toolbar: Tools/DownUnder Studios/Inventory System/Tools/Open Inventory Editor Or by the shortcut key CTRl + SHIFT + ALT + I

Declaration

```
public static void OpenWindow()
```

RefreshUI()

This Method will Update the UI when their are changes made.

Declaration

```
public void RefreshUI()
```

Save()

This Method will Save the inventory's changes. Only if the editor is not in play mode.

Declaration

```
public void Save()
```

SetWindowSize(Int32, Int32)

This Method will set the Editor Window min & max size to a fix size. With both params being the width & height respectively.

Declaration

```
public void SetWindowSize(int width, int height)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	width	The width of the Window.
System.Int32	height	The height of the Window.

UpdateUI()

This Method is a unity method, it will check if SelectedInventory is null or not. If not null set the displays to flex and window size to 1200-500 else set the displays to none and window size to 570-30

Declaration

```
public void UpdateUI()
```

Namespace Manager

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Classes

GameManager

This Class is the Game Mamager and it manages the Save, Load & Restock Events

InputManager

This Class

InventorySystemManager

This Class is the Manager for the Inventory System.

MouseManager

This Class is the Mouse Manager for the Inventory System.

UIControllerManager

This Class is the Controller of the Inventory System's MVC.

UIModelManager

This Class is the model for the Inventory System's MVC.

UIViewManager

This Class is the View of the Inventory System's MVC.

Delegates

GameManager.LoadEvent<T>

This Variable is the delegate for the Loading of inventories.

GameManager.Restock

This Variable is the delegate for the trader restock.

GameManager.SaveEvent<T>

This Variable is the delegate for the saving of inventories.

InventorySystemManager.UIChanged

This Variable is the delegate for updating UIs when changed.

MouseManager.KeyPressed

This Variable is a delegate for the KeyDownEvent.

MouseManager.ScreenClick

This Variable is a delegate for the MouseDownEvent.

UIControllerManager.ControllerUpdate

This Variable is the delegate for the Controller Update.

Class GameManager

This Class is the Game Mamager and it manages the Save, Load & Restock Events

Inheritance

System.Object
GameManager

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public class GameManager : MonoBehaviour
```

Fields

instance

This Variable is the Singleton Instance of this Class.

Declaration

```
public static GameManager instance
```

Field Value

TYPE	DESCRIPTION
GameManager	

is3D

This Variable will be used to select 3D or 2D

Declaration

```
public bool is3D
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

isLogger

This Variable will be used to allow the Logger.

Declaration

```
public bool isLogger
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

player

This Variable is the player gameobject.

Declaration

```
public GameObject player
```

Field Value

TYPE	DESCRIPTION
GameObject	

restocktimer

This Variable will be the Timer that will activate the onRestock event.

Declaration

```
public Timer restocktimer
```

Field Value

TYPE	DESCRIPTION
Timer	

restockTimer

This Variable is the set time between trader inventories restocking.

Declaration

```
public float restockTimer
```

Field Value

TYPE	DESCRIPTION
System.Single	

Methods

Awake()

This Method is a unity method, and will set the instance, find the player and set the Timer for the restock.

Declaration

```
public void Awake()
```

LoadInventories(List<Inventory>)

This Method will send the inventories to the subscriber so they can find their inventory. Call this Method when loading

Declaration

```
public void LoadInventories(List<Inventory> inventories)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<Inventory>	inventories	

SaveInventories()

This Method will save the Inventories that have subscribed to onSaveInventory to a list. Call this Method when saving

Declaration

```
public List<Inventory> SaveInventories()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List<Inventory>	Return a list of all the inventories that have subscribed to onSaveInventory

Events

onLoadInventory

This Variable is called when loading all inventories

Declaration

```
public static event GameManager.LoadEvent<Inventory> onLoadInventory
```

Event Type

TYPE	DESCRIPTION
GameManager.LoadEvent<Inventory>	

onRestock

This Variable is the event for the Traders to subscribe to for restock.

Declaration

```
public static event GameManager.Restock onRestock
```

Event Type

TYPE	DESCRIPTION
GameManager.Restock	

onSaveInventory

This Variable is called when saving all inventories

Declaration

```
public static event GameManager.SaveEvent<Inventory> onSaveInventory
```

Event Type

TYPE	DESCRIPTION
GameManager.SaveEvent<Inventory>	

Delegate GameManager.LoadEvent<T>

This Variable is the delegate for the Loading of inventories.

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void LoadEvent<T>(List<T> inventory);
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<T>	inventory	

Type Parameters

NAME	DESCRIPTION
T	

Delegate GameManager.Restock

This Variable is the delegate for the trader restock.

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void Restock();
```

Delegate GameManager.SaveEvent<T>

This Variable is the delegate for the saving of inventories.

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public delegate T SaveEvent<T>();
```

Returns

TYPE	DESCRIPTION
T	

Type Parameters

NAME	DESCRIPTION
T	

Class InputManager

This Class

Inheritance

System.Object
InputManager

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public class InputManager : MonoBehaviour
```

Fields

otherController

This Variable is the Other's Controller.

Declaration

```
public UIControllerManager otherController
```

Field Value

TYPE	DESCRIPTION
UIControllerManager	

playerController

This Variable is the player's Controller.

Declaration

```
public UIControllerManager playerController
```

Field Value

TYPE	DESCRIPTION
UIControllerManager	

Class InventorySystemManager

This Class is the Manager for the Inventory System.

Inheritance

System.Object
InventorySystemManager

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public class InventorySystemManager : MonoBehaviour
```

Fields

instance

This Variable is the Singleton Instance of this class.

Declaration

```
public static InventorySystemManager instance
```

Field Value

TYPE	DESCRIPTION
InventorySystemManager	

Methods

CloseInventory()

This Method close both other & player inventories if they are open.

Declaration

```
public void CloseInventory()
```

GetController(InventoryType, out UIControllerManager)

This Method will find an The controller based on the type.

Declaration

```
public bool GetController(InventoryType type, out UIControllerManager Controller)
```

Parameters

TYPE	NAME	DESCRIPTION
InventoryType	type	the inventory type
UIControllerManager	Controller	out UIControllerManager if found.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if the controller isn't null.

OpenInventory(UIModelManager, UIViewManager)

This Method will add the model & view to the other's inventory. And open both other & player inventories.

Declaration

```
public void OpenInventory(UIModelManager model = null, UIViewManager view = null)
```

Parameters

TYPE	NAME	DESCRIPTION
UIModelManager	model	the model that will be applied to the Other Controller.
UIViewManager	view	the view that will be applied to the Other Controller.

TryGetHotbarController(out UIControllerManager)

This Method will try to get the Hotbar Controller

Declaration

```
public bool TryGetHotbarController(out UIControllerManager hotbarController)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
UIControllerManager	hotbarController	out UIControllerManager if found.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if the hotbar controller is found.

TryGetMouseManager(out MouseManager)

This Method will try to get the Mouse Manager

Declaration

```
public bool TryGetMouseManager(out MouseManager mouseManager)
```

Parameters

TYPE	NAME	DESCRIPTION
MouseManager	mouseManager	out MouseManager if found.

Returns

TYPE	DESCRIPTION
System.Boolean	

TryGetOtherController(out UIControllerManager)

This Method will try to get the Other Controller

Declaration

```
public bool TryGetOtherController(out UIControllerManager otherController)
```

Parameters

TYPE	NAME	DESCRIPTION
UIControllerManager	otherController	out UIControllerManager if found.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if the other controller is found.

TryGetPlayerController(out UIControllerManager)

This Method will try to get the Player Controller

Declaration

```
public bool TryGetPlayerController(out UIControllerManager playerController)
```

Parameters

TYPE	NAME	DESCRIPTION
UIControllerManager	playerController	out UIControllerManager if found.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if the player controller is found.

UpdateUI()

This Method when run will run the onUIChanged event.

Declaration

```
public void UpdateUI()
```

Events

onUIChanged

This Variable is called when UIs are changed.

Declaration

```
public static event InventorySystemManager.UIChanged onUIChanged
```

Event Type

TYPE	DESCRIPTION
InventorySystemManager.UIChanged	

Delegate InventorySystemManager.UIChanged

This Variable is the delegate for updating UIs when changed.

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void UIChanged();
```

Class MouseManager

This Class is the Mouse Manager for the Inventory System.

