DRG Modding Guide

By Rauliken

(Red color means important, blue is a link inside the document, green is an external link)

(You can click blue text to go to that part, there's also blue links through the guide)

(You can also go back to the index by clicking "Back to Index" on the right when you see it)

INDEX

1. Introduction and tools that you need to use

(Back to Index)

There's still a lot of things to do in DRG even if you have all the cosmetics and promoted all characters to the max level. Welcome to DRG modding, you are going to like this game even more after learning how to mod it. If you find it too difficult just try the current mods that the modders have released and have some fun. I'm at 1k hours myself and I would have stopped playing if it wasn't for the amazing mods.

First you are gonna need a few tools and files to start modding. It's important to read all of them. You can always use the blue hyperlinks through the guide and in the index to quickly find what you need but I highly recommend just following the guide from top to bottom the first time.

A few VERY important clarifications before starting:

1. Your vanilla or modded save files are safe because mods will not corrupt/mess with it unless you use the tools maliciously. If you are concerned about this, make save backups or use the backup save backup button that the game provides. (files are in "your drg path\Deep Rock Galactic\FSD\Saved\SaveGames")
2. I highly recommend downloading the DRG Modding Tools zip with all the tools from our discord now so you can follow the guide with the tools next to you.

1. FSD-WindowsNoEditor.pak

-> This is the main file of the game, which contains all the rest of the files. It's "paked" with Unreal Engine so we need to extract the files if we want to modify them.

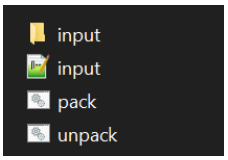
-> You can find it in "…\Program Files (x86)\Steam\steamapps\common\Deep Rock Galactic\FSD\Content\Paks". You obviously need the game installed and NEVER DELETE THIS FILE, just copy it if you need it later. If you don't know how to access this folder go to Steam, right click DRG > Properties > Local Files > Browse Local Files. If you want to make a lot of mods it is recommended to make a backup of this file or make a backup of the extracted files inside it when we get to that point.

-> The mods that you create need to have a certain name to be loaded by the game which should be the name that you want followed by "_P". When you start up the game, Unreal Engine will load the main file, and the extra .pak files with that suffix. You can also make subfolders inside "FSD->Mods" to organize the mods better.

2. DRGPacker

-> This tool will allow us to extract the files from FSD-WindowsNoEditor.pak and will also let us pack our own files. The files that you extract will have the extensions .uexp and .uasset which are Unreal Engine files. More on this on 2.1 Unpacking the game's files.

-> You can find it in the DRG Modding discord, you will need to tweak some file paths inside the .txt and .bat files before using. You will also need files from UEE.



3. HxD

-> This software is a hexadecimal number editor that we can use to see the contents of the files and change values. You are going to need this if you want to make some basic mods that change a value inside the game. More on this on 3.1 Hex mods.

-> You can find it here https://mh-nexus.de/en/hxd/.

4. EmptyContentHierarchy

-> These are all the folders inside FSD-WindowsNoEditor.pak but empty, just the folders. This will be useful later to pack files or to make audio and blueprint mods.

-> You can find it in the DRG Modding discord.

5. UModel

-> This tool will allow you to extract the audio files, textures, 3d models… This is different from DRG Packer because that one just extracts the .uexp and .uasset files but if you want to dive deeper and get the sounds, icons, 3d models… you are going to need this. If you just want to edit values with HxD there's no need for this tool.

-> You can find it here https://www.gildor.org/en/projects/umodel.

6. DRG Parser

-> This tool will give you a readable (not in hexadecimal) .json file from a pair of .uexp and .uasset files. Pretty useful if you want to change a value but don't know exactly what you are looking for. You can do everything without this, but it helps sometimes. However, there are some files that cannot be parsed correctly so we need to change the values of those blindly with HxD or learn blueprint modding.

-> You can find it in the DRG Modding discord.

(Back to Index)

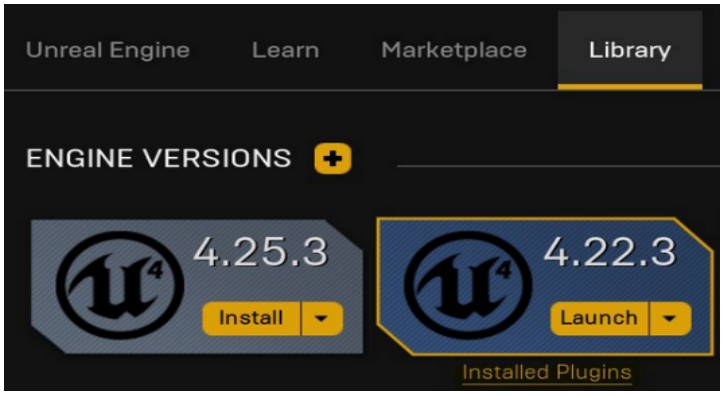
7. Unreal Engine Editor (UEE)

-> If you want to make mods to replace audio and textures you need this. You'll need it anyways to use DRG Packer. You can also create your own Unreal Engine blueprints (which are way more complex than hex editing mods or audio mods but can do more stuff).

-> You can find it here https://www.unrealengine.com/en-US/ . Unfortunately you need to download the Epic Games Store even though we just want Unreal Engine and not the games but there's also a github with a quick google search that tells you how to download it without the store.

-> You are going to need the version that DRG is made with, which is version 4.25. Get the latest subversion so 4.25.X, X being whatever is the max in there.

Go to Library, add a new version with the + symbol -> click the arrow next to install -> and the version. You only need the basic installation so you can configure it and remove the stuff that you don't need so the installation takes only like 12GB of storage instead of +20GB.

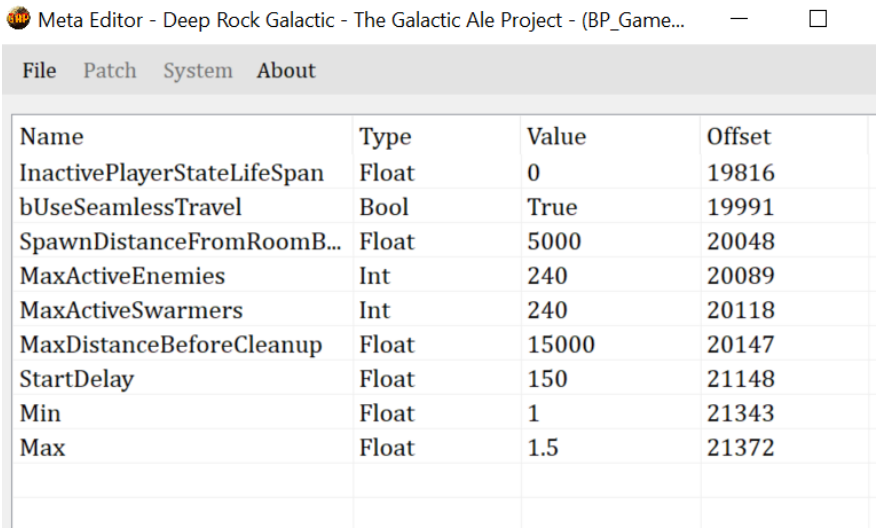


8. Other tools

(Back to Index)

-> Those were the main tools you need but there's al so some other useful ones like the DRGMetaEditor for easier value editing, save editors, console enabler and commands… in the DRG Modding discord.

I highly recommend trying at least the DRGMetaEditor, which will let you edit files much more easily than with the HxD editor. Warning: it doesn't replace it because it still can't edit values like array, which you can do with HxD.



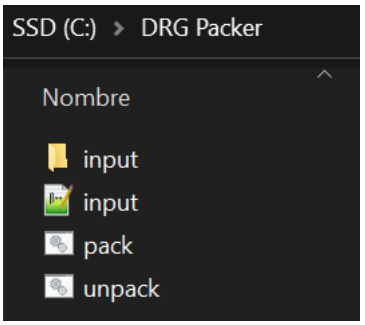
Name	Type	Value	Offset
InactivePlayerStateLifeSpan	Float	0	19816
bUseSeamlessTravel	Bool	True	19991
SpawnDistanceFromRoomB...	Float	5000	20048
MaxActiveEnemies	Int	240	20089
MaxActiveSwarmers	Int	240	20118
MaxDistanceBeforeCleanup	Float	15000	20147
StartDelay	Float	150	21148
Min	Float	1	21343
Max	Float	1.5	21372

2. Things to learn before modding

2.1 Introduction to DRG Packer

(Back to Index)

First you are going to need to download the zip with most of the tools in our discord but also Unreal Engine Editor because we are going to use one of the tools provided by it, called UnrealPak.exe, for DRG Packer to work.



This is our fresh DRG Packer folder, you are going to need to change a few file paths inside input, pack and unpack because it depends on how you have the files organized in your PC.

In input.txt you just need to change the first path between " " with the path to where you want the DRG Packer folder to be, can leave the same as default if you move the folder to drive C and name it DRGPacker.

input.txt

```
1  "C:\DRGpacker\input\" "../../../FSD/"
2
```

In unpack.bat you wanna change the first path between " " to your Unreal Engine Editor installation folder and then the UnrealPak.exe inside of it. You can see the default one is in Program Files (x86)->UE_4.25. Write the one where yours was installed.

unpack.bat

```
1  "C:\Program Files (x86)\UE_4.25\Engine\Binaries\Win64\UnrealPak.exe" %1 -extract C:\DRGpacker\unpacked
2  pause
```

In pack.bat you wanna change the first path between " " like before, to the UEE installation folder. The second and third path should should be the place where you want the output mod (.pak) to be, in this case the main folder of DRG Packer, and last path should be the one to the input.txt that we edited before, also in the main DRG Packer folder. You can see an example:

If you get any errors when packing related to "cannot find… whatever" then you wrote your own paths wrong. Follow the ones in the examples but adapt them to where you have the stuff in your computer. You can easily check paths if you click on the Windows textbox on top of the files:

pack.bat

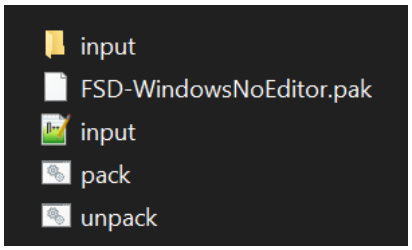
```
1  "C:\Program Files (x86)\UE_4.25\Engine\Binaries\Win64\UnrealPak.exe" "C:\DRGpacker\new_P.pak" -Create="C:\DRGpacker\input.txt" -co
2  pause
```

If the guide didn't help you can always check the video tutorial here:
https://www.youtube.com/watch?v=yfb4lJ0lYjY

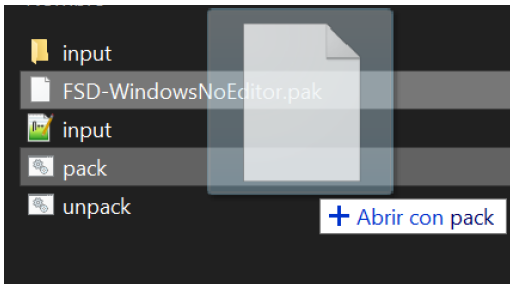
2.2 Unpacking the game's files

(Back to Index)

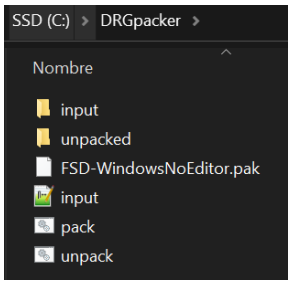
First of all we want to extract all the files from FSD-WindowsNoEditor.pak_ Start by going to your DRG Packer folder and copy and paste FSD-WindowsNoEditor.pak in the root directory. It should be like this when you are done:

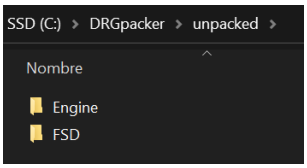


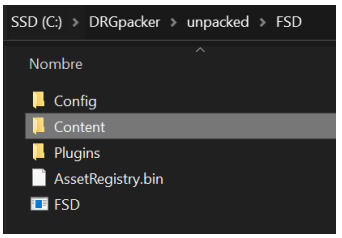
Now drag and drop FSD-WindowsNoEditor.pak into the .bat file "unpack".



Now wait for the extraction to complete. The command line window will say "Press any key to continue…" when it's done you will see a new folder called "unpacked". The files of the game are inside it:







We are not going to use the contents of the "Engine" folder, the other folder inside FSD or the .pak file again so you can delete those. Just leave FSD->Content, and it's contents.

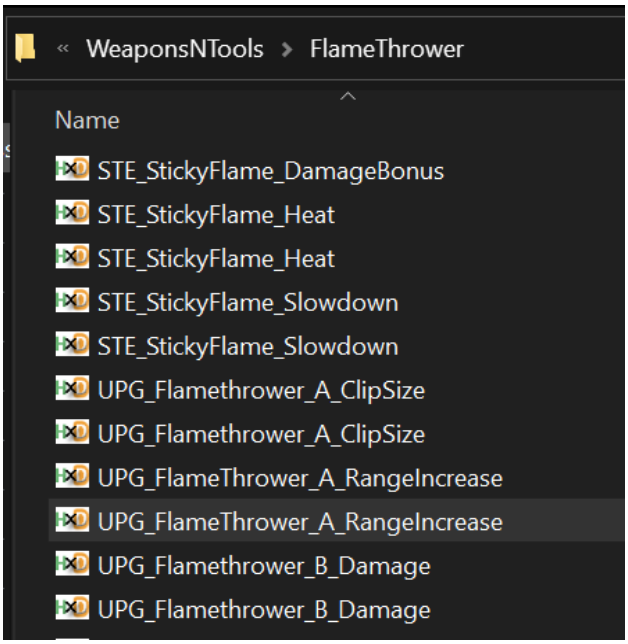
You should probably make a backup of FSD with all the extracted files.

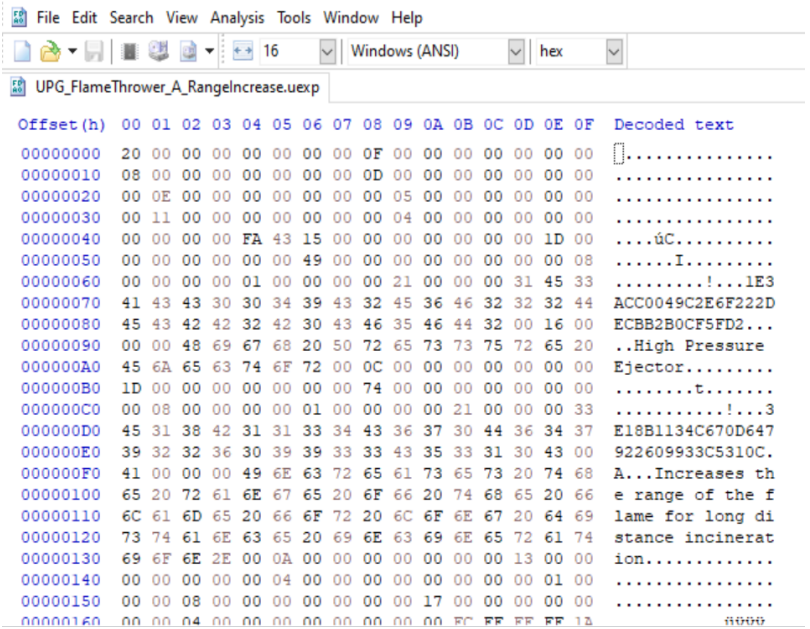
2.3 Checking the content of the files

(Back to Index)

If you go to the recently created “FSD” folder and then inside “Content” you will find the extracted folders. All of these contain the pairs of .uexp and .uasset files that we are going to need.

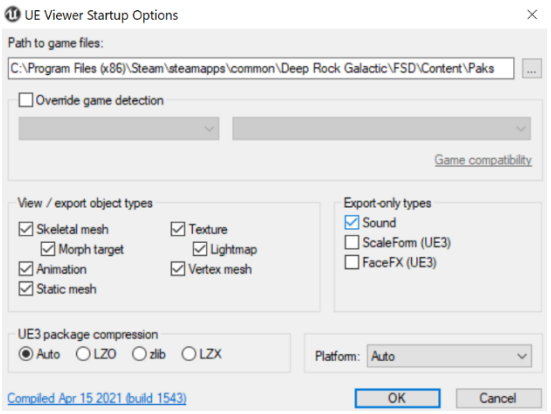
For example, if we go to “FSD\Content\WeaponsNTools\FlameThrower” we can see all the files for the flamethrower weapon. For hex editing values like upgrades or base stats we can open some of the .uexp files with HxD to see the contents in hexadecimal. (You can make HxD the default program so the .uexp files always open with it).



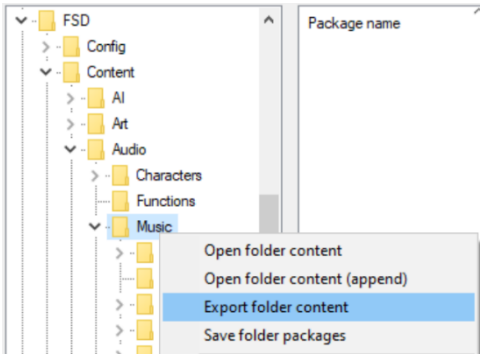


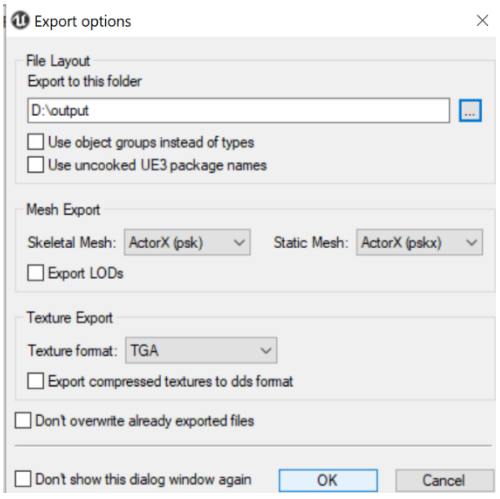
We are going to be working on this file later in 3.1 Hex mods to learn how to change the values and search for them in the screen full of numbers on the right side.

If you want to extract other things like sounds or textures you will need to use UModel and extract the desired file/s. For example, we open it to get one of Molly's sound files. You can select FSD-WindowsNoEditor.pak in your game folder or select the directory (my choice here) where you just extracted the .pak with DRG Packer, and we select these settings and OK:

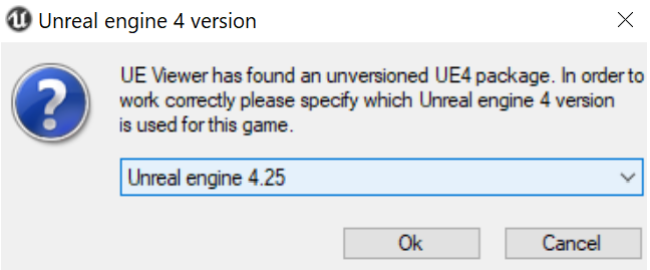


We navigate the folder tree until we find our file or a folder with all the files we need (you can extract the whole Audio folder for example if you want all the sounds in the game), and after selecting extract we select an output folder:





And then just select the Unreal Engine 4.25 when you get this pop up. If it's a sound, 3d model, texture file... you will get a few files in the extraction folder.



IMPORTANT: If you pick the version at the start with the "Override detection" option it will result in a 0/0. Don't use that and instead use the pop up above when extracting.

2.4 Packing your mod files

(Back to Index)

Using the same DRG Packer we used to unpack the main file, we are going to learn how to pack the files that we changed/added into a new .pak file.

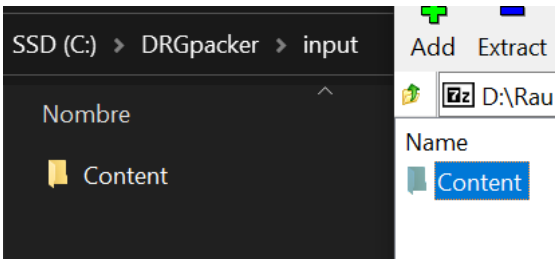
2.4.1 File packing method

(Back to Index)

I'm going to use the example from the flamethrower file before (imagine we already edited it even though that comes later in the tutorial). You should probably read about Empty Content Hierarchy if you haven't already.

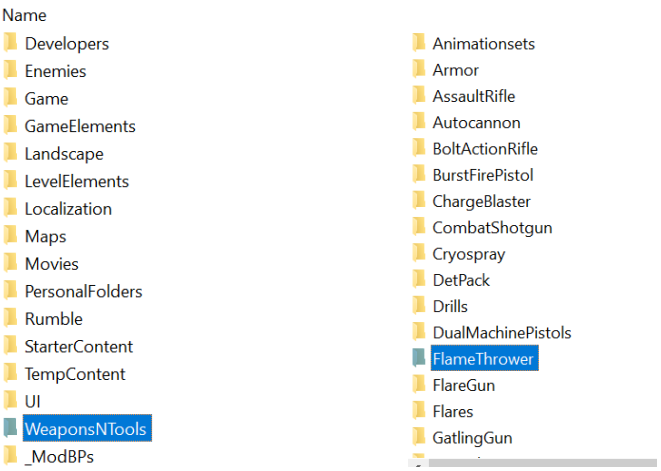
In the flamethrower example, the file is UPG_FlameThrower_A_RangeIncrease which is inside FSD\Content\WeaponsNTools\FlameThrower. To pack mods correctly we are going to need to replicate the same folder structure that we got on the output folder "unpacked" but inside the "input" folder.

You can just open the .zip from Empty Content Hierarchy (which has all the folders from the game but empty) and drop the "Content" folder in it, to "input" and replace the placeholder one there.

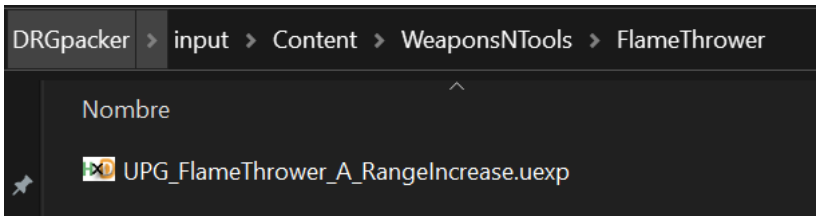


In this case we just want to work with 1 weapon so you don't need to have the rest of the folders. You can just drop the "WeaponsNTools" folder inside input -> Content.

Then we can delete everything but the FlameThrower folder too. I recommend having a backup of each folder structure for each mod you make, for easier packing, instead of just having all of the empty folders and having to search where you put the files.



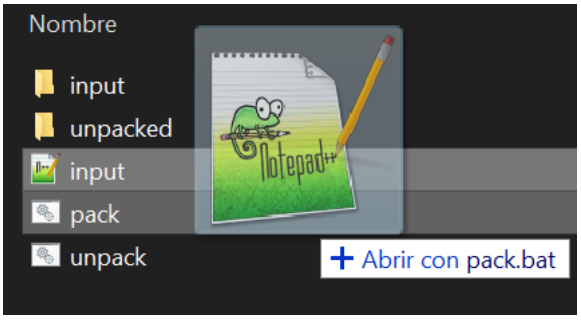
This will be the final path of the flamethrower file.



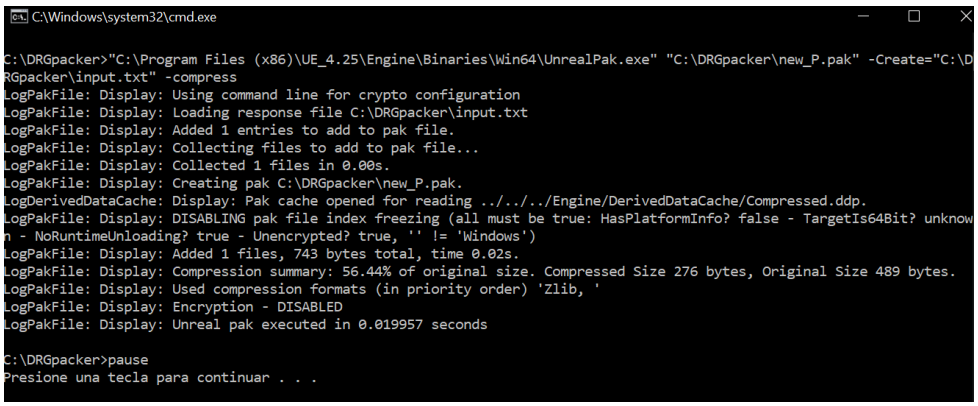
You can do the process before any number of files that you modded.

Be careful when leaving folders in input -> Content after packing because maybe you'll start making a new mod and forgot you still had the old mod files in there.

Now just drag and drop the .txt into pack. You will never need to change the contents of input, pack or unpack again after



You will get this window where you can see if there were any errors. Check the "Added x files" line. If you have for example 1 file in the mod check that the number is correct so it says "Added 1 files". If there's more or less than what you expect you probably forgot to put some files in the input folder.



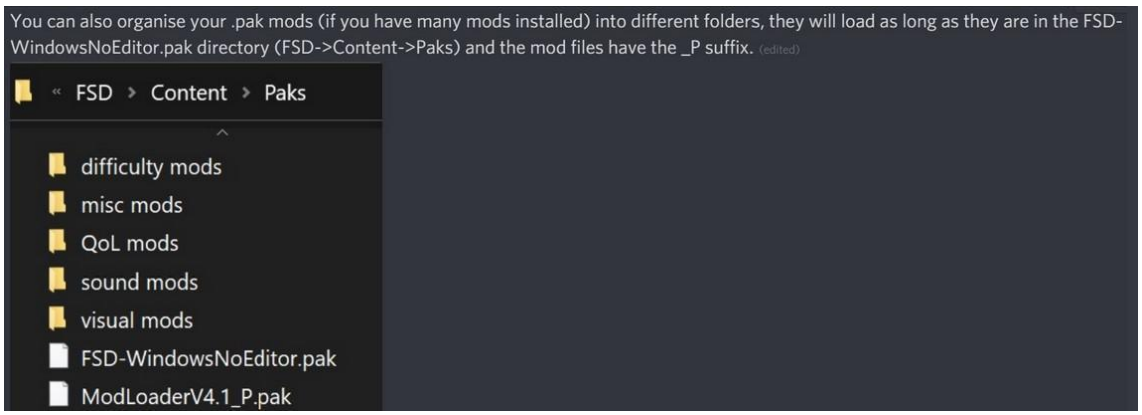
So, the summary of packing is that we only need to add or remove files inside the EmptyContentHierarchy (FSD and subfolders) with the correct folder structure compared to the vanilla files. It can get pretty messy if you forget to add/remove files when making new mods so, (again) I recommend having a backup of each folder structure for each mod you make, for better organization and future updating.

2.4.2 Mod installation

(Back to Index)

To install a mod, simply move the .pak than you made in the previous step (2.4.1) to"…\Steam\steamapps\common\Deep Rock Galactic\FSD\Content\Paks"

If you don't know how to open the game's folder follow this guide:
https://steamcommunity.com/sharedfiles/filedetails/?id=760447682



(Yes, installing the mod is just moving your mod folders in there. Unreal Engine automatically replaces the original file in FSD-WindowsNoEditor.pak when the game loads, with the file that you modified in your .pak).

3. How to mod

(Back to Index)

There's different types of mods that you can make for DRG. Hex mods (touch values and are easy to make if you know what you are looking for , there's also mods that change audio, textures… and blueprint mods to create Unreal Engine blueprint files with more complex behaviour.

3.1 Hex mods

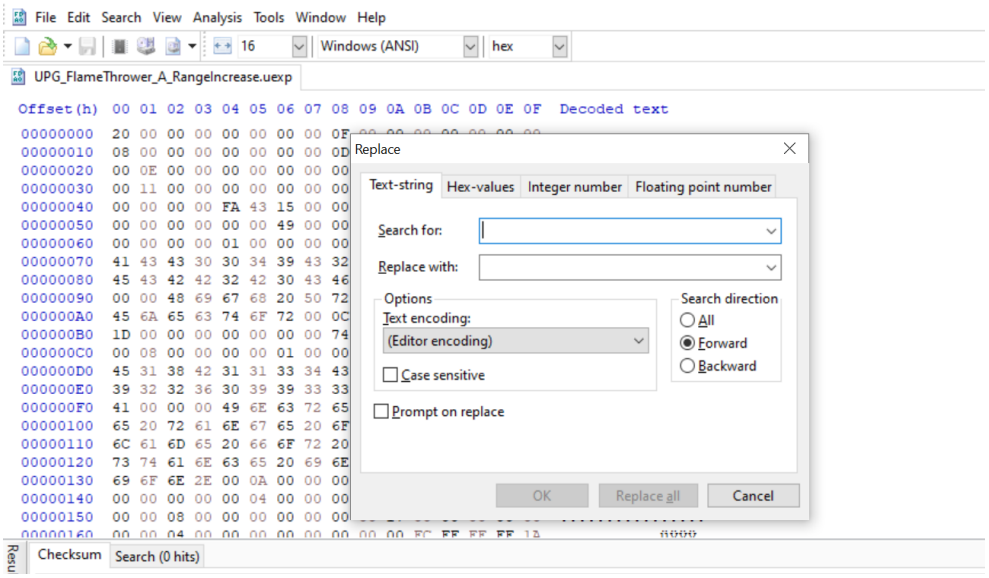
(Go to quick guide)

We have the hex mods, which edit values inside the .uexp files with HxD and replace the original file with your new one when the game loads. These values can be integers, text, booleans, floats… Most of the values should be easy to find just by using the armory inside DRG but you need to know a few things.

We need to know the value that we are going to change but also the file where it is located. You can check 4.1 Where's the file for X thing located? if it doesn't seem obvious.

I'm gonna be working with the file that I used in packing example before, in 2.4.1 File packing method. In this case our mod is going to have only one file which is called "UPG_FlameThrower_A_RangeIncrease.uexp" and as you can tell by the prefix "UPG" this is the file that controls the range upgrade for driller's flamethrower.

If we open the file with HxD and use CTRL+F or better CTRL+R (which allows us to search and replace a value we get this window:



You may think "Oh, I just checked the armory and the upgrade says +5 range so I will just search 5 in the .uexp file and replace it with whatever". NO, most values are easy to change but some follow certain rules so the value might not be what you think it is.

