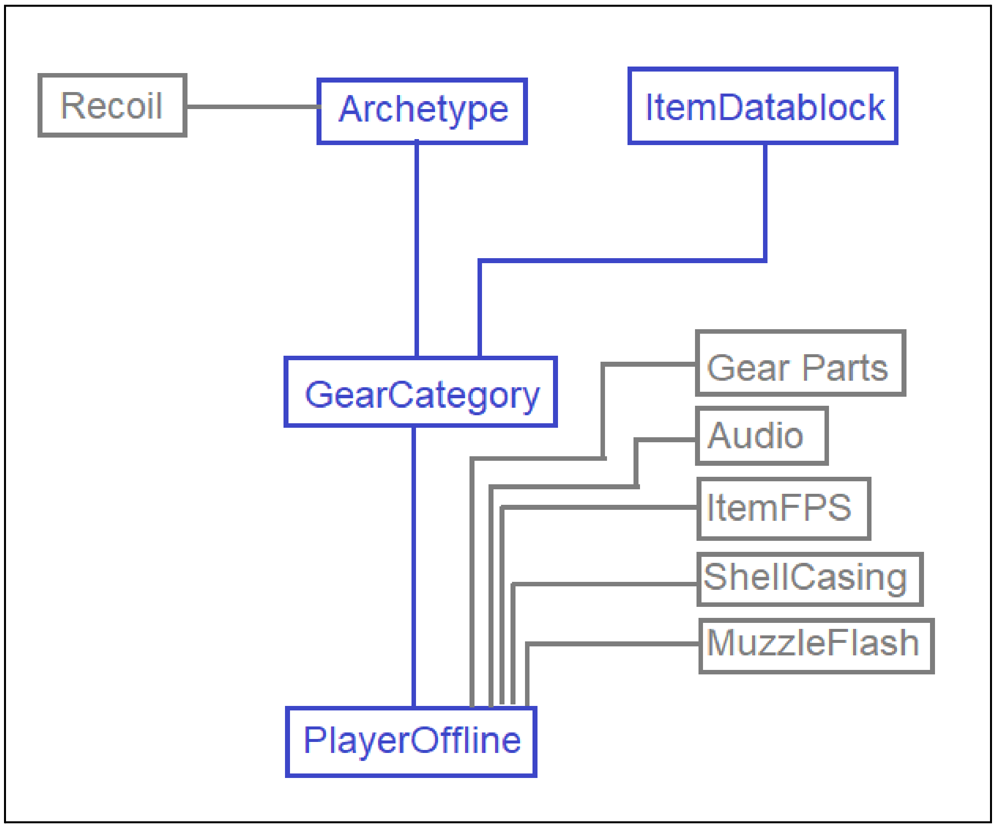
How to make custom weapons for GTFO

Written by Mccad00

In GTFO, almost all content and systems are highly configurable via json files called Datablocks, which provide the game with stats, behavior, and just general data for a large variety of systems. In particular, weapons are split into a large number of datablocks which provide stats, determine behavior, recoil data, and even configure the appearance of weapons.

Before going any further, it's important that we understand the hierarchy of datablocks that set up weapons, and understand what each of these datablocks control.



PlayerOfflineGear: Creates an inventory item by assigning parts and models to a 'GearCategory' entry.

GearCategory: Groups together 4 'Archetype' entries and 1 item datablock entry.

Archetype: Provides stats, a recoil datablock entry, and metadata to a weapon.

Item: Determines the inventory slot, controls HUD elements, and what type of weapon to use.

Archetype

At the heart of every weapon in GTFO is the Archetype datablock, which determines a gun's statistics and metadata. There is a lot to unpack here and it's almost all self explanatory, so I'll just break down each of the variables and what they control.

```
"PublicName": The weapon's name in the inventory (eg Assault Rifle)
"Description": The description in the lobby screen

"FireMode": 0 - Semi, 1 - Burst, 2 - Auto, 3 - SemiBurst (broken)
"RecoilDataID": The RecoilDatablock entry to use
"DamageBoosterEffect": Used by boosters to determine what to buff
"Damage": The weapon's base damage

"DamageFalloff": {
    "x": The distance at which the damage dropoff begins
    "y": The distance at which the weapon does minimum damage
},

"StaggerDamageMulti": Multiplier of the base damage for stagger
"PrecisionDamageMulti": Multiplier of the base damage for weakspots

"DefaultClipSize": The number of bullets in a magazine
"DefaultReloadTime": The time in seconds it takes to reload
"CostOfBullet": Divide this weapon type's “AmmoMaxCap” from the
player datablock by this value to determine the reserve ammo capacity
"ShotDelay": The time in seconds between each shot of the weapon

"PiercingBullets": True or false, whether the gun pierces
"PiercingDamageCountLimit": The amount of times a bullet can pierce

"HipFireSpread": The inaccuracy cone's angle when hipfiring
"AimSpread": The inaccuracy cone's angle when aiming down sights

"EquipTransitionTime": The time in seconds to equip the weapon
"EquipSequence": See FireSequence and ReloadSequence

"AimTransitionTime":  The time in seconds to aim down sights
"AimSequence": : See FireSequence and ReloadSequence
```

