Blade and Sorcery Basic Scripting Guide

whats up gamer nation, Basalt (Basalt#4272) here with another sicc bas guide!!!!!!

First and foremost, this guide will not teach you coding. I want that crystal clear, for learning the coding language of Blade and Sorcery I recommend the following playlist: https://m.youtube.com/playlist?list=PLyJiOytEPs4eQUuzs3PhM_7yU63jdibtf
for reference, i only watched about 9 of these to get a decent grasp, i recommend watching more of em though.

What this document *will* teach you is how to start scripting modded weapons and maps for This Particular Game™

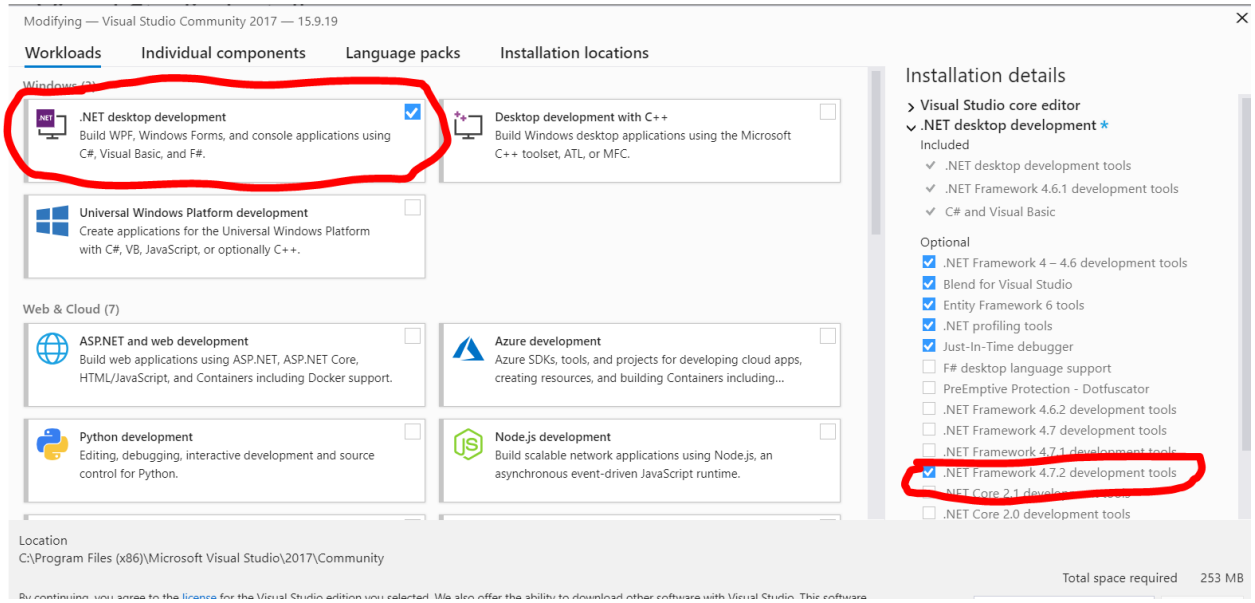
When I first started modding I didn't know how to code, it wasn't until way after I got my modder role that I decided to learn. It was kinda stupidly complicated to figure out

how to start coding for this game, so I figured ill make a quick guide of how to just get started.

Right, so first of all, even if you don't know how to script yet, this tutorial will help regardless for when you inevitably can because you are smart and you are capable of doing the things you want.