First of all, you should be searching with CTRL+F for an integer number (3rd tab) or a floating point number (4th tab). Most of them are usually floats but if you don't find the desired value, 2 things could be happening:

1. The value is not in that file.
2. The value is not the one that you think it was.

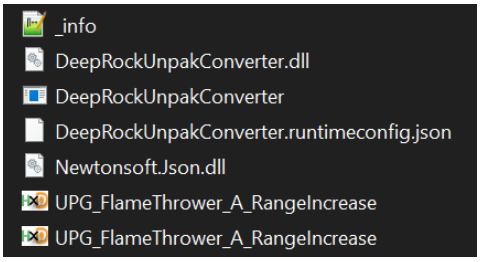
In this case if you search for "5" in "UPG_FlameThrower_A_RangeIncrease.uexp" in the integer tab you will get 1 result but even if you change it, you're game is probably going to crash or the mod will do nothing. Try floats first.

Did you search "5" as float and didn't find anything? We are pretty sure it should be here because the file is literally prefixed by "UPG_" like come on. Then the value is probably not 5 even though the armory inside the game says 5.

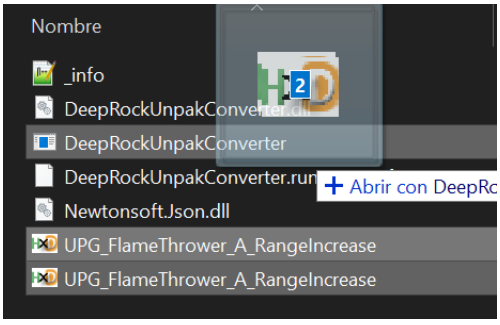
This is where the tool DRG Parser that I talked about in the intro comes to help. This tool will allow us to transform all the hex numbers that you see in HxD into a readable .json file. We don't always need (optional step) but it's pretty useful in some situations, like this, where we can't find a value. (I wasn't going to use an easy one for the example right? :P).

You need this (.NET Core 3.1) for the parser to work:
https://dotnet.microsoft.com/download/dotnet-core/thank-you/runtime-desktop-3.1.8-windows-x64-installer

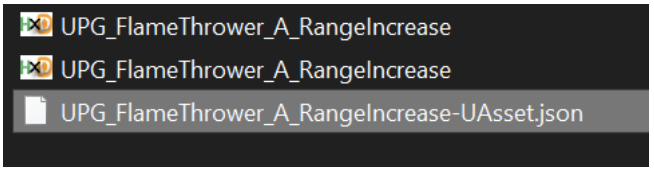
Once we have DRG Parser we need the both the .uexp and .uasset file of the thing we want to parse. We move all to a folder and then open the command line in windows.



Now we just drag both files into the .exe:



If there weren't any errors, you will get a .json file in the same folder that you can open with a .txt editor.



UPG_FlameThrower_A_RangeIncrease-UAsset.json


```
{
  "FileSummaryOffset": "0x0",
  "FileSummary": {
    "Tag": "0x9E2A83C1",
    "FileVersionUE4": 516,
    "FileVersionLicenseeUE4": 0,
    "CustomVersion": {
      "CustomVersion": null
    },
    "TotalHeaderSize": 1393,
    "PackageFlags": "PKG_RequiresLocalizationGather, PKG_FilterEditorOnly",
    "FolderName": "None",
    "NameCount": 33,
    "NameOffset": 193,
    "LocalizationId": null,
    "GatherableTextDataCount": 0,
    "GatherableTextDataOffset": 0,
    "ExportCount": 1,
    "ExportOffset": 1265,
    "ImportCount": 7,
    "ImportOffset": 1069,
    "DependsOffset": 1369,
    "SoftPackageReferencesCount": 0,
    "SoftPackageReferencesOffset": 0,
    "SearchableNamesOffset": 0,
    "ThumbnailTableOffset": 0,
    "Guid": {
      "GUID": "511125D7-455F3AAA-640BED9C-2AA652E1"
    },
    "Generations": [
```


Surprise! We transformed all those random numbers into something readable. Here we can search for values with CTRl+F or more importantly, search for strings like "range" so we can find the OFFSET or our value in HxD.

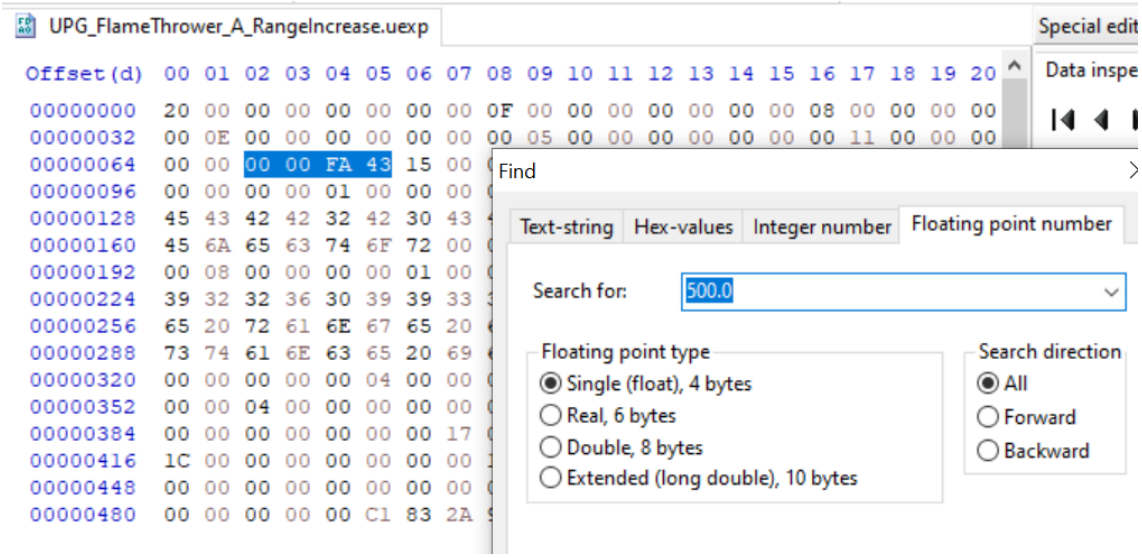
UPG_FlameThrower_A_RangeIncrease-UAsset.json


```
    "Outer": "UObject",
    "Template": "Default__FlameThrowerUpgrade",
    "Class": "FlameThrowerUpgrade",
    "Object": "UPG_FlameThrower_A_RangeIncrease",
    "File Offset": 0,
    "Struct Size": 485,
    "Potential Properties": [
      {
        "Property": "UpgradeType",
        "Type": "EnumProperty",
        "Offset": 0,
        "Size": 8,
        "Array Index": 0,
        "Tag Data": {
          "Name": "EFlameThrowerUpgradeType"
        },
        "Value Offset": 33,
        "Data Value": "EFlameThrowerUpgradeType::LongReach"
      },
      {
        "Property": "Amount",
        "Type": "FloatProperty",
        "Offset": 41,
        "Size": 4,
        "Array Index": 0,
        "Value Offset": 66,
        "Data Value": 500.0
      },
```


We now this is an upgrade and inside the properties we have one called "Amount", which inside has a data value (line 414) and a value offset.

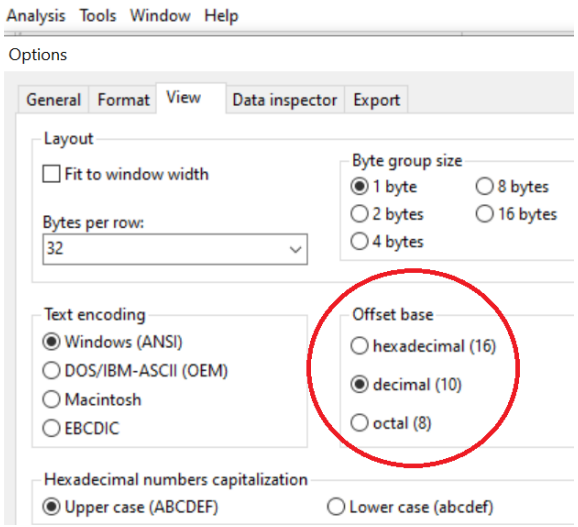
If you look at “Data Value:500.0” you will see that the value we were searching for is a float because it says “FloatProperty” above and that it was written in centimeters instead of meters so the +5 meter upgrade is coded as 500 centimeters in this upgrade file. (Some values are pretty easy to find with just the armory but as you just saw some of them need the parser or knowledge about how the developers code some values).

Now we search for 500 in the last tab inside HxD and we found it!



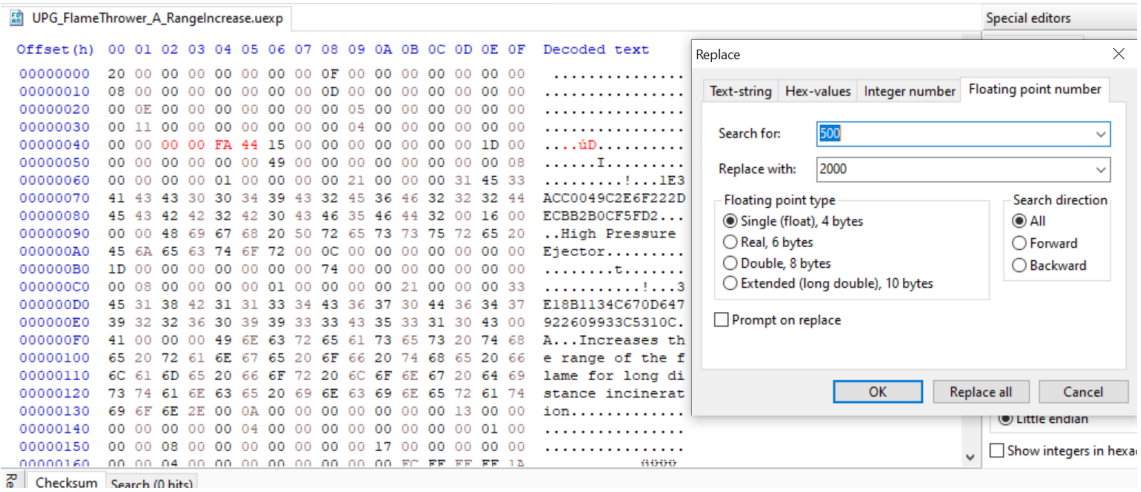
As you can see the offset is 66, this wasn’t very useful knowledge here. The offset will be really useful in files with repeated values, because we know the exact line where the value is. If you want to search by offset use CTRL+G.

You should change the offset base from hexa to decimal (because the parser uses decimal offsets) in Tools-> Options ->View.



Now that we are sure this file has the value, press CTRL+R so it’s find and replace instead of just find, and input the value that you want (don’t forget the Search

direction on “All” if you moved the writing bar/cursor “|” inside the fie). When you



press OK, it will turn red:

Finally, we just save the file with CTRL+S or just File->Save and we are ready to pack our mod. You should know now how to pack that file with one of the methods that I showed you earlier in 2.4 Packing your mod files. I used this exact file in both examples to make it easier to understand the whole process.

After packing, just follow 2.4.2 Mod installation and you are ready to try your new flamethrower upgrade that will add +20 meters instead of +5. As you can see in the armory when I hover over the range upgrade it now gives us 20 extra range. You can try the range OC too for some insane flame reach :P

A few important clarifications about HEX MODS:

1. If you don't want to hex edit simple values all the time because it can be a chore use the DRGMetaEditor to easily edit the values (it has an interface+in-built parser). This won't work for now with arrays or other complex values so this hex modding guide is still relevant.
2. If your game crashes, don't send the crash report to the devs. You probably changed the wrong value. Also NEVER change file size, if HxD says that you are changing file size, go back. You will always crash if you change the size.
3. Most but not all mods made this way are client side, the meaning of this is that unless the host has the mod, or you are the host, it won't work. It's also like this for other mods like the difficulty mods (haz 6 etc) and blueprint mods. Audio mods are always client side.
4. As I explained in the packing tutorial, you can have more than 1 file in a mod. You don't need to make 10 .pak files if you modify 10 files, just make a single one by adding more files with the correct paths to the input->Content folder.
5. You can change a lot of things with hex editing, not only the weapon's stats. If for example you change a perk, you will need to try the changes inside a mission to make sure they work because those don't show up in the armory.
6. You can use the quick guide if you need to remember one of the steps but don't want to read the whole detailed guide again.

3.2 Other mods (Audio, textures…)

(Back to Index)

The old guide was outdated and now there's a new better guide made by the user Buckminsterfullerene#6666 in our modding discord (if you have any questions ask him). Don't forget to download the pdf because you can't use the links inside if you view it through the browser. You can find the latest pdf in his github:

https://github.com/Buckminsterfullerene02/DRG-Modding/blob/main/DRG%20Mods%20-%20A%20Guide%20To%20Audio%20Modding.pdf

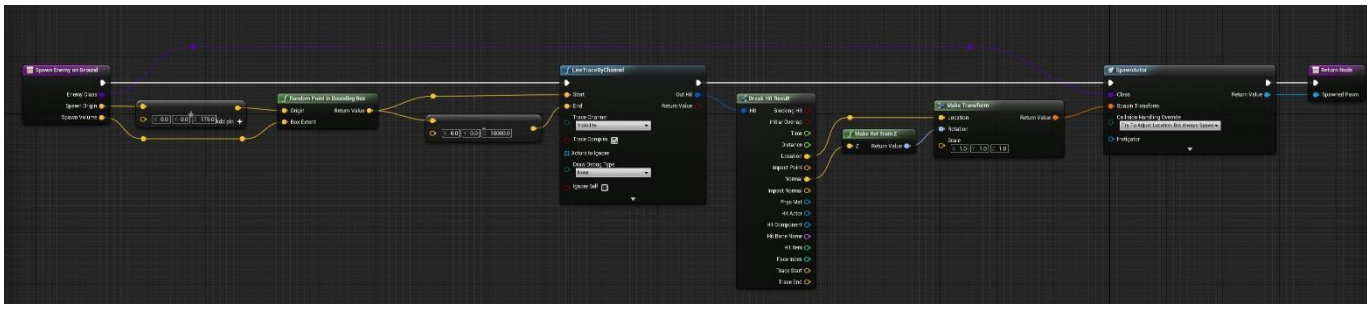
3.3 Blueprint mods

(Back to Index)

Alright this is the big boss. If you followed 3.2, you should know how to use basic stuff from UEE like the Content Browser and how to package projects.

The Blueprints Visual Scripting system in Unreal Engine is a gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor. Basically, you don't need to use a programming language.

For example, this is how it looks like to spawn an enemy:



The user Buckminsterfullerene#6666 in our modding discord has made a great guide to introduce you to these mods. Don't forget to download the pdf because you can't use the links inside if you view it through the browser. You can find the latest pdf in his github:

https://github.com/Buckminsterfullerene02/DRG-Modding/tree/main/BP%20Modding%20Guide

You can also find more info in ArcticEcho's github where he has the mod loader, used to load blueprint mods into the game.

https://github.com/ArcticEcho/DRG-Mod-Loader

If you need further assistance there's also a ton of info in the #mod-chat channel in our discord and there's even a channel where people stream themselves making blueprint mods sometimes. There's also the Unreal Engine documentation and some blueprint tutorials that you can find online.

This guide still has some extra things below and quick guides step by step for hex and replacement mods and a huge amount of specific modding guides.

3.4 Quick guides (summary)

(Back to Index)

Here's a quick guide of all the steps for HEX MODS: (Back to long guide)

1. Extract .pak files with DRG Packer by dragging the .pak into "unpack".

2. (Optional) Parse a pair of .uasset and .uexp with DRG Parser if you want more info about the value you want to change.

3. Open the .uexp file you want to change with a HxD and edit (find and replace option) the values you want. Never change the file size!

4. Now to turn it back into .pak again follow these steps: (an example for the barrel kick force mod that modifies only PST_BarrelKicking.uexp):

    4.1 Create the same folder structure inside DRGpacker -> input -> Content as the original extracted files had and add the modified files, so:

    ……/GameElements/PawnStats/PST_BarrelKicking.uexp

    4.2 Drag and drop the .txt to "pack" and check that there are no errors.

5. Put the generated new_P.pak file in:

    …\Steam\steamapps\common\DeepRockGalactic\FSD\Content\Paks

6. Rename it to whatever you want so: "*InsertNameHere*_P.pak" Keep the "_P"!

Here's a quick guide of all the steps for AUDIO MODS: (Back to long guide)

1. Make an empty project in Unreal Engine Editor 4.25.3

2. Add the folder structure just for the audio file (Cue) you want to change or for the whole game with EmptyContentHierarchy (you can check where the audio file is with UModel or an extracted folder with all the game files).

3. Make a new Cue file in your folder structure (same folder as the original Cue you want to change) with the same name (without uexp or uasset extensions).

4. Use the blueprint system from unreal to modify the Cue file to your liking with sounds of the game or custom sounds that you add to your unreal project, you also need to add the sound files that you want to the project.

5. Go to File > Package Project > Packaging Settings and uncheck "Use pak file".

6. Cook your project (File > Package project > Windows).

7. Now create a new pack using the .uassets and .uexp files that you got from the cooked project inside "Conten", using 2.4.1 File packing method.

4.Extra guides made by other modders

4.1 DRG Modding Discord

(Back to Index)

This is where we have our modding community. We post mods, previews, guides, chat about the game and have a good time.

Here's the invite link: https://discord.gg/p4UGSnU

Here's the git where the mods are hosted: https://github.com/ArcticEcho/DRG-Mods

4.2 Inventory and Property Swapping Guide - by Chibba

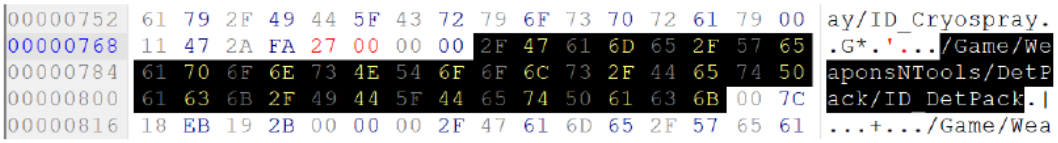
(Back to Index)

By Chibba

(Green color means technical terms, orange is DRG (or Unreal Engine) specific and red is important)

String Swapping can be used in many places but for this guide I will focus on InventoryLists (changing the class loadouts i.e. their weapons and tools, or even armor and grenades).

The files will be located in Character\InventoryLists and there is one for every class. We only need the *.uasset file to work with.



In there we need to find the strings we want to replace. For InventoryLists every component is referenced in **two** different places. First the path to the component as seen in the above image (paths are best found in the json of the class file you want to change, but I also added a list of all weapons and tools at the end of the guide).

Note that there is also a string-length for every string (the two red bytes in the image), this number tells the game how long the following string will be and needs to be adjusted in case it is changed.

Every path is enclosed with **00** (dot terminator in hex) which act as a sort of boundary and are necessary (so don't overwrite them, always be aware of the available string-length).

Now if we want to change the path (black in the image) we click in the text field to the **right** of every editor and just type over the existing text (bytes). If the new string is shorter than the original one we can click in the byte area on the **left** (where all the hex numbers are) and press 0 twice. These **00** will show up as dots in the text area, repeat until all bytes up to the dot terminator are doted.

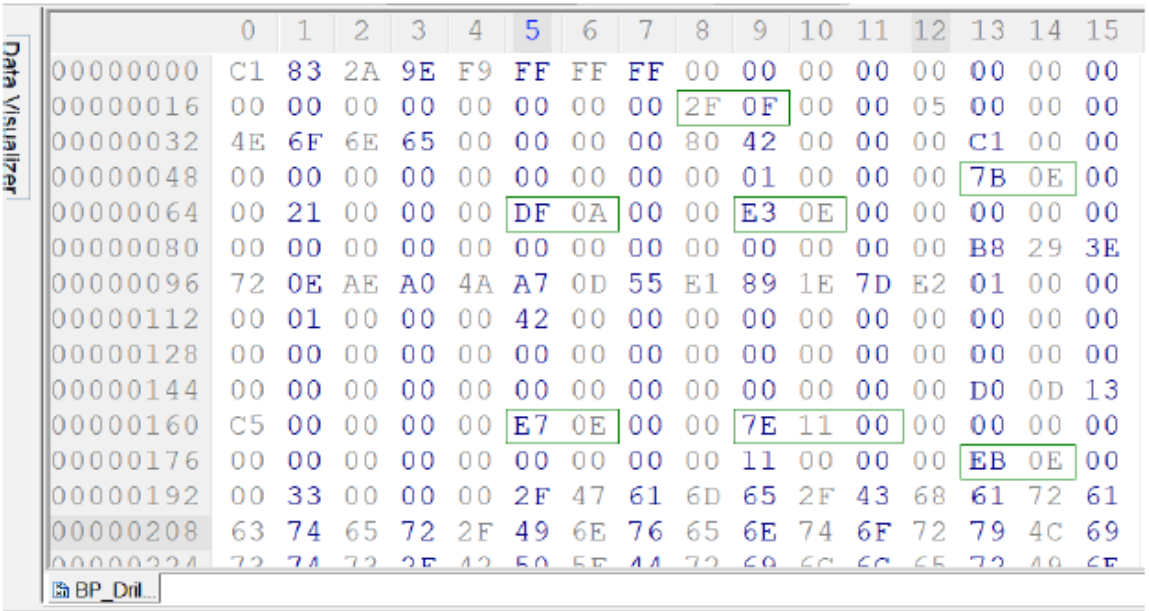
The second place we need to change is the ID which appears in a different location in the file but follows the exact same structure as the path.

Since we can't pass the dot terminator we are limited by the string-length of the original string which is a huge restriction for the components we can swap in.

--

To remove that restriction **oscateexor**#2362 found a method to add bytes to the *.uasset file and therefore allow longer strings (all components) to be swapped in.

First determine how many bytes you need. If your path needs three more characters (one byte per character) and your ID needs two more characters then your file will have to be expanded with five more bytes. Meaning all offsets (see below) need to be increased by 5.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	C1	83	2A	9E	F9	FF	FF	FF	00	00	00	00	00	00	00	00
00000016	00	00	00	00	00	00	00	00	2F	0F	00	00	05	00	00	00
00000032	4E	6F	6E	65	00	00	00	00	80	42	00	00	00	C1	00	00
00000048	00	00	00	00	00	00	00	00	00	01	00	00	00	7B	0E	00
00000064	00	21	00	00	00	DF	0A	00	00	E3	0E	00	00	00	00	00
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	B8	29	3E
00000096	72	0E	AE	A0	4A	A7	0D	55	E1	89	1E	7D	E2	01	00	00
00000112	00	01	00	00	00	42	00	00	00	00	00	00	00	00	00	00
00000128	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000144	00	00	00	00	00	00	00	00	00	00	00	00	00	D0	0D	13
00000160	C5	00	00	00	00	E7	0E	00	00	7E	11	00	00	00	00	00
00000176	00	00	00	00	00	00	00	00	00	11	00	00	00	EB	0E	00
00000192	00	33	00	00	00	2F	47	61	6D	65	2F	43	68	61	72	61
00000208	63	74	65	72	2F	49	6E	76	65	6E	74	6F	72	79	4C	69

Bookmarks

Address	Length	Length	Description
00000024	2	02	TotalHeaderSize
00000061	2	02	ExportOffset
00000069	2	02	ImportOffset
00000073	2	02	DependsOffset
00000165	2	02	AssetRegistryDataOffset
00000169	3	03	BulkDataStartOffset
00000189	2	02	PreloadDependencyOffset

There are eight offsets (here does not mean position in the file but rather offsets = values the game uses) that need to be adjusted, seven of those can be seen in the above image. Their names can also be found in the json file, but they are always on the same offset (position in the file) so looking them up in the json file is optional.

The last offset is the SerialOffset (not in the image) which holds the same value as the TotalHeaderSize = the filesize of the *.uasset in bytes. The offset for the SerialOffset changes depending on the file and needs to found with ctrl f (search).

In most parsed files line 32 holds the ExportCount which tells us how many SerialOffset need to be changed to expand the file. For InventoryLists (the example in this guide) ExportCount is 1.

Other files (for example WPN files) usually have between 30 and 90, so expanding one of those takes some effort. But it enables great things like for example changing projectiles or damage types.

To sum it up when expanding a file and creating new paths and IDs we need to:

1. Figure out the number of new bytes we need
2. Change all necessary offsets
3. Adjust the string-length number before the string
4. Insert the bytes (in HxD: Edit -> Insert Bytes)
5. Create the new path and ID or whatever new string

Driller:

FlameThrower/ID_FlameThrower (15) [27]

Cryospray/ID_Cryospray (12) [21]

Pistol/ID_Pistol (9) [15]

ChargeBlaster/ID_ChargeBlaster (16) [29]

Drills/ID_DoubleDrills (15) [21]

DetPack/ID_DetPack (10) [17]

Engineer:

CombatShotgun/ID_CombatShotgun (16) [29]

SMG/ID_SMG (6) [9]

GrenadeLauncher/ID_GrenadeLauncher (18) [33]

LineCutter/ID_LineCutter (13) [23]

PlatformGun/ID_PlatformGun (14) [25]

SentryGun/SentryGun_Engineer/ID_SentryGun (12) [39]

Gunner:

GatlingGun/ID_Gatling (10) [20]

Autocannon/ID_Autocannon (13) [23]

Revolver/ID_Revolver (11) [19]

BurstFirePistol/ID_BurstPistol (14) [29]

ZipLineGun/ID_ZiplineGun (13) [23]

ShieldGenerator/ID_ShieldGenerator (18) [33]

Scout:

AssaultRifle/ID_AssaultRifle (15) [27]

BoltActionRifle/ID_M1000 (8) [23]

SawedOffShotgun/ID_SawedOffShotgun (18) [33]

DualMachinePistols/ID_DualMPs (10) [28]

GrapplingGun/ID_GrapplingGun (15) [27]

FlareGun/ID_FlareGun (11) [19]

Here () is the length of the ID and [] the length of the path. If both are >= than the target component numbers then it can be swapped to without adding new bytes.

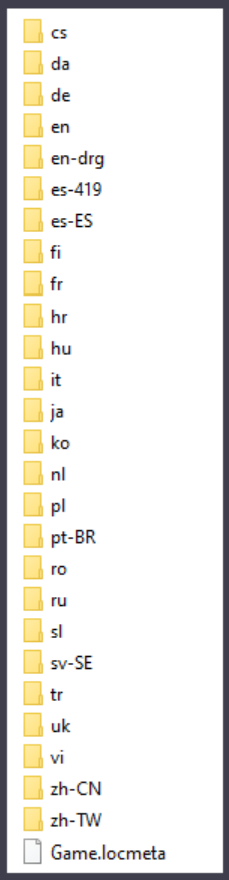
4.3 String Replacement - by Bebe[ru]

(Back to Index)

Replacing strings

Guide for making custom text in-game

Some theory first. This method works due to the fact that DRG has different translations, and as a result it has to make strings more customizeable. It also does have multiple localizations so in theory there should be less conflicts, but that's not the point.
We're going to replace these localization packs with our own ones.



In content hierarchy localization files are located at /content/localization/game. Inside these folders there are .locres files. They'll be our target today. Especially en one.
We'll be using locres.zip that's in modding-tools channel. Unpack it and use export exe file onto .locres
You'll get nice strings to change, and you're allowed to do anything with these as long as you don't change file structure (e.g. place an enter somewhere)
To convert it back, use import. Now you can create .pak file with it just like with any other mod!

There are some issues with this method:

a) Some strings can't be changed (You can check it via Ctrl+O in game)
b) Installing several mods replacing same .locres will most likely result in crashes or unintended behaviour, so make a note if you do change a .locres in your mod.
c) The way of .locres interaction with crowdin is unknown, so different localizations may have different structures so you can't just replace one localization with another

Oh, about crowdin. Crowdin is a platform DRG is translated on, any translateable string can be found here. You're free to explore it as long as you don't write in your own translations (Trust me, translators hate it when a random guy joins in, creates mess and disappears to never be seen again)
Sometimes we get leaks before official leaking there

In order to check if the string you want to change is changeable at all, you can use crowdin search to search for them or... You can use that strange in-game proofreader mode you can enable in settings. Ctrl + O shows you code for each string, which is used in crowdin to search for originals.
If a string doesn't appear as #12345, that means it's not translateable and can't be changed.

Here are some examples of these unchangeable strings:

#12527

#12529 Vietnamese #10394

#04759
Armor Paintjob : #09942

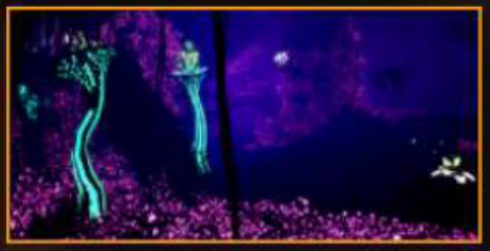
The Azure Weald is a chilly place, every bit as hostile as the rest of Hoxxes IV - but still, we cannot deny it is oddly beautiful. Dazzling, bioluminescent clusters of lichen light up the darkness, illuminating this fertile and overgrown Region in flickering neon colors. Don't let that lull you into a false sense of security however - the Azure Weald brings to mind the deep jungle, in all its splendor and horror - and much like its terrestrial counterparts, it is absolutely not out to make friends. Kill or be killed - that is the law in this place.

#12455

The alluring Mobula Cave Angel is a non-hostile inhabitant of the Azure Weald, and feeds exclusively on the bioluminescent algae so common in the local flora.

#12450

The magnificent wooden structures of the Hollow Bough really puts the biological oddity that is Hoxxes on display. It is estimated the weight of even one is in the billions of tons.



BIOLUMINESCENT FAUNA

While "glowing stuff" is hardly unique to the Azure Weald, the sheer proliferation of it in this region continues to mystify the Research Department.

#12692

To the best of our knowledge, the invasive red vines seen in the Hollow Bough is a recent development. Competing species of mega-flora? Seems to fit the general gist of Hoxxes pretty well.