Archetype (Cont.)

```
"BurstDelay": The delay in seconds between each full burst
"BurstShotCount": The amount of shots to fire in a burst

"ShotgunBulletCount": The amount of pellets to fire
"ShotgunConeSize": The radius of the shotgun’s cone (Integer)
"ShotgunBulletSpread": The spread between each pellet (Integer)

"SpecialChargetupTime": The time it takes to charge up before firing
"SpecialCooldownTime": The time it takes to cool down after firing
"SpecialSemiBurstCountTimeout": Unused or broken

"Sentry_StartFireDelay": The time before a sentry starts firing
"Sentry_RotationSpeed": The speed a sentry can rotate
"Sentry_DetectionMaxRange": The maximum distance a sentry can target
"Sentry_DetectionMaxAngle": The maximum angle a sentry can pivot
"Sentry_FireTowardsTargetInsteadOfForward": Whether or not a sentry
gun leads it’s shots (requires more testing to determine)
"Sentry_LongRangeThreshold": Unused?
"Sentry_ShortRangeThreshold": Unused?
"Sentry_LegacyEnemyDetection": Whether or not to use the new or old
sentrygun targeting behavior
"Sentry_FireTagOnly": Whether or not to only shoot biotracked enemies
"Sentry_PrioTag": Whether or not to prioritize biotracked enemies

"Sentry_StartFireDelayTagMulti": Delay multi for biotracked enemies
"Sentry_RotationSpeedTagMulti": Rotation multi for biotracked enemies
"Sentry_DamageTagMulti": Damage multi for biotracked enemies
"Sentry_StaggerDamageTagMulti": Stagger multi for biotracked enemies
"Sentry_CostOfBulletTagMulti": Ammo multi for biotracked enemies
"Sentry_ShotDelayTagMulti": Firerate multi for biotracked enemies

"name": The weapon’s internal name in the files, unused ingame
"internalEnabled": Whether or not this weapon is enabled ingame
"persistentID": The weapon’s ID for reference in other datablocks
```

Note: Assign a unique persistent ID to your weapon that doesn’t occur anywhere else in the datablock. Keep track of this ID as you’ll be using it in the GearCategory datablock.

GearCategory - A stupid system for dumbassess

Perhaps the most baffling datablock involved in custom weapons and gear is the GearCategory datablock. The GearCategory system is designed to group together 4 archetype datablock entries into one category and assigns them an entry in the Item Datablock, which is used to assign the weapons behavior on the HUD and a slot in the inventory to fit into. Let's take a look at an example.

```
"PublicName": "", Unused by guns, the ingame name for all other gear
"Description": "",Unused by guns, the lobby description for all other gear

"BaseItem": 108,                   The item datablock entry to reference
"HUDIcon": "",                     Unused
"FPSArmPoseName": "",              Unused
"ThirdPersonFullbodyMovement": 0,  Third person animation set:

0 = Rifle
1 = Pistol
2 = Melee
3 = CarryHeavy

"SemiArchetype": 1,          The 0th archetype in this category
"BurstArchetype": 3,         The 1st archetype in this category
"AutoArchetype": 5,          The 2nd archetype in this category
"SemiBurstArchetype": 16,    The 3rd archetype in this category

Note: These don't actually have any bearing on firemode, you can put
whatever you want into any of these. The number (0-3) of the archetype in
this category is indexed by the PlayerOffline gearjson.

"PartAlignPriority": [], Determines which parts align to eachother.
"name": "Rifle",         Internal name, unused
"internalEnabled": true, whether or not this category is enabled ingame
"persistentID": 1    This category's ID for reference in other datablocks
```

Once again, assign a unique persistent ID to your gear category to prevent conflicts with existing game data. Keep track of this persistent ID as you'll need to reference it in the PlayerOffline gear datablock