Inheritance

System.Object
MouseManager

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public class MouseManager : MonoBehaviour
```

Fields

icon

This Variable is the Icon that is from an inventory picked.

Declaration

```
public Icon icon
```

Field Value

TYPE	DESCRIPTION
Icon	

item

This Variable is the InventoryItem that is from an inventory picked.

Declaration

```
public InventoryItem item
```

Field Value

TYPE	DESCRIPTION
InventoryItem	

key

This Variable is the Interger Key that is from an inventory picked.

Declaration

```
public int key
```

Field Value

TYPE	DESCRIPTION
System.Int32	

pickupParent

This Variable is the VisualElement Parent of the pickups.

Declaration

```
public VisualElement pickupParent
```

Field Value

TYPE	DESCRIPTION
VisualElement	

Properties

MouseController

This Method will find an The controller based on the type.

Declaration

```
public UIControllerManager MouseController { get; }
```

Property Value

TYPE	DESCRIPTION
UIControllerManager	

Methods

AddItem(Icon, Int32, InventoryType, InventoryItem)

This Method will pick up the Item Icon from the inventory.

Declaration

```
public bool AddItem(Icon icon, int key, InventoryType type, InventoryItem item)
```

Parameters

TYPE	NAME	DESCRIPTION
Icon	icon	the Icon that will be picked up and attached to the mouse.
System.Int32	key	the key of the slot that will be used to return the item back to the inventory.
InventoryType	type	the type of inventory that will be used to get the MouseController.
InventoryItem	item	the Item that will be picked out of the inventory.

Returns

TYPE	DESCRIPTION
System.Boolean	

AddItemDataToPlayerInventory(InventoryItem)

This Method will add an item to the player or hotbar inventory.

Declaration

```
public bool AddItemDataToPlayerInventory(InventoryItem item = null)
```

Parameters

TYPE	NAME	DESCRIPTION
InventoryItem	item	This param will be the item added to the inventory. if null then create a random item.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if the item was added to inventory.

CloseMouse()

This Method will return the Item to it's Inventory. DetailScreen will have it's display turned off. and RemoveItem Method.

Declaration

```
public void CloseMouse()
```

OpenItemDetailScreen(Slot, InventoryItem)

This Method will display up the DetailScreen for the item in the inventory.

Declaration

```
public void OpenItemDetailScreen(Slot slot, InventoryItem item)
```

Parameters

TYPE	NAME	DESCRIPTION
Slot	slot	This param will be used to set the location of the Detail screen spawn point.
InventoryItem	item	This param will provide the data to be displayed.

RemoveItem()

This Method will remove the Item Icon from the mouse.

Declaration

```
public void RemoveItem()
```

UpdateCount()

This Method will update the Icon attached to the mouse.

Declaration

```
public bool UpdateCount()
```

Returns

TYPE	DESCRIPTION
System.Boolean	

Events

onKeyPressed

This Variable is the KeyPressed event is to identify when any key is pressed.

Declaration

```
public event MouseManager.KeyPressed onKeyPressed
```

Event Type

TYPE	DESCRIPTION
MouseManager.KeyPressed	

onScreenClick

This Variable is the ScreenClick event is to identify when the mouse interacts with the screen.

Declaration

```
public event MouseManager.ScreenClick onScreenClick
```

Event Type

TYPE	DESCRIPTION
MouseManager.ScreenClick	

Delegate MouseManager.KeyPressed

This Variable is a delegate for the KeyDownEvent.

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void KeyPressed(KeyDownEvent e);
```

Parameters

TYPE	NAME	DESCRIPTION
KeyDownEvent	e	

Delegate MouseManager.ScreenClick

This Variable is a delegate for the MouseDownEvent.

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void ScreenClick(MouseDownEvent e);
```

Parameters

TYPE	NAME	DESCRIPTION
MouseDownEvent	e	

Class UIControllerManager

This Class is the Controller of the Inventory System's MVC.

Inheritance

System.Object
UIControllerManager

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public class UIControllerManager : MonoBehaviour
```

Properties

GetElementNames

This Property will get the list of element names from model.

Declaration

```
public List<string> GetElementNames { get; }
```

Property Value

TYPE	DESCRIPTION
System.Collections.Generic.List<System.String>	

GetInventoryGold

This Property will get the Gold Amount from the Inventory.

Declaration

```
public int GetInventoryGold { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

GetInventoryName

This Property will get the Name of the Inventory.

Declaration

```
public string GetInventoryName { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

GetInventorySize

This Property will get the Size of the Inventory.

Declaration

```
public int GetInventorySize { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

GetInventoryType

This Property will get the Type of Inventory.

Declaration

```
public InventoryType GetInventoryType { get; }
```

Property Value

TYPE	DESCRIPTION
InventoryType	

HasModel

This Property will check is if a model is attached.

Declaration

```
public bool HasModel { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

HotbarController

This property is for the hotbar Controller.

Declaration

```
public UIControllerManager HotbarController { get; }
```

Property Value

TYPE	DESCRIPTION
UIControllerManager	

IsOpen

This Property will check if the UI is open.

Declaration

```
public bool IsOpen { get; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

OtherController

This property is for the other Controller.

Declaration

```
public UIControllerManager OtherController { get; }
```

Property Value

TYPE	DESCRIPTION
UIControllerManager	

PlayerController

This property is for the player Controller.

Declaration

```
public UIControllerManager PlayerController { get; }
```

Property Value

TYPE	DESCRIPTION
UIControllerManager	

Methods

AddInventoryItem(out List<Int32>, InventoryItem, Int32, Boolean)

This Method will add the InventoryItem in the dictionary.

Declaration

```
public bool AddInventoryItem(out List<int> keys, InventoryItem item, int index = -1, bool isMerge = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<System.Int32>	keys	out the list of int keys that were changed.
InventoryItem	item	the item to be added to the Dictionary.
System.Int32	index	if index isn't -1 it will try adding the item in that dictionary key entry. else try the first available key entry.
System.Boolean	isMerge	if isMerge will check if the item will merge with any other of the item located in the Dictionary.

Returns

TYPE	DESCRIPTION
System.Boolean	return bool if item is added or merged in to the Dictionary.

ClearSlot(Int32)

This Method allow for the clearing of an icon form a slot based on it's key.

Declaration

```
public void ClearSlot(int key)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	The Slot's at key.

Close()

This Method will Close Inventory UI and save. if Other Controller is the Loot type and is empty or can Destroy then run Destroy method from model. else RefreshUI() Method. if Other Controller set model & view to null.

Declaration

```
public void Close()
```

FindInventoryItem(Int32)

This Method will find the InventoryItem in the dictionary's key.

Declaration

```
public InventoryItem FindInventoryItem(int key)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	The Slot's at key.

Returns

TYPE	DESCRIPTION
InventoryItem	Return the InventoryItem

LootAll()

This Method will loot all of the items from Other's Inventory to the Player's Inventory.

Declaration

```
public void LootAll()
```

LootGold()

This Method will transfer the gold from Other's Inventory to the Player's Inventory.

Declaration

```
public void LootGold()
```

Open(UIModelManager, UIViewManager)

This Method will Open Inventory UI and save. It will run ApplyConnections. then check if IsOpen & model.canOpen. if true then Genreate the Dictionary in model. And run Display in View ot true.

Declaration

```
public void Open(UIModelManager newModel = null, UIViewManager newView = null)
```

Parameters

TYPE	NAME	DESCRIPTION
UIModelManager	newModel	the New UIModelManager being added (can be null).
UIViewManager	newView	the New UIViewManager being added (can be null).

RefreshUI()

This Method will be called after each interaction. if hasChanged Property is true then run onControllerUpdate event. And set hasChanged to false.

Declaration

```
public void RefreshUI()
```

RemoveInventoryItem(Int32)

This Method will remove the InventoryItem in the dictionary's key.

Declaration

```
public void RemoveInventoryItem(int key)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	The Slot's at key.

ReturnItem()

This Method will Return the item from the mouse to the inventory.

Declaration

```
public void ReturnItem()
```

SaveInventory()

This Method will up date the InventoryStorage from the Dictionary.

Declaration

```
public void SaveInventory()
```

SellMisc()

This Method will Sell all of the Misc type items from Player's Inventory to the Trader's Inventory.

Declaration

```
public void SellMisc()
```

SetDisplayState(Boolean)

This Method will update the IsOpen Property

Declaration

```
public void SetDisplayState(bool state)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	state	

SlotInteraction(Slot, MouseDownEvent)

This Methid is called when the palyer interacts with a Slot.

Declaration

```
public void SlotInteraction(Slot slot, MouseDownEvent e)
```

Parameters

TYPE	NAME	DESCRIPTION
Slot	slot	This is the Slot that was clicked.
MouseDownEvent	e	This Param is a MouseDownEvent event and is need to determine if left or right click from mouse.

Sortby(SortByEnum)

This Method will Sort the inventory data then update the UI's Slots.

Declaration

```
public void Sortby(SortByEnum sortBy)
```

Parameters

TYPE	NAME	DESCRIPTION
SortByEnum	sortBy	

TryFindInventoryItem(Int32, out InventoryItem)

This Method will try to find the InventoryItem in the dictionary's key.

Declaration

```
public bool TryFindInventoryItem(int key, out InventoryItem slotItem)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	The Slot's at key.
InventoryItem	slotItem	out the InventoryItem that is removed.

Returns

TYPE	DESCRIPTION
System.Boolean	Return if InventoryItem is located.

UpdateListSlots(List<Int32>)

This Method will update a list of slots.

Declaration

```
public void UpdateListSlots(List<int> keys)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<System.Int32>	keys	

UpdateSlot(Int32, InventoryItem)

This Method will update a slot.

Declaration

```
public void UpdateSlot(int key, InventoryItem item)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	The Slot's at key.
InventoryItem	item	the item that is going to update the slot.

UpdateUI()

This Method will update elements from the View.

Declaration

```
public void UpdateUI()
```

Events

onControllerUpdate

This variable is the event that will be run whenever there is a change made.

Declaration

```
public event UIControllerManager.ControllerUpdate onControllerUpdate
```

Event Type

TYPE	DESCRIPTION
UIControllerManager.ControllerUpdate	

Delegate UIControllerManager.ControllerUpdate

This Variable is the delegate for the Controller Update.

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void ControllerUpdate();
```

Class UIModelManager

This Class is the model for the Inventory System's MVC.

