Notizen

Empfohlene Naming Convention:

ModulTyp/ModulName/Subkategorie1/Subkategorie2 (etc.)

- Durch verwendung von “/” entstehen im Toolkit automatisch dropdownmenüs

Inspector can now show Tooltips, which can easily be added by using the tooltip attribute on properties that are toolkit accessible.

[Tooltip("Tooltip text")]

SOEvents_ItemSystem Documentation

Overview

Setup

1. Import the Package.
2. Make sure there is a singular Instance of SO_EditorSettings at “Assets/ItemSystem/Main Module/SystemStructure/Editor/Tabs and Windows/Settings”.
   It should already be there, but without it the toolkit won’t work as intended.
3. Make sure there is a singular Instance of SO_StatLoader at “Assets/ItemSystem/Sub Modules/StatModule”.
   It should already be there, but without it the toolkit won’t work as intended.
4. Open the Toolkit and make sure the Folder Paths are setup correctly to where you want your Instances and JSON files to go.
5. Add an Instance of ItemEventHandler to your Scene.
6. Check out the Examples inside the Folder “ExampleSetup”.
7. Have fun :)

General System Hierarchy

Manager Classes

ItemEventHandler

Handles all Item System related event calls, like Triggers for certain effects getting called.

Registration of Effects and Triggers within the Item System happens automatically, no need to do so manually.

This needs to be present as a Singleton Instance (before Runtime) in every Scene you want to utilize the Item System in.

ItemCoroutineHandler

Manages Item System Coroutines like periodically fired Effects or Stack Effects.

Registration of Effects and Triggers within the Item System happens automatically, no need to do so manually.

You should -*not*- add this to the scene manually, it gets created on Runtime if needed.

Interfaces

IItemUser

This Interface integrates the Item System into your custom Character Controller. If any Character or other script should be utilizing Items, add this Interface to it.

Automatically registers Items, Effects and Stats, so the System can correctly address them.
No need to manually access the Effect Registry unless you want to build something really out of the ordinary.

You can Access stats by using Userstats.[“StatModuleName”].Value

Stats get converted into Runtime_Stat at Startup to allow changes to character stats without changing the stats of items etc.

Stats that can be added up (numerical Values), automatically get added into a single value, while Stats that can't be added (like strings or bools) always use the stat from the highest hierarchy level. (See "General System Hierarchy") (Character -> Type -> Class -> Item).

This can be changed in the Settings of the Toolkit.

Holds a reference to the implementing GameObject, allowing you to access it from a call utilizing the Interface instead of another class or object reference, granting system compatibility across multiple Item User Classes. Access it through "ImplementingUser".

Holds a reference to the Team of the user to differentiate friend and foe.

IItemModule

Used for generalized access to different Item Modules within the Toolkit. Add this interface to any new Module you want to utilize inside the Toolkit.
This is not necessary if you derive from one of the abstract module classes, as they already implement this Interface.

IItemModuleBase

Used to filter Module derivates and Module instances, so only base types get used in certain parts of the toolkit. This has to be added to all base classes of entirely new Module types.

IProjectile

Used for Generalized Access to different Projectile Classes. Implement this into any Class you want to utilize as Projectile within the Item System.

Attributes

ConditionalHideAttribute

This attribute allows you to show/hide certain Values within the Item Toolkit based on the value of another Property.

[ConditionalHide(nameof(Property), Values to be visible at)]

Example:

[ConditionalHide(nameof(EffectTarget), 1, 2, 3)]

This would show the attributed value if the Property "EffectTarget" holds the value 1,2 or 3. Effect Target is an Enum Value, so the numbers reference certain Enum Indices.

ItemToolkitAccess

Add this to every Property you want to be editable within the Item Toolkit Inspector.

[ItemToolkitAccess]

The Toolkit and its Modules

Item Modules

In this Tab of the Item Toolkit, you find all the core modules necessary to create a basic Item and its effects.

Items

The core class of the system. In here you will reference the different modules that hold the actual functionality and stats. This class does nothing on its own, it is basically just a container to hold and manage the sub modules.

Stats (see “Stats”) you add to this module will only count towards the item itself, not the selected submodules like classes, so other items will not be affected by them.

I would not recommend creating multiple versions of this base class if possible; Rather use the sub modules to fill it with the in-game functionality you need.

Classes

This module further defines what kind of item you are dealing with. Use this to define broader classes like weapon, armor or pickup. The base does not contain any additional functionality apart from holding types (see “Types”) and the option to add stats (see “Stats”) that should be applied to all items that are part of this class.

Classes always hold a list of all types that can fall into their categorization and do not define which type an item later utilizes. This selection can be found on the item itself. (see “Items”)

Create derived types from the abstract base class to utilize this module, here you can then also add custom functionality if you want. This should be necessary in most use cases though.

After you created a derived Type of the abstract base class, you can then create an Instance of this derived type from within the Item System Tool. I would not recommend doing so manually outside of it.

Types

This is the most granular of the hierarchy modules. This defines subtypes of a class. (see “Classes”)

For the example I will assume that the types are subtypes of the class “Weapon”. Types could then further define those items into “one handed melee” or “two handed magical ranged” … I think you get the gist.

