# How do I make “Hex Mods” with DAUM?

The guide will help you start using DAUM to make UE mods that change game’s paked blueprint assets.

Before reading and using the guide, please learn the basics of UE modding. Make a simple hex mod from the main modding guide.

# First Steps

Grab a DAUM release [here](https://github.com/DarthPointer/DAUM/releases). And follow the [installation instructions](https://github.com/DarthPointer/DAUM/wiki/Installing-the-Tools).

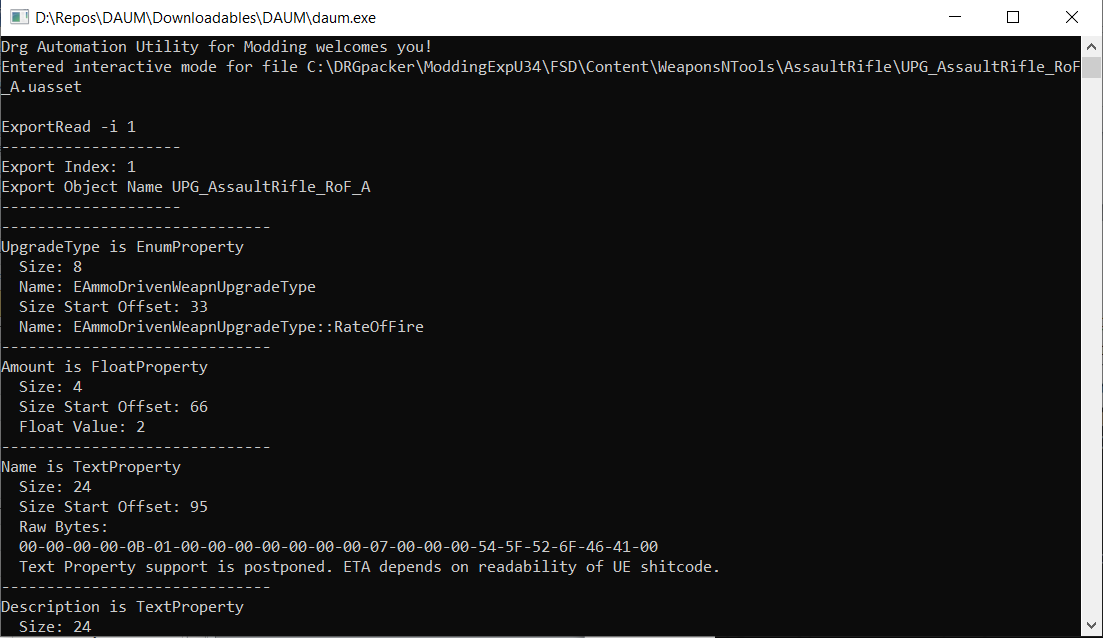
For your convenience you can associate .uasset file type with daum.exe. To do this, double-click any uasset and select the daum.exe you installed (browse for it in the default app selection menu that will pop up). After that you could open a file with DAUM immediately double-clicking the file to open.

## Interactive Mode

Drag’n’drop a .uasset you want to open into the daum.exe. Once again, a .uasset. Not a .uexp, NEVER! Or double-click a .uasset if you have the file extension-application association thing done.

WARNING! In order to operate files, you need the uasset-uexp file pair be located in same folder. If you are going to open XXX.uasset, you need its XXX.uexp to be nearby.

There is no need to move them into the DAUM application folder.

You should get a DAUM CLI window awaiting for your commands. Let’s start with the safest one. Type the following and hit enter: ExportRead -i 1. 

Depending on what file you have opened, you can get output of different size, the picture here is just an example. The bare minimum are Export Index and Objec Name. And no exception should arise.

It does not matter much to understand what has just happened here, we only do that to see that you have no technical issues.

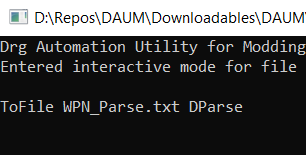
All the other commands work this same way. You type them, hit enter and things happen. Some commands don’t have any “visible” output, some do, the ExportRead one has a lot of output.