Inheritance

System.Object
UIModelManager

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public class UIModelManager : MonoBehaviour
```

Fields

CanDestroy

This Variable if true then Destroy Method can be run if isOpen Variable is false.

Declaration

```
public bool CanDestroy
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

canOpen

This Variable if true then the UI can open.

Declaration

```
public bool canOpen
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

elementNames

This Variable is a list of strings that are the names of the UI elements that will be accessed in the View.

Declaration

```
public List<string> elementNames
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<System.String>	

inventory

This Variable is the Inventory that is used in the UI.

Declaration

```
public Inventory inventory
```

Field Value

TYPE	DESCRIPTION
Inventory	

inventoryDictionary

This Variable is the Dictionary for the UI.

Declaration

```
public InventoryDictionary inventoryDictionary
```

Field Value

TYPE	DESCRIPTION
InventoryDictionary	

isOpen

This Variable if true then the UI is open.

Declaration

```
public bool isOpen
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

AddInventory(Inventory)

This Method will add an Inventory to this model.

Declaration

```
public void AddInventory(Inventory inventory)
```

Parameters

TYPE	NAME	DESCRIPTION
Inventory	inventory	the Inventory that will be added to this model.

Destroy()

This Method will Destroy parent gameobject.

Declaration

```
public void Destroy()
```

GenerateDictionary()

This Method will Generate the Dictionary for the Model.

Declaration

```
public void GenerateDictionary()
```

InventorySpace(out Int32, InventoryItem)

This Method will check for space in the inventory.

Declaration

```
public bool InventorySpace(out int space, InventoryItem item)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	space	out will return at the end the amount of the Item param can be added to the Dictionary.
InventoryItem	item	the InventoryItem that will be checked against to find the amount of items that can be added to the Dictionary.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if there is space in the inventory for the item.

SortBy(SortByEnum, Boolean)

This Method will sort the dictionary by SortByEnum.

Declaration

```
public bool SortBy(SortByEnum sortBy, bool isDescending = false)
```

Parameters

TYPE	NAME	DESCRIPTION
SortByEnum	sortBy	This param that is what will sorted by.
System.Boolean	isDescending	if the sorting is in ascending or descending.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true so the view can update the slots.

Class UIViewManager

This Class is the View of the Inventory System's MVC.

Inheritance

System.Object
UIViewManager

Namespace: InventorySystem.Manager

Assembly: cs.temp.dll.dll

Syntax

```
public class UIViewManager : MonoBehaviour
```

Fields

currentSortOrder

This Variable is the current Sorting order.

Declaration

```
public SortByEnum currentSortOrder
```

Field Value

TYPE	DESCRIPTION
SortByEnum	

isDescending

This Variable is used to determine the sorting.

Declaration

```
public bool isDescending
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Methods

Display(Boolean, UIControllerManager)

This Method is to set the View on and off. if State is true then update the View. else close down.

Declaration

```
public void Display(bool State, UIControllerManager controller)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Boolean	State	This param will turn the view on or off.
UIControllerManager	controller	the Controller that is connected.

GetSlotFromKey(Int32)

This Method will get the Slot from a key.

Declaration

```
public Slot GetSlotFromKey(int key)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	the key of the slot & Dictionary.

Returns

TYPE	DESCRIPTION
Slot	

TryGetSlotFromKey(Int32, out Slot)

This Method will try ands get the Slot from a key.

Declaration

```
public bool TryGetSlotFromKey(int key, out Slot slot)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	the key of the slot & Dictionary.
Slot	slot	out return This param at the end.

Returns

TYPE	DESCRIPTION
System.Boolean	return true if the Slot was found.

UpdateAllSlots(UIControllerManager)

This Method will Update all slots.

Declaration

```
public void UpdateAllSlots(UIControllerManager controller)
```

Parameters

TYPE	NAME	DESCRIPTION
UIControllerManager	controller	the Controller that is connected.

UpdateElement(UIControllerManager)

This Method will update elements from the UIList with data from the Controller.

Declaration

```
public virtual void UpdateElement(UIControllerManager controller)
```

Parameters

TYPE	NAME	DESCRIPTION
UIControllerManager	controller	the Controller that is connected.

UpdateListSlots(UIControllerManager, List<Int32>)

This Method will update all the slots from a lost of keys.

Declaration

```
public void UpdateListSlots(UIControllerManager controller, List<int> keys)
```

Parameters

TYPE	NAME	DESCRIPTION
UIControllerManager	controller	the Controller that is connected.
System.Collections.Generic.List<System.Int32>	keys	This param are the keys of the slots & Dictionary.

UpdateSlot(Int32, InventoryItem)

This Method will remove the Slot from the slotList at the key param with the InventoryItem param.

Declaration

```
public void UpdateSlot(int key, InventoryItem item)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	key	the key of the slot & Dictionary.
InventoryItem	item	the item that will be used.

Namespace Script

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Characters

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Static

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

UI

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Script

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Namespace Characters

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Classes

Enemy

This Class is the Enemy Script

Player

This Abstract Class is the parent script thats applied to a player Character

Player2D

This Class is the script applied to a 2D player Character

Player3D

This Class is the script applied to a 3D player Character

Delegates

Enemy.DeathEvent

This Variable is a delegate for when an enemy is killed.

Class Enemy

This Class is the Enemy Script

Inheritance

System.Object
Enemy

Namespace: InventorySystem.Script.Characters

Assembly: cs.temp.dll.dll

Syntax

```
public class Enemy : MonoBehaviour
```

Fields

health

This Variable is the heath of the enemy.

Declaration

```
public int health
```

Field Value

TYPE	DESCRIPTION
System.Int32	

isDead

This Variable if true will stop Death Method from being run.

Declaration

```
public bool isDead
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

Properties

Health

This Property gets health & sets health, if health == 0 & isDead == false then set isDead to true & run Death Method.

Declaration

```
public int Health { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Events

onDeath

This Variable is the DeathEvent that can be subscribe too. When run will run all method subscribed to it.

Declaration

```
public event Enemy.DeathEvent onDeath
```

Event Type

TYPE	DESCRIPTION
Enemy.DeathEvent	

Delegate Enemy.DeathEvent

This Variable is a delegate for when an enemy is killed.

Namespace: InventorySystem.Script.Characters

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void DeathEvent();
```

Class Player

This Abstract Class is the parent script thats applied to a player Character

Inheritance

System.Object
Player
Player2D
Player3D

Namespace: InventorySystem.Script.Characters

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class Player : MonoBehaviour
```

Fields

canMove

This Variable if true will allow movement.

Declaration

```
public bool canMove
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

moveSpeed

This Variable is for the movement speed of the player.

Declaration

```
public float moveSpeed
```

Field Value

TYPE	DESCRIPTION
System.Single	

Class Player2D

This Class is the script applied to a 2D player Character

Inheritance

System.Object
Player
Player2D

Inherited Members

Player.moveSpeed
Player.canMove

Namespace: InventorySystem.Script.Characters
Assembly: cs.temp.dll.dll

Syntax

```
public class Player2D : Player
```

Fields

animator

This Variable is to allow animation of the player.

Declaration

```
public Animator animator
```

Field Value

TYPE	DESCRIPTION
Animator	

rb

This Variable is for the RidgidBody2D.

Declaration

```
public Rigidbody2D rb
```

Field Value

TYPE	DESCRIPTION
Rigidbody2D	

Class Player3D

This Class is the script applied to a 3D player Character

Inheritance

System.Object
Player
Player3D

Inherited Members

Player.moveSpeed
Player.canMove

Namespace: InventorySystem.Script.Characters
Assembly: cs.temp.dll.dll

Syntax

```
public class Player3D : Player
```

Fields

controller

This Variable is the Character Controller.

Declaration

```
public CharacterController controller
```

Field Value

TYPE	DESCRIPTION
CharacterController	

playerHand

This Variable is the gameobject for the hand. When item is equipped it will be apply to the hand.

Declaration

```
public GameObject playerHand
```

Field Value

TYPE	DESCRIPTION
GameObject	

Namespace Static

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Classes

GenerateData

This Static Class is used to Generate Inventories & InventoryItems from ItemData files.

Class GenerateData

This Static Class is used to Generate Inventories & InventoryItems from ItemData files.

Inheritance

System.Object
GenerateData

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: InventorySystem.Script.Static

Assembly: cs.temp.dll.dll

Syntax

```
public static class GenerateData
```

Methods

CreateInventory(out Inventory, Int32, List<ItemData>, Int32, Boolean)

This Method will create an Inventory add items to an inventory's storageData.

Declaration

```
public static void CreateInventory(out Inventory inventory, int slotLine, List<ItemData> itemDataList, int
inventoryGold, bool isRandom = false)
```

Parameters

TYPE	NAME	DESCRIPTION
Inventory	inventory	out Inventory file after it is created.
System.Int32	slotLine	how many slot will the UI have per row. so the inventory's can be determined along with number of items to be added.
System.Collections.Generic.List<ItemData>	itemDataList	list of ItemDat files that will be added to the inventory. between(1 - Max Stack).

TYPE	NAME	DESCRIPTION
System.Int32	inventoryGold	the amount of gold the inventory will have or will be the max random range if isRandom is true.
System.Boolean	isRandom	if true each item will be given a random key. else 1 by 1.