GearCategoryDataBlock: Part align priority

Sometimes, multiple parts in a GearJson will share the same aligns as each other, which will result in one of the parts defaulting to have that align. In order to control which parts get the align in cases like this, we need to edit the PartAlignPriority in the GearCategoryDataBlock. The GearCategory which is linked to your weapon via the ‘Category’ component type will determine which GearCategory you need to modify for this. Let’s look at the Magnum’s part align priority to see how it works:

```
"PartAlignPriority": [
{
    "AlignType": 7,
    "PartPrio": [
    12
    ]
},
{
    "AlignType": 8,
    "PartPrio": [
    12
    ]
}
],
```

`"AlignType":` This corresponds to the same Enum as aligns in the gear datablocks.
`"PartPrio":` A list of gear component types which determines the hierarchy of which parts to assign aligns to.

PlayerOfflineGear

In GTFO, almost every single piece of equipment is assembled programmatically in order to get more mileage out of the limited number of assets they have. Weapons, tools, and even some consumables are put together using this same system. For the sake of simplicity, I'll be referring to these assembled items as 'Gear'. Every assembled Gear item in GTFO is set up in the PlayerOfflineGearDataBlock via a string called GearJson.

Pictured: An exploded view of a typical GTFO weapon, using the Gear system

GearJson formatting

To get started understanding the format of GearJson, let's take a look at the GearJson string for the Pistol:

```
"GearJSON":
"{\"Ver\":1,\"Name\":\"Shelling S49\",
\"Packet\":{\"Comps\":{\"Length\":16,
\"a\":{\"c\":2,\"v\":8},
\"b\":{\"c\":3,\"v\":108},
\"c\":{\"c\":4,\"v\":8},
\"d\":{\"c\":5,\"v\":1},
\"e\":{\"c\":6,\"v\":1},
\"f\":{\"c\":7,\"v\":1},
\"g\":{\"c\":8,\"v\":16},
\"h\":{\"c\":9,\"v\":15},
\"i\":{\"c\":10,\"v\":27},
\"j\":{\"c\":11,\"v\":27},
\"k\":{\"c\":12,\"v\":38},
\"l\":{\"c\":16,\"v\":1001},
\"m\":{\"c\":19,\"v\":1002},
\"n\":{\"c\":23,\"v\":8},
\"o\":{\"c\":25,\"v\":3}},
\"MatTrans\":{\"tDecalA\":{\"position\":{\"x\":-0.098,\"y\":-0.07},\"
scale\":0.05},\"tDecalB\":{\"position\":{\"x\":-0.098,\"y\":-0.07},\"
scale\":0.04},\"tPattern\":{\"position\":{\"x\":-0.09,\"y\":-0.03},\"
angle\":-90.0,\"scale\":0.1}},\"publicName\":{\"data\":\"Shelling
S49\"}}}",
```

I've taken the liberty here to add line breaks into this string for the sake of demonstration, but please note that in your datablock, this entire string will be on a single line. The entire end of this string, the section labeled "MatTrans" is unused by the game, so to further simplify, I'll break down the GearJson into its three main building blocks

`"{\"Ver\":1,\"Name\":\"Shelling S49\",`
This line assigns the GearJson a ‘descriptive name’, which is displayed in your inventory and in the lobby screen.

`\"Packet\":{\"Comps\":{\"Length\":16,`
This line tells the game how many gear components your part contains. If this number doesn’t match the actual number of gear components, the game won’t be able to read the GearJson string correctly.

`\"a\":{\"c\":2,\"v\":8},`
This line defines a gear component to add to our GearJson, which is broken up into 3 values:

`\"a\"` - The index, in alphabetical order, of this gear component. Notice in our example that every component starts with a unique letter identifier.

`\"c\":2` - The component type, such as FrontPart, ReceiverPart, or StockPart. In this component, type is ‘2’ which corresponds to GearCategory. If you attempt to use the same component type on multiple components, only one of them will load.

`"V\":8` - The component value. This tells the game which component to load for a given type. In this component, since the type was GearCategory, this will load the GearCategory with persistentID 8.

You can have as many of these gear components as you want in any given gearJson, provided you use correct syntax. Because of the clusterfuck nature of this string, I highly advise against writing any of these from scratch, but rather copy pasting an existing GearJson string from vanilla, and editing the values and syntax to suit your needs.

Gear Component Types

Note: Unused/Unusable component types are omitted.