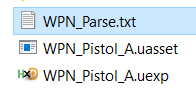
# Task 1. Replace X with Y, Trivial Case.

Let’s start learning with a simple mod that replaces one upgrade in upgrades tree for a different upgrade. Following mini-guide will replace Subata’s Improved Propellant (T3.A) with minigun’s +2 damage upgrade (relevant stats and names as of U34).

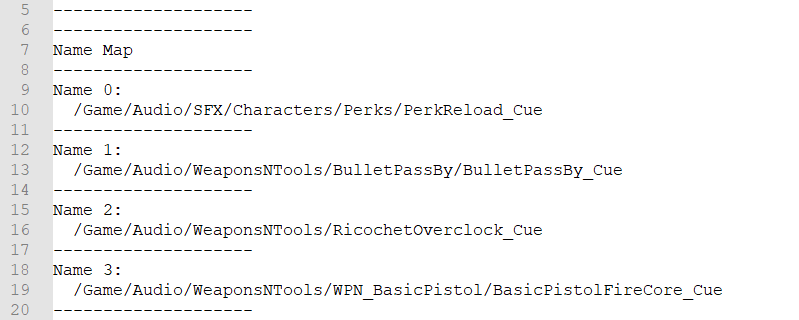
## Replacement Targets

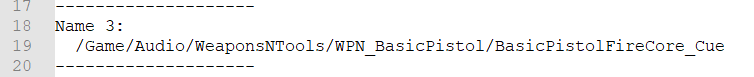
Copy original Subata’s WPN (WeaponsNTools/Pistol/WPN\_Pistol\_A) files into your working folder. Open the new .uasset file with DAUM. Get a parse of its contents. Type “ToFile WPN\_Parse.txt DParse” and hit enter.



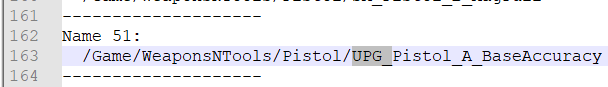
You should get a new file nearby the files you operate, named WPN\_Parse.txt. 

In this file we have to search for “names” to replace. All the names are located together in a list that starts this way:

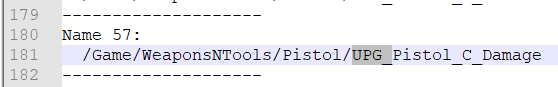


Each element looks like this: 

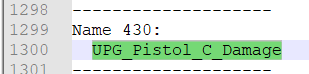
This parse displays each name’s index (“Name 3:” means it is the name by index “3”) and the name string. We are looking for 2 names, both have UPG\_ inside it and they refer that it is the improved propellant upgrade we are looking for. Use search for “UPG\_”.



Like this one, but is definitely not the one we need, it is Tier A accuracy.



Here we go, Tier C damage. Let’s refer that name as a “filename”. The other one we need is just the last part of it.

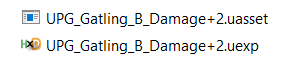


Let’s call it “object name”.

In fact there are many other things happening on the way the upgrade “makes” to get into the tree, but we will stop here as for that case we don’t need to dig deeper.

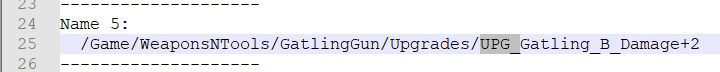
## Replacements Source

Now we need to know relevant names for the minigun’s upgrade. You can either analyse the minigun’s WPN the same way or search through files in the minigun’s folder in the unpaks.

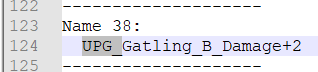
Going back to WeaponsNTools, then deeper, into GatlingGun. In that specific case (Minigun) we have all the upgrades being in a separate “Upgrades” folder. 

These are the files of damage upgrade we are looking for. For convenience purposes you can parse the UPG\_Gatling\_B\_Damage+2.uasset the same way we used for the Subata’s WPN. We can actually deduce new names we need from file names, but taking them from parses is easier. The guide uses copying from parses.