4.4 Windows Store Modding - by Smelly Gritz

(Back to Index)

MOD Win10 Version of Unreal Engine Games

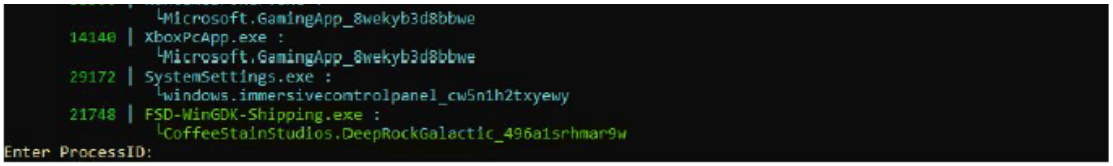
Uninstalls default version of Deep Rock (where XBOX Game Pass installed it for you)
Get-AppxPackage | where{$_.packagefullname -like '*deeprock*'} | Remove-AppxPackage

Registers the new appx package that we just created in new install directory
cd $New_Inst_dir
Add-AppxPackage -Register appxmanifest.xml

This is what it should look like in Powershell

```
# The new location where Deep Rock will be installed. You can set this to whatever you want
$New_Inst_dir = 'D:\Games\DRG_Win10'

# File DUMP location from step 5. Make sure you edit this to match the correct path
$DUMP_location = 'C:\Users\(Username)\AppData\Local\Packages\(Package Family Name)\TempState\DUMP'

# This creates new installation directory
mkdir $New_Inst_dir

# Copies all files from dump location to new install location
cd $DUMP_location
xcopy $DUMP_location $New_Inst_dir /E /T
xcopy $DUMP_location $New_Inst_dir /R /H /E /G

# Uninstalls default version of Deep Rock (where XBOX Game Pass installed it for you)
Get-AppxPackage | where{$_.packagefullname -like '*deeprock*'} | Remove-AppxPackage

# Registers the new appx package that we just created in new install directory
cd $New_Inst_dir
Add-AppxPackage -Register appxmanifest.xml

```

7. Game should now be fully installed in the $New_Inst_Dir as annotated in Powershell. Mod as you would the steam files.



5. It will extract the contents of those files to:
'C:\Users\(Username)\AppData\Local\Packages\(Package Family Name)\TempState\DUMP'
It should open this location automatically after extraction
6. Now open PowerShell ISE as admin and copy the following into the script pane. You will have to edit the file paths to match what yours are. Then run with (F5):

The new location where Deep Rock will be installed. You can set this to whatever you want
$New_Inst_dir = 'D:\Games\DRG_Win10'

File DUMP location from step 5. Make sure you edit this to match the correct path
$DUMP_location = 'C:\Users\(Username)\AppData\Local\Packages\(Package Family Name)\TempState\DUMP'

This creates new installation directory
mkdir $New_Inst_dir

Copies all files from dump location to new install location
cd $DUMP_location
xcopy $DUMP_location $New_Inst_dir /E /T
xcopy $DUMP_location $New_Inst_dir /R /H /E /G

4.5 File Prefix List – by Elythnwaen

(Back to Index)

The list below shows all the prefixes that the files in the game can have, makes it easier to know what you are looking for, so you don't need to parse all files. Thanks to Elythnwaen for making it:

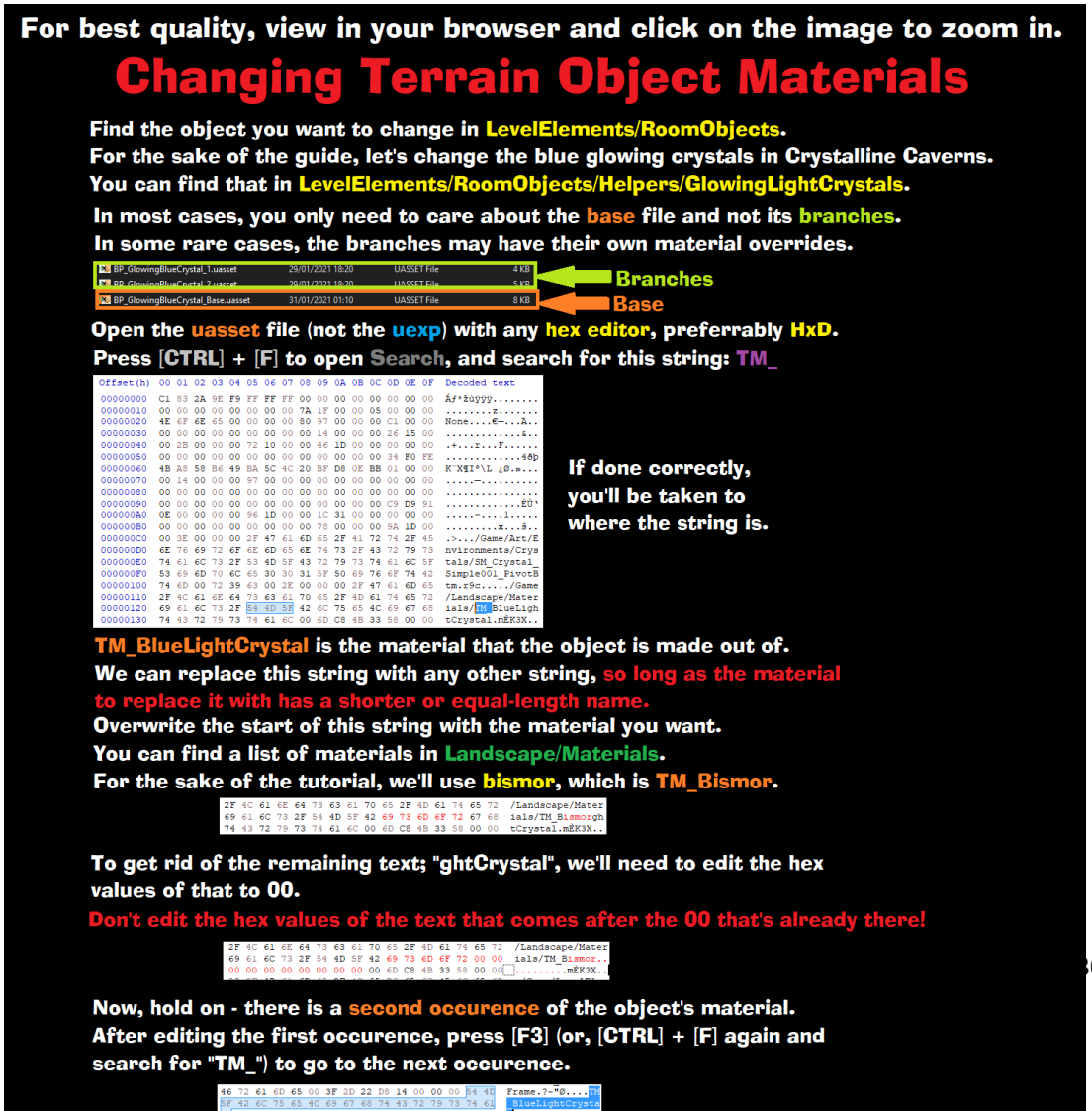
1P: 1st Person animation
3P: 3rd Persion animation
APB: Animated BluePrint
ACH: ACHievement
AFE: AFfliction Effect
AFL: AFfLiction
AFLS: AFfLiction SFX
AIC: AI Component
ANIM: ANIMation
APB: Animated BluePrint, but with a typo
ASP: Aim offset blend SPace 1D
BA: Bosco Ability
BB: BlackBoard data
BG: BackGround
BP: BluePrint
BS: Blend Space
BT: Behavior Tree
CD: Critter Definition
CG: Community Goal
CGC: Community Goal Category
CGF: Community Goal Faction
CI: Cave Influencer
CP: CamPaign
CRB: Commnuity Reward Bundle
CRC: Carved Resource Creator
CRS: Commnuity Reward Setup
CRV: CuRVe
CSC: Cave Script Component
DB: DeBris
DD: Deep Dive
DE: Drink Effects
DEC: DECorators
DFC: DiFfiCulty
DIC: Debris InfluenCer
DLC: DownLoadable Content
DMG: DaMaGe
DNA: missionDNA (all stats defining a mission: Complexity, Duration...)
ED: Enemy Descriptor
EG: Enemy Group descriptor
EMMD: Enemy Miners Manual Data
ENE: ENEmy
ENUM: ENUmeration
ERT: Event Reward Type
ESI: Enemy Showroom Item
EV: EVent
EWC: Enemy Wave Controller
FM: ai ForMation
FP: First Person

GD: Game Data
GM: GaMe
HG: HUD visibility Group
HUD: Head Up Display
IAS: Item character Animation Set
ID: IDentifier (could be for enemies, items...)
ITM: ITeM
KPI: Key Performance Indicator
LIB: LIBrary
LVL: Level
M: Material
MAG: MAGazine
MD: Mission Description
MF: Material Function
MI: Material Instance constant
MIL: MILstones
MMUT: Missions MUTator
MP: Material Particle
MS: Mission Stats
MSC: Mission Stats Category
MUT: MUTators
OBJ: OBJectives
OC: OverClocks
OSB: Overclock Schematic Bank
P: Particles
PAF: Pawn AFfliction
PAO: Pawn Affliction Overlay
PERK: PERK
PM: Physical Material
PRJ: PRoJectile
PRW: PRevieW
PST: Pawn STats
PXP: PickaXe Part
RCR: Carved Resource Creator
RES: RESource
RFND: ReFuND
RMA: RooM generator
RMG: RooM Generator
RT: Reactive Terrain
SCAT: Schematic CATegory
SCC: skin SChematic Collection
SCE: SChematic Element
SCR: SChematic Rarity
SE: Special Event
SER: SERvices
SK: SKeleton
SKB: SKin Bank
SKS: SKin Sets
SM: Static Mesh
SQ: SeQuences
ST: Sound Track
STAT: STATs (achievements related)
STE: STatus Effect
T: Textures
TAT: TATtoos
TBF: Temporary BuFf
TM: Terrain Materials

TP: Tunnel Parameters
TS: Tunnel Settings
TSK: TaSKs
TSS: Tunnel Segments Settings
TTP: Terrain TyPe
UAS: Use Animation Settings
UI: User Interface
UPC: UPgrade Category
UPG: UPgrade Group
VAN: VANity
VP: Victory Pose
VSB: Vanity Schematic Bank
W: Widget
WND: WiNdow Widget
WP: Widget Parts
WPN: WeaPoN
WRN: WaRNing

4.6 Terrain Changing Guide – by jen walter

(Back to Index)



4.7 Damage Type List – by Elythnwaen

(Back to Index)

DNG_Ice: Frost dmg + Cold dmg (Health and Temp. 100%/50%)
DMG_Burn: Fire dmg (Health only)
DMG_Heat: Heat dmg (Temp. only)
DMG_Fire: Fire dmg + Heat dmg (Health and Temp. 100%/100%)
DMG_Cold: Cold dmg (Temp. only)
DMG_Thorns: Thorn perk
DMG_Poison: Poison dmg
DMG_Pysical: Melee dmg
DMG_Kinetic: Kinetic dmg
DMG_Falling: Fall dmg
DMG_Freezing: Frost dmg (Health only)
DMG_Electric: Electric dmg
DMG_NoOxygen: Suffocation dmg
DMG_Explosive: Explosive dmg
DMG_Radiation: Radiation dmg
DMG_Disintegrate: Pure dmg, used for SYiH and Temperature shock
DMG_InteralDamage: Used for the new Explosive Reload OC
DMG_EventExplosion: Tritilyte bombs

4.8 Model Replacement – by Pacagma

(Back to Index)

Welcome to my guide!

Today I'll teach you how to do Model Replacement and Hat Replacement for DRG.

Important: If you wanna see the gifs check them in the docs version, where they work:
https://docs.google.com/document/d/16aBTZ_Qx614CrHd6rS4KZ7K9tZn0p3CGOzbfaZ5wJ_o/edit

Let's get started.

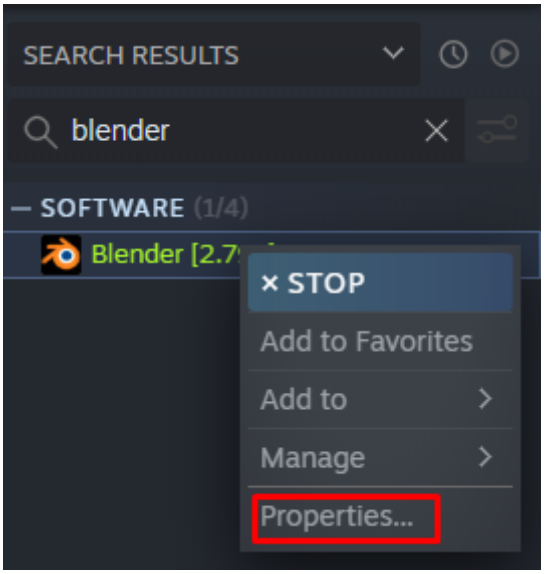
You'll need a Free 3D software called **Blender**, go ahead and download it.

(https://store.steampowered.com/app/365670/Blender/)

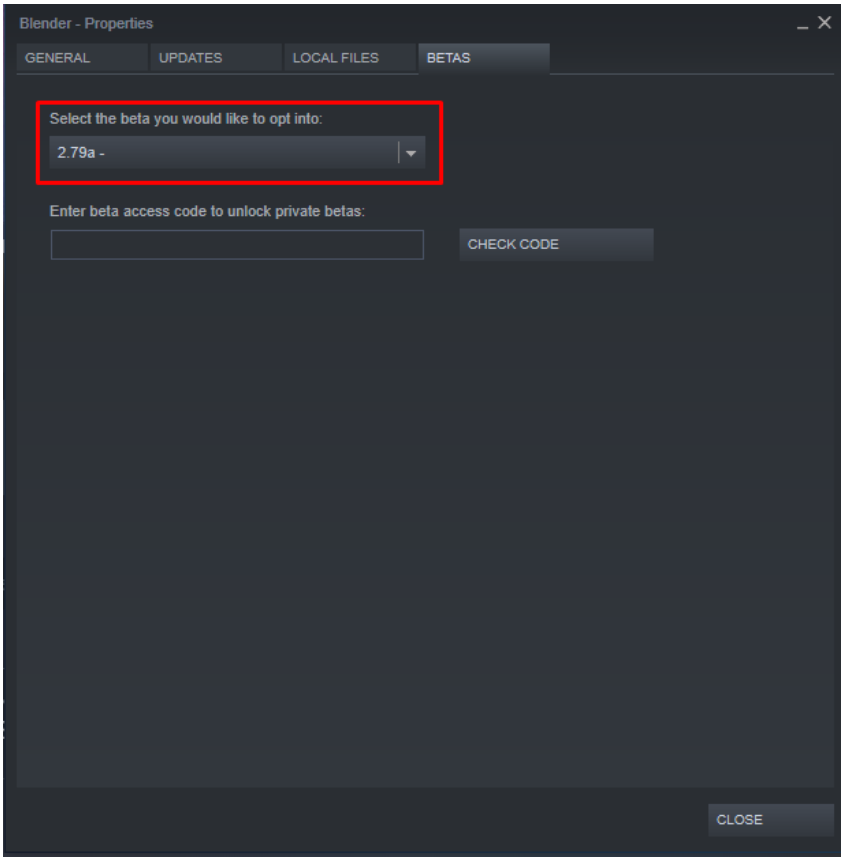
Now I use personally use the 2.79 version of blender because of I got used on it the most.

To use the exact same version that i do, do the following steps:

1. Right click on **Blender** and select **Properties...**

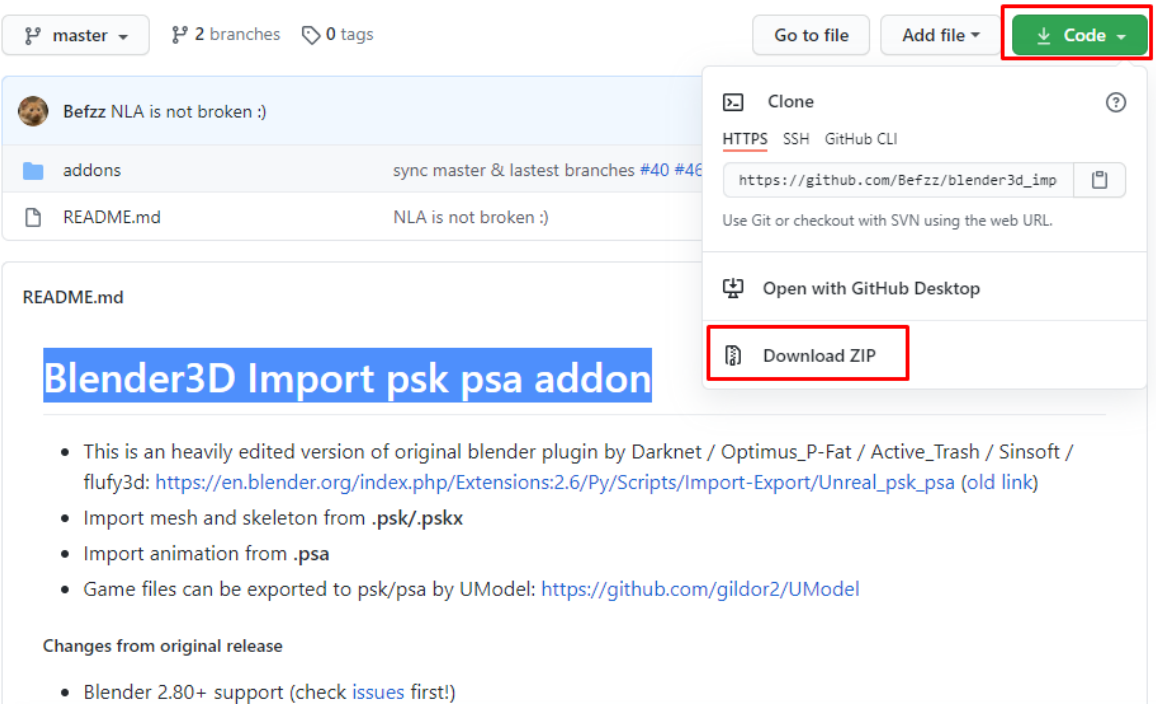


2. Go to **Betas** tab and select to **2.79a-**



Wait for it to re download to that version and you're good to go for using Blender.

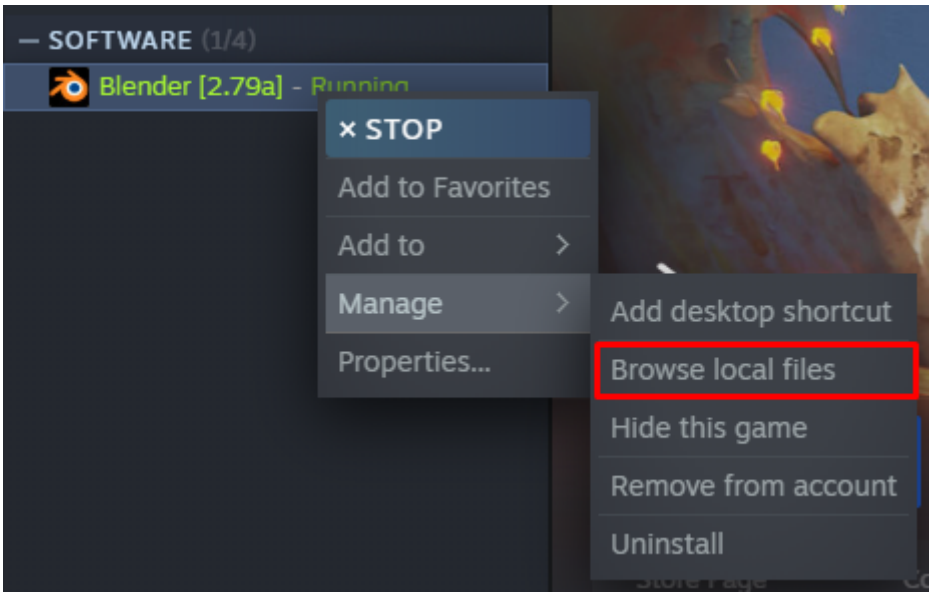
Now to import the DRG models into blender you'll need an Addon called **Blender 3D import psk psa addon,** download it here. (https://github.com/Befzz/blender3d_import_psk_psa)



Now let me show you how to install the addon to use it in Blender.

1. Open up the directory of where your Blender is installed.

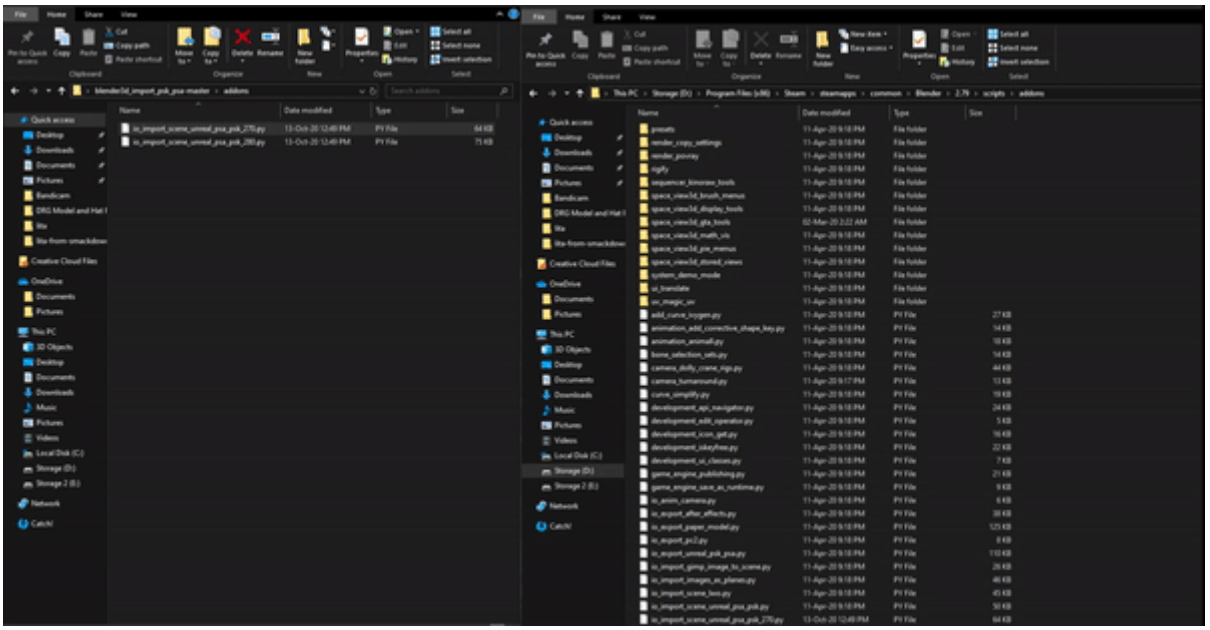
To do that, do the following: Right click on Blender and select Manage> Browse local files



Now navigate to the folders called **2.79/scripts/addons**

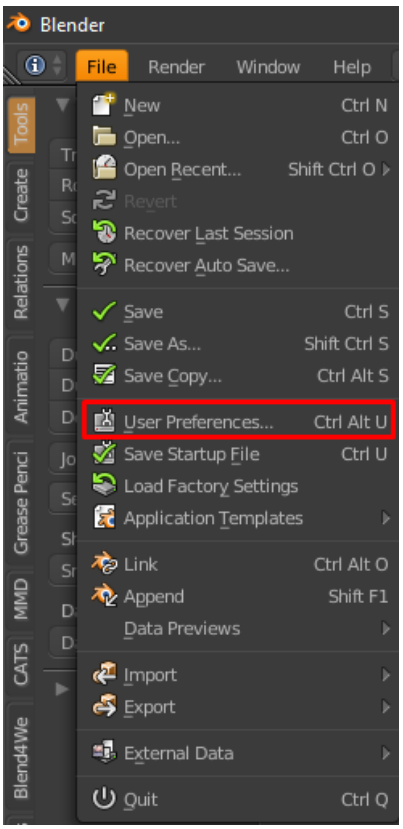
Extract the **blender3d_import_psk_psa-master.zip** anywhere then in addons folder drag and drop the **io_import_scene_unreal_psa_psk_270.py** into **2.79/scripts/addons**

Here’s a gif of it:

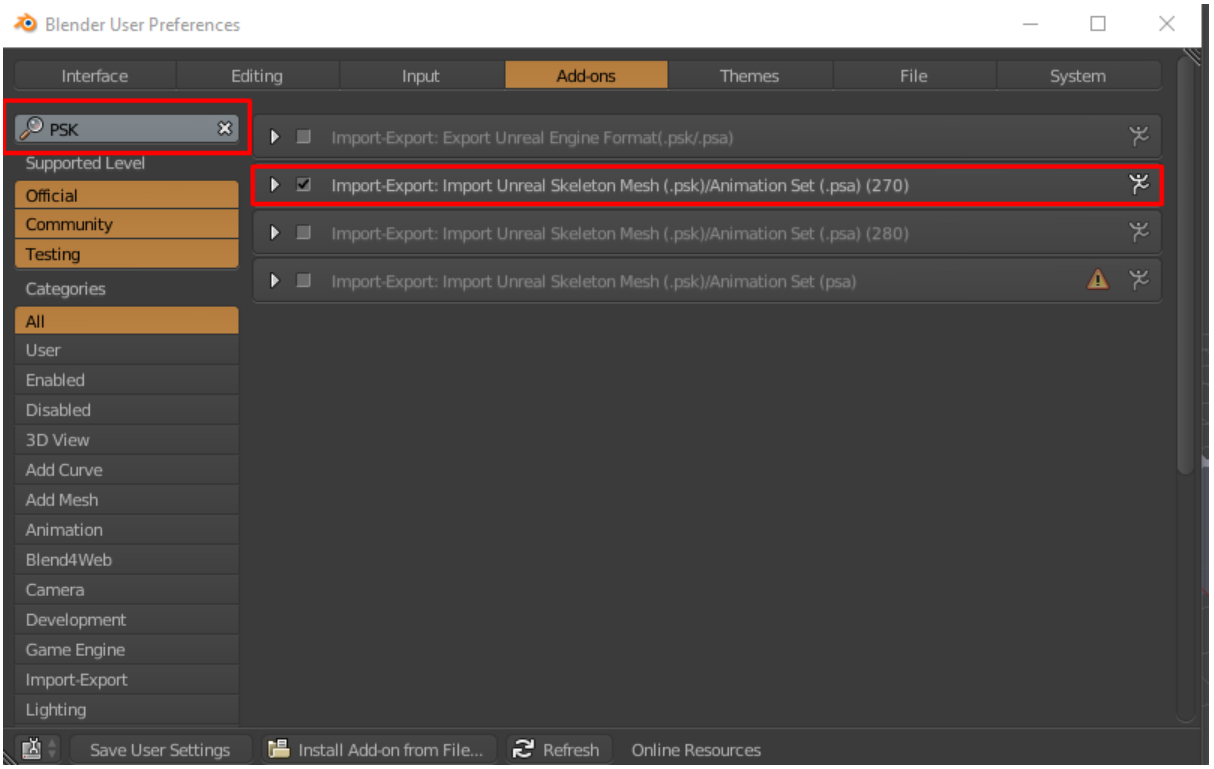


2. Next step is to enable the addon in Blender, do the following:

In Blender on top right where it says **File** then go to **User Preferences…**



In Search bar type **PSK** and Tick: **Import-Export: Import Unreal Skeleton Mesh (.psk.)/Animation Set (.psa) (270)**



Now you're ready to import DRG Models into Blender.