1	FireMode	Value between 0 and 3, tells bulletweapon items which Archetype to load in the GearCategory datablock
2	Category	PersistentID of the GearCategory to load
3	BaseItem	PersistentID of the ItemDatablock entry to load.
4	ItemFPSSettings	PersistentID of the ItemFPS to load. Used for First Person positioning
5	AudioSetting	PersistentID of the WeaponAudio to load
6	MuzzleFlash	PersistentID of the WeaponMuzzleFlash to load
7	ShellCasing	PersistentID of the WeaponShellCasing to load
12	FrontPart	The front, barrel portion of a weapon
16	ReceiverPart	The receiver of a weapon
19	StockPart	The grip or stock of a weapon
21	SightPart	The sight of a weapon; some front parts have this function built in
23	MagPart	The magazine of a weapon. Not all parts support this
25	FlashlightPart	The flashlight of a weapon
27	ToolMainPart	The base part which the rest of the tool parts align to
30	ToolGripPart	The grip or stock of a tool; this is interchangeable with StockPart
33	ToolDeliveryPart	Typically the business end of a tool
37	ToolPayloadPart	Varies widely between tools
40	ToolTargetingPart	The ‘targeting module’ placed on some tools
42	ToolScreenPart	The display screen, only functional if the gear uses the correct baseitem
44	MeleeHeadPart	The head of the hammer
46	MeleeNeckPart	The part that connects the head to the handle of a hammer
48	MeleeHandlePart	The handle of a hammer
50	MeleePommelPart	The base of a hammer handle

Note: Even though the devs typically only use parts where it’s appropriate, you can actually use any of these parts on any piece of gear. Tool parts are usable on guns.

Gear Datablocks: General

The gear datablocks are used to assign behavior, animations, models, and aligns to gear parts, for use in GearJson. The most simple, and arguably the most important part of these datablocks is ‘General’, which tells the game what model and children to load, and what hand animations to use.

```
"General": {
    "Model":
"Assets/AssetPrefabs/Items/Gear/Parts/Fronts/Front_Short_Shotgun_2.prefab",
    "Children": [],
    "GearCategoryFilter": 0,
    "LeftHandGripAnim": "Short_Shotgun_2_Player_Idle",
    "RightHandGripAnim": "Short_Shotgun_2_Player_Idle",
    "AssetBundle": 20,
    "BundleShard": 3
},
```

`"Model":` The path of the part prefab to load
`"Children":` A list of child prefabs to load
`"GearCategoryFilter": 0,` Unused
`"LeftHandGripAnim":` The animation clip to play for the player’s arms on layer 5
`"RightHandGripAnim":` The animation clip to play for the player’s arms on layer 6
`"AssetBundle": 20,` The asset bundle to load the prefabs from.
`"BundleShard": 3` The asset shard to load the prefabs from.

Generally when copying prefabs into this datablock, it’s important to make sure you assign the AssetBundle and BundleShard to the same ones as the prefab you copied.

Gear Datablocks: aligns

Gear parts are attached to each other using aligns, which are empty Game Objects placed onto gear parts which aligned parts are parented to. In order to change the usage of these aligns, you need to edit, or make a copy of the gear datablock entry for the desired part. Let's look at the aligns of Front_UZI_1:

```
"Aligns": [
    {
        "AlignType": 0,
        "AlignName": "Receiver_M"
    },
    {
        "AlignType": 1,
        "AlignName": "Front_SE"
    },
    {
        "AlignType": 2,
        "AlignName": "Front_Mag"
    },
    {
        "AlignType": 5,
        "AlignName": "Front_Flash"
    },
    {
        "AlignType": 7,
        "AlignName": "LeftHand"
    },
    {
        "AlignType": 6,
        "AlignName": "Sight_Align"
    },
    {
        "AlignType": 4,
        "AlignName": "Receiver_Sight"
    }
],
```

`"AlignType": 0,` The Type of the align, which determines which part attaches to this

`"AlignName": "Receiver_M"` The name of the game object which this align attaches parts to. Unfortunately, we can't add or modify these game objects in this datablock, but we can tell the game what parts to place onto these aligns.