Search for UPG\_ and see following name definitions:



aka “filename”



aka “object name”

We will have to replace original file and object names with these ones in the Subata’s WPN.

## Taking Action

Replacements to do:

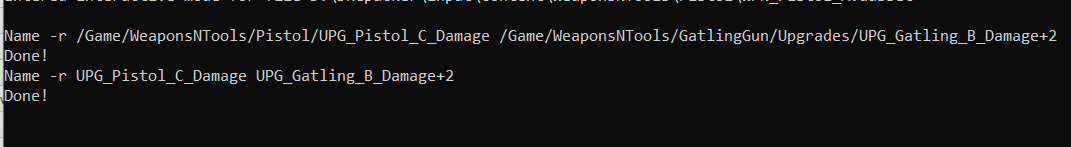
/Game/WeaponsNTools/Pistol/UPG\_Pistol\_C\_Damage -> /Game/WeaponsNTools/GatlingGun/Upgrades/UPG\_Gatling\_B\_Damage+2

UPG\_Pistol\_C\_Damage -> UPG\_Gatling\_B\_Damage+2

The DAUM commands to do this are:

Name -r /Game/WeaponsNTools/Pistol/UPG\_Pistol\_C\_Damage /Game/WeaponsNTools/GatlingGun/Upgrades/UPG\_Gatling\_B\_Damage+2

Name -r UPG\_Pistol\_C\_Damage UPG\_Gatling\_B\_Damage+2



Make a new parse (ToFile WPN\_Parse.txt DParse). And see these names there (Search for UPG\_Gatling\_B\_Damage+2).

DAUM generates a bunch of files in the usage process. In case everything is done successfully, you only need the .uasset and .uexp ones. Although it is OK to pak all the files into your mod. Pak the mod and install it.

Ta-daaa!

## Name Map

We have only interacted with one part of the UE cooked files. The part is Name Map. It is a list of names that are declared to be used further in other parts of the files. These names can stand for types, property and method names, object names, object names and many other things. These names are referred via indices so we are able to do the whole replacement only replacing the names that are referred from somewhere else.

## Summary

You have replaced an object, replacing the names.

New commands used: “Name -r [old name] [new name]” (**n**ame **r**eplace), “ToFile” and “DParse”.

“Name r” replaces a name definition.

“DParse” reads the whole file (uasset-uexp file pair) and spams the readings into the console window. You can try calling it w/o the ToFile part.

“ToFile” redirects console output to the file you specify after it and executes the command given at the end. After the command execution it returns the console output back.

In general, there are few cases where names replacement is not the only thing to be done. If you have succeeded with this mod example and fail to make a different name replacement, it might be not your fault. Ask for help in that case or continue your (loooong) way through this guide.

## Think About It

You can also swap some stuff within the same file, using name replacement. If you have a file with X and Y names and you want to swap them, you can do

Name -r X Z

Name -r Y X

Name -r Z Y

X gets replaced with Z, then X takes place of Y and finally Y occupies position previously used for X. Just make sure that Z name is different from all the other names in the file so the last command does not change a different Z that is present in the original file.

# Task 2. Alter a Value not Defined Explicitly.

Each class property has a default value. And if the used value is the default value defined somewhere in the class hierarchy, you will not see it defined in the place you are looking for it.