CreateInventoryItem(String, Int32)

This Method will create an InventoryItem and add an amount to it.

Declaration

```
public static InventoryItem CreateInventoryItem(string targetItemID, int amount = 0)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	targetItemID	the Item ID.
System.Int32	amount	the amount to be added to the InventoryItem.

Returns

TYPE	DESCRIPTION
InventoryItem	Return the InventoryItem that was created.

CreateRandomInventoryItem()

This Method will create a random InventoryItem and add a random amount to it.

Declaration

```
public static InventoryItem CreateRandomInventoryItem()
```

Returns

TYPE	DESCRIPTION
InventoryItem	Return the InventoryItem that was created.

FindItemData(String)

This Method will to located the ItemData by ID.

Declaration

```
public static ItemData FindItemData(string targetItemID)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	targetItemID	

Returns

TYPE	DESCRIPTION
ItemData	Return ItemData file after it is found.

GetItems()

This Method will get all ItemData from the Resources folder and add them to the items list.

Declaration

```
public static void GetItems()
```

TryFindItemData(String, out ItemData)

This Method will try to located the ItemData by ID.

Declaration

```
public static bool TryFindItemData(string targetItemID, out ItemData itemData)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	targetItemID	the ID of the ItemData to be found.
ItemData	itemData	out ItemData file after it is found.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if ItemData file is found.

Namespace UI

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Classes

InventoryInteraction

This abstract Class is the parent of the Loot, Storage, Trader. Interaction classes

LootInteraction

This Class will be attached to Enemy gameobject that contains an Enemy script.

StorageInteraction

This Class is added to any gameobject to be used as a storage.

TraderInteraction

This Class will be attached to a gameobject that will be the Trader.

Class InventoryInteraction

This abstract Class is the parent of the Loot, Storage, Trader. Interaction classes

Inheritance

System.Object
InventoryInteraction
LootInteraction
StorageInteraction
TraderInteraction

Namespace: InventorySystem.Script.UI

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class InventoryInteraction : MonoBehaviour
```

Fields

generateInventory

This Variable is used for initialising the timer & inventory. GenerateInventory class is used to open and close Inventory UI for Other Controller Don't set as this is should be located on the Prefab.

Declaration

```
protected GenerateInventory generateInventory
```

Field Value

TYPE	DESCRIPTION
GenerateInventory	

gold

This Variable is used as a max range for how much gold will be added to a new inventory. No gold for Hotbar or Storage types : unless you change the storage type to hold gold.

Declaration

```
protected int gold
```

Field Value

TYPE	DESCRIPTION
System.Int32	

inventory

This Variable is Optional. If their is no Inventory file then it will create one with the itemIDs list. If itemIDs is empty then the inventory will be empty. If Inventory file is created then it will be temporary. suitable for loot.

Declaration

```
protected Inventory inventory
```

Field Value

TYPE	DESCRIPTION
Inventory	

inventoryID

This Variable is the Id that will be used when you load your game

Declaration

```
protected static string inventoryID
```

Field Value

TYPE	DESCRIPTION
System.String	

inventoryName

This Variable is the name that will be apply to the inventory on creation

Declaration

```
protected string inventoryName
```

Field Value

TYPE	DESCRIPTION
System.String	

isRandom

This Variable is used to randomise the which slots and amount of gold

Declaration

```
protected bool isRandom
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

itemIDs

This Variable list contains ItemData files that will be used when generating a new Inventory

Declaration

```
protected List<ItemData> itemIDs
```

Field Value

TYPE	DESCRIPTION
System.Collections.Generic.List<ItemData>	

prefab2D

This Variable is the prefab for 2D game. Apply a prefab from inspector.

Declaration

```
protected GameObject prefab2D
```

Field Value

TYPE	DESCRIPTION
GameObject	

prefab3D

This Variable is the prefab for 3D game. Apply a prefab from inspector.

Declaration

```
protected GameObject prefab3D
```

Field Value

TYPE	DESCRIPTION
GameObject	

slotLine

This Variable is used to determine how many slot an inventory has per line. The Inventory-Screen.uxml supports 8 slots in each row.

Declaration

```
protected int slotLine
```

Field Value

TYPE	DESCRIPTION
System.Int32	

time

This Variable is the time for the timer used for loot decay timer. Set this Value to -1 when you don't want the Loot types to decay(Destroy)

Declaration

```
protected int time
```

Field Value

TYPE	DESCRIPTION
System.Int32	

timer

This Variable is the timer used for Loot and determines when to decay(Destroy)

Declaration

```
protected Timer timer
```

Field Value

TYPE	DESCRIPTION
Timer	

Properties

NewID

This property Give a new ID for Inventory.

Declaration

```
public string NewID { get; }
```

Property Value

TYPE	DESCRIPTION
System.String	

Methods

InitInventory()

This Method will create a child GameObject from the lop Prefab and Initialise the scripts attached.

Declaration

```
public void InitInventory()
```

Load(List<Inventory>)

This Method load your Inventory.

Declaration

```
public virtual void Load(List<Inventory> inventories)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Collections.Generic.List<Inventory>	inventories	

OnDisable()

This Method is the Unity OnDisable.

Declaration

```
public virtual void OnDisable()
```

OnEnable()

This Method is the Unity OnEnable.

Declaration

```
public virtual void OnEnable()
```

Save()

This Method Save your Inventory.

Declaration

```
public virtual Inventory Save()
```

Returns

TYPE	DESCRIPTION
Inventory	

Start()

This Method is the Unity Start.

Declaration

```
public virtual void Start()
```

Class LootInteraction

This Class will be attached to Enemy gameobject that contains an Enemy script.

Inheritance

System.Object
InventoryInteraction
LootInteraction

Inherited Members

InventoryInteraction.inventoryName
InventoryInteraction.inventoryID
InventoryInteraction.itemIDs
InventoryInteraction.inventory
InventoryInteraction.prefab2D
InventoryInteraction.prefab3D
InventoryInteraction.slotLine
InventoryInteraction.gold
InventoryInteraction.generateInventory
InventoryInteraction.isRandom
InventoryInteraction.timer
InventoryInteraction.time
InventoryInteraction.NewID
InventoryInteraction.InitInventory()
InventoryInteraction.Save()
InventoryInteraction.Load(List<Inventory>)

Namespace: InventorySystem.Script.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class LootInteraction : InventoryInteraction
```

Methods

OnDisable()

This Method is a unity method, and will remove InitInventory Method to the onDeath event in the Enemy Script.

Declaration

```
public override void OnDisable()
```

Overrides

InventoryInteraction.OnDisable()

OnEnable()

This Method is a unity method, and will remove InitInventory Method to the onDeath event in the Enemy Script.

Declaration

```
public override void OnEnable()
```

Overrides

InventoryInteraction.OnEnable()

Start()

This Method is a unity method, and will setup the Inventory & get Enemy Script attached to this gameobject.

Declaration

```
public override void Start()
```

Overrides

InventoryInteraction.Start()

Class StorageInteraction

This Class is added to any gameobject to be used as a storage.

Inheritance

System.Object
InventoryInteraction
StorageInteraction

Inherited Members

InventoryInteraction.inventoryName
InventoryInteraction.inventoryID
InventoryInteraction.itemIDs
InventoryInteraction.inventory
InventoryInteraction.prefab2D
InventoryInteraction.prefab3D
InventoryInteraction.slotLine
InventoryInteraction.gold
InventoryInteraction.generateInventory
InventoryInteraction.isRandom
InventoryInteraction.timer
InventoryInteraction.time
InventoryInteraction.NewID
InventoryInteraction.OnEnable()
InventoryInteraction.OnDisable()
InventoryInteraction.InitInventory()
InventoryInteraction.Save()
InventoryInteraction.Load(List<Inventory>)

Namespace: InventorySystem.Script.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class StorageInteraction : InventoryInteraction
```

Methods

Start()

This Method is a unity method, and will setup the Inventory & InitInventory Method.

Declaration

```
public override void Start()
```

Overrides

InventoryInteraction.Start()

Class TraderInteraction

This Class will be attached to a gameobject that will be the Trader.