Gear Datablocks: align types

0	Muzzle	Weapon’s muzzle flash
1	ShellEject	Weapon’s ejection port
2	Magazine	Corresponds to the MagPart gear component
4	Sight	Corresponds to the SightPart gear component
5	Flashlight	Corresponds to the FlashlightPart gear component
6	SightLook	The aim down sights camera if no sight part is present
7	LeftHand	The player’s left hand
8	RightHand	The player’s right hand
9	Receiver	Corresponds to the ReceiverPart gear component
10	Front	Corresponds to the FrontPart gear component
12	ToolGrip	Corresponds to the ToolGripPart gear component
13	ToolDelivery	Corresponds to the ToolDeliveryPart gear component
14	ToolPayload	Corresponds to the ToolPayloadPart gear component
15	ToolTargeting	Corresponds to the ToolTargetingPart gear component
16	ToolScreen	Corresponds to the ToolScreenPart gear component
17	ToolDetection	unknown
18	ToolScanning	unknown
20	MeleeHead	Corresponds to the MeleeHeadPart gear component
40	RotationPivot	Presumably used by sentry guns to position the contained weapon parts
41	GroundPlacement	Presumably used to position the base of a sentry gun

Gear Datablocks: FireSequence and ReloadSequence

To handle animations, the GearDatablocks lists of WeaponAnimSequenceItems, which provide timing, behavior, and a string of the animation to play under a given context. Let's look at the Short Shotgun reload sequence:

```
"ReloadSequence": [
    {
        "TriggerTime": 0.0,
        "Type": 0,
        "StringData": "ShotgunReload_bump"
    },
    {
        "TriggerTime": 0.0,
        "Type": 5,
        "StringData": "carbine_reload_3_foley"
    },
    {
        "TriggerTime": 0.95,
        "Type": 5,
        "StringData": "shotgun_reload_1_shell_in"
    },
    {
        "TriggerTime": 1.0,
        "Type": 6
    },
    {
        "TriggerTime": 1.5,
        "Type": 5,
        "StringData": "shotgun_reload_2_cock"
    },
    {
        "TriggerTime": 2.0,
        "Type": 7
    }
],
```

`"TriggerTime": 0.0,` The delay, from the start of the sequence, to play the clip.

`"Type": 5,` The type, which determines the behavior of the animation.

`"StringData": "shotgun_reload_1_shell_in"` The string of the animation clip.

Note: This formatting and behavior is consistent between reload and firing sequence.

Gear Datablocks: Animation Types

0	WeaponMovementAnim	Animate the first person weapon holder
1	FrontPartAnim	Animate the weapon’s front part
2	LeftHandAnim	Animate the FPS arms (animation layer 5)
3	LeftHandMagAnim	Animate the FPS arms using the magazine’s animation. *No animation clip input required*
4	ReceiverAnim	Animate the weapon’s receiver part
5	Sound	Play a sound event using the animation string as a sound event ID
6	DoUpdateAmmo	Reload the weapon’s clip. *No animation clip input required*
7	Empty	Don’t do anything. Typically used at the very end of a sequence *No animation clip input required*
8	StockAnim	Animate the weapon’s stock part
9	RightHandAnim	Animate the FPS arms (animation layer 6)

GearPartCustomization: Modifying gear part transforms

In the vanilla game, it's possible to use whatever parts you want on any gun, and to choose what aligns those parts snap to, but it is impossible to manually adjust the position, scale, or rotation of parts. To get around this limitation, we can use my 'GearPartCustomization' plugin, which allows us to get far more milage out of the vanilla weapon parts.

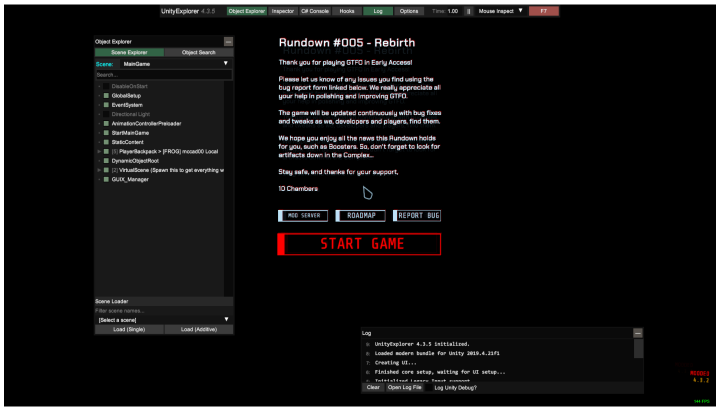
With the GearPartCustomization plugin installed, we'll get a new file in our custom folder titled GearPartTransform.json. Let's take a look at the default settings which generated in our file. Because the datablock is large, I'll break it down into individual properties.