But wait, we need to extract the models with a tool called **UModel,** download it here. (https://www.gildor.org/en/projects/umodel)

Scroll down to downloads and pick the **Win32 version**

Downloads

Win32 version	Size 964.3 KB	Updated 24 October, 2020 - 16:04	804668 downloads
Linux version	Size 537.76 KB	Updated 19 October, 2020 - 13:45	25038 downloads

Readme with changelog

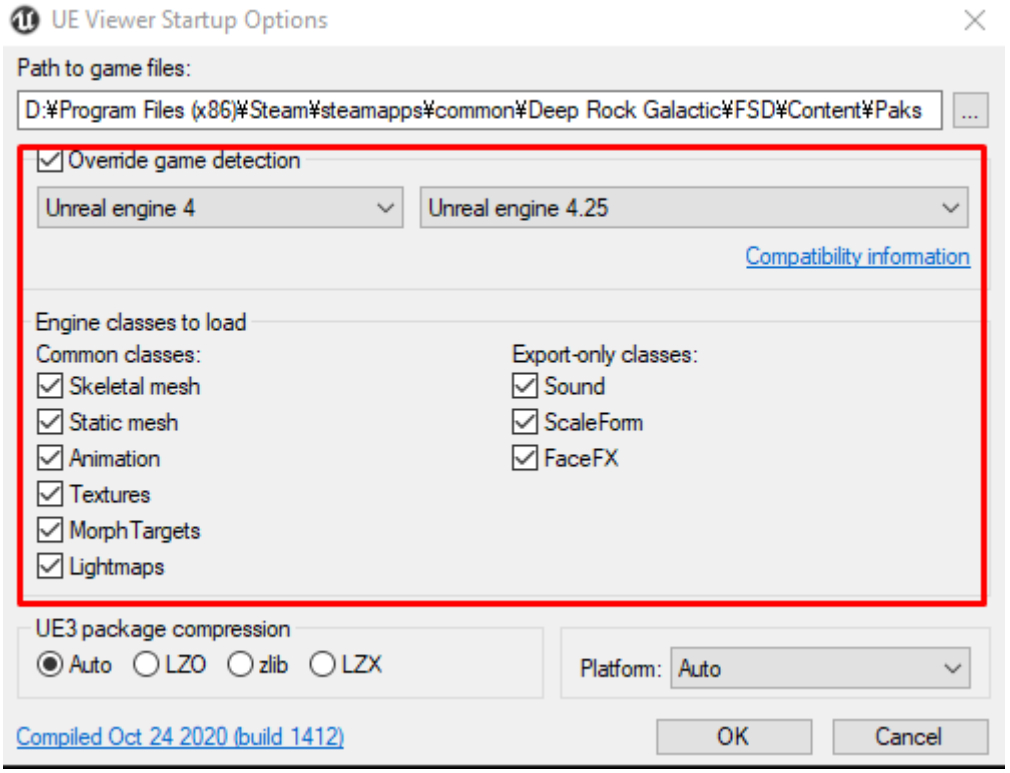
Extract the **umodel_32.zip** anywhere, open the folder and open up the **umodel.exe**

Now do the following:

1. In **Path to game files .** Paste the directory of where your **FSD-WindowsNoEditor.pak** is.

(depends on what drive is the game installed at) Mine is in : **D:\Program Files (x86)\Steam\steamapps\common\Deep Rock Galactic\FSD\Content\Paks**

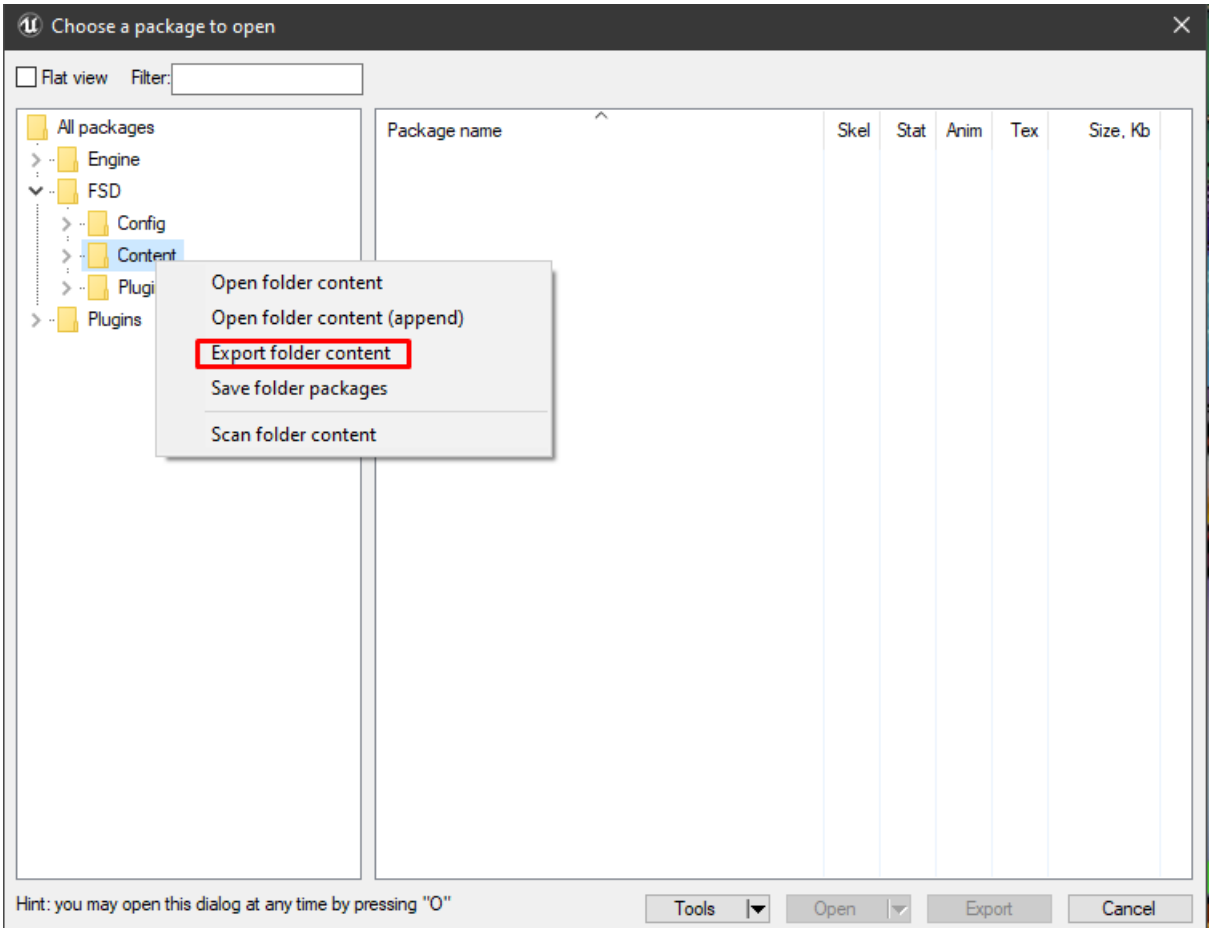
Next do the following as what i've did in the picture:



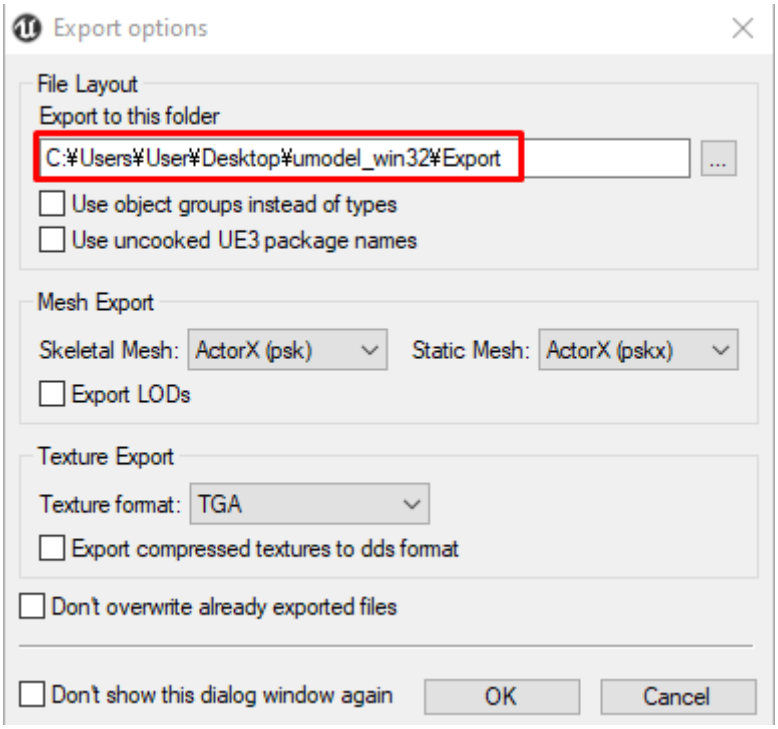
(I pretty much export every single thing from the game, that's why i ticked **Sound** with it too, for **Scale Form** and **FaceFX**, i have no idea what they do, but i still ticked it because why not?)

Next Click **OK**

Now expand **FSD** and you'll find the folder called **Content.** Right click on it and click **Export folder content.**



You'll have the **Export options** window.



Make a folder called **Export** inside **umodel_win32** folder to be more organized.

Click **OK** and wait for it to complete the export.

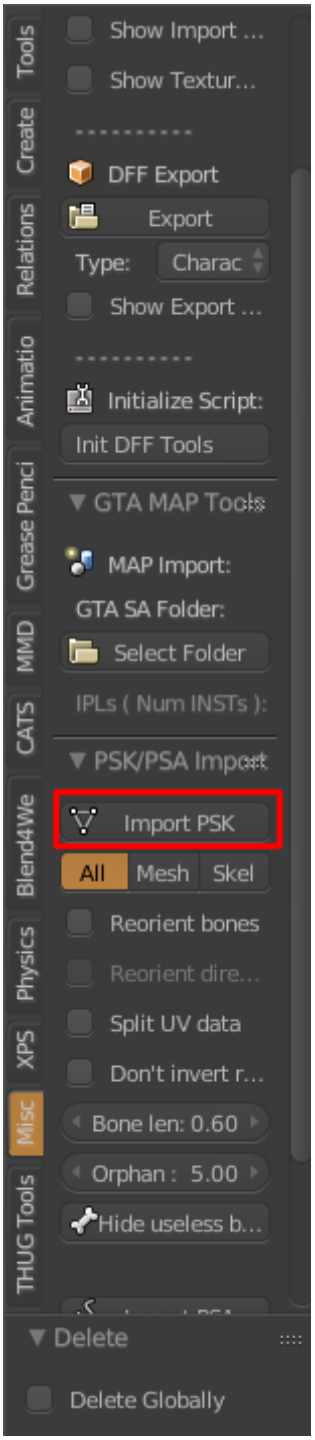
Time to replace a armor.

Be sure to delete everything from the scene first.

Let’s say we wanna replace Driller’s MK1 armor with the Streamer armor.

We will need to open the model mesh called **SK_Vanity_ArmorStreamer.psk**

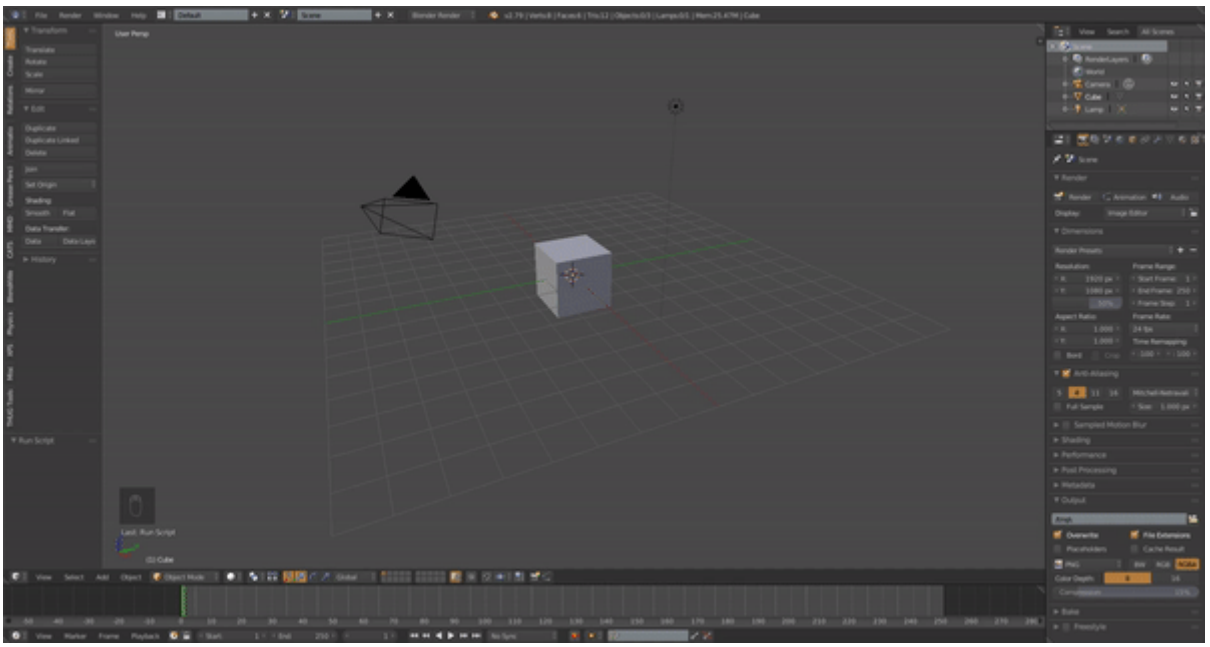
First find the tab called **Misc** and select **Import PSK**



Navigate to the folder where you exported the files using **Umodel** then go to **Character\Vanity2\Armor** and find **SK_Vanity_ArmorStreamer.psk**

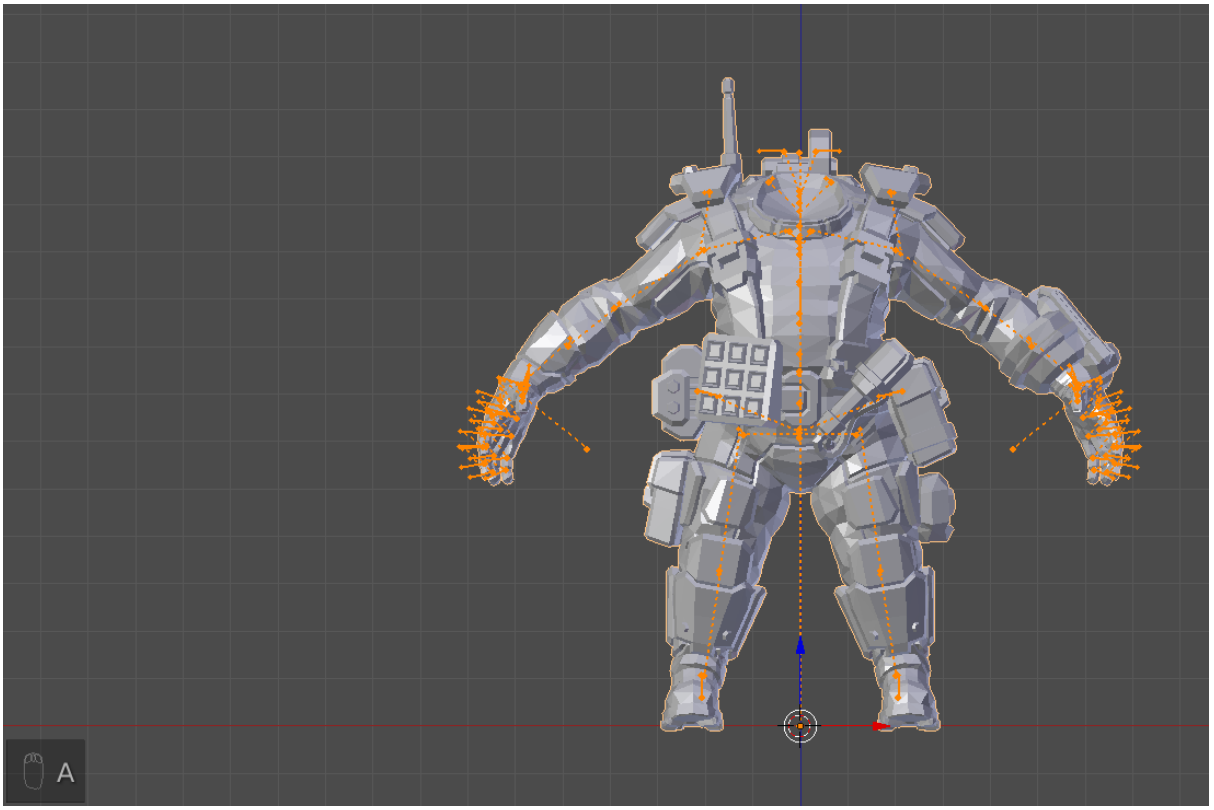

E:\DRG Mods\Extracted\Character\Vanity2\Armor\
SK_Vanity_ArmorStreamer.psk
..
InactivePaintjobs
Paintjobs_ArmorDefaults
Paintjobs_Loose
Paintjobs_Promotions
SK_Vanity_ArmorDriller001.psk 497 KiB
SK_Vanity_ArmorDriller001_Beard.psk 540 KiB
SK_Vanity_ArmorDriller001_FP.psk 154 KiB
SK_Vanity_ArmorDriller002.psk 505 KiB
SK_Vanity_ArmorDriller002_Beard.psk 507 KiB
SK_Vanity_ArmorDriller002_FP.psk 154 KiB
SK_Vanity_ArmorDriller003.psk 548 KiB
SK_Vanity_ArmorDriller003_FP.psk 232 KiB
SK_Vanity_ArmorDriller004.psk 778 KiB
SK_Vanity_ArmorDriller004_Beard.psk 759 KiB
SK_Vanity_ArmorDriller004_FP.psk 232 KiB
SK_Vanity_ArmorDriller005.psk 912 KiB
SK_Vanity_ArmorDriller005_Beard.psk 896 KiB
SK_Vanity_ArmorDriller005_FP.psk 232 KiB
SK_Vanity_ArmorDriller_DLC01.psk 759 KiB
SK_Vanity_ArmorDriller_DLC01_Beard.psk 758 KiB
SK_Vanity_ArmorDriller_DLC01_FP.psk 157 KiB
SK_Vanity_ArmorDriller_DLC03.psk 802 KiB
SK_Vanity_ArmorDriller_DLC03_FP.psk 153 KiB
SK_Vanity_ArmorEngineer001.psk 516 KiB
SK_Vanity_ArmorEngineer001_FP.psk 179 KiB
SK_Vanity_ArmorEngineer002.psk 544 KiB
SK_Vanity_ArmorEngineer002_FP.psk 174 KiB
SK_Vanity_ArmorEngineer003.psk 511 KiB
SK_Vanity_ArmorEngineer003_FP.psk 206 KiB
SK_Vanity_ArmorEngineer004.psk 691 KiB
SK_Vanity_ArmorEngineer004_FP.psk 236 KiB
SK_Vanity_ArmorEngineer005.psk 737 KiB
SK_Vanity_ArmorEngineer005_FP.psk 242 KiB
SK_Vanity_ArmorEngineer_DLC01.psk 693 KiB
SK_Vanity_ArmorEngineer_DLC01_Beard.psk 684 KiB
SK_Vanity_ArmorEngineer_DLC01_FP.psk 197 KiB
SK_Vanity_ArmorEngineer_DLC03.psk 627 KiB
SK_Vanity_ArmorEngineer_DLC03_FP.psk 153 KiB
SK_Vanity_ArmorGunner001.psk 507 KiB
SK_Vanity_ArmorGunner001_Beard.psk 492 KiB
SK_Vanity_ArmorGunner001_FP.psk 190 KiB
SK_Vanity_ArmorGunner002.psk 509 KiB
SK_Vanity_ArmorGunner002_Beard.psk 482 KiB
SK_Vanity_ArmorGunner002_FP.psk 206 KiB
SK_Vanity_ArmorGunner003.psk 528 KiB
SK_Vanity_ArmorGunner003_FP.psk 178 KiB
SK_Vanity_ArmorGunner004.psk 826 KiB
SK_Vanity_ArmorGunner004_Beard.psk 817 KiB
SK_Vanity_ArmorGunner004_FP.psk 216 KiB
SK_Vanity_ArmorGunner005.psk 936 KiB
SK_Vanity_ArmorGunner005_Beard.psk 918 KiB
SK_Vanity_ArmorGunner005_FP.psk 237 KiB
SK_Vanity_ArmorGunner_DLC01.psk 799 KiB
SK_Vanity_ArmorGunner_DLC01_Beard.psk 794 KiB
SK_Vanity_ArmorGunner_DLC01_FP.psk 224 KiB
SK_Vanity_ArmorGunner_DLC03.psk 762 KiB
SK_Vanity_ArmorGunner_DLC03_Beard.psk 690 KiB
SK_Vanity_ArmorGunner_DLC03_FP.psk 182 KiB
SK_Vanity_ArmorScout001.psk 374 KiB
SK_Vanity_ArmorScout001_FP.psk 154 KiB
SK_Vanity_ArmorScout002.psk 478 KiB
SK_Vanity_ArmorScout002_FP.psk 192 KiB
SK_Vanity_ArmorScout003.psk 550 KiB
SK_Vanity_ArmorScout003_FP.psk 257 KiB
SK_Vanity_ArmorScout004.psk 780 KiB
SK_Vanity_ArmorScout004_Beard.psk 759 KiB
SK_Vanity_ArmorScout004_FP.psk 271 KiB
SK_Vanity_ArmorScout005.psk 733 KiB
SK_Vanity_ArmorScout005_Beard.psk 898 KiB
SK_Vanity_ArmorScout005_FP.psk 243 KiB
SK_Vanity_ArmorScout_DLC01.psk 667 KiB
SK_Vanity_ArmorScout_DLC01_Beard.psk 663 KiB
SK_Vanity_ArmorScout_DLC01_FP.psk 201 KiB
SK_Vanity_ArmorScout_DLC03.psk 659 KiB
SK_Vanity_ArmorScout_DLC03_FP.psk 121 KiB
SK_Vanity_ArmorStreamer.psk 764 KiB
SK_Vanity_ArmorStreamer_FP.psk 184 KiB

Here’s a quick GIF of importing the model (I enabled screencast keys so you can see what i’m pressing):

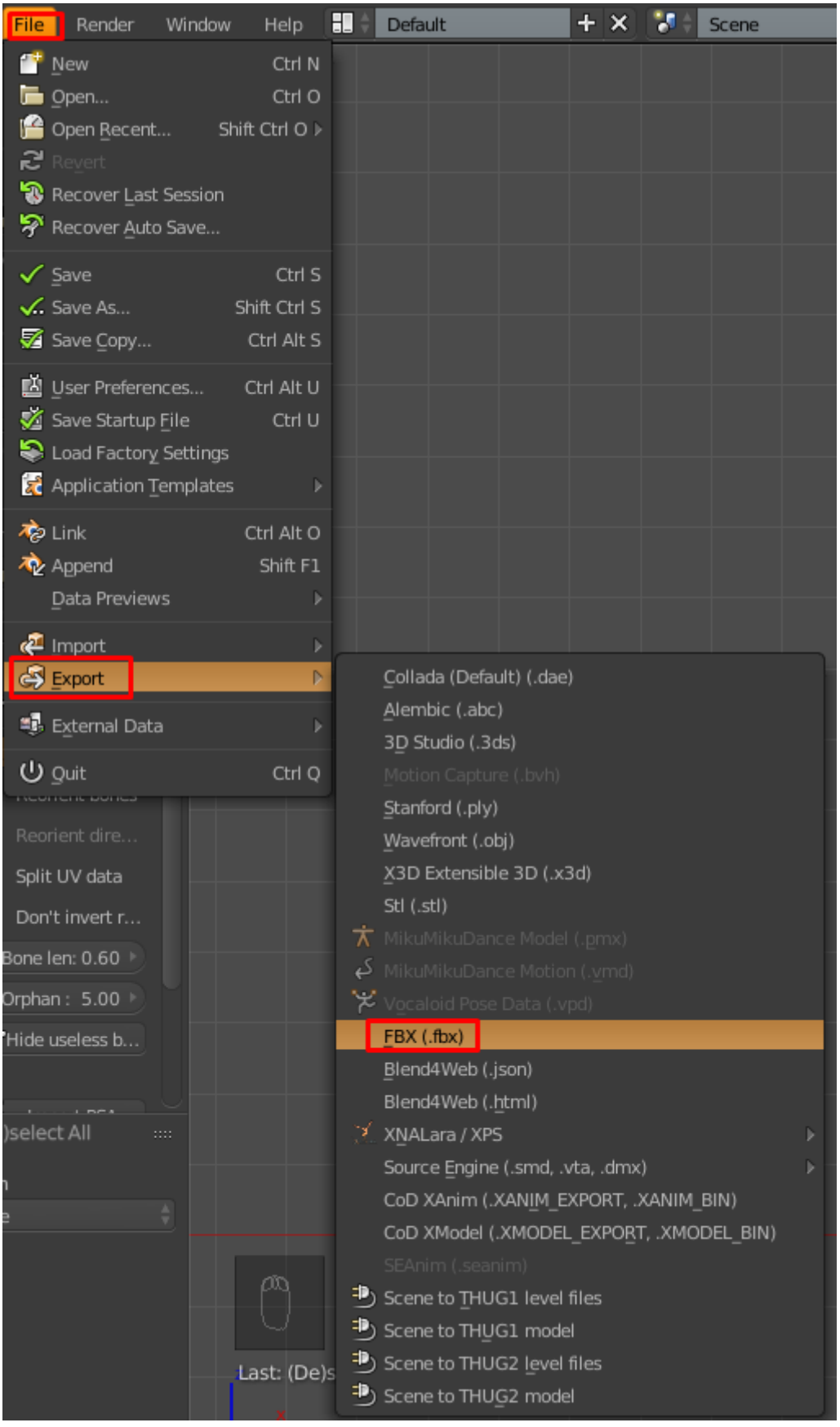


Now let’s export the model so we can import it in **Unreal Editor**

Select everything by tapping **A** key in the scene



Now go to **File>Export>FBX(.fbx)** and call it whatever you want then **Export**


File
Render
Window
Help
Default
Scene
New Ctrl N
Open... Ctrl O
Open Recent... Shift Ctrl O
Revert
Recover Last Session
Recover Auto Save...
Save Ctrl S
Save As... Shift Ctrl S
Save Copy... Ctrl Alt S
User Preferences... Ctrl Alt U
Save Startup File Ctrl U
Load Factory Settings
Application Templates
Link Ctrl Alt O
Append Shift F1
Data Previews
Import
Export
External Data
Quit Ctrl Q
Collada (Default) (.dae)
Alembic (.abc)
3D Studio (.3ds)
Motion Capture (.bvh)
Stanford (.ply)
Wavefront (.obj)
X3D Extensible 3D (.x3d)
Stl (.stl)
MikuMikuDance Model (.pmx)
MikuMikuDance Motion (.vmd)
Vocaloid Pose Data (.vpd)
FBX (.fbx)
Blend4Web (.json)
Blend4Web (.html)
XNALara / XPS
Source Engine (.smd, .vta, .dmx)
CoD XAnim (.XANIM_EXPORT, .XANIM_BIN)
CoD XModel (.XMODEL_EXPORT, .XMODEL_BIN)
SEAnim (.seanim)
Scene to THUG1 level files
Scene to THUG1 model
Scene to THUG2 level files
Scene to THUG2 model
Reorient dire...
Split UV data
Don't invert r...
Bone len: 0.60
Orphan : 5.00
Hide useless b...
)select All
Last: (De)s

Now to know what's the Driller's MK1 Armor name in the files, it's:
SK_Vanity_ArmorDriller001.psk and **SK_Vanity_ArmorDriller001_Beard.psk**

Now you're wondering, why is there another one called with **_Beard** ? They made it like that because when you switch the beard in game it will still be on the same armor.

Keep in mind that only Driller and Gunner has **_Beard** at the end.

Here's a list of all of the names for MK Armors:

SK_Vanity_ArmorClassName001.psk - MK1

SK_Vanity_ArmorClassName002.psk - MK2

SK_Vanity_ArmorClassName003.psk - MK3

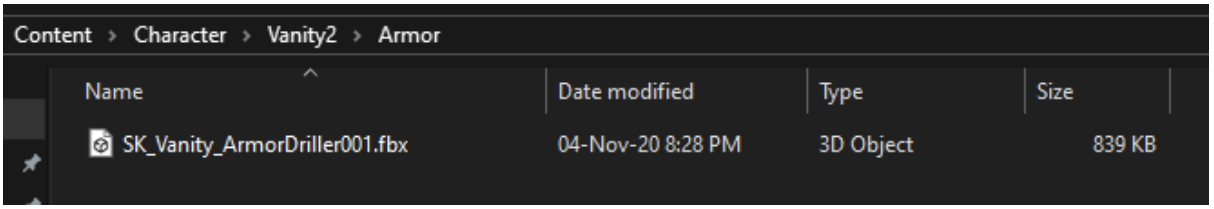
SK_Vanity_ArmorClassName004.psk - MK4

SK_Vanity_ArmorClassName005.psk - MK5

(I'll update the list for the rest of the armor's and hats some other time.)

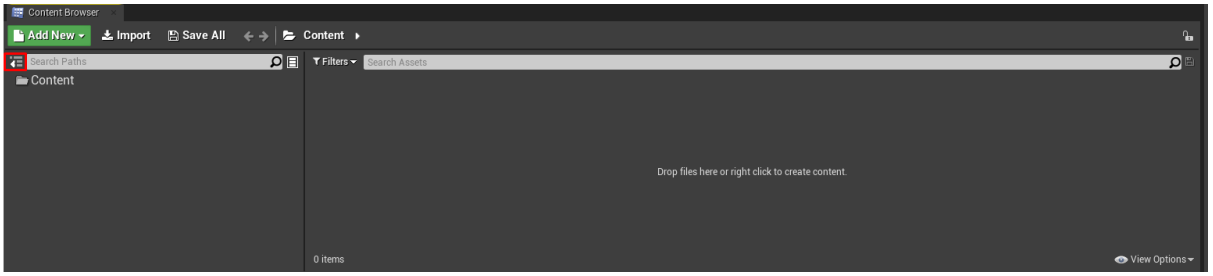
Now make a folder and call it **Content** then make the following folders inside it:

Character/Vanity2/Armor. Place your FBX file in it and rename it to
SK_Vanity_ArmorDriller001.fbx

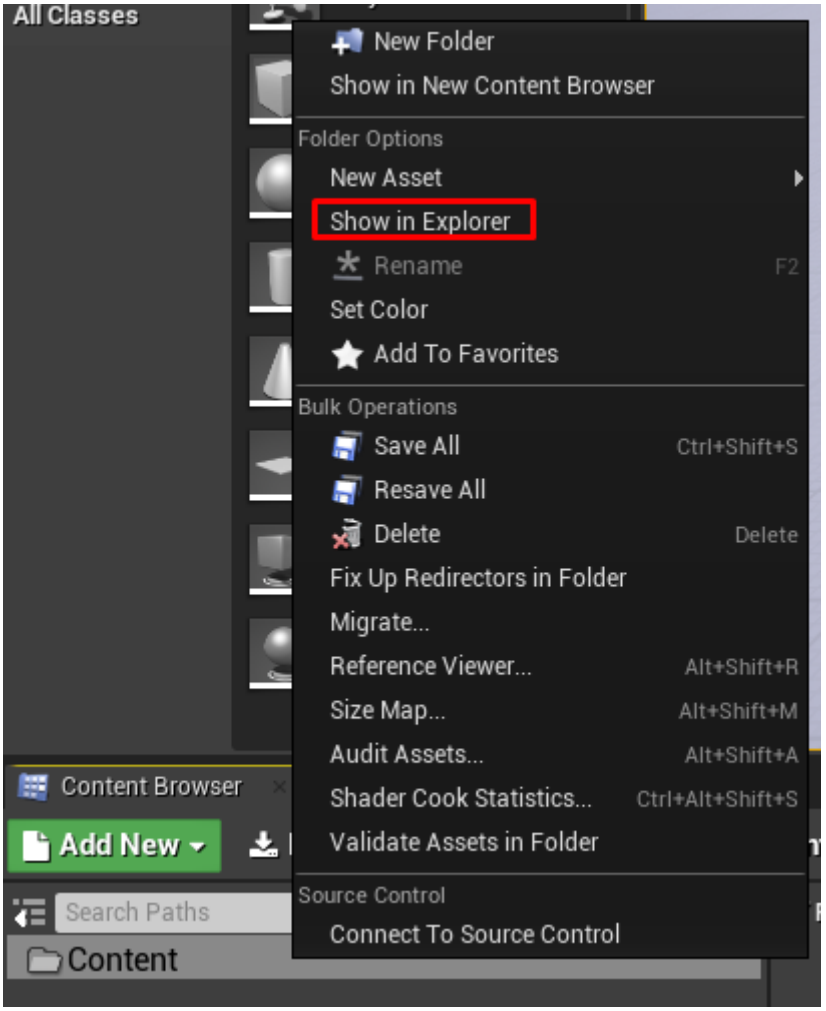


Now let's import the model in Unreal Editor.