Inheritance

System.Object
InventoryInteraction
TraderInteraction

Inherited Members

InventoryInteraction.inventoryName
InventoryInteraction.inventoryID
InventoryInteraction.itemIDs
InventoryInteraction.inventory
InventoryInteraction.prefab2D
InventoryInteraction.prefab3D
InventoryInteraction.slotLine
InventoryInteraction.gold
InventoryInteraction.generateInventory
InventoryInteraction.isRandom
InventoryInteraction.timer
InventoryInteraction.time
InventoryInteraction.NewID
InventoryInteraction.InitInventory()
InventoryInteraction.Save()
InventoryInteraction.Load(List<Inventory>)

Namespace: InventorySystem.Script.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class TraderInteraction : InventoryInteraction
```

Methods

OnDisable()

This Method is a unity method, and will remove StartCoroutine for Restock from generateInventory Variable to the onRestock event in the GameManager.

Declaration

```
public override void OnDisable()
```

Overrides

InventoryInteraction.OnDisable()

OnEnable()

This Method is a unity method, and will add StartCoroutine for Restock from generateInventory Variable to the onRestock event in the GameManager.

Declaration

```
public override void OnEnable()
```

Overrides

InventoryInteraction.OnEnable()

Start()

This Method is a unity method, and will setup the Inventory & InitInventory Method.

Declaration

```
public override void Start()
```

Overrides

InventoryInteraction.Start()

Namespace World

This is a Namespace in InventorySystem Namespace, Assembly-CSharp

Classes

ConsumableHotbarItem

This Class is attached to a gameobject prefab and when the mouse is click will consume 1 of the item.

DroppedItem

This Class is a sub class of WorldItem and is apply to a prefab for any item in the world that can be picked up.

GenerateInventory

This Class Is located on the prefabs that will be Instantiated by the InventoryInteraction classes.

Hotbar

This Class is attached to the hotbar UI and will be the way the player interacts with the hotbar beyond the Controller.

HotbarItem

This Class

WeaponHotbarItem

This Class is a sub class of HotbarItem and Implaments IWeapon and is attached to any weapon.

WorldItem

This abstract Class is the root of the WorldItem scripts that are attached to any Item that will inhabit your game.

Delegates

WorldItem.UseItem

This delegate is to be used by any subclass that needs an event.

Class ConsumableHotbarItem

This Class is attached to a gameobject prefab and when the mouse is click will consume 1 of the item.

Inheritance

System.Object
WorldItem
HotbarItem
ConsumableHotbarItem

Implements

IUse
IConsume

Inherited Members

HotbarItem.canUse
HotbarItem.onUseItem
HotbarItem.Use()
WorldItem.animator
WorldItem.item
WorldItem.Destroy()

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public class ConsumableHotbarItem : HotbarItem
```

Properties

Amount

This Property get the amount Variable. and sets amount to value;

Declaration

```
public int Amount { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

Methods

Consume()

This Method

Declaration

```
public void Consume()
```

OnDisable()

This Method is a unity method, and will remove Consume Method to the onUseItem event.

Declaration

```
public void OnDisable()
```

OnEnable()

This Method is a unity method, and will add Consume Method to the onUseItem event.

Declaration

```
public void OnEnable()
```

Implements

IUse
IConsume

Class DroppedItem

This Class is a sub class of WorldItem and is apply to a prefab for any item in the world that can be picked up.

Inheritance

System.Object
WorldItem
DroppedItem

Implements

IPickUp

Inherited Members

WorldItem.animator
WorldItem.item
WorldItem.Destroy()

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public class DroppedItem : WorldItem
```

Properties

canPickUp

This variable if true will all this item to be picked up.

Declaration

```
public bool canPickUp { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

Pickup()

This Method will add this item to the players inventory.

Declaration

```
public void Pickup()
```

Implements

IPickUp

Class GenerateInventory

This Class Is located on the prefabs that will be Instantiated by the InventoryInteraction classes.

Inheritance

System.Object
GenerateInventory

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public class GenerateInventory : MonoBehaviour
```

Methods

CloseInventory()

This Method will Close both Player & Other Inventories.

Declaration

```
public void CloseInventory()
```

Decay()

This Method will run Destroy in the Model Class or set CanDestroy to true;

Declaration

```
public void Decay()
```

Init(Inventory)

This Method will initialise the inventory and data with the model.

Declaration

```
public void Init(Inventory inventory)
```

Parameters

TYPE	NAME	DESCRIPTION
Inventory	inventory	

OpenInventory()

This Method will Open both Player & Other Inventories. And add the Model & View to Other's Controller.

Declaration

```
public void OpenInventory()
```

Restock()

This Method is a Coroutine that will restock the attached Inventory.

Declaration

```
public IEnumerator Restock()
```

Returns

TYPE	DESCRIPTION
System.Collections.Generic.IEnumerator	

Class Hotbar

This Class is attached to the hotbar UI and will be the way the player interacts with the hotbar beyond the Controller.

Inheritance

System.Object
Hotbar

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public class Hotbar : MonoBehaviour
```

Properties

CurrentSlot

This property will get currentSelect Variable & if currentSelect isn't value then set currentSelect to value and run SelectItem Method.

Declaration

```
public int CurrentSlot { get; }
```

Property Value

TYPE	DESCRIPTION
System.Int32	

HotbarItem

This property will get hotbarItem Variable & if hotbarItem isn't null then run Destroy method then check if different then set hotbarItem to value.

Declaration

```
public GameObject HotbarItem { get; }
```

Property Value

TYPE	DESCRIPTION
GameObject	

SelectedItem

This property will get selectedItem Variable & if selectedItem isn't value then set selectedItem to value and run EquipItem Method.

Declaration

```
public InventoryItem SelectedItem { get; }
```

Property Value

TYPE	DESCRIPTION
InventoryItem	

Methods

FixedUpdate()

This Method is a unity method, and is used to set the CurrentSlot property from the mouse scroll.

Declaration

```
public void FixedUpdate()
```

Start()

This Method is a unity method, and on start it will try and get mouse. If it does then add ClickMouseButton Method to the mouse onScreenClick event. And add SlotNumber Method to the mouse onKeyPressed event. then run SelectItem Method.

Declaration

```
public void Start()
```

Class HotbarItem

This Class

Inheritance

System.Object
WorldItem
HotbarItem
ConsumableHotbarItem
WeaponHotbarItem

Implements

IUse

Inherited Members

WorldItem.animator
WorldItem.item
WorldItem.Destroy()

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public class HotbarItem : WorldItem
```

Properties

canUse

This Variable if true will allow onUseItem to be run.

Declaration

```
public bool canUse { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

Use()

This Method will run the onUseItem event.

Declaration

```
public void Use()
```

Events

onUseItem

This Variable is the event that can be subscribed to. When run it will run any methods that have subscribed.

Declaration

```
public event WorldItem.UseItem onUseItem
```

Event Type

TYPE	DESCRIPTION
WorldItem.UseItem	

Implements

IUse

Class WeaponHotbarItem

This Class is a sub class of HotbarItem and Implaments IWeapon and is attached to any weapon.

Inheritance

System.Object
WorldItem
HotbarItem
WeaponHotbarItem

Implements

IUse
IWeapon

Inherited Members

HotbarItem.canUse
HotbarItem.onUseItem
HotbarItem.Use()
WorldItem.animator
WorldItem.item
WorldItem.Destroy()

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public class WeaponHotbarItem : HotbarItem
```

Properties

canAttack

This Property if true will allow the item can Attack.

Declaration

```
public bool canAttack { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

canDefend

This Property if true will allow the item can Defend.

Declaration

```
public bool canDefend { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

canParry

This Property if true will allow the item can Parry.

Declaration

```
public bool canParry { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

State

This Property get weaponState Variable & set weaponState as value.

Declaration

```
public WeaponState State { get; set; }
```

Property Value

TYPE	DESCRIPTION
WeaponState	

Methods

Attack()

This Method will set the state of the weapon to Attack if the state is in idle.

Declaration

```
public void Attack()
```

Defend()

This Method will set the state of the weapon to Defend if the state is in idle.

Declaration

```
public void Defend()
```

Idle()

This Method will return the weapon back to the idle state. Have this Run in the animation event for the Attack animation.

Declaration

```
public void Idle()
```

OnDisable()

This Method is a unity method, and will remove SetWeaponState Method to the onUseItem event.

Declaration

```
public void OnDisable()
```

OnEnable()

This Method is a unity method, and will Add SetWeaponState Method to the onUseItem event.

Declaration

```
public void OnEnable()
```

Parry()

This Method will set the state of the weapon to Parry if the state is in idle.

Declaration

```
public void Parry()
```

SetWeaponState()

This Method is will choice which of the 4 states to enter.

Declaration

```
public void SetWeaponState()
```

Implements

IUse
IWeapon

Class WorldItem

This abstract Class is the root of the WorldItem scripts that are attached to any Item that will inhabit your game.

Inheritance

System.Object
WorldItem
DroppedItem
HotbarItem

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public abstract class WorldItem : MonoBehaviour
```

Fields

animator

This Variable is the Animator for any animations that you require.

Declaration

```
public Animator animator
```

Field Value

TYPE	DESCRIPTION
Animator	

item

This Variable is the ItemData that this gameobject is created from.

Declaration

```
public ItemData item
```

Field Value

TYPE	DESCRIPTION
ItemData	

Methods

Destroy()

This Method will Destroy this Gameobject.

Declaration

```
public void Destroy()
```

Delegate WorldItem.UseItem

This delegate is to be used by any subclass that needs an event.

Namespace: InventorySystem.Script.World

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void UseItem();
```

Namespace ItemSystem

Data

This is a Namespace in ItemSystem Namespace, Assembly-CSharp

Editor

This is a Namespace in ItemSystem Namespace, Assembly-CSharp

Interface

This is a Namespace in ItemSystem Namespace, Assembly-CSharp

Namespace Data

This is a Namespace in ItemSystem Namespace, Assembly-CSharp

Classes

ItemData

This Class is a ScriptableObject and is used to represent an item that can be used in the world or in the inventory.

Enums

ItemType

This Enum is used to determine the type of the Item. To use this enum in UI Builder ItemSystem.Data.ItemType, Assembly-CSharp

WeaponState

This Enum is used to determine the Current State of the Weapon is in. To use this enum in UI Builder ItemSystem.Data.WeaponState, Assembly-CSharp

Class ItemData

This Class is a ScriptableObject and is used to represent an item that can be used in the world or in the inventory.

Inheritance

System.Object
ItemData

Namespace: ItemSystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
public class ItemData : ScriptableObject
```

Fields

animator

This Variable is the animator controller that the item will use.