`"OfflineID":` The persistentID of the PlayerOfflineGear entry to customize
`"Name":` The name of the customization settings. For organizational purposes only
`"InternalEnabled":` Determines whether or not to load this customization setting
`"Parts":` The list of parts on the gear item which we want to customize. Parts contains the following properties:

`"PartHolderObject":` The Property within the GearPartHolder to modify
`"PartType":` The Component Type to modify. Only used if PartHolderObject is unset.
`"Enabled":` Determines whether or not to enable or disable this part's Game Object
`"PartTransform":` Modify the position, scale, and rotation of our desired part.
`"Children":` The list of children on this part to modify. Many gear parts contain children for things like additional models, aligns, and animations. If you don't want to customize the model's children, leave this list empty. Children contains the following properties:

`"ChildName":` The name of the child object to modify.
`"Enabled":` Determines whether or not to enable or disable this child's Game Object
`"PartTransform":` Modify the position, scale, and rotation of the child part.
`"Children":` The list of children on this child object which we'd like to modify. Many parts have nested children multiple levels deep, so you need to modify the children of children to access those.

This might seem pretty daunting, as you'll be editing a potentially very large amount of data, but thanks to Unity Explorer, this process is heavily simplified and relatively easy.



Rundown #005 - Rebirth

Thank you for playing GTFO in Early Access!

Please let us know of any issues you find using the bug report form linked below. We really appreciate all your help in polishing and improving GTFO.

The game will be updated continuously with bug fixes and tweaks as we, developers and players, find them.

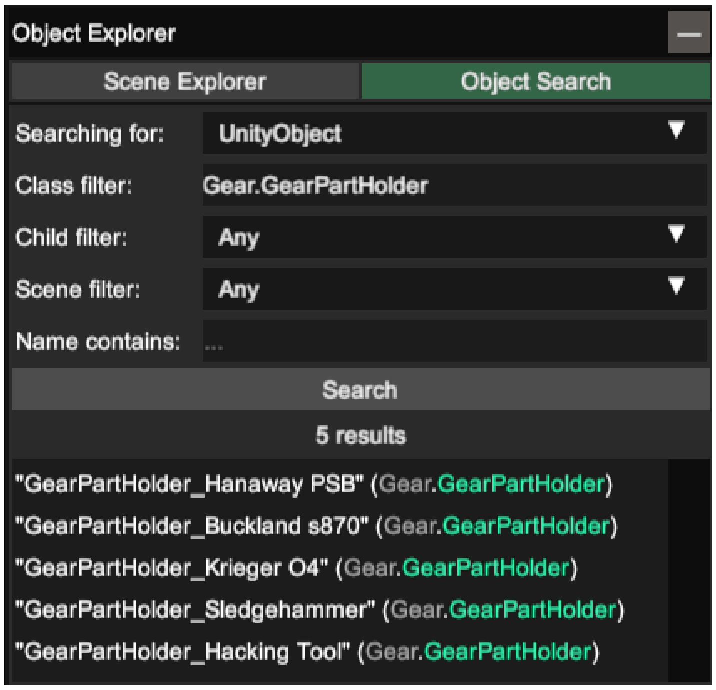
We hope you enjoy all the news this Rundown holds for you, such as Boosters. So, don't forget to look for artifacts down in the Complex...

Stay safe, and thanks for your support,

10 Chambers

```
Log
9: UnityExplorer 4.3.5 initialized.
8: Loaded modern bundle for Unity 2019.4.21f1
7: Creating UI...
6: Finished core setup, waiting for UI setup...
```

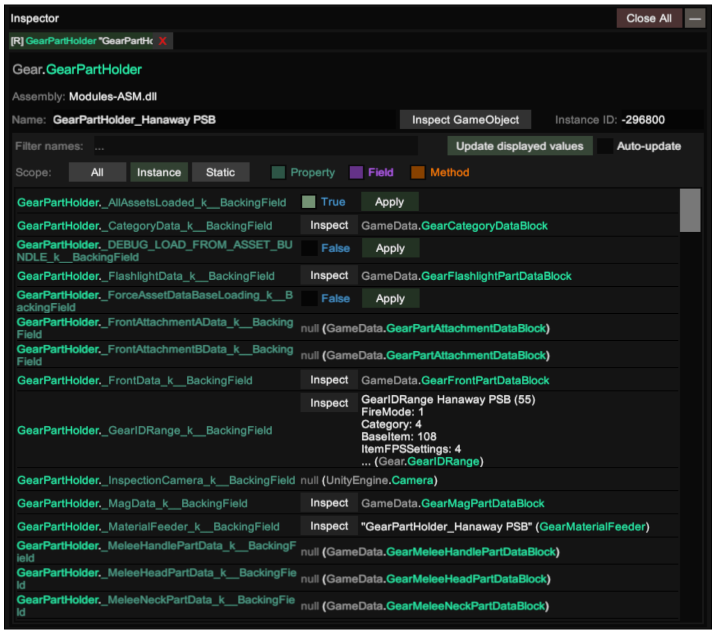
When we first boot up the game with unity explorer, we'll see something like this. For now, you can just press F7 to close the unity explorer menu and start the game. Pick a level, pick the gear you'd like to customize, and drop in.