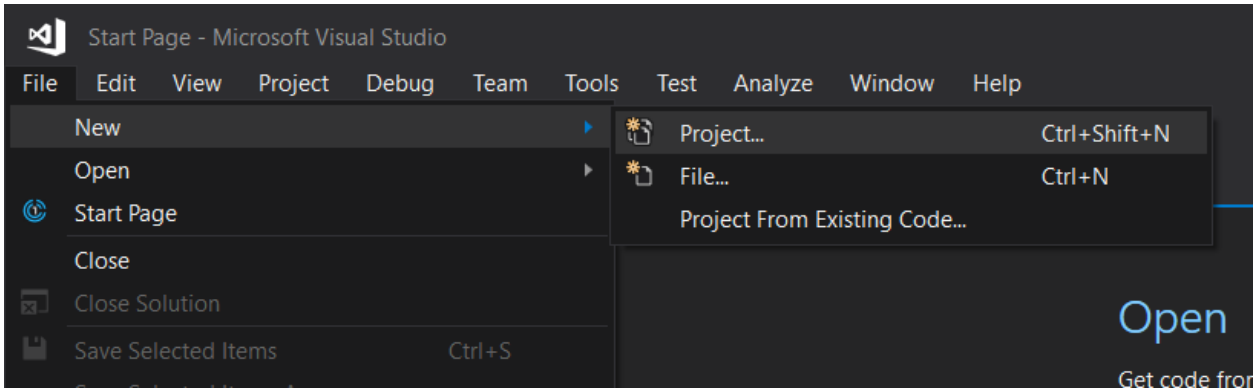
So, first you gotta have the right Visual Studio version. For god knows whatever reason Microsoft has decided to wipe away the very existence of Visual Studio Community 2017. Because of that, you have to install visual studio from the following google drive link: https://drive.google.com/open?id=1bcP4UFo_9bqzU_Li0QpaGJA5wwaYCgB_

Make sure these checkboxes are ticked when installing so you can actually like, do anything.