Declaration

```
public AnimatorOverrideController animator
```

Field Value

TYPE	DESCRIPTION
AnimatorOverrideController	

canStack

This Variable determines if th items can be stacked.

Declaration

```
public bool canStack
```

Field Value

TYPE	DESCRIPTION
System.Boolean	

itemCost

This Variable is the cost of the item.

Declaration

```
public int itemCost
```

Field Value

TYPE	DESCRIPTION
System.Int32	

itemDescription

This Variable is the description of the item.

Declaration

```
public string itemDescription
```

Field Value

TYPE	DESCRIPTION
System.String	

itemID

This Variable is a unique id for the item. This will be used to identify the InventoryItem and Items in the world.

Declaration

```
public string itemID
```

Field Value

TYPE	DESCRIPTION
System.String	

itemImage

This Variable is the Image of the item for the inventory icons.

Declaration

```
public Sprite itemImage
```

Field Value

TYPE	DESCRIPTION
Sprite	

itemMaxStack

This Variable is how many of the item and fit into an inventory stack.

Declaration

```
public int itemMaxStack
```

Field Value

TYPE	DESCRIPTION
System.Int32	

itemName

This Variable is the name of the item.

Declaration

```
public string itemName
```

Field Value

TYPE	DESCRIPTION
System.String	

itemType

This Variable the type of item it is.

Declaration

```
public ItemType itemType
```

Field Value

TYPE	DESCRIPTION
ItemType	

itemWeight

This Variable is the weight of the item.

Declaration

```
public float itemWeight
```

Field Value

TYPE	DESCRIPTION
System.Single	

prefabWorldItem

This Variable is a prefab of the item for 3D games.

Declaration

```
public GameObject prefabWorldItem
```

Field Value

TYPE	DESCRIPTION
GameObject	

prefabWorldItem2D

This Variable is a prefab of the item for 2D games.

Declaration

```
public GameObject prefabWorldItem2D
```

Field Value

TYPE	DESCRIPTION
GameObject	

Methods

CreateWorldObject(Transform)

This Method will Instantiate the prefabWorldItem prefab.

Declaration

```
public GameObject CreateWorldObject(Transform target)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	target	The parent of this object.

Returns

TYPE	DESCRIPTION
GameObject	The game object of the item.

CreateWorldObject2D(Transform)

This Method will Instantiate the prefabWorldItem2D prefab.

Declaration

```
public GameObject CreateWorldObject2D(Transform target)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	target	The parent of this object.

Returns

TYPE	DESCRIPTION
GameObject	The game object of the item.

Enum ItemType

This Enum is used to determine the type of the Item. To use this enum in UI Builder ItemSystem.Data.ItemType, Assembly-CSharp

Namespace: ItemSystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
public enum ItemType
```

Fields

NAME	DESCRIPTION
Armour	
Consumable	
Materials	
Misc	
Weapon	

Enum WeaponState

This Enum is used to determine the Current State of the Weapon is in. To use this enum in UI Builder ItemSystem.Data.WeaponState, Assembly-CSharp

Namespace: ItemSystem.Data

Assembly: cs.temp.dll.dll

Syntax

```
public enum WeaponState
```

Fields

NAME	DESCRIPTION
Attack	
Defend	
Idle	
Parry	

Namespace Editor

This is a Namespace in ItemSystem Namespace, Assembly-CSharp

Classes

CreateItem

This Class is an EditorWindow used to create an ItemData file.

EditItem

This Class is an EditorWindow used to edit any ItemData file.

ItemIDScrubber

This Class is an EditorWindow used to check all ItemData in the Resource folder and make sure the ID's are not conflicted.

Delegates

CreateItem.GiveItem

This Variable is a delegate that passes the new ItemData file.

Class CreateItem

This Class is an EditorWindow used to create an ItemData file.

Inheritance

System.Object
CreateItem

Namespace: ItemSystem.Editor

Assembly: cs.temp.dll.dll

Syntax

```
public class CreateItem : EditorWindow
```

Fields

createButton

This Variable is the Button that will be used to Create the ItemData file and pass in the data from the input fields.

Declaration

```
public Button createButton
```

Field Value

TYPE	DESCRIPTION
Button	

itemCanStackToggle

This Variable is to determine if the item will be able to stack.

Declaration

```
public Toggle itemCanStackToggle
```

Field Value

TYPE	DESCRIPTION
Toggle	

itemCostField

This Variable is the cost that the item will have.

Declaration

```
public IntegerField itemCostField
```

Field Value

TYPE	DESCRIPTION
IntegerField	

itemDescriptionField

This Variable is the description of the item will have.

Declaration

```
public TextField itemDescriptionField
```

Field Value

TYPE	DESCRIPTION
TextField	

itemImageField

This Variable is the image that the item will use in the inventory.

Declaration

```
public ObjectField itemImageField
```

Field Value

TYPE	DESCRIPTION
ObjectField	

itemMaxStackField

This Variable is the max stack size that the item will have.

Declaration

```
public IntegerField itemMaxStackField
```

Field Value

TYPE	DESCRIPTION
IntegerField	

itemNameField

This Variable is the name of the item will have.

Declaration

```
public TextField itemNameField
```

Field Value

TYPE	DESCRIPTION
TextField	

itemtype

This Variable is the type that the new item will be.

Declaration

```
public EnumField itemtype
```

Field Value

TYPE	DESCRIPTION
EnumField	

itemVisualSprite

This Variable is a display of the image that the item will use.

Declaration

```
public SpriteElement itemVisualSprite
```

Field Value

TYPE	DESCRIPTION
SpriteElement	

itemWeightField

This Variable is the weight that the item will have.

Declaration

```
public FloatField itemWeightField
```

Field Value

TYPE	DESCRIPTION
FloatField	

newItemData

This Variable is the new ItemData file that is created.

Declaration

```
public ItemData newItemData
```

Field Value

TYPE	DESCRIPTION
ItemData	

Methods

CreateGUI()

This Method is a unity method that will be run when the window is created.

Declaration

```
public void CreateGUI()
```

CreateItemFile()

This Method Will create the item from the inputs you have made.

Declaration

```
public void CreateItemFile()
```

GetUIElements()

This Method will get the Elements in the UI. And set RegisterValueChangedCallback if any other element's values are changed.

Declaration

```
public void GetUIElements()
```

OpenWindow()

This Static Method will open the CreateItem. Though the UnityEditor Toolbar: Tools/DownUnder Studios/Item System/Tools/Create Item Or by the shortcut key CTRl + SHIFT + ALT + C

Declaration

```
public static void OpenWindow()
```

ResetField()

This Method will reset the input field for the item.

Declaration

```
public void ResetField()
```

SetWindowSize(Int32, Int32)

This Method will set the Editor Window min & max size to a fix size. With both params being the width & height respectively.

Declaration

```
public void SetWindowSize(int width, int height)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	width	The width of the Window.
System.Int32	height	The height of the Window.

Events

OnItemChange

This Variable is the GiveItem that is subscribed to by the InventoryWindow Method AddItemData. when the InventoryWindow is open it and an ItemData file is created it will add the new ItemData file ot the AddItemData Method.

Declaration

```
public static event CreateItem.GiveItem OnItemChange
```

Event Type

TYPE	DESCRIPTION
CreateItem.GiveItem	

Delegate CreateItem.GiveItem

This Variable is a delegate that passes the new ItemData file.

Namespace: ItemSystem.Editor

Assembly: cs.temp.dll.dll

Syntax

```
public delegate void GiveItem(ItemData newItemData);
```

Parameters

TYPE	NAME	DESCRIPTION
ItemData	newItemData	

Class EditItem

This Class is an EditorWindow used to edit any ItemData file.

Inheritance

System.Object
EditItem

Namespace: ItemSystem.Editor

Assembly: cs.temp.dll.dll

Syntax

```
public class EditItem : EditorWindow
```

Fields

itemCanStackToggle

This Variable is the Toggle that will be used to determine if the item can stack.

Declaration

```
public Toggle itemCanStackToggle
```

Field Value

TYPE	DESCRIPTION
Toggle	

itemCostField

This Variable is the IntegerField that will be used to get the cost for the item.

Declaration

```
public IntegerField itemCostField
```

Field Value

TYPE	DESCRIPTION
IntegerField	

itemDescriptionField

This Variable is the TextField that will be used to get the description for the item.

Declaration

```
public TextField itemDescriptionField
```

Field Value

TYPE	DESCRIPTION
TextField	

itemImageField

This Variable is the ObjectField that will be used to get the Sprite for the item.

Declaration

```
public ObjectField itemImageField
```

Field Value

TYPE	DESCRIPTION
ObjectField	

itemMaxStackField

This Variable is the IntegerField that will be used to get the max stack size for the item.

Declaration

```
public IntegerField itemMaxStackField
```

Field Value

TYPE	DESCRIPTION
IntegerField	

itemNameField

This Variable is the TextField that will be used to get the name for the item.

Declaration

```
public TextField itemNameField
```

Field Value

TYPE	DESCRIPTION
TextField	

itemtype

This Variable is the EnumField that will be used to get the type of the item.