Searching for:	UnityObject
Class filter:	Gear.GearPartHolder
Child filter:	Any
Scene filter:	Any
Name contains:	...

```
5 results
"GearPartHolder_Hanaway PSB" (Gear.GearPartHolder)
"GearPartHolder_Buckland s870" (Gear.GearPartHolder)
"GearPartHolder_Krieger O4" (Gear.GearPartHolder)
"GearPartHolder_Sledgehammer" (Gear.GearPartHolder)
"GearPartHolder_Hacking Tool" (Gear.GearPartHolder)
```

When we get into the level, open up Unity Explorer again by pressing F7, and go to the object explorer window. Select 'Object Search', set the scope of the search to UnityObject, the class filter to Gear.GearPartHolder, and click search.
In our search results, we'll find the gear part holder of everything we have equipped in our inventory. To continue, click on the GearPartHolder of the gear you want to customize. I'll be modifying the Double Tap Rifle



Gear.GearPartHolder

Assembly: Modules-ASM.dll

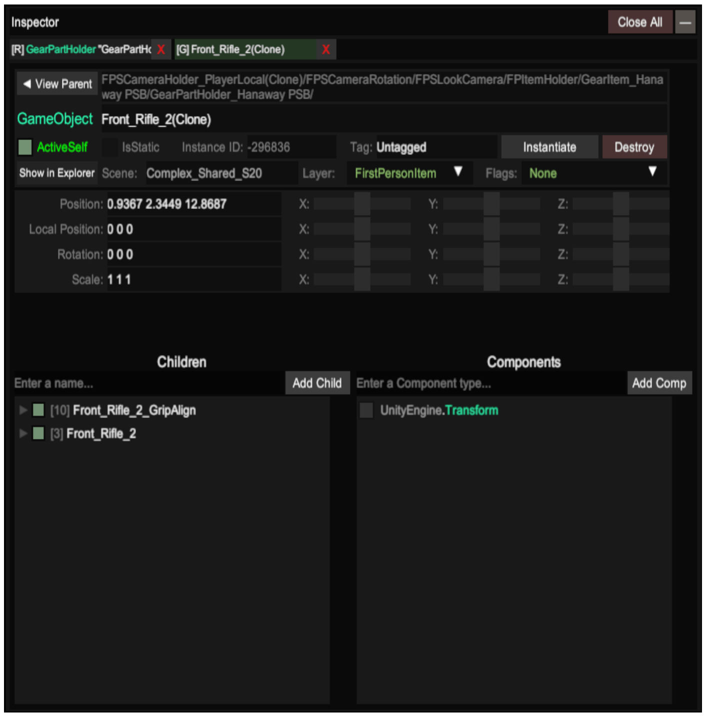
Name: GearPartHolder_Hanaway PSB   Instance ID: -296800

Filter names: ...