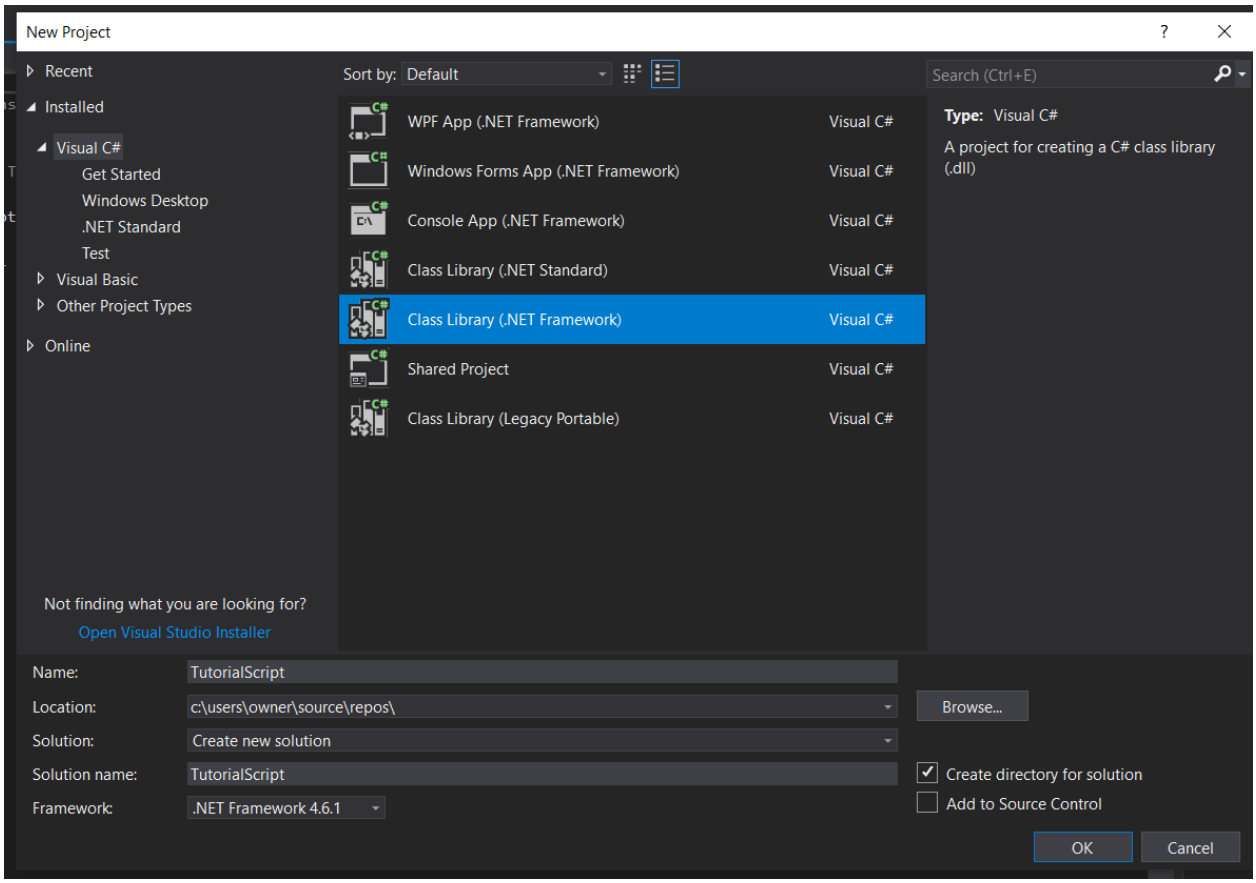


Once it's installed, sign in with your Microsoft account.

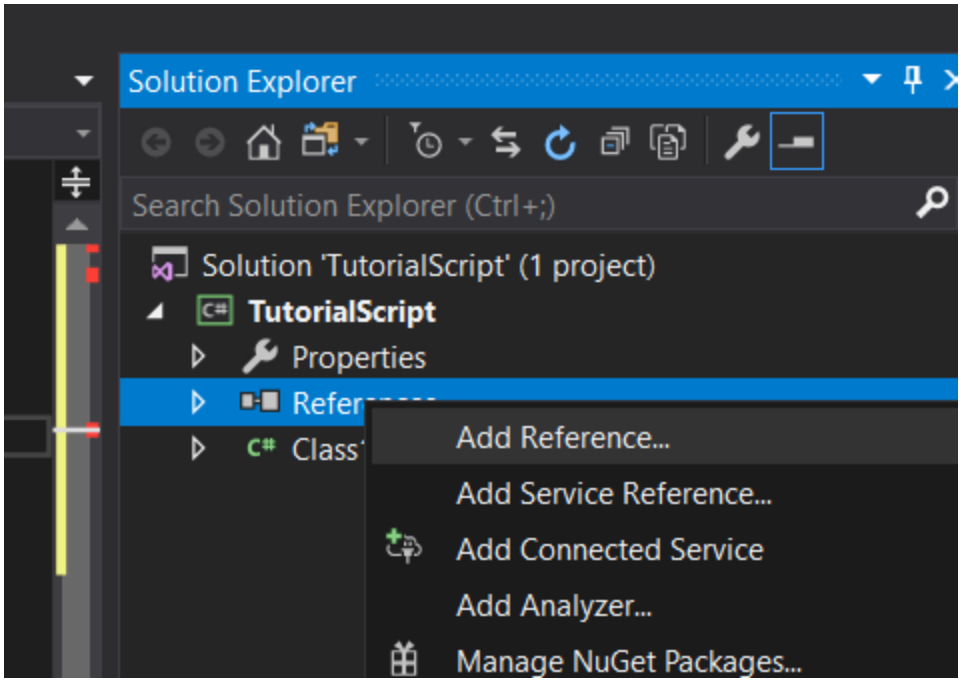
After that, create a new project like so



Select .NET Framework, and rename it to the name of your choosing
(Note: the name you choose will be the name of the .dll when you build it)