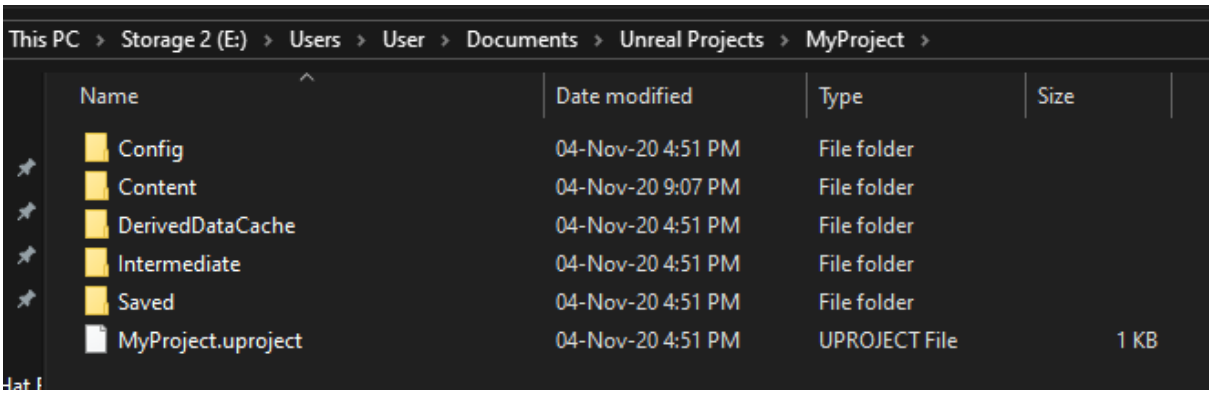
Click on **Show or hide the resources panel**



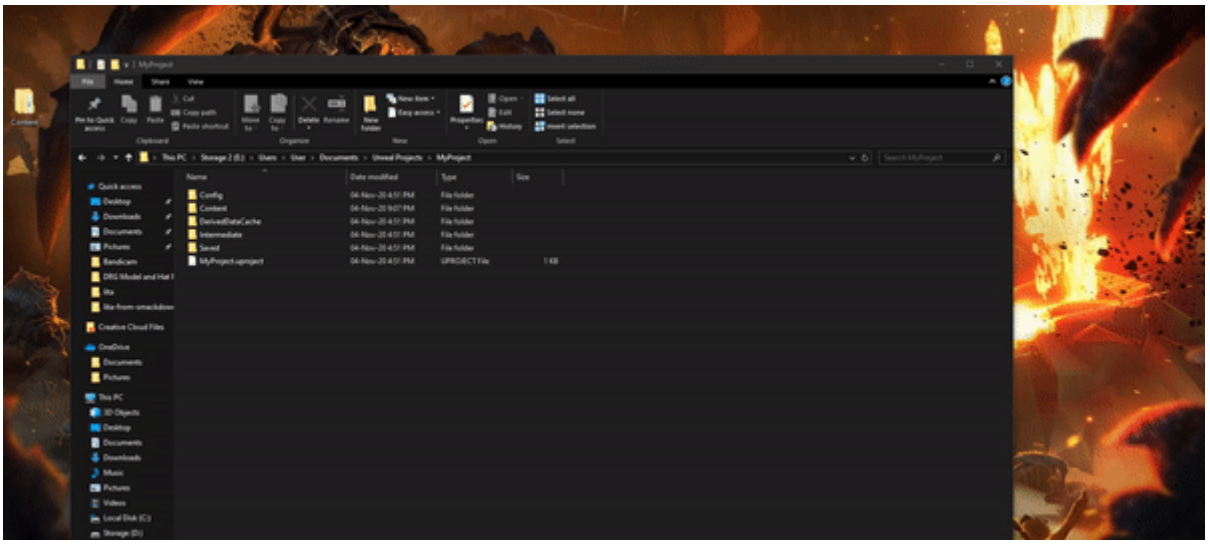
Now open the **Content** folder by right clicking on the **Content** folder then **Show Explorer**



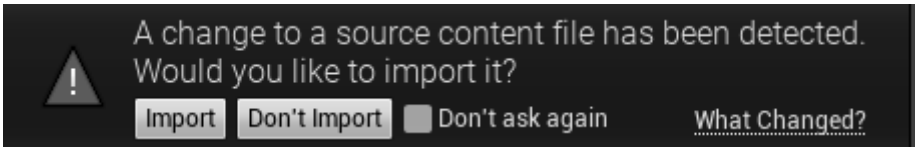
It will open the folder of your **Project**



Now place your Content folder where your’re files are in

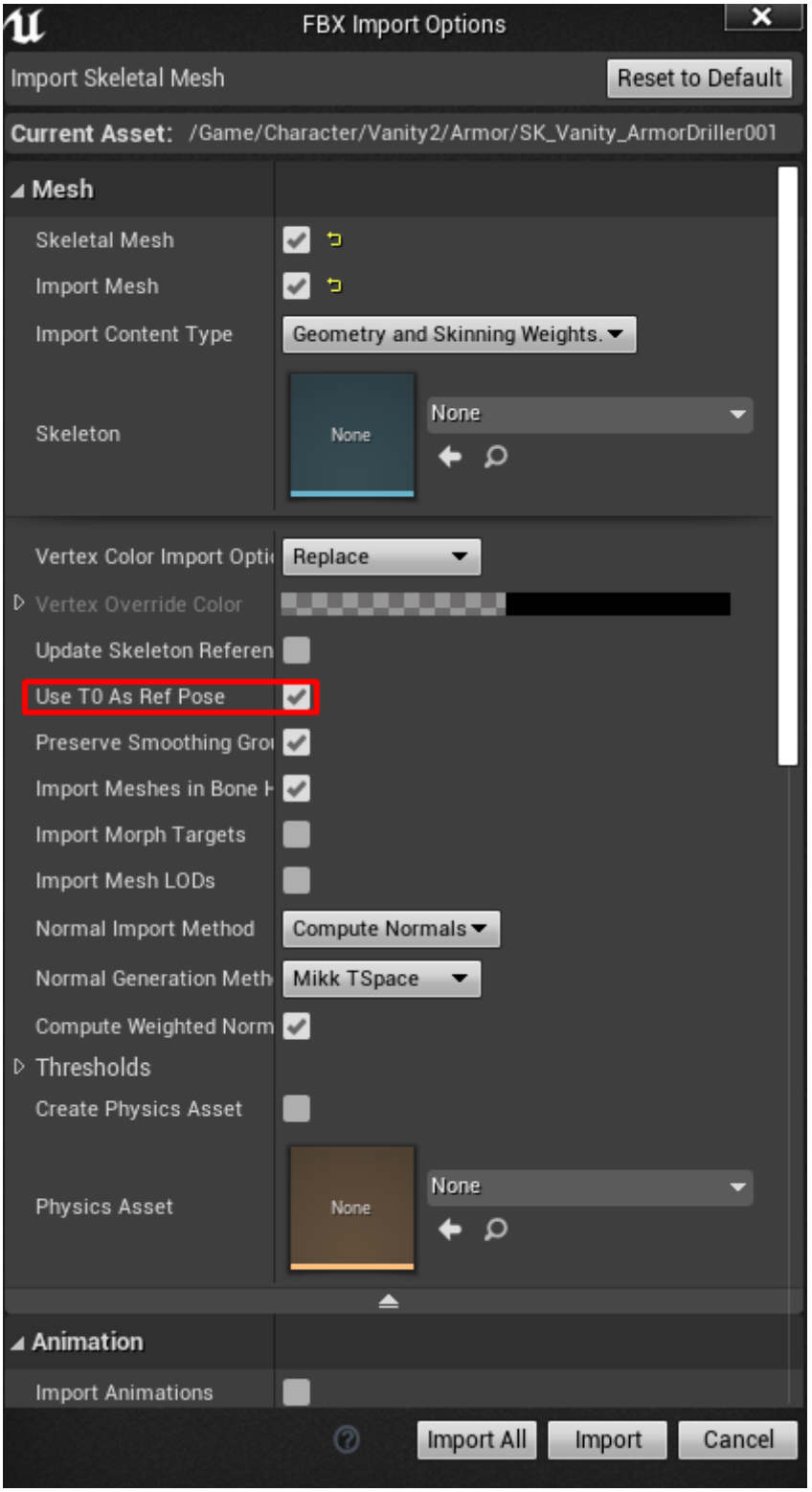


When you go in Unreal Editor this message will pop up at the bottom right.



Click **Import**

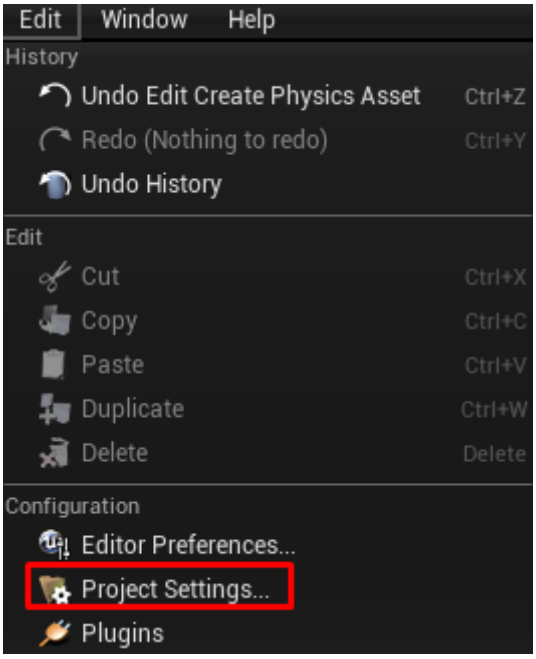
This tab will open next:



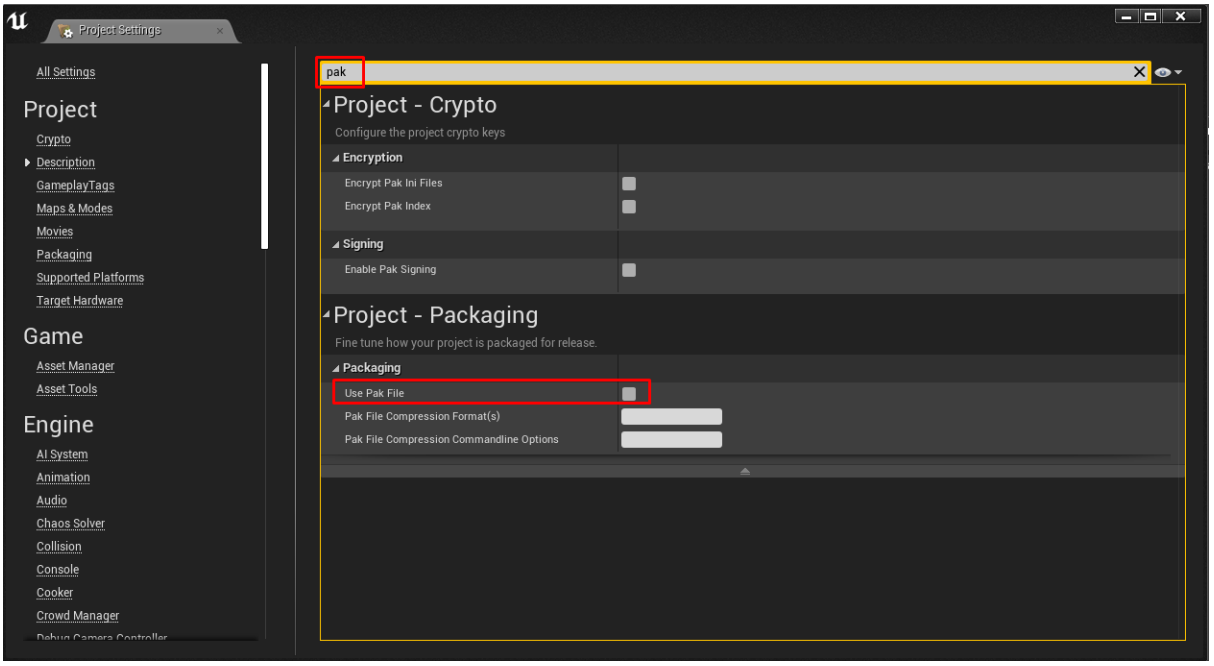
Now make sure that **Use T0 As Ref Pose** is ticked (This is important so it import's all the bones)

Now let's finally pack it to test in game, first do this before packing it.

Click on **Edit>Project Settings…**

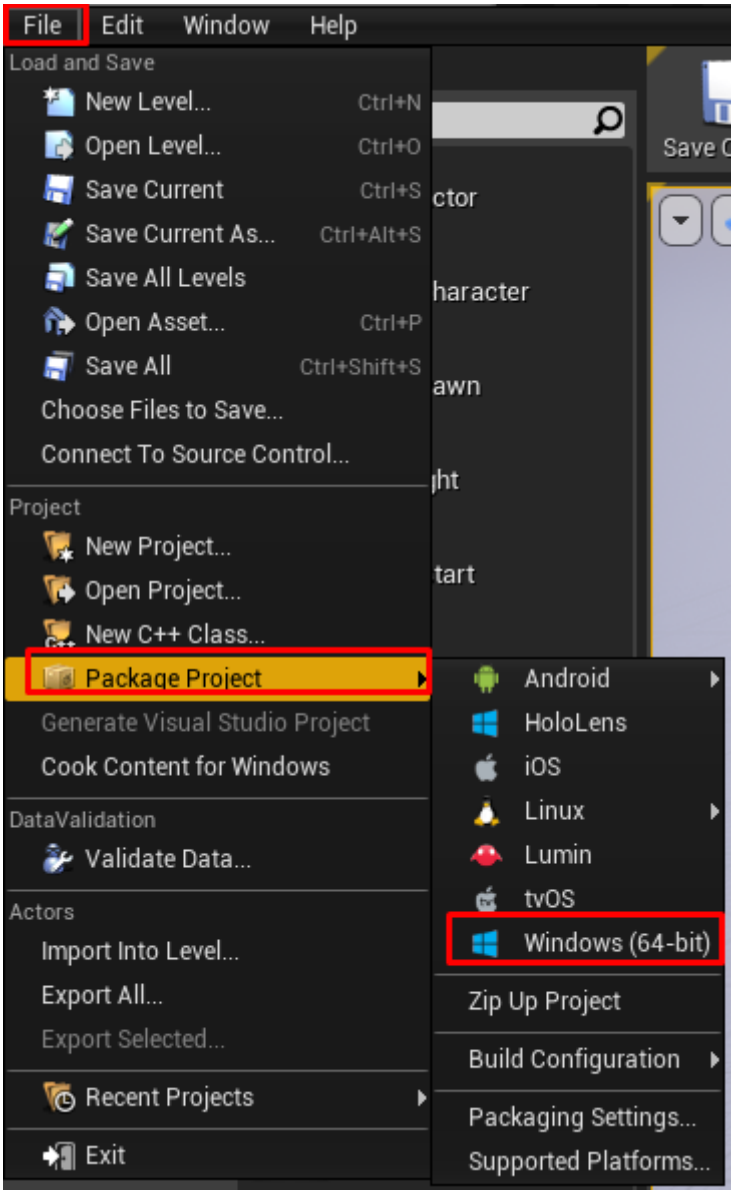


Next, on the search bar, search for **pak** and untick **Use Pak File**



Now we're ready.

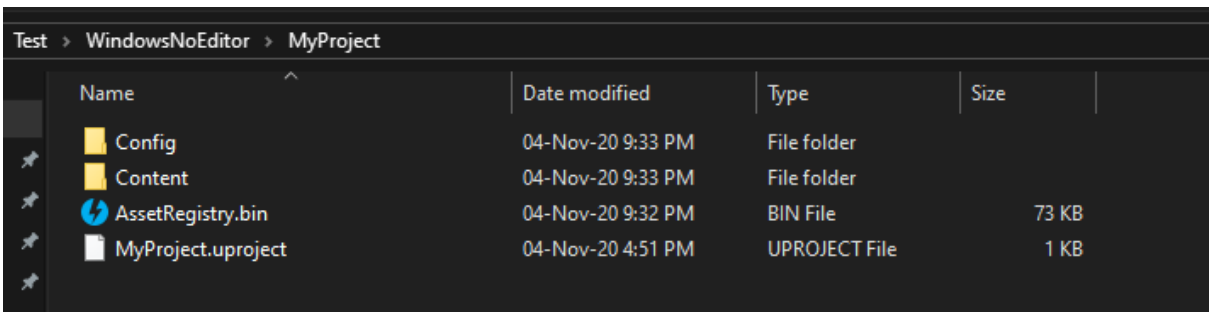
Click on **File>Package Project>Windows**



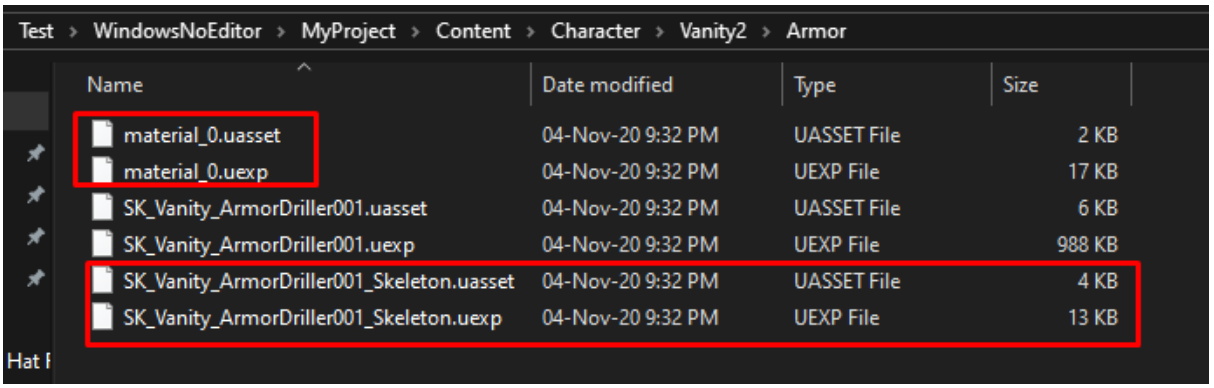
Make a folder somewhere, select it and pack it.

Wait for it to be done.

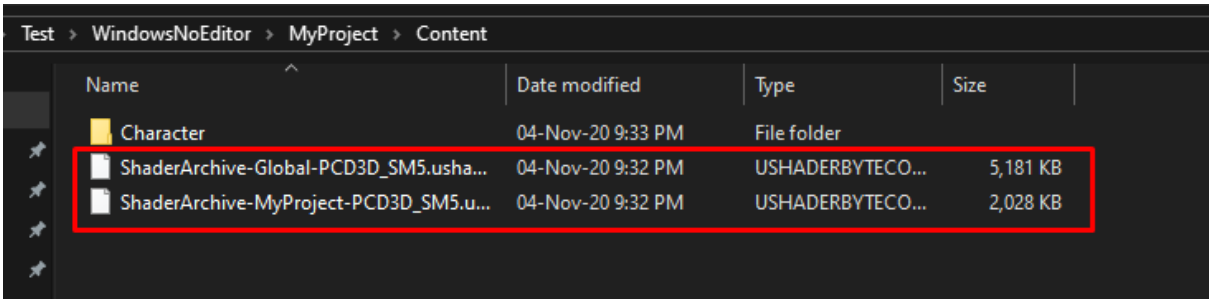
Navigate to the folder where it packed, here's a example of mine:



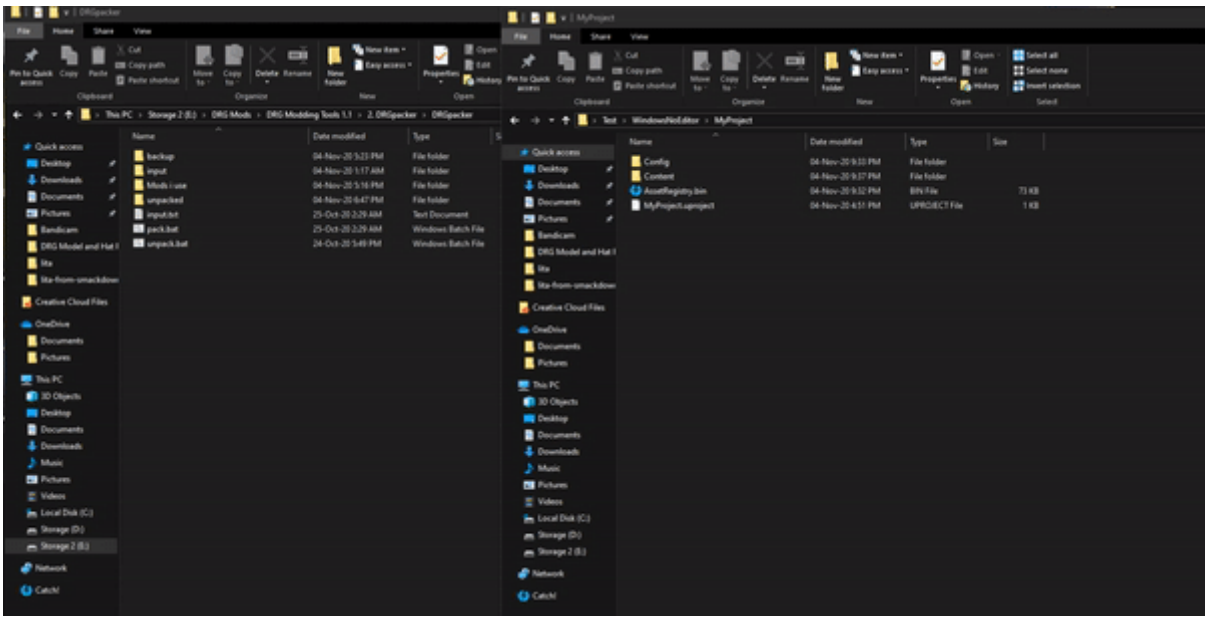
Inside the **Character/Vanity2/Armor** delete the following files that you don't need:



Now go back to the inside of the **Content** folder and delete these 2 files as well:

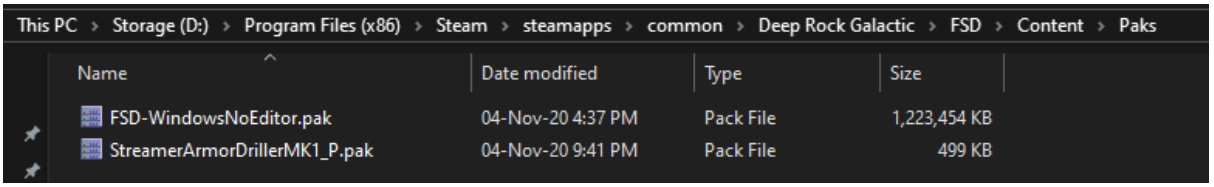


Now place the **Content** folder in **input** folder and drag and drop it into **pack.bat**



You'll get a new file called **new_P.pak**, you can call it whatever you want, example: **StreamerArmorDrillerMK1_P.pak**

Let’s install the mod, drag and drop your mod to **D:\Program Files (x86)\Steam\steamapps\common\Deep Rock Galactic\FSD\Content\Paks**



Let’s open the game and see the armor.

And there it is!

Now let’s make ourself’s a hat replacement!

Let’s replace Scout’s Ar Headset with a critter, let’s choose maggot.

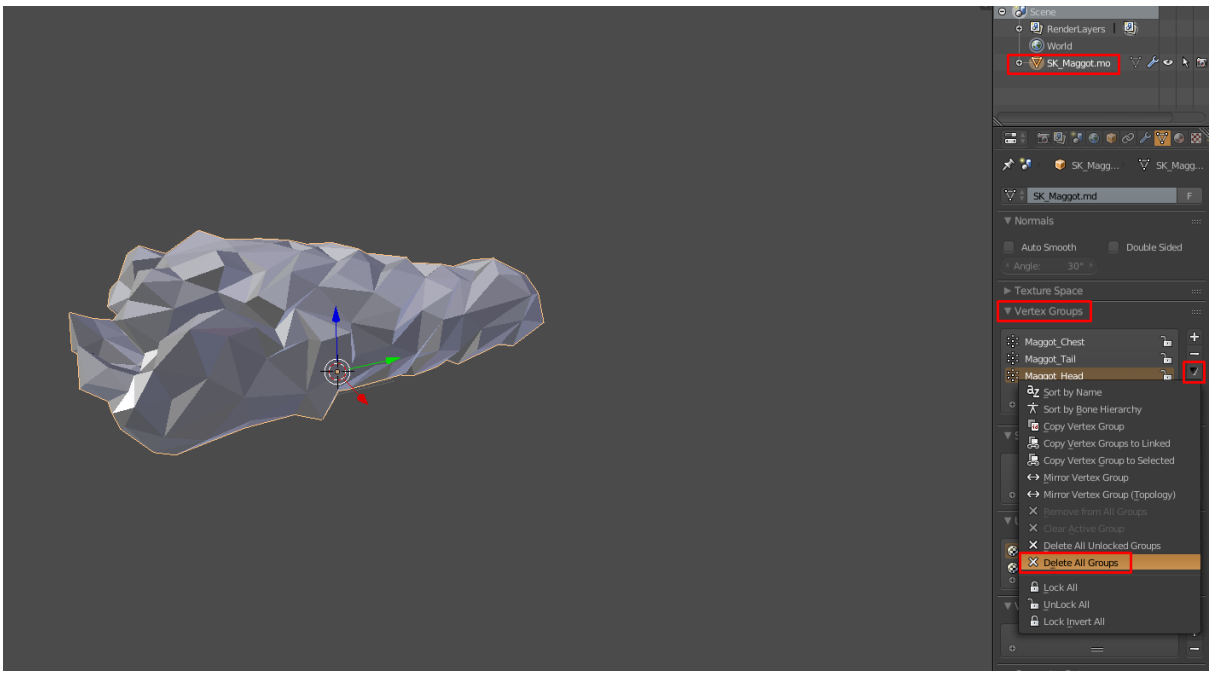
Now you should import the Dwarf’s head to use it as reference, it’s called: **SK_Vanity_Head001.psk** in **Character/Vanity2/Heads**

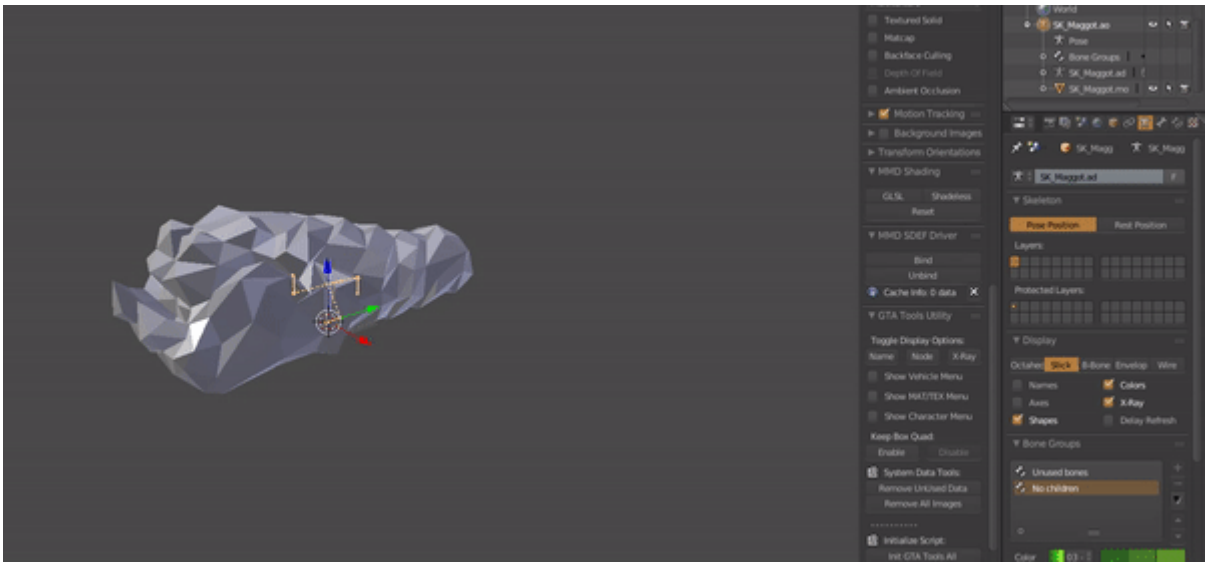
Go ahead and save the project as something, i'll name it **head.blend**

Make a new project and import the maggot, it's located in **Critters/Maggot** and the model is called **SK_Maggot.psk**

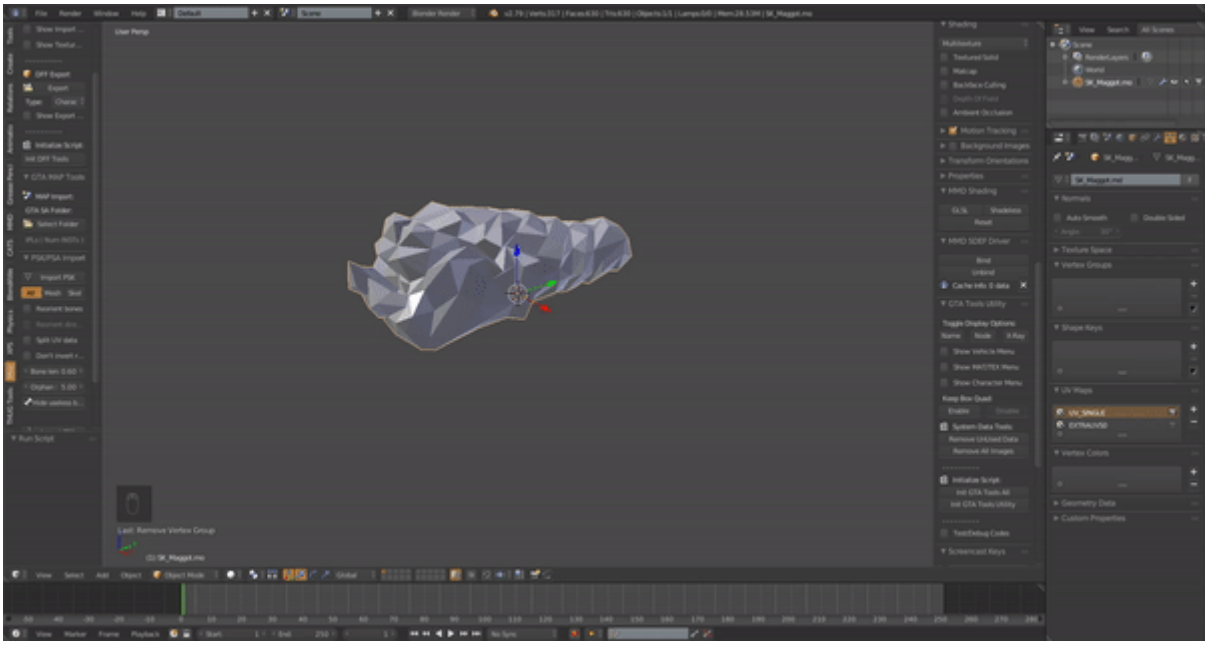
When you import the maggot then do the following,

Delete the skeleton,select the maggot and click **Delete All Groups**





Now with the maggot selected, Copy him (CTRL+C) and open the **head.blend** and paste it there (CTRL+V)



Now let's scale it down and rig it to the head.