Declaration

```
public EnumField itemtype
```

Field Value

TYPE	DESCRIPTION
EnumField	

itemVisualSprite

This Variable is the SpriteElement that will show the item image tat will be used by the inventory.

Declaration

```
public SpriteElement itemVisualSprite
```

Field Value

TYPE	DESCRIPTION
SpriteElement	

itemWeightField

This Variable is the FloatField that will be used to get the weight of the item.

Declaration

```
public FloatField itemWeightField
```

Field Value

TYPE	DESCRIPTION
FloatField	

targetItemData

This Variable is the ItemData file that will be edited.

Declaration

```
public static ItemData targetItemData
```

Field Value

TYPE	DESCRIPTION
ItemData	

Methods

CreateGUI()

This Method is a unity method that will be run when the window is created.

Declaration

```
public void CreateGUI()
```

GetUIElements()

This Method will get the Elements in the UI. And set RegisterValueChangedCallback if any other element's values are changed.

Declaration

```
public void GetUIElements()
```

OnOpenAsset(Int32, Int32)

This Static Method will open the EditItem window when an ItemData file is double clicked.

Declaration

```
public static bool OnOpenAsset(int instanceID, int line)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	instanceID	not used.
System.Int32	line	not used.

Returns

TYPE	DESCRIPTION
System.Boolean	not used.

OpenWindow()

This Static Method will open the EditItem. The window can be open by double clicking an ItemData file.

Declaration

```
public static void OpenWindow()
```

SetItemData(ItemData)

This static Method will set the target ItemData. and open the window.

Declaration

```
public static bool SetItemData(ItemData item)
```

Parameters

TYPE	NAME	DESCRIPTION
ItemData	item	The ItemData that will be edited.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true if targetItemData is set and window open.

SetWindowSize(Int32, Int32)

This Method will set the Editor Window min & max size to a fix size. With both params being the width & height respectively.

Declaration

```
public void SetWindowSize(int width, int height)
```

Parameters

TYPE	NAME	DESCRIPTION

TYPE	NAME	DESCRIPTION
System.Int32	width	The width of the Window.
System.Int32	height	The height of the Window.

UpdateItemData()

This Method will update ItemData file with the new data from the input fields.

Declaration

```
public void UpdateItemData()
```

Class ItemIDScrubber

This Class is an EditorWindow used to check all ItemData in the Resource folder and make sure the ID's are not conflicted.

Inheritance

System.Object
ItemIDScrubber

Namespace: ItemSystem.Editor

Assembly: cs.temp.dll.dll

Syntax

```
public class ItemIDScrubber : EditorWindow
```

Fields

clearButton

This Variable is the UI Button that when pressed will clear the Log.

Declaration

```
public Button clearButton
```

Field Value

TYPE	DESCRIPTION
Button	

currentLogNumber

This Variable is the number of logs made.

Declaration

```
public int currentLogNumber
```

Field Value

TYPE	DESCRIPTION
System.Int32	

items

This Variable is an array of all ItemData in the project.

Declaration

```
public ItemData[] items
```

Field Value

TYPE	DESCRIPTION
ItemData[]	

logDataString

This Variable is the sting that hold all of the logs that will be added into a file upon saving.

Declaration

```
public string logDataString
```

Field Value

TYPE	DESCRIPTION
System.String	

LogEntryParent

This Variable is the parent where all log elements will be parented to.

Declaration

```
public VisualElement LogEntryParent
```

Field Value

TYPE	DESCRIPTION
VisualElement	

SaveButton

This Variable is the UI Button that when pressed will save the Log.

Declaration

```
public Button SaveButton
```

Field Value

TYPE	DESCRIPTION
Button	

startButton

This Variable is the UI Button that when pressed will Start the scrubbing process.

Declaration

```
public Button startButton
```

Field Value

TYPE	DESCRIPTION
Button	

Methods

Clear()

This Method will clear the Log.

Declaration

```
public void Clear()
```

CreateGUI()

This Method is a unity method that will be run when the window is created.

Declaration

```
public void CreateGUI()
```

CreateLogEntry(String)

This Method Will create a log entry from a string log.

Declaration

```
public void CreateLogEntry(string log)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	log	this is the log that is passed in.

GetAllInstances<T>()

This Generic Method is used to get all of the ItemData from the project.

Declaration

```
public static T[] GetAllInstances<T>()
    where T : ScriptableObject
```

Returns

TYPE	DESCRIPTION
T[]	return an array of T.

Type Parameters

NAME	DESCRIPTION
T	The type of ScriptableObject.

Save()

This Method will save the Log to a .txt file. (With time stamp).

Declaration

```
public void Save()
```

SetWindowSize(Int32, Int32)

This Method will set the Editor Window min & max size to a fix size. With both params being the width & height respectively.

Declaration

```
public void SetWindowSize(int width, int height)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	width	The width of the Window.
System.Int32	height	The height of the Window.

ShowExample()

This Static Method will open the CreateItem. Though the UnityEditor Toolbar: Tools/DownUnder Studios/Item System/Tools/Item ID Scrubber Or by the shortcut key CTRl + SHIFT + ALT + S

Declaration

```
public static void ShowExample()
```

Start()

This Method will cross check every ItemData file for conflicting IDs. And return a log for each ID that is changed.

Declaration

```
public void Start()
```

Namespace Interface

This is a Namespace in ItemSystem Namespace, Assembly-CSharp

Interfaces

IPickUp

This Interface should be used to pickup an item.

Interface IPickUp

This Interface should be used to pickup an item.

Namespace: ItemSystem.Script.Interface

Assembly: cs.temp.dll.dll

Syntax

```
public interface IPickUp
```

Properties

canPickUp

This Variable will determine if the Item can be picked up.

Declaration

```
bool canPickUp { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

Pickup()

This Method will be used to when that player wants to pick up the item.

Declaration

```
void Pickup()
```

Namespace UtilitySystem

Extension

This is a Namespace in UtilitySystem Namespace, Assembly-CSharp

Tool

This is a Namespace in UtilitySystem Namespace, Assembly-CSharp

UI

This is a Namespace in UtilitySystem Namespace, Assembly-CSharp

Namespace Extension

This is a Namespace in UtilitySystem Namespace, Assembly-CSharp

Classes

FloatExtension

This Static Class extends the float Variable.

IntExtension

This Static Class extends the integer Variable.

TransformExtension

This Static Class extends the Transform type.

Class FloatExtension

This Static Class extends the float Variable.

Inheritance

System.Object
FloatExtension

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: UtilitySystem.Extension

Assembly: cs.temp.dll.dll

Syntax

```
public static class FloatExtension
```

Methods

FlipValue(Single)

This Method will change an negative number to positive and positive to negative.

Declaration

```
public static float FlipValue(this float value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Single	value	the float that will be flipped.

Returns

TYPE	DESCRIPTION
System.Single	return the flipped float.

RoundDown(Single)

This Method will round down the float to an interger.

Declaration

```
public static int RoundDown(this float value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Single	value	the float that will be round down.

Returns

TYPE	DESCRIPTION
System.Int32	Return an integer of the float round down.

RoundUp(Single)

This Method will round up the float to an interger.

Declaration

```
public static int RoundUp(this float value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Single	value	the float that will be round up.

Returns

TYPE	DESCRIPTION
System.Int32	Return an integer of the float round up.

Class IntExtension

This Static Class extends the integer Variable.

Inheritance

System.Object
IntExtension

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: UtilitySystem.Extension

Assembly: cs.temp.dll.dll

Syntax

```
public static class IntExtension
```

Methods

FlipValue(Int32)

This Method will change an negative number to positive and positive to negative.

Declaration

```
public static int FlipValue(this int value)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	value	the integer that will be flipped.

Returns

TYPE	DESCRIPTION
System.Int32	return the flipped integer.

HighestValue(Int32, Int32)

This Method will take the highest of 2 integers.

Declaration

```
public static int HighestValue(this int value, int compare)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	value	The integer that will check.
System.Int32	compare	The integer that will be compare.

Returns

TYPE	DESCRIPTION
System.Int32	Return the highest integer.

LowestValue(Int32, Int32)

This Method will take the lowest of 2 integers.

Declaration

```
public static int LowestValue(this int value, int compare)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	value	The integer that will check.
System.Int32	compare	The integer that will be compare.

Returns

TYPE	DESCRIPTION
System.Int32	Return the lowest integer.

Class TransformExtension

This Static Class extends the Transform type.

Inheritance

System.Object
TransformExtension

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: UtilitySystem.Extension

Assembly: cs.temp.dll.dll

Syntax

```
public static class TransformExtension
```

Methods

GetChildren(Transform, Boolean)

This Method will get a list of the children from a gameobject.