GearPartHolder._AllAssetsLoaded_k__BackingField	True	Apply
GearPartHolder._CategoryData_k__BackingField	Inspect	GameData.GearCategoryDataBlock
GearPartHolder._DEBUG_LOAD_FROM_ASSET_BUNDLE_k__BackingField	False	Apply
GearPartHolder._FlashlightData_k__BackingField	Inspect	GameData.GearFlashlightPartDataBlock
GearPartHolder._ForceAssetDataBaseLoading_k__BackingField	False	Apply
GearPartHolder._FrontAttachmentAData_k__BackingField		null (GameData.GearPartAttachmentDataBlock)
GearPartHolder._FrontAttachmentBData_k__BackingField		null (GameData.GearPartAttachmentDataBlock)
GearPartHolder._FrontData_k__BackingField	Inspect	GameData.GearFrontPartDataBlock
GearPartHolder._GearIDRange_k__BackingField	Inspect	GearIDRange Hanaway PSB (55) FireMode: 1 Category: 4 BaseItem: 108 ItemFPSSettings: 4 ... (Gear.GearIDRange)
GearPartHolder._InspectionCamera_k__BackingField		null (UnityEngine.Camera)
GearPartHolder._MagData_k__BackingField	Inspect	GameData.GearMagPartDataBlock
GearPartHolder._MaterialFeeder_k__BackingField	Inspect	"GearPartHolder_Hanaway PSB" (GearMaterialFeeder)
GearPartHolder._MeleeHandlePartData_k__BackingField		null (GameData.GearMeleeHandlePartDataBlock)
GearPartHolder._MeleeHeadPartData_k__BackingField		null (GameData.GearMeleeHeadPartDataBlock)
GearPartHolder._MeleeNeckPartData_k__BackingField		null (GameData.GearMeleeNeckPartDataBlock)

With the gear part holder open, we'll be able to see all of the properties and fields of the gear part holder. We can also click Inspect GameObject to view the Game Object itself, where we can move, rotate, and scale the gun to make it more easy to view while editing.

While there are tons of fields here, very few of them are actually meaningful to us. Let's inspect GearPartHolder.FrontPart:



```
Inspector                                                        Close All
[R] GearPartHolder "GearPartHc X   [G] Front_Rifle_2(Clone) X

◄ View Parent  FPSCameraHolder_PlayerLocal(Clone)/FPSCameraRotation/FPSLookCamera/FPItemHolder/GearItem_Hana
               way PSB/GearPartHolder_Hanaway PSB/
GameObject  Front_Rifle_2(Clone)
ActiveSelf   IsStatic   Instance ID: -296836   Tag: Untagged   Instantiate   Destroy
Show in Explorer  Scene: Complex_Shared_S20   Layer: FirstPersonItem ▼   Flags: None ▼

      Position: 0.9367 2.3449 12.8687   X:   Y:   Z:
Local Position: 0 0 0                   X:   Y:   Z:
      Rotation: 0 0 0                   X:   Y:   Z:
         Scale: 1 1 1                   X:   Y:   Z:

              Children                              Components
Enter a name...              Add Child   Enter a Component type...    Add Comp
► [10] Front_Rifle_2_GripAlign           UnityEngine.Transform
► [3] Front_Rifle_2
```

We can now see the front part's position, local position, rotation, and scale, as well as the attached child objects. I highly recommend that you don't use the sliders, as at the scale of this game, even minute adjustments to the sliders can move parts a tremendous distance.
Note: Don't change the position, change the local position.

When we change these values here, we'll see that our weapon is modified in game as well. Once you're happy with how it looks, you can input these values into your GearPartTransform in the datablock.

ItemFPSSettings - First person weapon positioning

In order to modify the position of weapons in the first person camera, you need to either assign an existing ItemFPS (There are plenty of good ones in vanilla) or write a custom one. Let’s look at an example of the ItemFPSSettings datablock:

```
"localPosHip": The position of the weapon while hipfiring
"localRotHip":    The angle of the weapon while hipfiring
"localPosRelaxed": The position of the weapon while in the idle animation
"localRotRelaxed": The rotation of the weapon while in the idle animation
"localPosZoom": The position of the weapon when aiming down sights
"localRotZoom": The rotation of the weapon when aiming down sights

"bodyOffsetLocal": The root position; other positions are offset from this
"bodyRotationOffsetLocal": The root rotation

"ItemCameraFOVDefault": Requires further research
"ItemCameraFOVZoom":
"LookCameraFOVZoom":

"canAim": Whether or not this weapon can be aimed with
"onlyStartAimOnPressed": Unused?
"canRelax": Whether or not the weapon's idle animation is used
"customDelayUntilRelax": The time until the idle animation begins
"allowRotToAimPos": Can the weapon can rotate into the zoom position
"rotToAimPosMinDis": Too confusing for my tiny brain to comprehend
"transitionToAim": 0 =  quick, 1 = slow

"idleAnimData": The animation and speed for standing still
"walkAnimData": The animation and speed for walking
"runAnimData": The animation and speed for running

"DofDefault": The depth of field settings (camera blur) for hipfiring
"DofAim": The depth of field settings (camera blur) for aiming down sights

"name": Internal name of these settings, unused ingame
"internalEnabled": Whether or not this setting is enabled ingame
"persistentID": The unique ID of this itemFPS setting
```