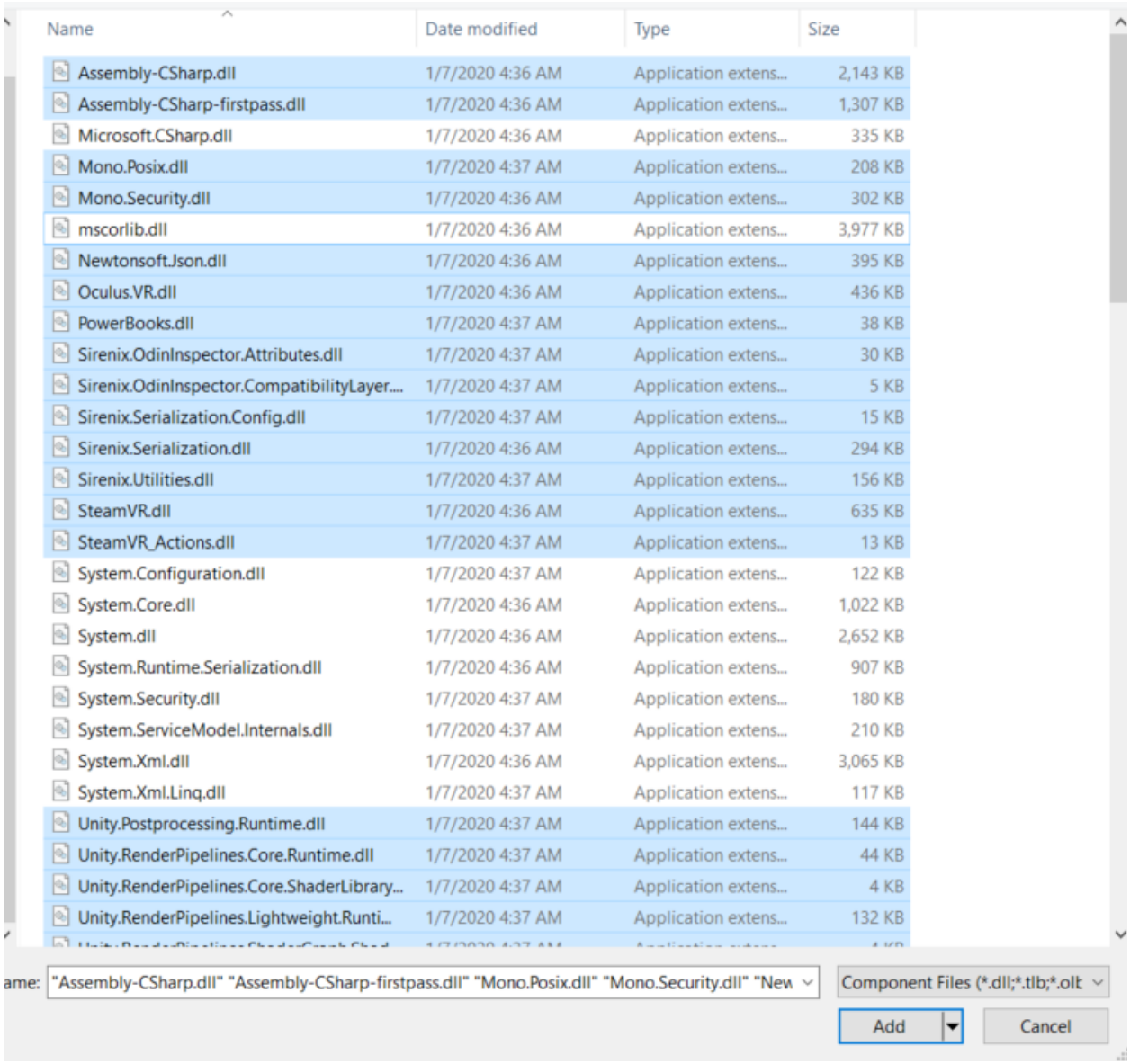


Next, look on the right side, right click “References” and click “Add Reference”



Hit Browse and navigate to your Blade and Sorcery folder. For Steam users it's located at "C:\Program Files (x86)\Steam\steamapps\common\Blade & Sorcery" by default. Click on BladeAndSorcery_Data > Managed.

Select every .dll in that folder, **EXCEPT** anything that begins with System. and the two files called Microsoft.CSharp.dll and mscorlib.dll.



Click Add, then OK

Delete all the shit text that is generated for you when you make a new project and paste the following

```
using UnityEngine;
using BS;


namespace RenameThis
{
	public class RenameThisToo : MonoBehaviour
	{

	public void Awake()
	{

	}


	public void Update()
	{

	}

	}

}
```