Declaration

```
public static List<GameObject> GetChildren(this Transform transform, bool includeInactive = false)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	transform	This is the transform that will get it's children.
System.Boolean	includeInactive	If the param is true it will also includeInactive gameobject. else ignore.

Returns

TYPE	DESCRIPTION
System.Collections.Generic.List<GameObject>	return a list of GameObjects.

ResetTransformation(Transform)

This Method will reset the transform.

Declaration

```
public static void ResetTransformation(this Transform transform)
```

Parameters

TYPE	NAME	DESCRIPTION
Transform	transform	This is the transform that will get reset.

Namespace Tool

This is a Namespace in UtilitySystem Namespace, Assembly-CSharp

Classes

CustomLogger

Timer

This Class is a Timer that can be used to activate an action after an amount of time.

Class CustomLogger

Inheritance

System.Object
CustomLogger

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: UtilitySystem.Tool

Assembly: cs.temp.dll.dll

Syntax

```
public static class CustomLogger
```

Properties

logEnabled

To runtime toggle debug logging [ON/OFF].

Declaration

```
public static bool logEnabled { get; set; }
```

Property Value

TYPE	DESCRIPTION
System.Boolean	

Methods

Break()

Pauses the editor.

Declaration

```
public static void Break()
```

Error(String)

A variant of Log that logs an error message to the console.

Declaration

```
public static void Error(string message)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	String or object to be converted to string representation for display.

Error(String, UnityEngine.Object)

A variant of Log that logs an error message to the console.

Declaration

```
public static void Error(string message, UnityEngine.Object context)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	String to be converted to string representation for display.
UnityEngine.Object	context	Object to which the message applies.

Log(String)

Logs a message to the Unity Console By Type.

Declaration

```
public static void Log(string message)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	String or object to be converted to string representation for display.

Log(String, UnityEngine.Object)

Logs a message to the Unity Console.

Declaration

```
public static void Log(string message, UnityEngine.Object context)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	String or object to be converted to string representation for display.
UnityEngine.Object	context	Object to which the message applies.

LogException(Exception)

A variant of Log that logs an exception message to the console.

Declaration

```
public static void LogException(Exception exception)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Runtime Exception.

LogException(Exception, UnityEngine.Object)

A variant of Log that logs an exception message to the console.

Declaration

```
public static void LogException(Exception exception, UnityEngine.Object context)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Exception	exception	Runtime Exception.
UnityEngine.Object	context	Object to which the message applies.

Warning(String)

A variant of Log that logs a warning message to the console.

Declaration

```
public static void Warning(string message)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	String or object to be converted to string representation for display.

Warning(String, UnityEngine.Object)

A variant of Log that logs a warning message to the console.

Declaration

```
public static void Warning(string message, UnityEngine.Object context)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	message	String or object to be converted to string representation for display.
UnityEngine.Object	context	Object to which the message applies.

Class Timer

This Class is a Timer that can be used to activate an action after an amount of time.

Inheritance

System.Object
Timer

Inherited Members

System.Object.ToString()
System.Object.Equals(System.Object)
System.Object.Equals(System.Object, System.Object)
System.Object.ReferenceEquals(System.Object, System.Object)
System.Object.GetHashCode()
System.Object.GetType()
System.Object.MemberwiseClone()

Namespace: UtilitySystem.Tool

Assembly: cs.temp.dll.dll

Syntax

```
[Serializable]
public class Timer
```

Constructors

Timer(Action, Single, Boolean)

This is a Constructor that will setup the timer.

Declaration

```
public Timer(Action action, float timer, bool isStopped = false)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Action	action	the Action that will be run when the timer finishes.
System.Single	timer	the amount of time it will take to finish.
System.Boolean	isStopped	This param if true will set the timer to stop from the start. (default false).

Methods

ResetTimer()

This Method will reset the timer.

Declaration

```
public void ResetTimer()
```

StartTimer()

This Method will start the timer.

Declaration

```
public void StartTimer()
```

StopTimer()

This Method will stop the timer.

Declaration

```
public void StopTimer()
```

UpdateTimer()

This Method will update the timer.

Declaration

```
public void UpdateTimer()
```

Namespace UI

This is a Namespace in UtilitySystem Namespace, Assembly-CSharp

Classes

Icon

This Class is a VisualElement that is created for the Inventory.

Icon.UxmlFactory

This new class is UxmlFactory and is needed to create a uxml tag for the UIDocument to read.

Slot

This Class is a VisualElement that is created for the Inventory.

Slot.UxmlFactory

This new class is UxmlFactory and is needed to create a uxml tag for the UIDocument to read.

SpriteElement

This Class is an VisualElement that is created for the Inventory Editor.

SpriteElement.UxmlFactory

This new class is UxmlFactory and is needed to create a uxml tag for the UIDocument to read.

Class Icon

This Class is a VisualElement that is created for the Inventory.

Inheritance

System.Object
Icon

Namespace: UtilitySystem.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class Icon : VisualElement
```

Constructors

Icon()

This is a Constructor and will set the style for the Icon.

Declaration

```
public Icon()
```

Icon(InventoryItem)

This is the constructor that will accept an InventoryItem It will set the Image & Style and add the label for amount if the item canStack.

Declaration

```
public Icon(InventoryItem item)
```

Parameters

TYPE	NAME	DESCRIPTION
InventoryItem	item	an InventoryItem that will be used to add the image and label if canStack is true.

Methods

ItemAmount(Int32)

This method will create a Label for the amount that the item currently has. Set the style of the Label and adds to the Icon.

Declaration

```
public void ItemAmount(int amount)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	amount	the amount that the item currently has

Style()

This Method will set the style of the Icon.

Declaration

```
public void Style()
```

UpdateIcon(Int32)

This Method will update the Icon with a new amount.

Declaration

```
public bool UpdateIcon(int amount)
```

Parameters

TYPE	NAME	DESCRIPTION
System.Int32	amount	The amount to update item amount.

Returns

TYPE	DESCRIPTION
System.Boolean	Return true when updated.

Class Icon.UxmlFactory

This new class is UxmlFactory and is needed to create a uxml tag for the UIDocument to read.

Inheritance

System.Object
Icon.UxmlFactory

Namespace: UtilitySystem.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class UxmlFactory : Icon.UxmlFactory<Icon, VisualElement.UxmlTraits>
```

Class Slot

This Class is a VisualElement that is created for the Inventory.

Inheritance

System.Object
Slot

Namespace: UtilitySystem.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class Slot : VisualElement
```

Constructors

Slot()

This is a Constructor and will set the style for the Slot.

Declaration

```
public Slot()
```

Fields

key

This Variable is the key that corresponds to the dictionary entry's key.

Declaration

```
public int key
```

Field Value

TYPE	DESCRIPTION
System.Int32	

Methods

AddslotLabel(String)

This Method will create a slat label to this Slot if it is a slot no the Hotbar.

Declaration

```
public void AddslotLabel(string text)
```

Parameters

TYPE	NAME	DESCRIPTION
System.String	text	the text that will be added to the slot label.

ClearIcon()

This Method will remove the icon from this slot.

Declaration

```
public void ClearIcon()
```

GetIcon()

This Method will get Icon attached to this Slot.

Declaration

```
public Icon GetIcon()
```

Returns

TYPE	DESCRIPTION
Icon	Return the Icon.

SetIcon(Icon)

This Method will add an Icon to this slot.

Declaration

```
public void SetIcon(Icon icon)
```

Parameters

TYPE	NAME	DESCRIPTION
Icon	icon	the Icon that will be added to this slot.

SetIcon(InventoryItem)

This Method will create a new Icon to this slot using an InventoryItem.

Declaration

```
public void SetIcon(InventoryItem item)
```

Parameters

TYPE	NAME	DESCRIPTION
InventoryItem	item	the InventoryItem that will be used to make the Icon.

Style()

This Method will set the style of the Slot.

Declaration

```
public virtual void Style()
```

TryGetIcon(out Icon)

This Method will try to get Icon attached to this Slot.

Declaration

```
public bool TryGetIcon(out Icon icon)
```

Parameters

TYPE	NAME	DESCRIPTION
Icon	icon	out Icon when found.

Returns

TYPE	DESCRIPTION
System.Boolean	return true if icon is found.

Class Slot.UxmlFactory

This new class is UxmlFactory and is needed to create a uxml tag for the UIDocument to read.

Inheritance

System.Object
Slot.UxmlFactory

Namespace: UtilitySystem.UI
Assembly: cs.temp.dll.dll

Syntax

```
public class UxmlFactory : Slot.UxmlFactory<Slot, VisualElement.UxmlTraits>
```

Class SpriteElement

This Class is an VisualElement that is created for the Inventory Editor.

Inheritance

System.Object
SpriteElement

Namespace: UtilitySystem.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class SpriteElement : Image
```

Constructors

SpriteElement()

This is a Constructor and will set the style for the SpriteElement.

Declaration

```
public SpriteElement()
```

SpriteElement(Sprite)

This is a Constructor and will set the style for the SpriteElement.

Declaration

```
public SpriteElement(Sprite sprite)
```

Parameters

TYPE	NAME	DESCRIPTION
Sprite	sprite	

Methods

RemoveSprite()

This Method will remove the sprite for this element.

Declaration

```
public void RemoveSprite()
```

SetSprite(Sprite)

This Method will set the sprite for this element.

Declaration

```
public void SetSprite(Sprite sprite)
```

Parameters

TYPE	NAME	DESCRIPTION
Sprite	sprite	

Class SpriteElement.UxmlFactory

This new class is UxmlFactory and is needed to create a uxml tag for the UIDocument to read.

Inheritance

System.Object
SpriteElement.UxmlFactory

Namespace: UtilitySystem.UI

Assembly: cs.temp.dll.dll

Syntax

```
public class UxmlFactory : SpriteElement.UxmlFactory<SpriteElement, Image.UxmlTraits>
```