Select the maggot, go in edit mode (press TAB), click and hold the blue arrow and move it up till it reaches to the end of the head, scale it down (press S and move your mouse down to scale it smaller) when you get the right position of it, do the following:

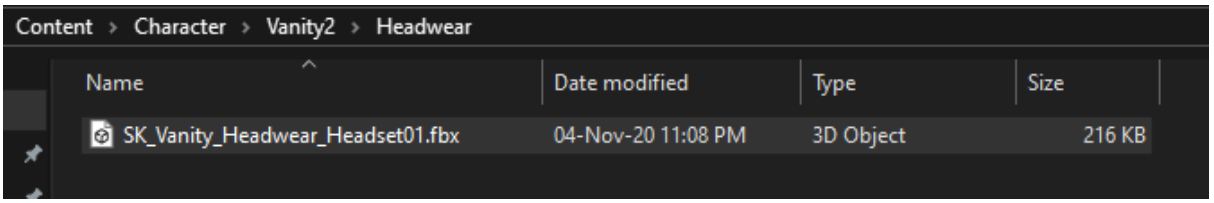
Select the maggot and the skeleton (Shift Select the maggot then dwarf's skeleton) (then press (CTRL+P) and select **Set Parent To (With Empty Groups)**

Test if it's attached properly in pose mode.

Also to rotate around the scene, hold middle mouse and move the mouse left and right.

Here's a gif of my process:

Now delete dwarf's head, select the skeleton and the maggot by pressing A and export the FBX. Call the FBX like this (since we're replacing Scout's AR Headset):
SK_Vanity_Headwear_Headset01.fbx place the fbx inside the **Headwear** folder



Now follow the same steps that i did at page 11 to 18 page to Import and Pak to test it yourself in game.

And here it is!

I hope this helped you out!

If you have any questions, you can DM me on Discord: Pacagma#1515

ROCK AND STONE, BROTHER!

4.9 Uexp-Uasset basics and weapon mod tree transformation – by Drillboy Jenkins

This is a short guide about basics of uasset-uexp files structure. It provides knowledge needed to create “extensive” hex-mods, for DRG. To find examples see guides about more particular stuff like editing upgrades tree or adding baseline weapon properties.

File Summary

The core of a uasset-uexp file pair is the summary in the beginning of the .uasset file. There you can see some generalized data about their contents (metadata if you want an IQ-meter to double its readings).

D: > DRGpacker > input > Content > WeaponsNTools > BoltActionRifle > {} WPN_M1000-UAsset.json > [ ] Objec


```
{
  "FileSummaryOffset": "0x0",
  "FileSummary": {
    "Tag": "0x9E2A83C1",
    "FileVersionUE4": 516,
    "FileVersionLicenseeUE4": 0,
    "CustomVersion": {
      "CustomVersion": null
    },
    "TotalHeaderSize": 24637,
    "PackageFlags": "PKG_FilterEditorOnly",
    "FolderName": "None",
    "NameCount": 449,
    "NameOffset": 193,
    "LocalizationId": null,
    "GatherableTextDataCount": 0,
    "GatherableTextDataOffset": 0,
    "ExportCount": 30,
    "ExportOffset": 20309,
    "ImportCount": 193,
    "ImportOffset": 14905,
    "DependsOffset": 23429,
    "SoftPackageReferencesCount": 0,
    "SoftPackageReferencesOffset": 0,
    "SearchableNamesOffset": 0,
    "ThumbnailTableOffset": 0,
    "Guid": {
      "GUID": "B89E7E41-48975E65-C98C4A90-F7FF3474"
    },
    "Generations": [
      {
        "ExportCount": 30,
        "NameCount": 449
      }
    ],
    "SavedByEngineVersion": {
      "Major": 0,
```


This is an example of a file summary. A part of summary, some of its contents don’t fit the screen. In case of extensive changes you will have to change some of the offsets and counts

here. You can parse a file pair with DRG Parser and open the .uasset with hex-editor and check that these offset and count values are actually found in the file.

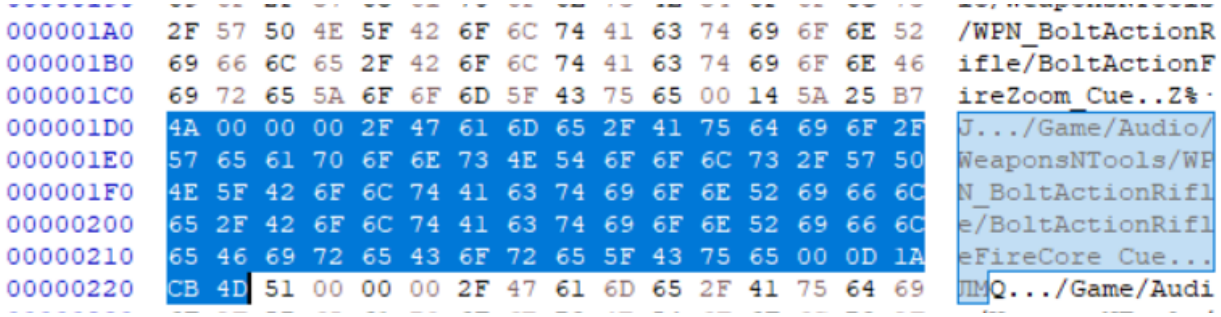
These offset values say where certain blocks start. And counts say how many records they contain. When you change the size of a block, offsets of all the blocks going after are changed. Thus you have to “fix” offsets and counts after “extensive” edits.

Name Map

Names table is what you see next after the summary ends. In parsed .json you will see stuff like this:


```
    {
      "Index": 2,
      "FileOffset": "0x139",
      "Name": "/Game/Audio/WeaponsNTools/WPN_BasicPistol/Pistol_NearEmpty_01_Cue",
      "NonCasePreservingHash": "0xD828",
      "CasePreservingHash": "0xCEAC"
    },
    {
      "Index": 3,
      "FileOffset": "0x183",
      "Name": "/Game/Audio/WeaponsNTools/WPN_BoltActionRifle/BoltActionFireZoom_Cue",
      "NonCasePreservingHash": "0x5A14",
      "CasePreservingHash": "0xB725"
    },
    {
      "Index": 4,
      "FileOffset": "0x1D0",
      "Name": "/Game/Audio/WeaponsNTools/WPN_BoltActionRifle/BoltActionRifleFireCore_C
```


These are name definitions. They are string “global”-IDs spammed into the names table. And then these names are referred in the files via indices in the names map.



This is an outlined name definition found in .uasset file. It has an actual string in the middle of it, prefixed with 4 and postfixed with 5 bytes.

The 5 bytes after the string are null-terminator bytes and 2 hashcodes 2 bytes each. The null-terminator is a byte with value of 0x00 used in most programs to designate end of string. The 2 hash codes can be used by the engine to validate data it reads. But as of now DRG does not validate the codes so you fill them with whatever values you want. I personally use FF FF for name definitions I add to make them easier to find.

The 4 bytes before the string are length mark. In the pic above it is 4A 00 00 00 starting at offset of 0x1D0. At offset of 0x1D4 the string starts and it lasts exactly 0x4A=74 bytes (INCLUDING NULL-TERMINATOR).

Other blocks use names form the names table using their indices in the table. The indices are not explicitly configured in the .uasset, but generated according to order of names (the first name met in the table gets index of 0, the next one 1 and so on).

Editing Name Map

Do not remove/add name definitions in the middle of the table. That will cause indices of all the following names change and you will have to fix every mention of those indices.

If you need to replace a name, then replace its string with a new one and the change size designator if needed.

If you need to add a name, do it at the end of the list. You can find where the table ends using imports offset.

If your changes change net size of the names table and names count, you should

1. Change all the summary offsets that are greater than nameOffset (and a header size field). There are 7 of them at offsets of 24, 61, 69, 73, 165, 169, 189.
2. Change name count values in the summary (there are 2 of them).
3. Change serialOffset field of every exported structure definitions in export map.

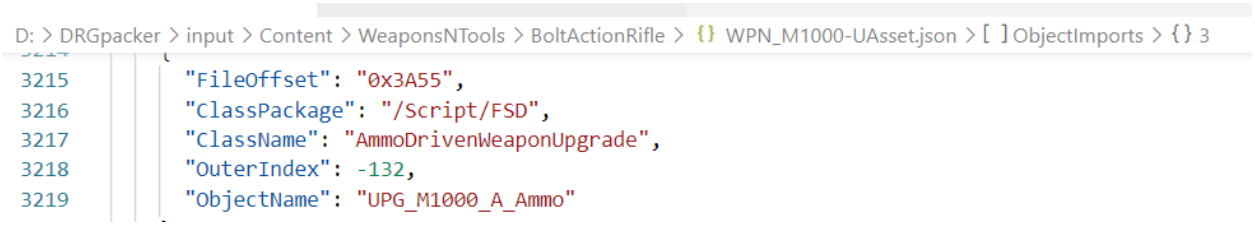
Import Map

Some of the structures in the files use not only names from name map, but also refer stuff from the import map.

```
3606        {
3607          "FileOffset": "0x4075",
3608          "ClassPackage": "/Script/FSD",
3609          "ClassName": "CombinedUpgrade",
3610          "OuterIndex": -136,
3611          "ObjectName": "UPG_M1000_B_Spread"
3612        },
3613        {
3614          "FileOffset": "0x4091",
3615          "ClassPackage": "/Script/FSD",
3616          "ClassName": "CrosshairAggregator",
```

Each import definition combines class package, class name and object name, referring them via indices of name map. Outer index refers another record in import map. But why is it negative? It works pretty much the same as in name map. It is defined by order of record in the import map. But starts with -1 and goes negative further. So that the first record in import map is referred as -1, second as -2 and so on.

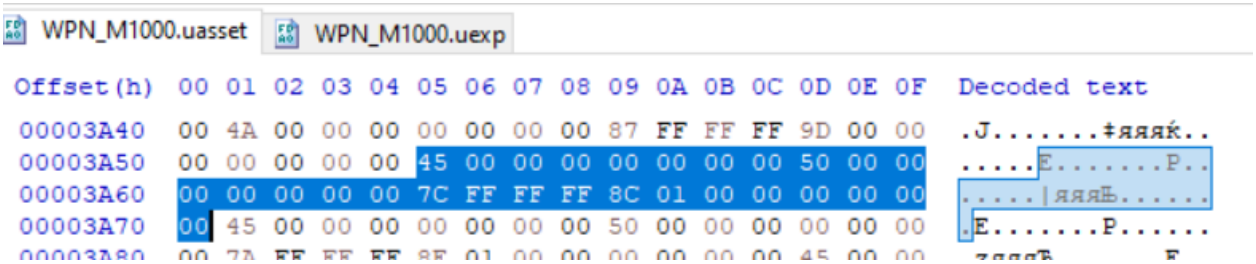
In order to get the needed index fast, I suggest using Visual Studio Code to read the .jsons. When your cursor is in one of the records, you can see its “traditional” index and even traverse to needed record by index clicking it.



If your cursor is placed in the record you want to refer, increase the traditional index by 1 and make it negative. That way, M1K’s A tier ammo upgrade in the pic above should be referred as -4.

Editing Import Map

File offset fields in the .json are generated by the parser and tell you where the import definition record is.



This is what the definition looks like in the .uasset. It starts with name index of class package. At relative offset of 8 (aka 8 bytes from the record’s start, asl 0x3A5D) we have index of class name. Then at relative offset of 16 we have a negative int. That is the outer index for this record. Some records have zero outer index. The last thing here starts at rel. offset of 20 – object name via index in name map.

Thus every import definition record is 28 bytes long and they don’t need a length designator.

Editing the import map is pretty much same as editing name map. The differences are:

1. Size changes only count changes too.
2. You don’t need to change import offset in the summary, so only summary offsets at 24, 61, 73, 165, 169, 189 should be changed.
3. Imports count is at 65.

REMEMBER: don’t remove records, only adjust existing or add at the end of the list (use export offset from the summary to find it).

Exported stuff (.uexp contents and Export Map of .uasset)

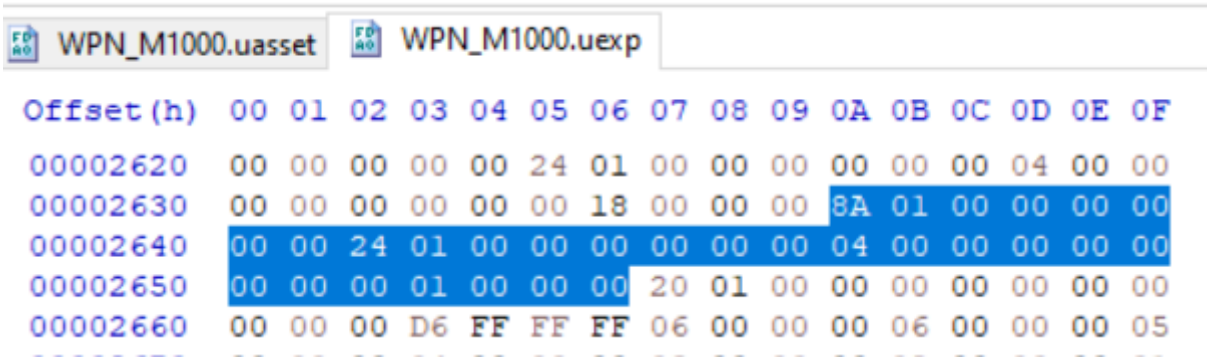

```
4558    ],
4559    "ExportMapOffset": "0x4F55",
4560    "ObjectExports": [
4561      {
4562        "FileOffset": "0x4F55",
4563        "ClassIndex": -29,
4564        "ClassIndexStr": "Function",
4565        "SuperIndex": 0,
4566        "SuperIndexStr": "Not_UStruct",
4567        "ThisIndex": 1,
4568        "TemplateIndex": -69,
4569        "TemplateIndexStr": "Default__Function",
4570        "OuterIndex": 7,
4571        "OuterIndexStr": "WPN_M1000_C",
4572        "ObjectName": "ExecuteUbergraph_WPN_M1000",
4573        "ObjectFlags": "RF_Public",
4574        "SerialSize": 565,
4575        "SerialOffset": 24637,
4576        "ExportFileOffset": 0,
4577        "ForcedExport": false,
4578        "NotForClient": false,
4579        "NotForServer": false,
4580        "WasFiltered": false,
4581        "PackagedGuid": {
4582          "GUID": "0-0-0-0"
4583        },
```


This is a pic of export map start and its first element. The hexadecimal file offset is the export definition offset in the .uasset. Serial size is amount of bytes in the .uexp storing the exported structure and serial offset is an offset in “through enumeration” – it would be the offset of exported stuff if you concat the .uexp at the end of the .uasset. You can also interpret that as .uasset length (header size from the summary) + .uexp offset. I have not encountered a need to work with other fields yet, let’s proceed to corresponding part of parsed .uexp contents.

```
5641    {
5642      "Object Flags": "RF_Public, RF_Transactional",
5643      "Parent": "BoltActionWeapon",
5644      "Outer": "UObject",
5645      "Template": "Default__BlueprintGeneratedClass",
5646      "Class": "BlueprintGeneratedClass",
5647      "Object": "WPN_M1000_C",
5648      "File Offset": 9757,
5649      "Struct Size": 540,
5650      "Potential Properties": [
5651        {
5652          "Property": "SimpleConstructionScript",
5653          "Type": "ObjectProperty",
5654          "Offset": 9757,
5655          "Size": 4,
5656          "Array Index": 0,
5657          "Value Offset": 9782,
5658          "Data Value": "Export:SimpleConstructionScript_1"
5659        },
5660        {
5661          "Property": "UberGraphFunction",
5662          "Type": "ObjectProperty",
5663          "Offset": 9786,
5664          "Size": 4,
5665          "Array Index": 0,
5666          "Value Offset": 9811,
5667          "Data Value": "Export:ExecuteUbergraph_WPN_M1000"
5668        }
5669      ]
5670    },
```


Pic of a structure the parser could partially crush into some readable data. Struct size is the same value as the serial size in the corresponding export definition (last 2 pics scope different objects, so they differ between them). Its substructures are properties, here we have 2 detected by the DRG Parser tool. Propertires in the .uexp tell their name and type as indices for names table. Then they define size of their body, which can actually be another substruct with other smaller structs inside you will have to “parse” manually because the parser can’t do it.

Following pic shows the way this UberGraphFunction looks inside. The last 4 bytes stand for object property with index of 1. How does it work? In this particular case UberGraphFunction’s export defnintion in the map goes first. The “guts” of export definitions are not studied by me yet so I can’t tell if this index goes from order or can be configured inside the definition.



Editing the Exports

There is not much I can say about it. At the offset of property start you should see index-ref to property name, then type and size designator. Further you will have value which can be an import/export index-ref for ObjectPorperty, simple 4-byte integer or a cascade of other propertires like in case of UpgradeTiers building upgrades tree.

When you edit (also delete or add) propertires, you must:

1. Update their size designators in the .uexp.
2. Update serial size in export definition of structure you edited.
3. Update serial offsets for all the following export definitions.
4. Not-really-needed as my experience shows, but I do it for some reason. One of the summary offsets (bulk data offset at 169) is pointing at the end of combined “through enumeration” of .uasset and .uexp. I have heard something about .ubulk files, guess it is used for them. If you find an .ubulk for some reason, then here is the start for your research.

Use Automation Tools

Like OffSetter I’ve made for this stuff. If you know what happens in the files, it will be easier to understand what your computer wants of you when you use this piece of garbage software engineering.

Mod trees

This tutorial is here to tell you about the way upgrades get into mod trees of equipment. It assumes you know the basics of working with .uasset-.uexp file pairs.

Things to do

All changes to a mod tree can be made as a composition of

1. Removing mods from the tree.
2. Swapping existing mods.
3. Adding new mods (be they borrowed from another weapon or you have made them on your own).
4. Replacing existing mods with new ones.

If you need to remove a mod, then shorten the corresponding mod tier in UpgradeTiers structure. Don't remove them from the .uasset file completely, fixing the index mess will take a lot of time! Proceed to the UpgradeTiers structure.

If you want to swap existing mods, you can swap their import references in UpgradeTiers. Proceed to UpgradeTiers structure.

If you want to replace existing mod with a new one, you need to replace 2 name defs. See steps 1-3 of following guide, but in step 1 replace names of existing upgrade you delete instead of adding new ones. For step 3 you will need net size and count changes.

In order to extend a tree, you should read the whole guide.

On the Way to the Tree

Making an upgrade tree use an upgrade involves some routine to do before pasting a reference in the UpgradeTiers structure. All (to my knowledge) upgrades are stored in UPG files with pretty descriptive names. Keep in mind that GSG seem not to be keen on updating filenames even if they get a bit outdated. For example, M1K has experienced some modswaps in vanilla and now many UPG-files seem to be mis-named compared to how the upgrades are placed in its tree.

Step 1. Name definitions. Nothing non-trivial. If you need an upgrade be in the tree of a weapon, the weapon's WPN_xxx.uasset needs to have 2 names defined for the upgrade on its own. One is full filename in the pak file system. And another one is name of the upgrade object, which is seems to be the filename without full path. For M1K's damage upgrade these are "/Game/WeaponsNTools/BoltActionRifle/UPG_M1000_A_Damage" and "UPG_M1000_A_Damage".

Step 2. Upgrade subclass name definition. If you check the import map, you will see that different upgrades can have different class names in import definitions. If the WPN you are adding an upgrade to does not have needed class name in name map, I guess you should also add it to refer further in import def for the upgrade. If you can test using "wrong" subtype and it turns out to work – ping or DM Drillboy Jenkins on discord.

Step 3. Offsetting. You can outline the new chunk in hex editor to see its length (don't forget to switch it to decimal). 2 or 3 names are added, depending on step 2.

Step 4. Import definitions. For the 2 name definitions from step 1 we also need make 2 import definitions. You can either reuse similar definitions replicating them at the end of the map or make them on your own.

1. Upgrade import.
    a. Class package should be index of "/Script/FSD" in the name map.
    b. Class name depends on upgrade, you can check similar upgrades or WPN where it is originally ised to find it out. Once again, it is name map index.
    c. Outer index should refer import index of filename import definition (indexation from first import definition starts from -1 and is decremented for each next element). You can leave it for a while and return when you have pasted filename import definition. Moreover, you can leave it whatever it is, finish

next import, proceed to offsetting and only then change the index, with the imports being seen in the new .json.

d. Object name is index ref to the “UPG” name def (like “UPG_M1000_A_Damage”).

2. Filename import.
   a. Class package is “/Script/CoreUObject”.
   b. Class name is “Package”.
   c. Outer index is 0.
   d. Object name is index ref to the “filename” name def (like “/Game/WeaponsNTools/BoltActionRifle/UPG_M1000_A_Damage”).

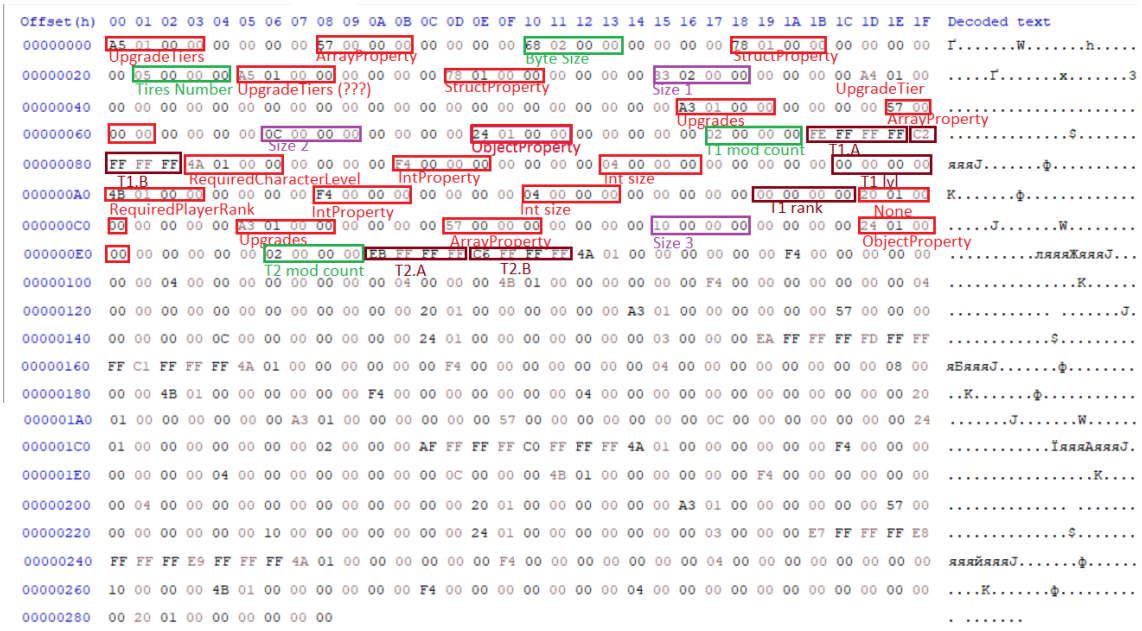
Step 5. Offsetting. Step 4 adds 2 import defs and 56 bytes (28 per each).

Done!

The instructions above use word “filename” referring to fact one of the string looks like a filepath. I have no idea how correct it is in terms of what happens inside the engine.

The UpgradeTiers Structure

UpgradeTiers is a property of “UpgradableItemComponent”, exported to WPN_xxx.uexp.



Used my paint MadSkillz that night. It is a scope of UpgradeTiers structure copied from its definition to the end.

Red highlights are things you should not change (at least I have not advanced far enough to see controllable and beneficial results from it). Most of them are names from name map.

Brown highlights are numbers you can tweak. TX.Y are import map indices of mods. As you can see they are organized in arrays corresponding to their position in mod tree. Seems that there is a rudimentary player rank limitation on upgrade tiers (all are zeros).

Green highlights show what you have to tweak here if you change sizes. The first green highlight is byte size of the body that takes all the scope from relative offset 0x21 to the end.

Purple highlights seem to be size marks. Size 1 changes between different files like if it was a size of something. Sizes 2 and 3 make sense as size of array properties value sizes. However, everything seems to be fine if you ignore adjusting the purple-highlighted sizes.

With all that information you could be able to swap mods between positions (exchanging indices), move mods from one tier to another (cutting import index from one array and appending it to other with proper changes to count marks).

Remember to make offset corrections and upgradable's serial size tweak if you change the size of UpgradeTiers!!!

Limitations

1. You can extend tiers count, but 6th tier (and chances are all the futher) are not functional.
2. Putting 4 mods in a tier worked fine for me. But I have no idea how to pick 5th mod. I have not tried it yet but chances are it will be off the borders of the screen.

4.10 Texture Replacement – by Pacagma

Welcome to my second guide!

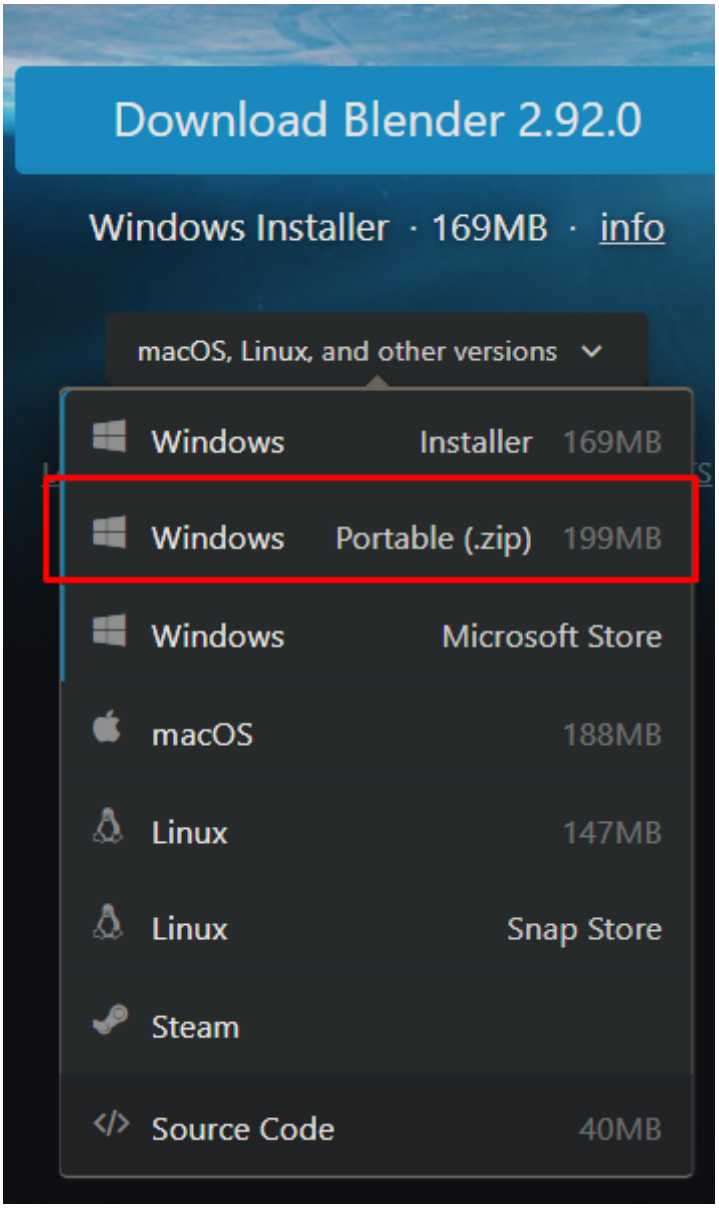
(If the gifs don't work view the guide here
https://docs.google.com/document/d/1eH1xUuTgzz_RbUZftyb9BQ85vD2_QgJa5XV5Lyeh9PE/edit)

Today I'll teach you how to do Custom Texture Replacement for DRG.

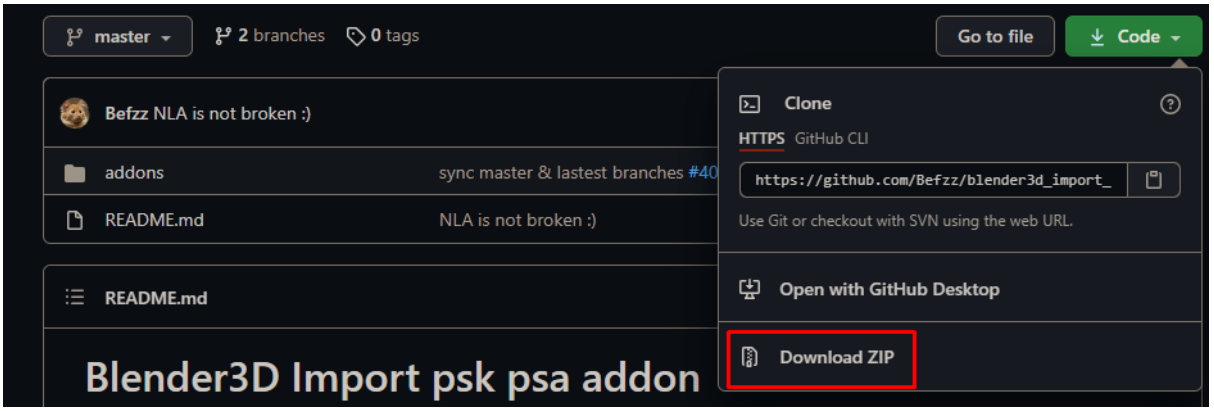
KEEP IN MIND THAT THIS ONLY WORKS WITH THE MODELS WITH NO SKELETON MESH AT THE MOMENT!!!

You'll need a Free 3D software called **Blender**, go ahead and download it.