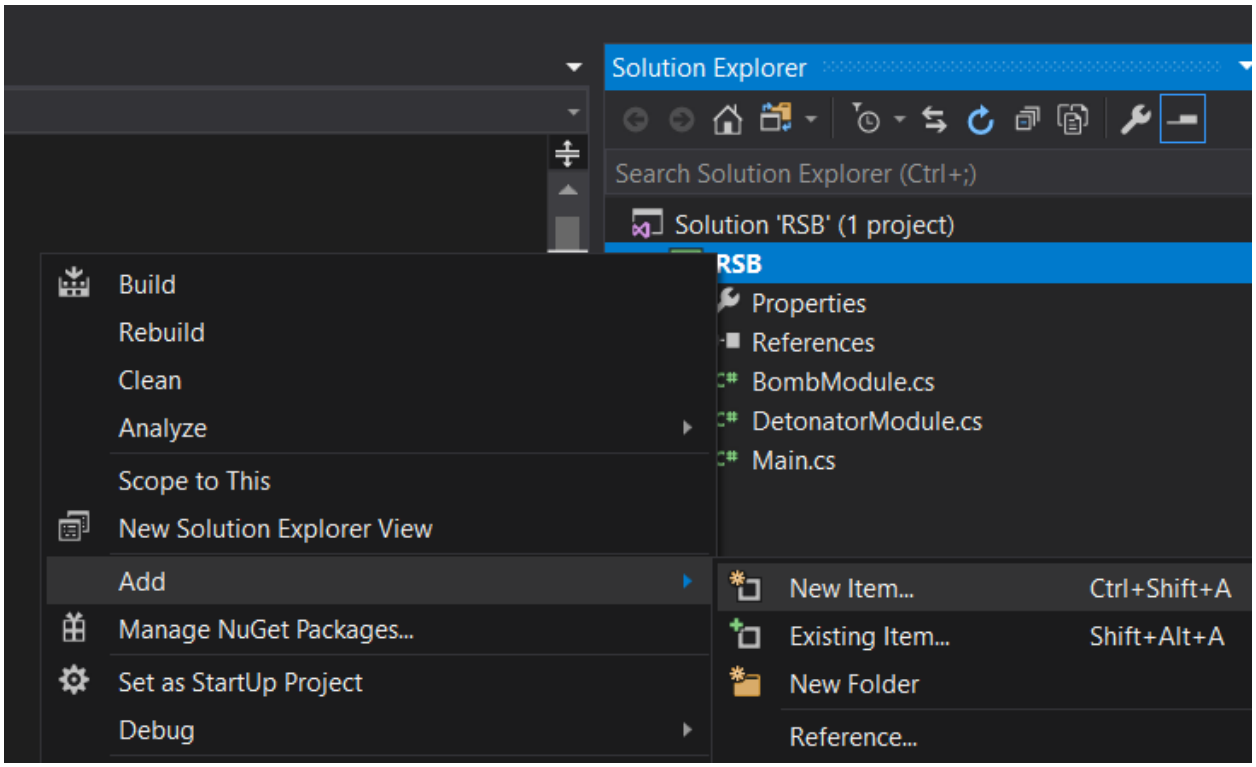
If you added the references properly, there should be no errors/red lines.

Now you can actually do your code.

The easiest way to find the classes is by typing “BS.” and scrolling through the list of options.

A fantastic place to find easy to read pieces of Blade and Sorcery code is https://github.com/Ebediam
Ebediam is a great scripter, known as the creator of Dark Chains and a very easy to use gun script called ItemShooter.dll. Some of it is outdated, but it really is an absolute treasure trove of scripting assets. Big thanks to Ebediam for making all their code open source, and I urge those who script things for this game to do the same.

Once you have your code finished, and there are no errors, you need to create a second class. The easiest way to do this and stay organized is to look to the right, right click on your current class, click Add, then New Item



Then select Class and hit Add

Next, if you're coding a weapon, paste in the new class the following code, and change YourNamespace and YourClass to that of your code, and rename KeepThisDifferent to something else.

```
using BS;

namespace YourNamespace
{
	public class KeepThisDifferent : ItemModule
	{
	public override void OnItemLoaded(Item item)
	{
	base.OnItemLoaded(item);
	item.gameObject.AddComponent<YourClass>();
	}
	}
}
```

If you're making a map, paste in the new class the following code, and change YourNamespace and YourClass to that of your code, and rename KeepThisDifferent to something else.

```
using BS;

namespace YourNamespace
{
	public class KeepThisDifferent : LevelModule
	{
	public override void OnLevelLoaded(LevelDefinition levelDefinition)
	{
	base.OnLevelLoaded(levelDefinition);
	levelDefinition.gameObject.AddComponent<YourClass>();
	}
	}
}
```

Note, `levelDefinition.gameObject.AddComponent<YourClass>();` is if you're modifying built in Unity options, such as gravity or physics options, I've used this once, but I'm not fully sure how often you'd use this, mostly because I've only scripted one map.