This module does also not contain any custom functionality on its own apart from stats (see “Stats”) that should be applied to all items of this type.

Create derived types from the abstract base class to utilize this module, here you can then also add custom functionality if you want. This should be necessary in most use cases though.

After you created a derived Type of the abstract base class, you can then create an Instance of this derived type from within the Item System Tool. I would not recommend doing so manually outside of it.

Effects

This module defines the actual functionality of an item effect. Utilize the abstract base class to create derived types that allow you to define the actual inner workings of an effect.

Example for the creation of a derived type: [BILD]

Effects get automatically registered within the system and utilized automatically when added to an item. There is no need to manually apply effects outside of the Item system itself.

After you created a derived Type of the abstract base class, you can then create an Instance of this derived type from within the Item System Tool. I would not recommend doing so manually outside of it.

Effects do not get called on their own. You will need to assign each effect a so-called Trigger (See “Trigger”), or they will sit idly and do nothing.

Trigger

Trigger modules do not contain any stats or functionality that impacts gameplay in any way. They are utilized to define when and how effects get called.

Create derived types of the abstract base class to define your own functionalities.

Example for the creation of a derived type: [BILD]

Triggers automatically register themselves as events inside the ItemEventHandler and in turn automatically register effects assigned to them. If you want a specific Trigger to be called, you simply have to access it through the Item handler and invoke it.

Example of Invoke: [BILD]

Triggers allow you to define custom conditions they should check before triggering the associated effects.

Example of Custom Condition: [BILD]

They also allow you to define other custom functionalities, should you need it. Below an example of a trigger, that calls its effects over and over again periodically, after being called once.

[BILD SO_EFFECT_TRIGGER_INTERVAL]

Order of calls is as follows:

Invoke -> Check if any effects are registered -> Check Condition -> Call Custom Functionality -> Apply Effect.

Item Containers

In this Tab of the Item Toolkit, you find all Sub Modules that are utilized to sort, pool, categorize etc. Items.

Item Slots

Holds references to an Item. When a new item gets added to the slot, it automatically checks if the item belongs to the defines Classes/Types.

Utilize this as part of you inventory system to easily allow for filtering and accessing items.

Item Pools

Holds references to a pool of selected Items. These items can then be attributed with a weight, that defines how likely it is to retrieve said item from the pool.

Provides you with an easy way to create custom loot tables.

Call the method “METHOD“ to retrieve a random Item from the pool, based on the set weights.

Stat Modules

This Tab allows you to easily define different types of stat values, you can later apply to items, classes or types.

Create derived types of the abstract base class to define your own types of stats. In most cases this should not be needed though, since the system comes with pre-defined modules of the most common stat types.

You can create Instances of these derived types from within the Item System Tool. I would not recommend doing so manually outside of it.

ToolTip Module

This Tab allows you to easily add dynamic tooltips, that utilize hyperlinks and stat loading, to your items.

ToolTipID: This value defines the ID you have to enter inside a tooltip text to hyperlink this tooltip.

HyperlinkText: This is the text that will be displayed as hyperlink, when the ID gets used inside a tooltip text.

ToolTipText: This is the actual content of the Tooltip. If you want to load stats or link other tooltips as hyperlinks, you have to do so in here.

Syntax for hyperlinking: {***ID***}

Syntax for property loading: [(Module:***MODULENAME***)(Property:***PROPERTYNAME***)]

Syntax for stat loading: [(Module:***MODULENAME***)(Stat:***STATNAME***)]

The tooltip system is capable of either generating the tooltip as string or as UI directly.

If you decide to generate a UI directly you can choose between creating UI from code, or filling in a prefab you created. (Currently no prefab functionality, due to switch to UITK)

You can do so by calling:

- UITKFactory.CreateNewUIWindow
  - Creates new Window from Code
- UITKFactory.CreateNewUIPopup
  - Creates new popup without buttons, drag or resize functionality
- UITKFactory.CreateNewUIFromPrefab
  - Fills in Prefab with content and functionality
- UITKFactory.ParseRichText
  - Converts string content to hyperlinks and loads item stats or properties, integrating them into the string, based on system syntax

Both options automatically allow for popup windows to be created on hyperlink interaction, for windows to be dragged and resized as well as a close window option.

To access the tooltip string required to generate UI, you have two options:

- Get the Tooltip from the SO directly
- Call UIToolTipRegistry.Instance.RetrieveTooltip() to get it from the centralized registry

To add or change hyperlink functionality/ mouse interactions you have to edit the UIController class. (Might decouple this by using the event handler in future versions)

For the UI system to work properly both the UIController and the UIToolTipRegistry have to be present in the scene.

File Manager

This tab allows you to export your modules into JSON files, or if you already did so, reimport them.

This can be utilized to implement modding support into your game.

Note: An External WPF tool to create and edit JSON Items is planned, as well as predefined loading mechanisms for JSON files. When I finally manage to get to this, you will be able to easily allow for users to mod your game's items. YEAY.

Tags

This tab allows you to create simple tags, you can then apply to the other modules inside the toolkit. They can be used to filter the modules you get shown in the toolkit, but also to further define allowed items for example inside item slots or item pools.

Settings

In here you find all the general settings needed to setup the item toolkit and its file directories.