(https://store.steampowered.com/app/365670/Blender/ or
https://www.blender.org/download/)



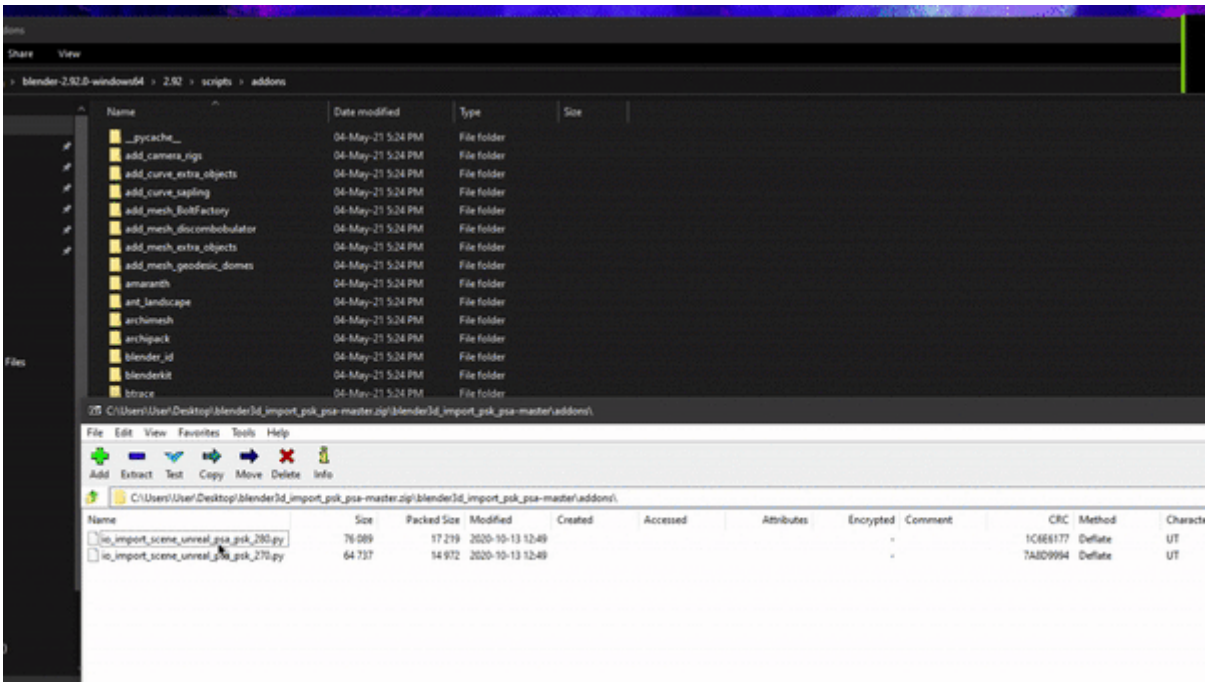
Now to import the DRG models into blender you'll need an Addon called **Blender 3D import psk psa addon,** download it here. (https://github.com/Befzz/blender3d_import_psk_psa)



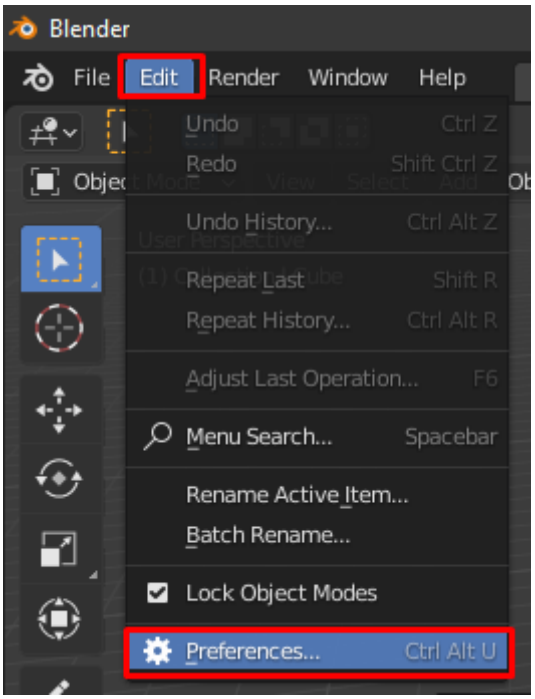
Now navigate to the folders called **2.92/scripts/addons**

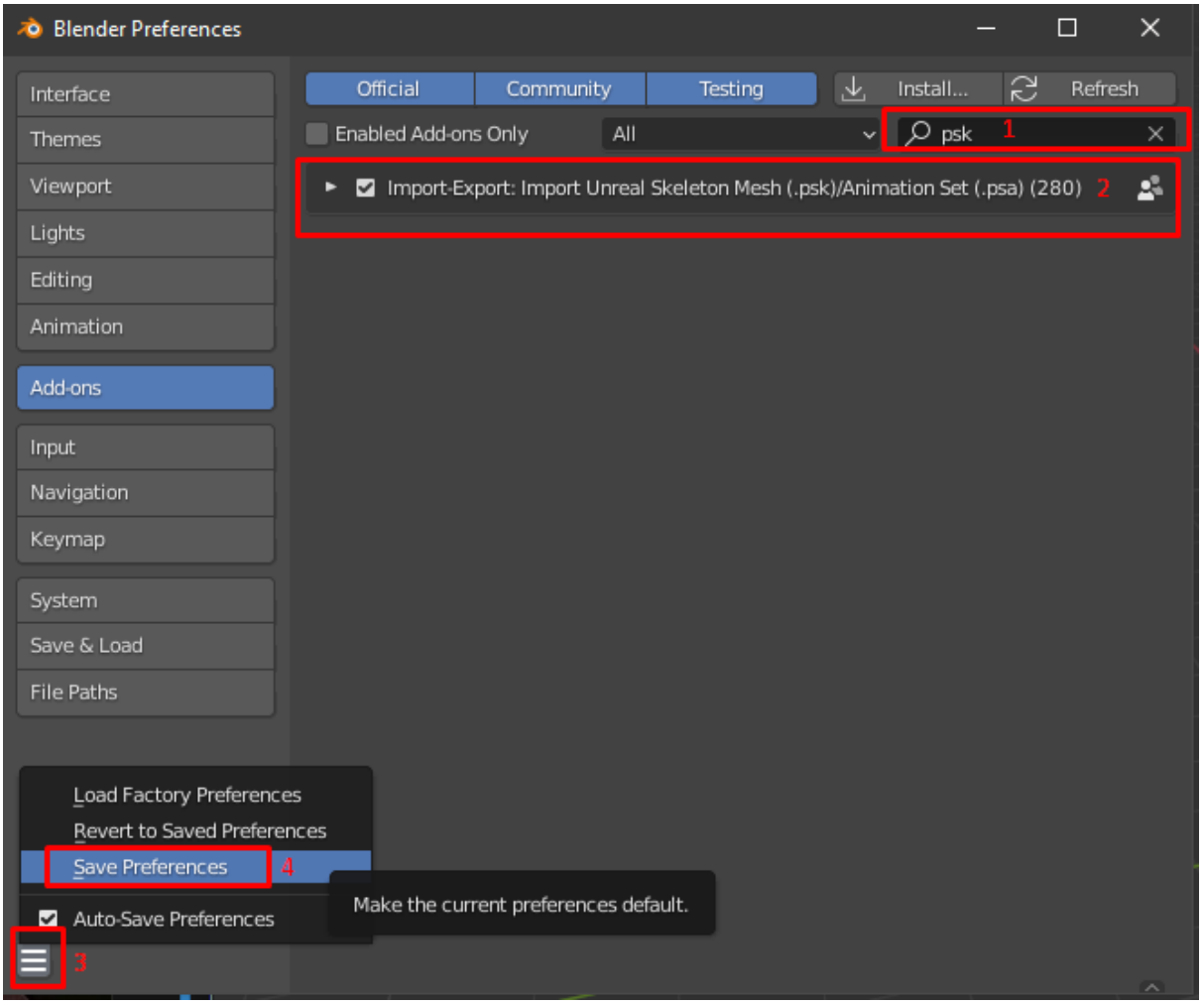
Extract the **blender3d_import_psk_psa-master.zip** anywhere then in addons folder drag and drop the **io_import_scene_unreal_psa_psk_280.py** into **2.92/scripts/addons**

GIF:

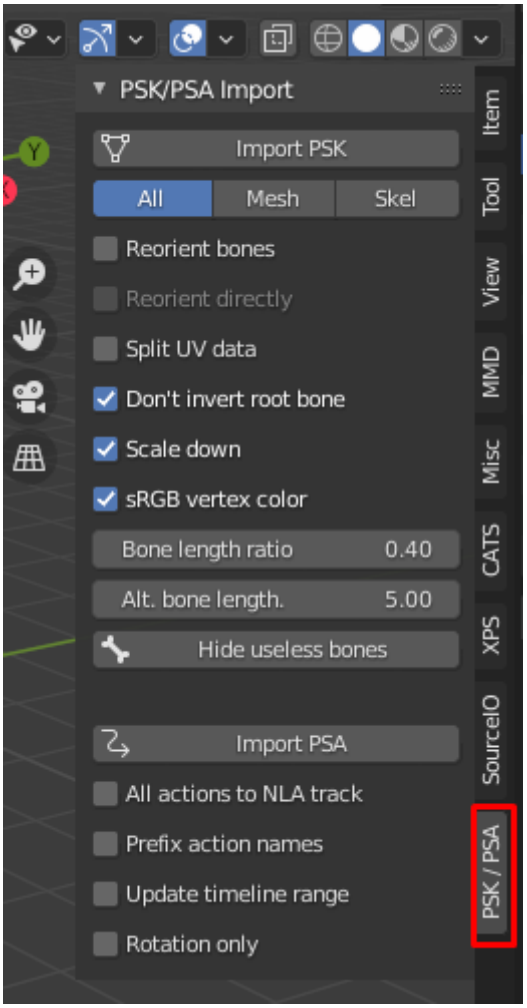


Now to enable the addon in blender do the following:





When that's done, press N on your keyboard and the addon should be on the right side, the last tab.



Now we need to extract the models with a tool called **UModel,** download it here. (https://www.gildor.org/en/projects/umodel)

Scroll down to downloads and pick the **Win32 version**

Downloads

Win32 version	Size 964.3 KB	Updated 24 October, 2020 - 16:04	804668 downloads
Linux version	Size 537.76 KB	Updated 19 October, 2020 - 13:45	25038 downloads

Readme with changelog

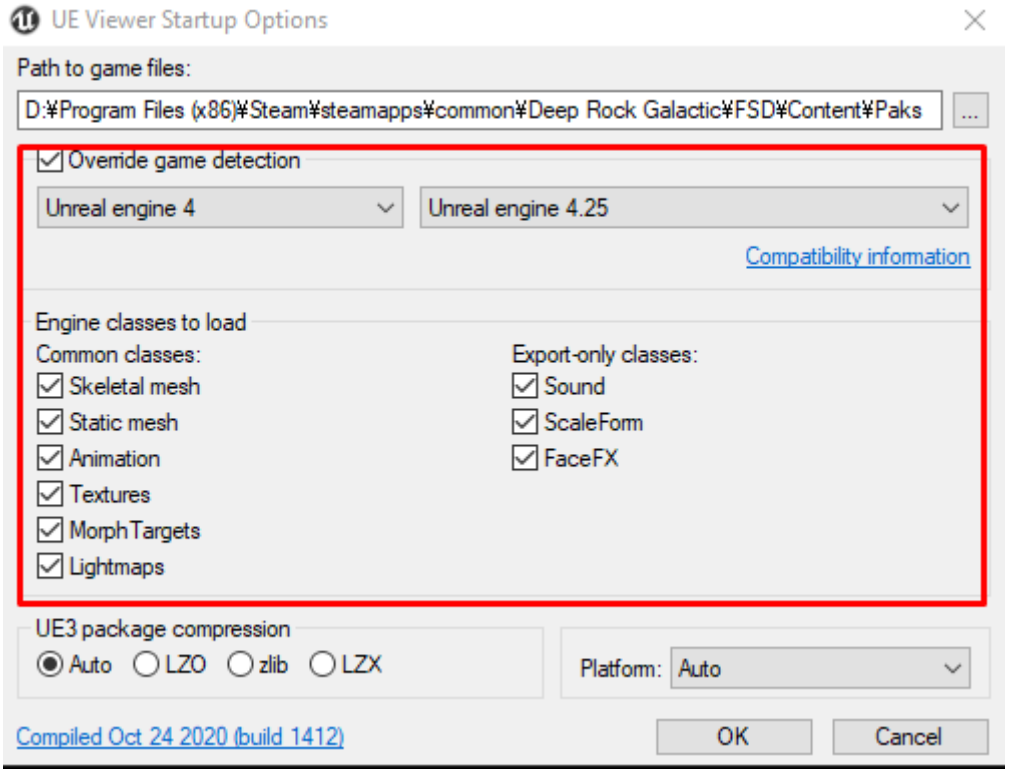
Extract the **umodel_32.zip** anywhere, open the folder and open up the **umodel.exe**

Now do the following:

1. In **Path to game files .** Paste the directory of where your **FSD-WindowsNoEditor.pak** is.

(depends on what drive is the game installed at) Mine is in : **D:\Program Files (x86)\Steam\steamapps\common\Deep Rock Galactic\FSD\Content\Paks**

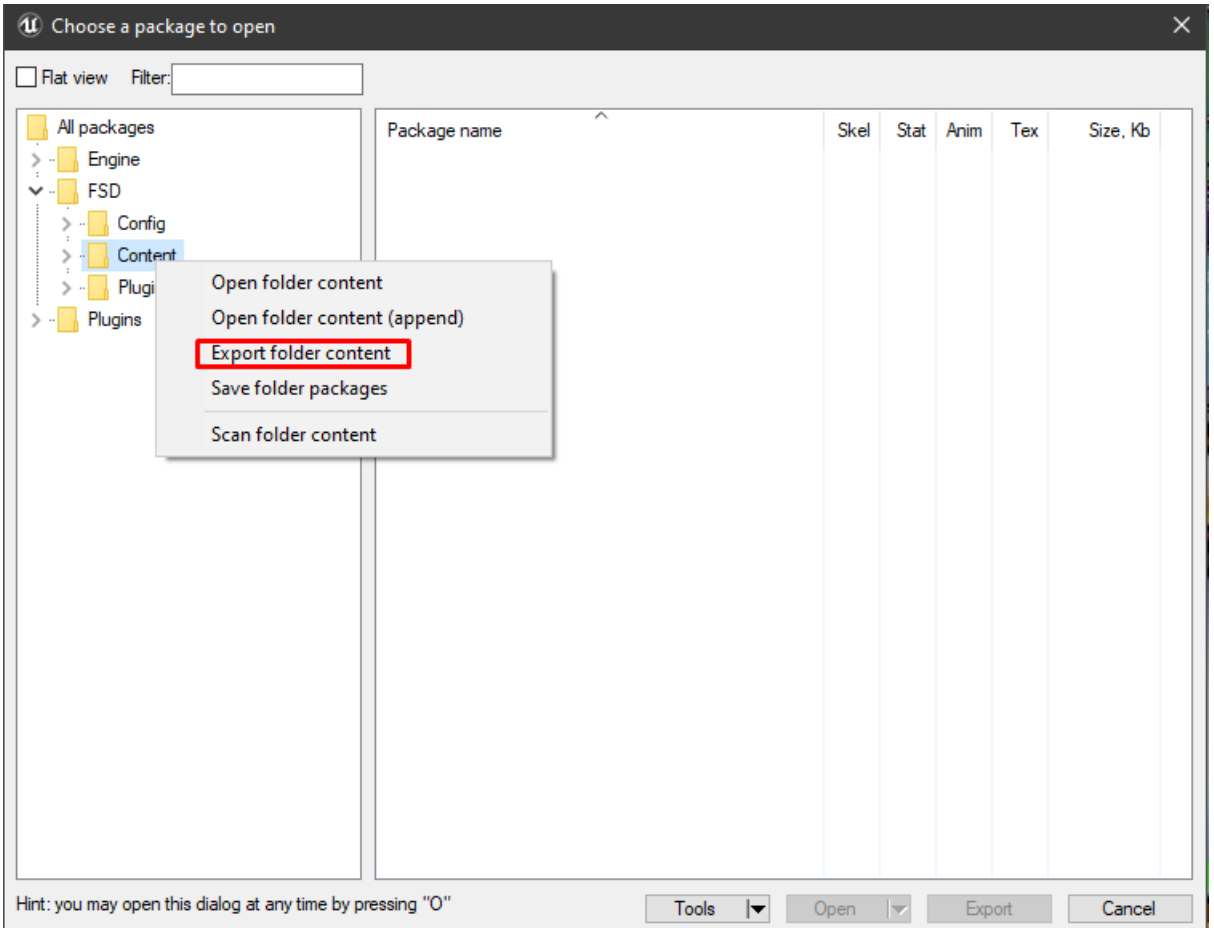
Next do the following as what i've did in the picture:



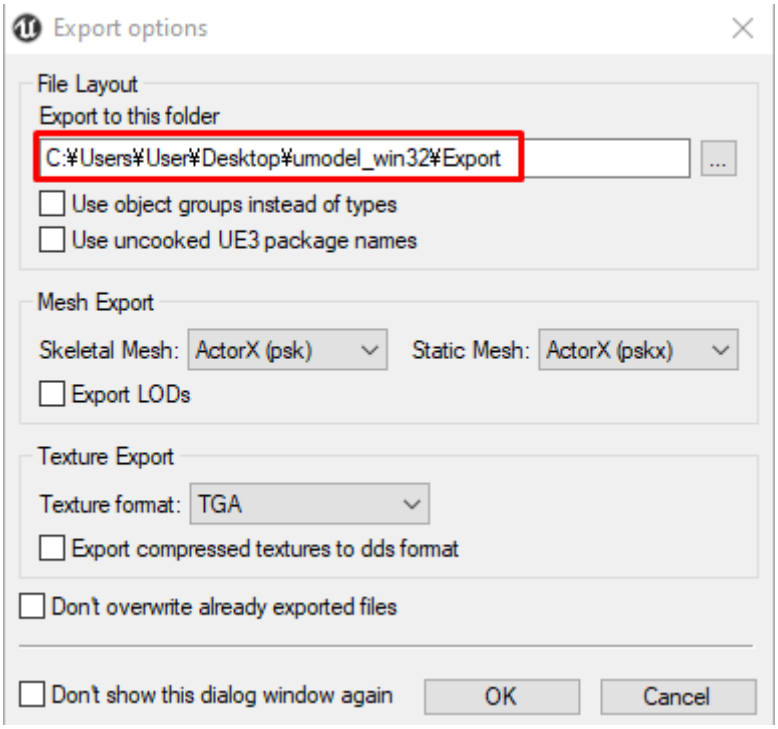
(I pretty much export every single thing from the game, that's why i ticked **Sound** with it too, for **Scale Form** and **FaceFX**, i have no idea what they do, but i still ticked it because why not?)

Next Click **OK**

Now expand **FSD** and you'll find the folder called **Content.** Right click on it and click **Export folder content.**



You'll have the **Export options** window.



Make a folder called **Export** inside **umodel_win32** folder to be more organized.

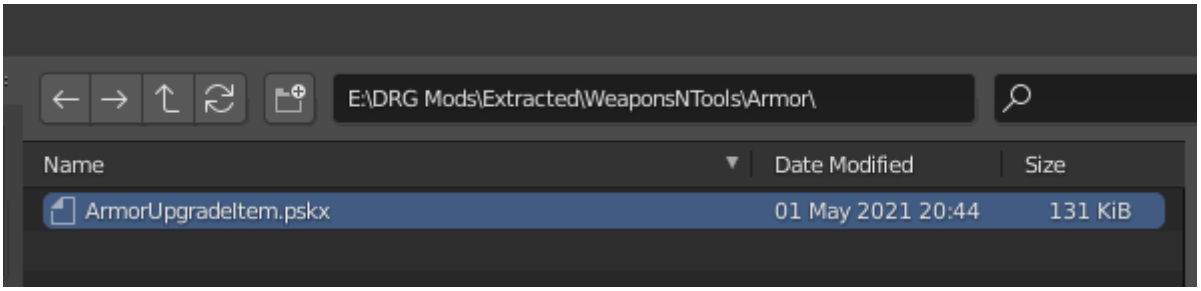
Click **OK** and wait for it to complete the export.

Time to replace a custom texture.

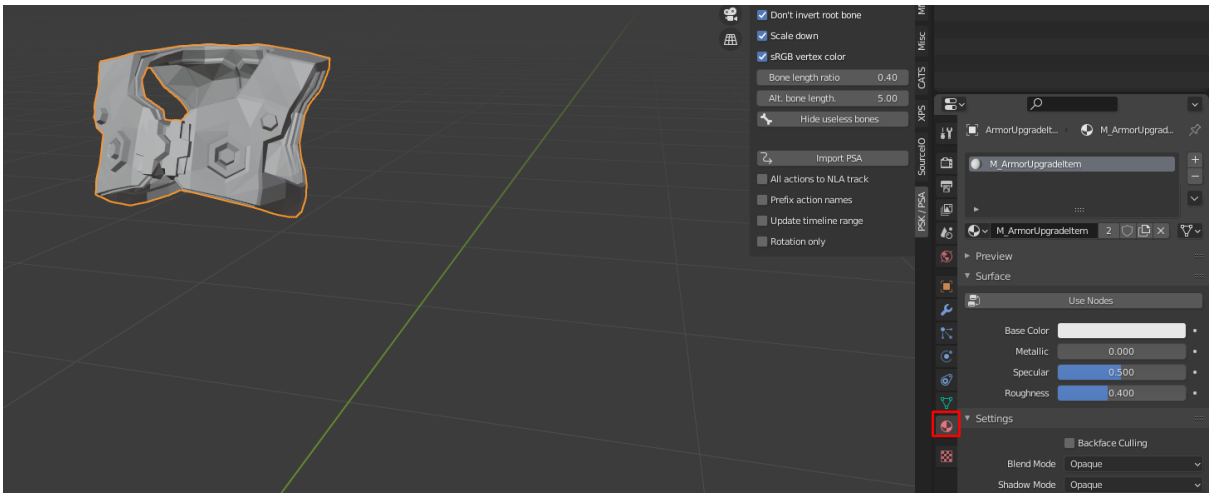
Be sure to delete everything from the scene first, Double tap A then Del

Let’s say we wanna change textures of the armor model upgrade in the terminal

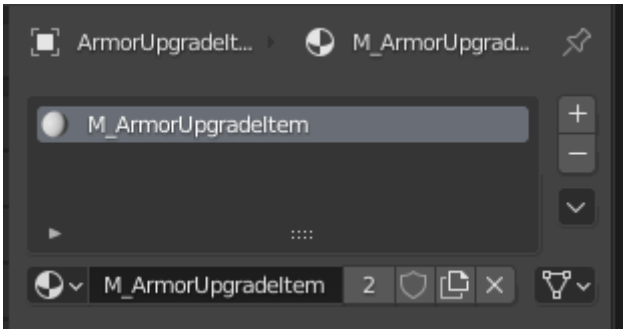
Click Import PSK then look in WeaponsNTools>Armor for the file called **ArmorUpgradeItem.pskx** and import it.



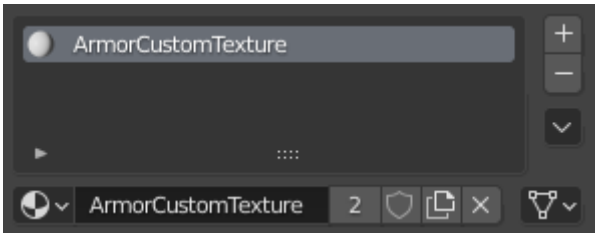
Now go to this material tab



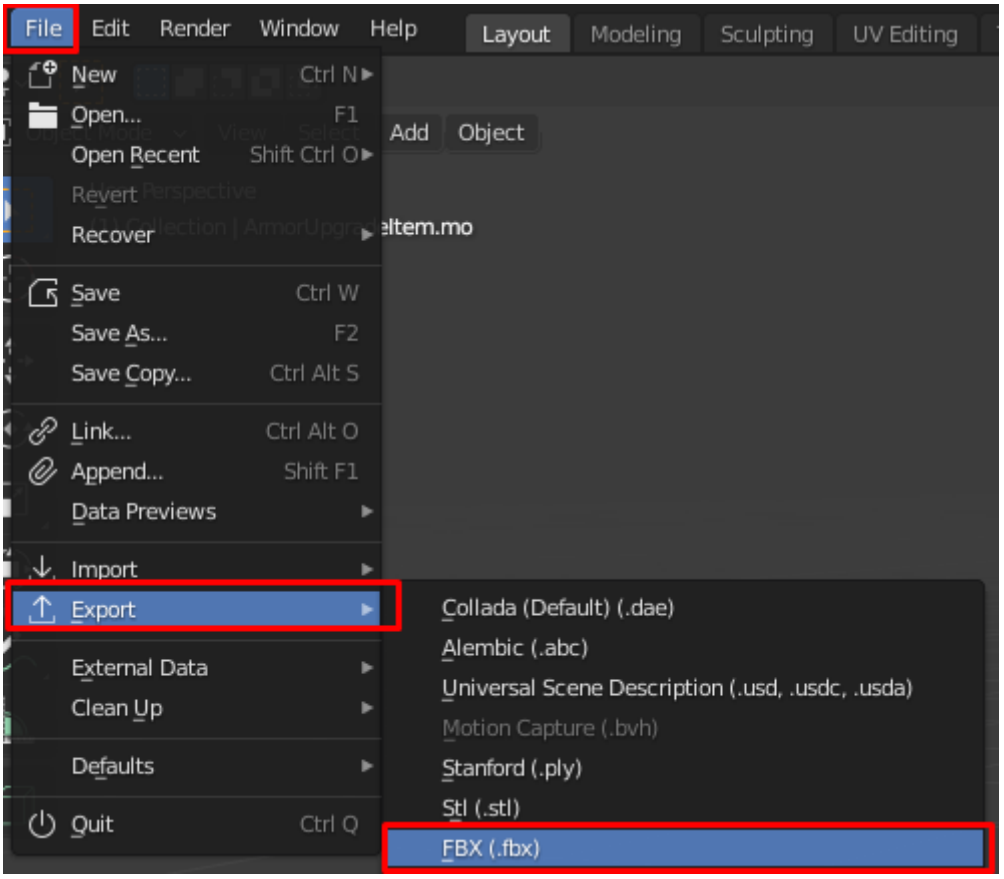
Now when you're on this section:



Change the **M_ArmorUpgradeItem** texture name to something like **ArmorCustomTexture**



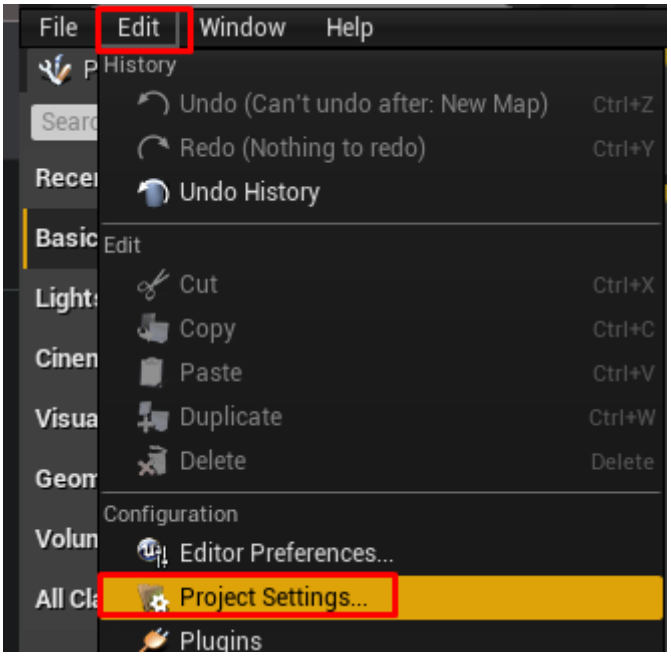
Now time to export the model, highlight the model in blender then go to File>Export>FBX(.fbx)



Export it anywhere and make sure the name is same as the model's name.

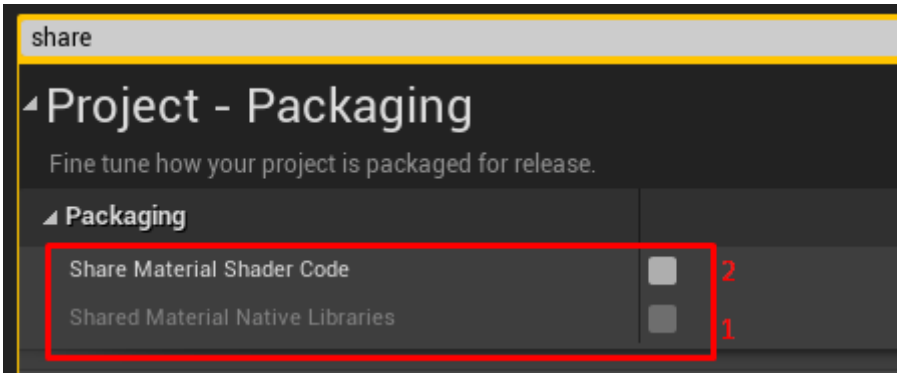
----------Now let's open Unreal Editor.

Go to Edit>Project Settings



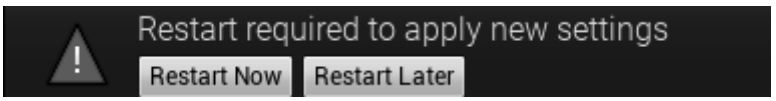
And search for “share”

Then uncheck these Two boxes.

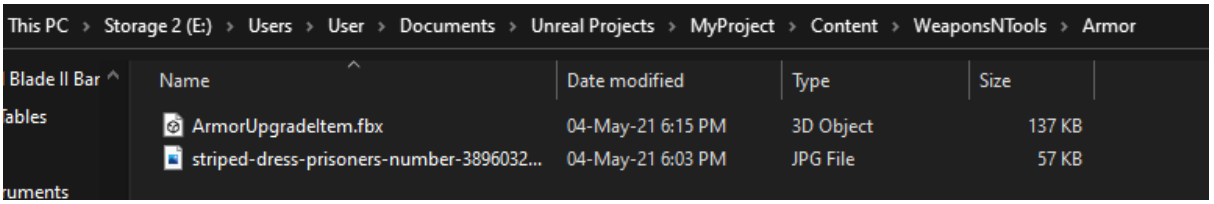


Reason why we need to uncheck these, it’s because if you don’t uncheck these, the game won’t load your custom textures.

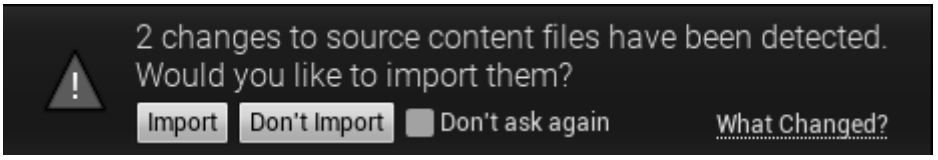
It’s gonna ask you to Restart unreal to apply the settings, Click **Apply Now**.