(The next segment within parenthesis are for maps, if you aren't making a map, you can ignore this until the end parenthesis.

So maps are a bit more complicated, it just be like that sometimes, you might not think that it be like the way that it be but it do.

If you want to find a particular GameObject, as expected, you have to find it's name with the following code in your main class:

```
private GameObject Door1Open;
private GameObject Door1Close;

        public void Awake()
        {
        Debug.Log("Door Script Awake");
        Door1Open = GameObject.Find("Door1Open");
        Door1Close = GameObject.Find("Door1Close");
        }
```

And then to reference the gameobject, you do it like

```
        Door1Close.GetComponent<Animation>().Play();
        Door1Close.GetComponent<AudioSource>().Play();
        Door2Close.GetComponent<Animation>().Play();
        Door2Close.GetComponent<AudioSource>().Play();
```

And for the record, I've got a working automatic door script, I'm just trying to figure out a good way to make it versatile and easy to use.

Then, to reference those objects in the second class, you use:

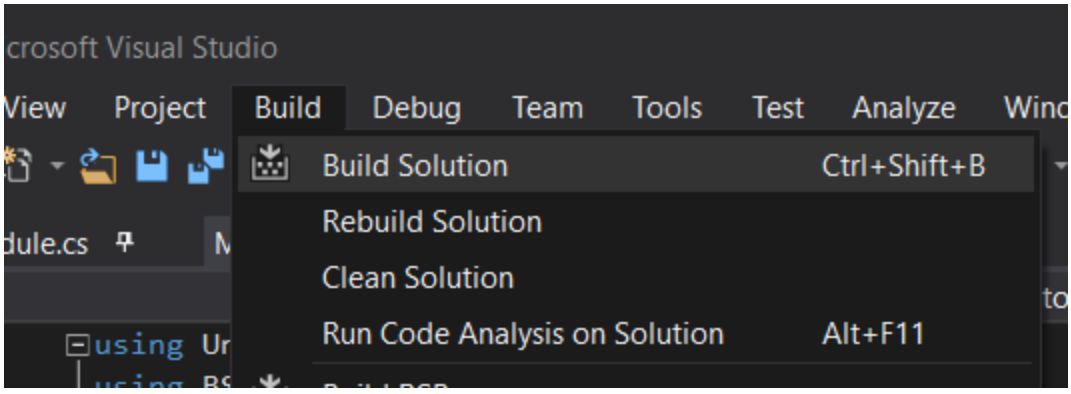
```
using BS;
```

```
using UnityEngine;

namespace AutoDoor
{
	public class Door1Module : LevelModule
	{
	public override void OnLevelLoaded(LevelDefinition levelDefinition)
	{
	base.OnLevelLoaded(levelDefinition);

	GameObject.Find("Door1").AddComponent<Door1>();
	}
	}
}
```

)

MAP SHIT IS DONE, THE REST APPLIES TO WEAPONS AND MAPS

Alright, now that you have both of your classes completed, look towards the top of VS and click Build>Build Solution