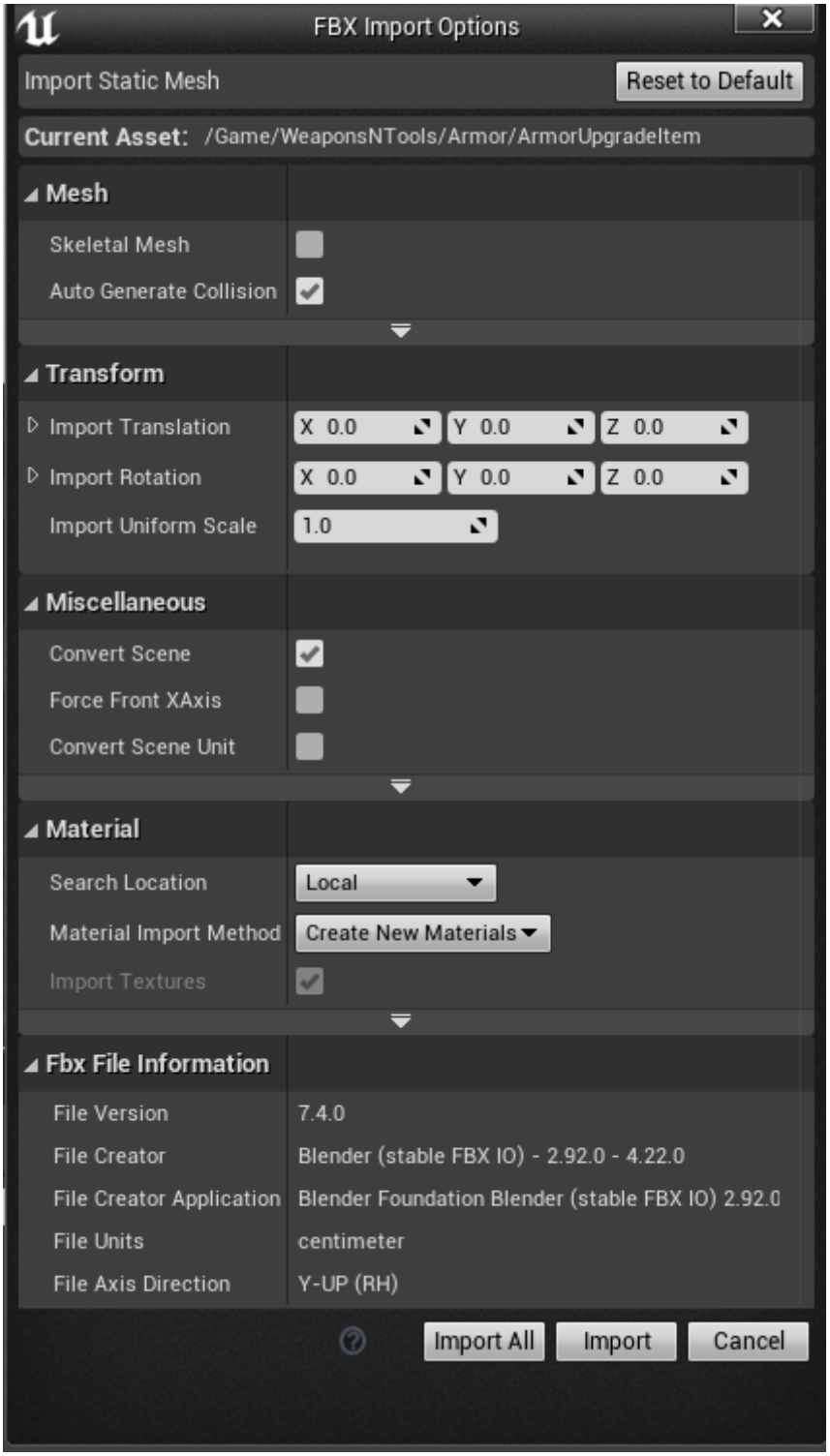


Now place the FBX file and the custom texture that you wanna use in your Unreal Project.

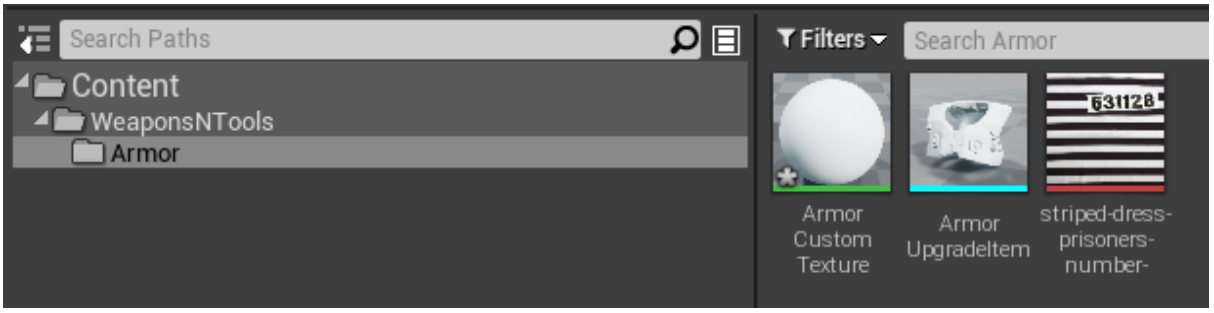


Now going back to Unreal these messages will appear, just Click **Import.**

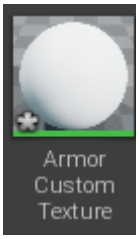




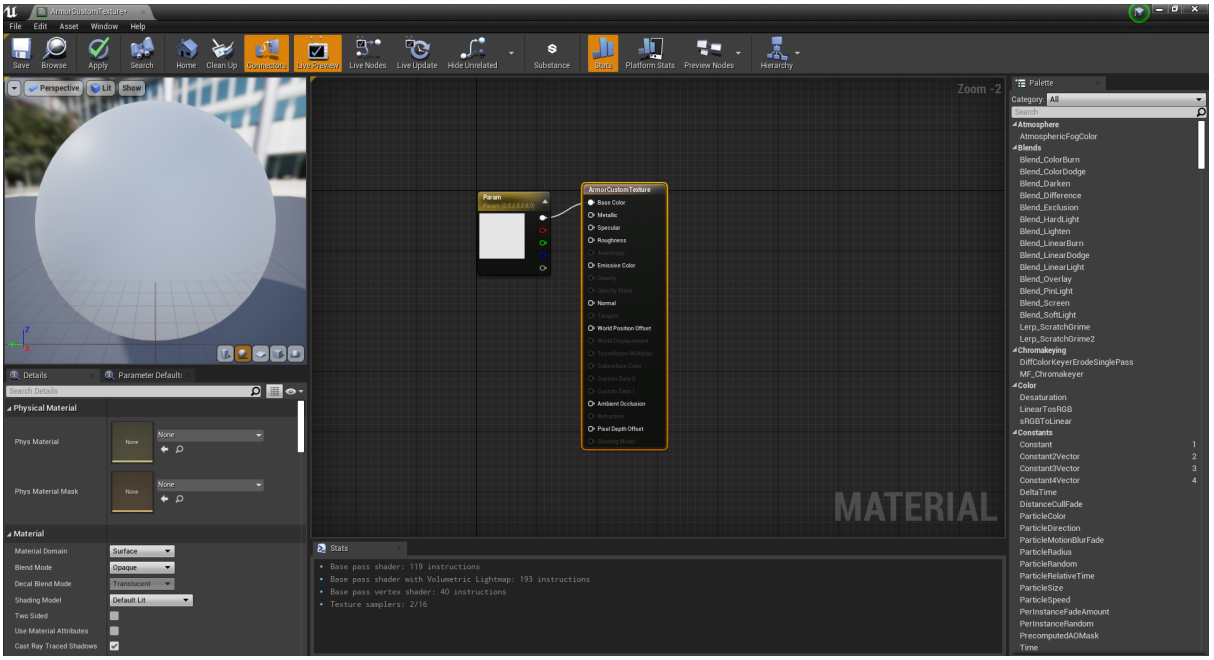
You should have now have this in your Content folder.



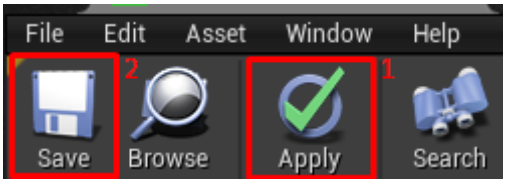
Double click on the material.



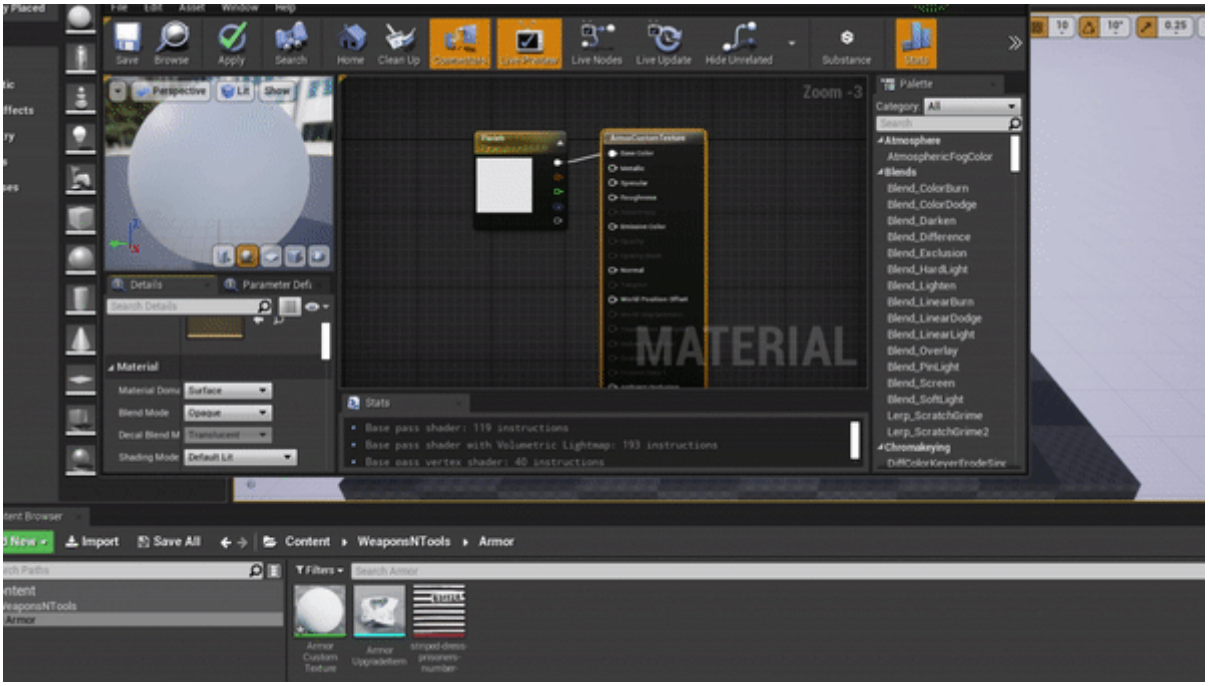
A Material tab will open which looks like this



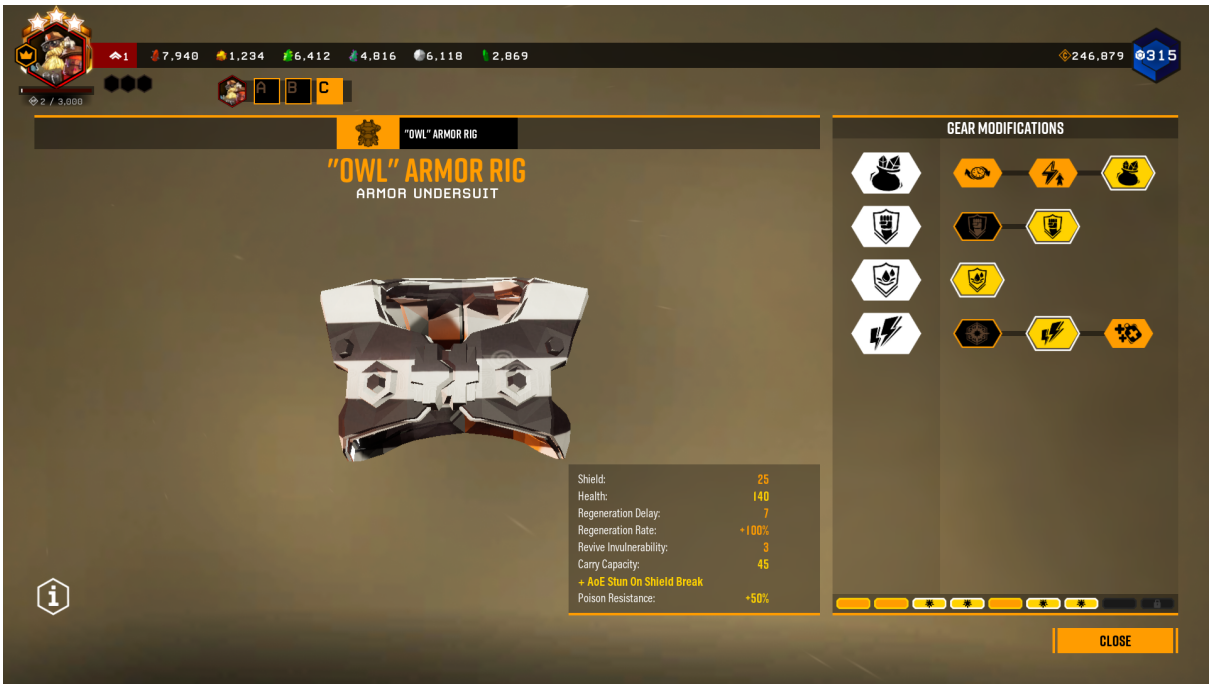
Delete the white color in the material, then drag and drop your custom texture in the material and connect the **RGB** node to **Base Color** when you're done with that Click **Apply** then **Save.**



GIF:

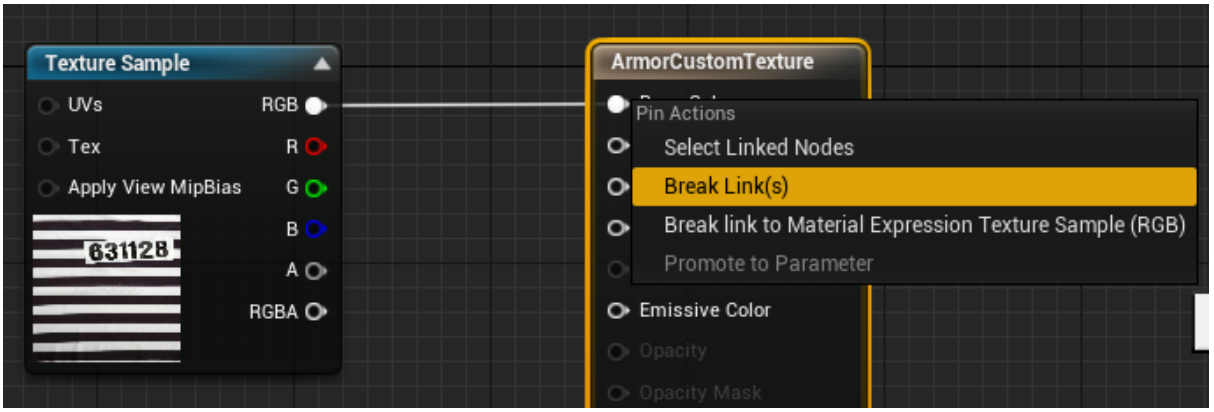


Pak the files and test it in the game!

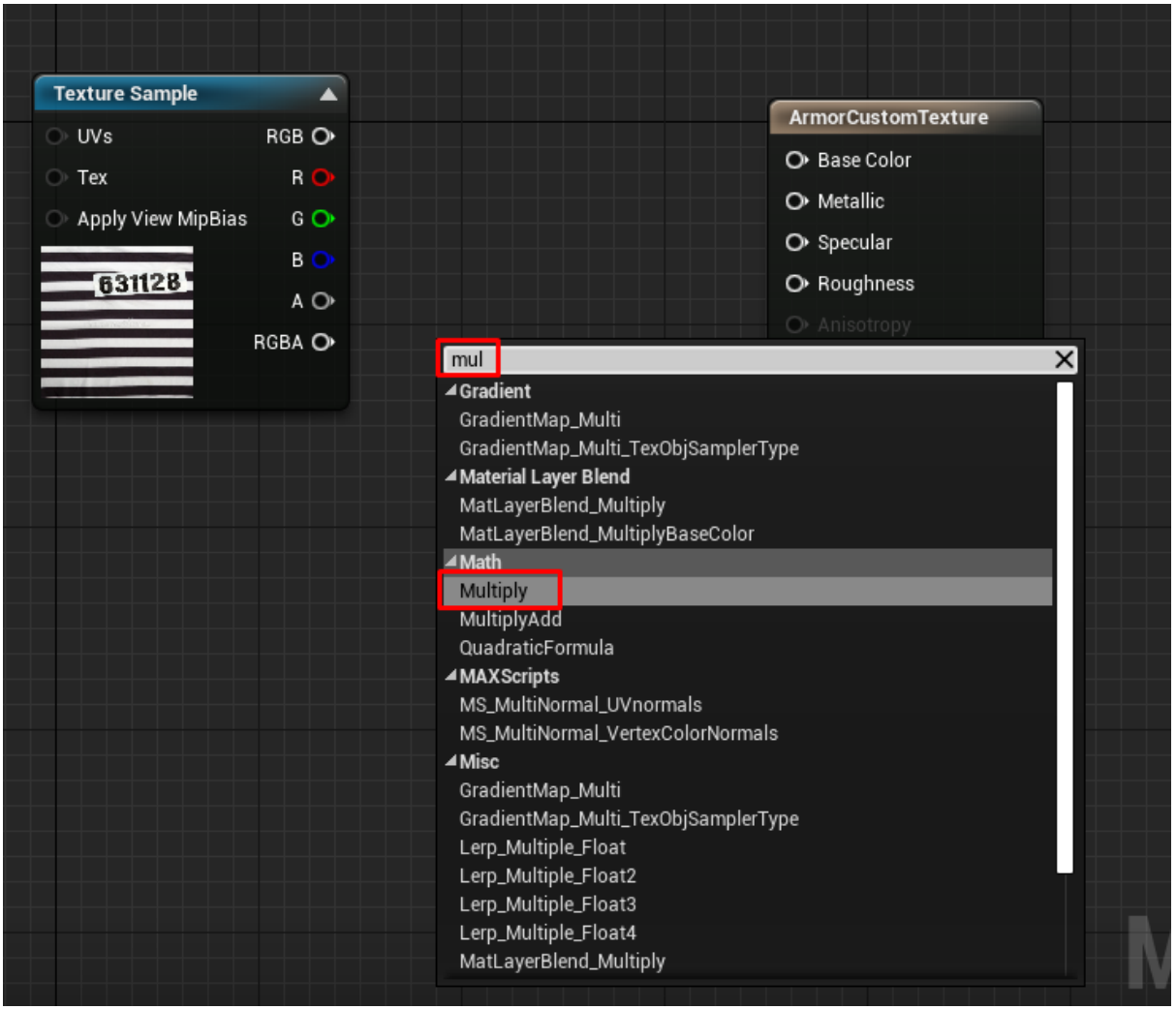


Before I end this guide, i'll teach you how to make a simple Emissive, Emissive means like a glowing texture.

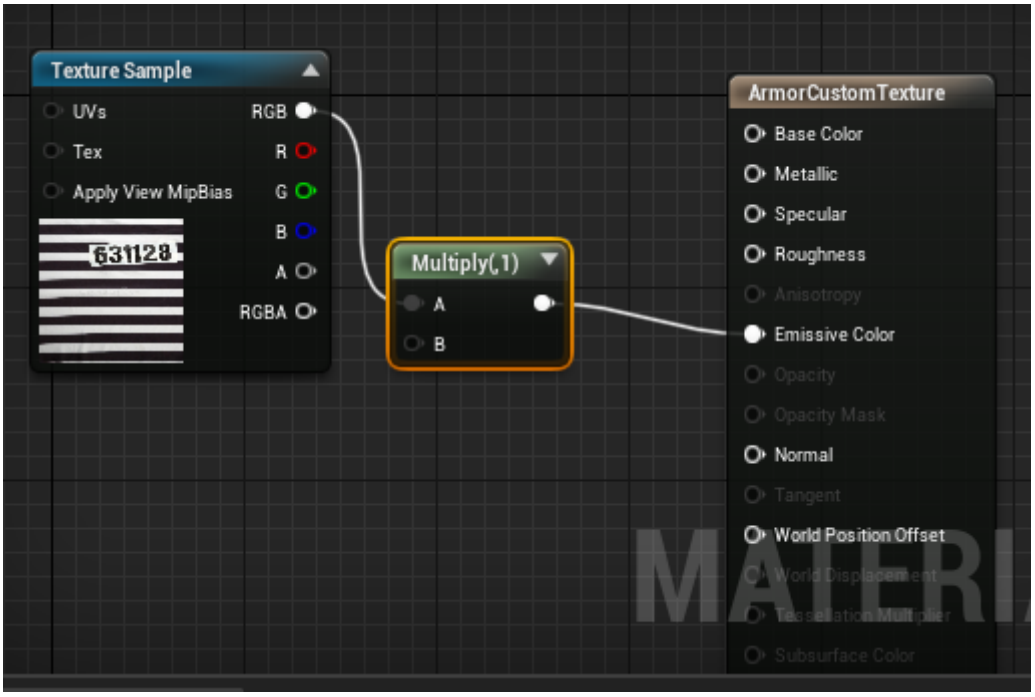
Right click on the **Base Color** and Click on **Break Link(s)**



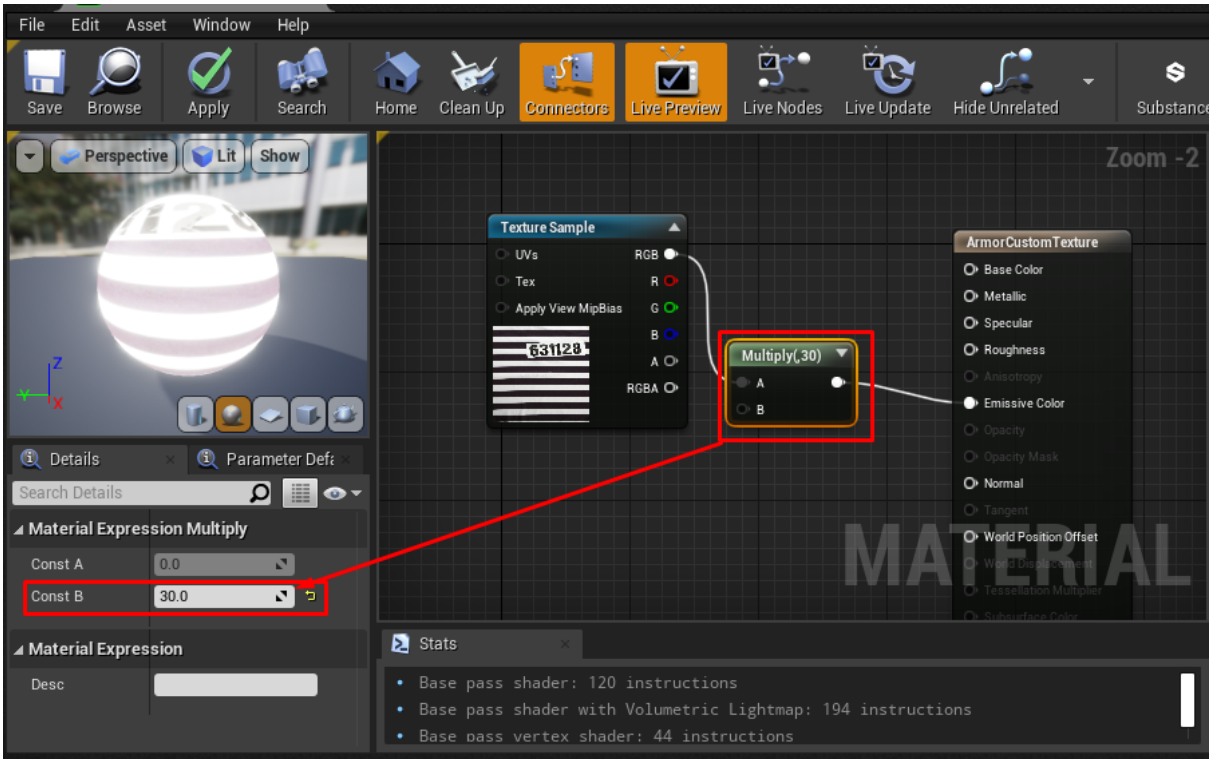
Now left click in the black bars and search for a node called **Multiply.**



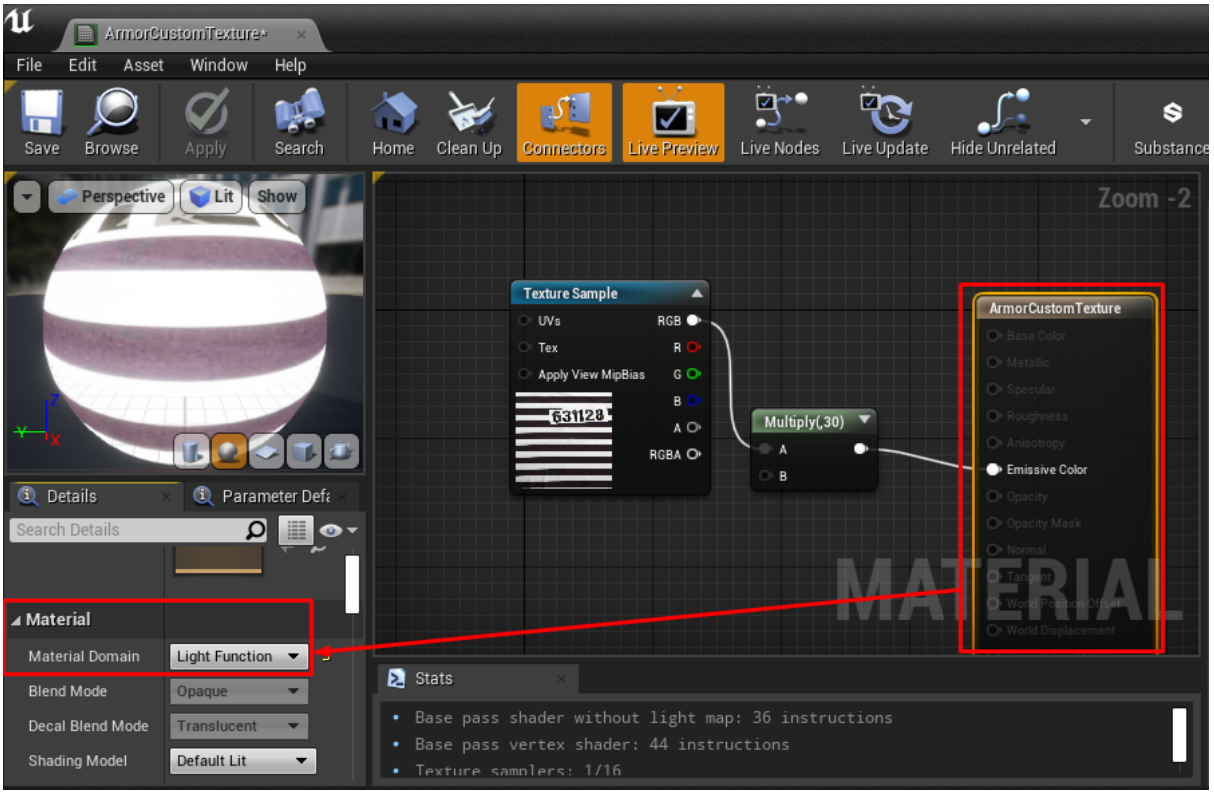
Connect the nodes like this.



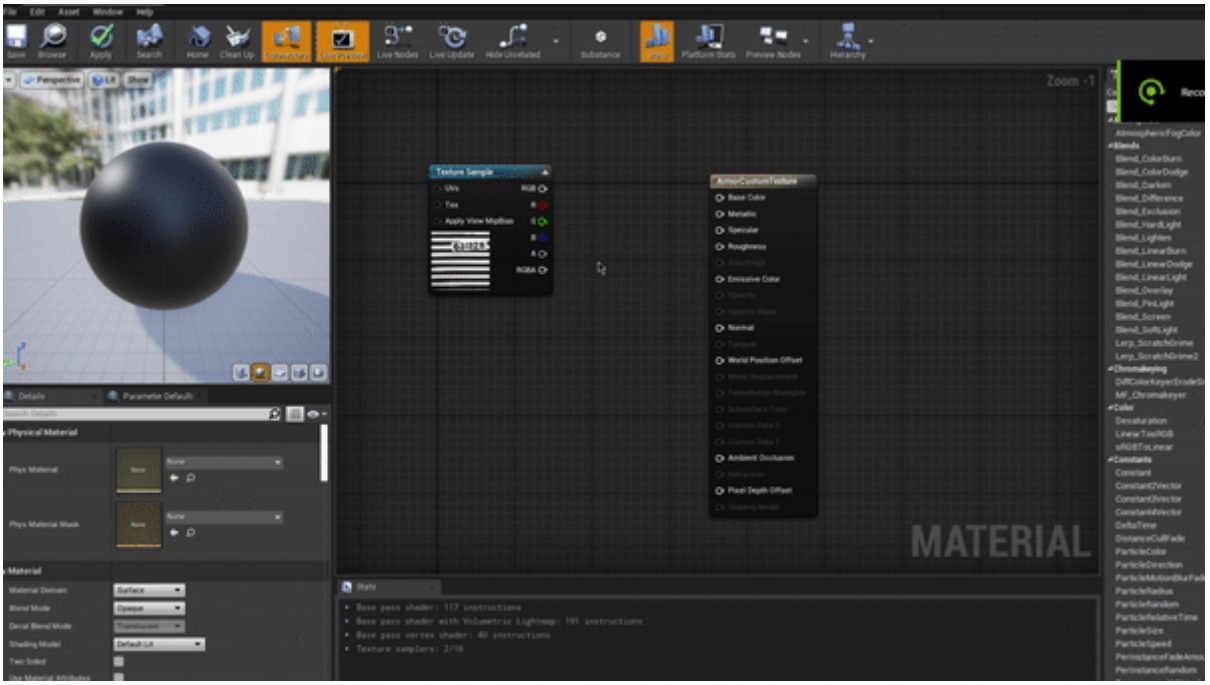
Now click on the **Multiply** node and change the **Const B** to something like **30.0**



Now for the last step to make sure the glowing actually works in the game, Select your material and Set the **Domain** to **Light function.**



GIF:



Pak and test it.

There's two Videos from Verex Showing how to do a blinking emissive similar how C4's lighting blink behaves in the game and doing RGB Color Changing.

For the Blinking: https://streamable.com/gtkfhx

For the RGB Color Changing: https://streamable.com/n7l31x

You can contact him on his Discord if you have any questions for these: SirVerex#5139

If you have any more questions about this guide, contact me on Discord too: Pacagma#1515

And that's all.

ROCK AND STONE, MODDER!