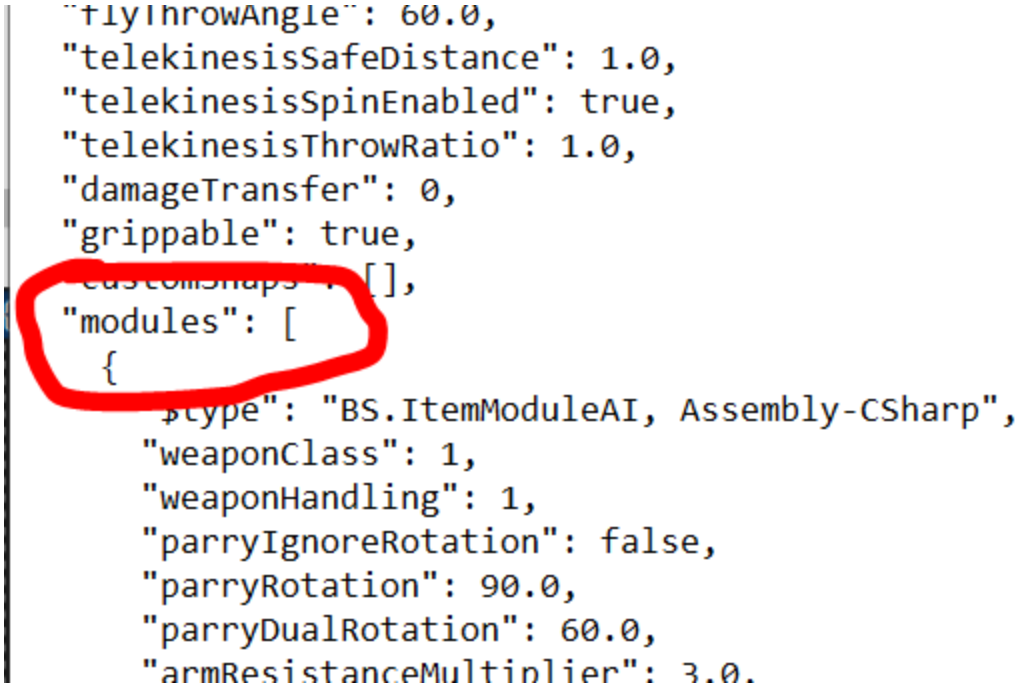


After that, navigate to the project folder. If you didn't change it's default file location you can find it at:

C:\Users\(username)\source\repos\projectname\projectname\bin\Debug

Find the .dll you built, it'll be the same name as the project, and drag it into your weapon folder. (ill get to maps in a sec)

Open up your Item, and find modules.



Just after the [, but before the {, paste and rename:

```
{
        "$type": "YourNamespace.ItemModuleName, DllName"
},
```

An example of this in code would be

```
using BS;

namespace YourNamespace
{
    public class ItemModuleName : ItemModule
    {
        public override void OnItemLoaded(Item item)
        {
            base.OnItemLoaded(item);
            item.gameObject.AddComponent<YourClass>();
        }
    }
}
```

As for Maps, you need to locate and open your Level, and find modules.

```
"mapOrbCubeMapPath": "@mappreviews:Market",
"modes": [
  {
    "name": "SandBox",
    "displayName": "{SandBox}",
    "description": "{SandBoxDescription}",
    "iconPath": "beach-bucket",
    "modules": [
      {
        "$type": "BS.LevelModuleCleaner, Ass
        "cleanerRate": 5.0
      },
      {
        "$type": "BS.LevelModuleDeath, Assem
```

Just after the [, but before the {, paste and rename:

```
{
```

```
        "$type": "YourNamespace.LevelModuleName, DllName"
},
```

The previous screenshot shows an item module, but it's exactly the same for a level module

AIGHT GRATS YOU GOT SHIT DONE!

The rest is up to you to do at your own discretion, however I will list some useful tips and tricks for coding your stuff.

The following is how to check if the trigger is pressed when holding the coded weapon.

```
Item item; //(these don't go in any function, just the overall class)
bool triggerPressed = false; //if you have no delay or function that relies on triggerPressed being true or false, this will mention it's not used. you dont really have to worry about it

public void Awake()
        {
        //Checks if handle is grabbed, and waits for trigger.
        item = GetComponent<Item>();
        Debug.Log("Weapon Awake");
        item.OnHeldActionEvent += HandleGrabbed;
        }

void HandleGrabbed(Interactor interactor, Handle handle, Interactable.Action action)
        {

        if (action == Interactable.Action.UseStart)
        {
        //If Trigger is pressed, trigger Detonate()
        Detonate();
        triggerPressed = true;
        Debug.Log("Trigger Pressed");
        }
        }
```

```
public void Detonate()
	{
	triggerPressed = false;
	//Your Code Here
	}
```

You can replace Interactable.Action.UseStart with Interactable.Action.AlternateUseStart to wait for Alt Use instead of Trigger.

The following is how to make a very simple timer, in conjunction with the trigger, code above, but the “triggerPressed” segment can be removed.

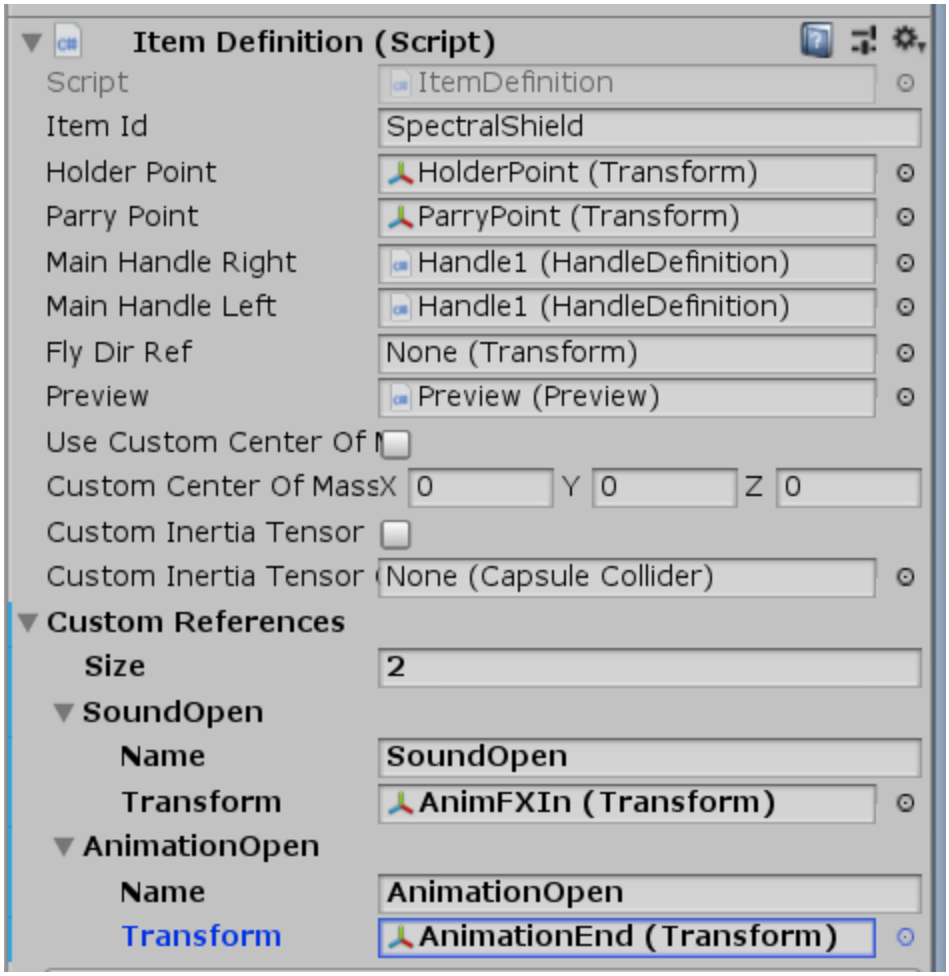
```
public void Awake()
	{
	 float triggerDelay = 0f;
	}

public void Update()
	{
	triggerDelay -= Time.deltaTime;

	If (triggerDelay <= 0f && triggerPressed == true )
	{
		Detonate();
	}
	}

public void Detonate()
	{
	triggerPressed = false;
	//Your Code Here
	}
```

So to make the timer work, you'd do something like " triggerDelay = 2f; " after your trigger. This sets the delay to 2 seconds, which then the update function reduces triggerDelay over time.

The following is how you'd find and trigger an animation, particle system, audio source, or any other component.

```
  public void Update()
        {
if (stateClosed == true)
        {

item.definition.GetCustomReference("SoundOpen").GetComponent<AudioSource>().Play();

item.definition.GetCustomReference("AnimationOpen").GetComponent<Animation>().Play();
                stateClosed = false;
                animDelay = 0.5f;
        }
        }
```

Then in unity, you'd add a custom reference to your item definition like so:



I'll add more tips and tricks as I discover them, but these are the basics. I also upload all the code I make onto the Nexus link for my weapons, like the Hidden Blade and Smiting. You can find my Nexus at https://www.nexusmods.com/bladeandsorcery/users/71722738?tab=user+files