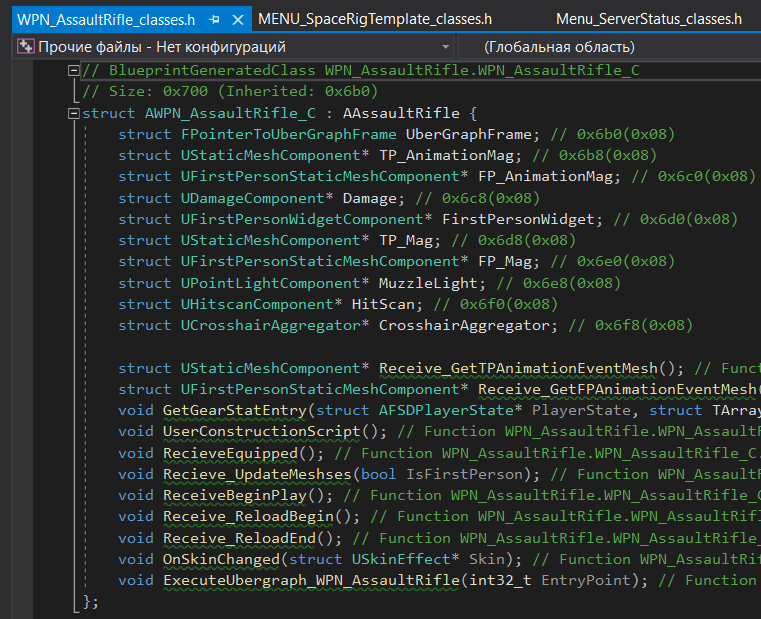
I suggest making a mod that alters baseline damage to armor of Gk2 rifle.

In order to conclude that a value exists, you may need to use other sources of information. That can be similar objects (an object A has property X, and object B is its “brother” but I don’t see X) or header dumps. “Header dumps” is a reverse-engineering product that tells us what classes and structures are defined in the game and what properties and functions they have. For our case, we are interested in a class relevant for Gk2 and its property that defines damage to armor.

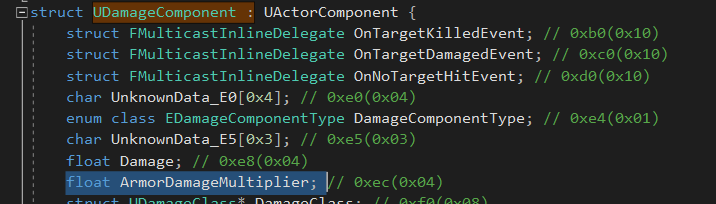
Getting the dumps is a separate task, let’s assume you simply ask for dumps at the modding server and get a copy.

## Locating the Target

The rifle’s object is called WPN\_AssaultRifle, so in the dumps we will search for a header file with similar name. That is “WPN\_AssaultRifle\_classes.h”.

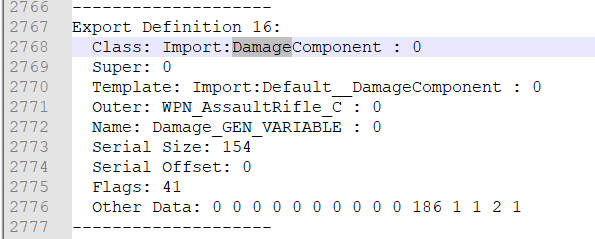


We don’t see any number to determine any stat here… and that is because they are all either defined in the parent class (see that AAsaultRifle) or inside the classes and structures that are parts of the class. Like the “UDamageComponent\* Damage”. Spoiler, this is our next destination. If you have opened the whole dumps folder with an IDE/editor you can ctrl+LMB the UDamageComponent. Or open “FSD\_classes.h” and search for “UDamageComponent :”.



Bingo! See that “float ArmorDamageMultiplier”. It means that there is a float property in each damage component. And we want to change that value.

Next we should find where the damage component of the Gk2 is. Search for “Damage” in the parse.



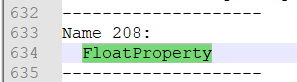
In the process we will need to know either the object name (“Damage\_GEN\_VARIABLE”) or Export Definition number (16).

This piece of parse is an “Export Definition”. Just remember that.

## Taking Action

We have to add the definition “ArmorDamageMultiplier” float into the damage component we located.

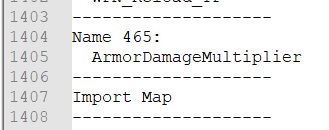
First step will be making sure the names we need exist. Search for “ArmorDamageMultiplier” in the parse. No matches – means we have to add this name. Then search for “FloatProperty”. Hopefully, it is present in the file.



So that we only have one name to add. Do this now. The command is “Name -a ArmorDamageMultiplier”.



You can reparse the file and see following there at the end of the name list:



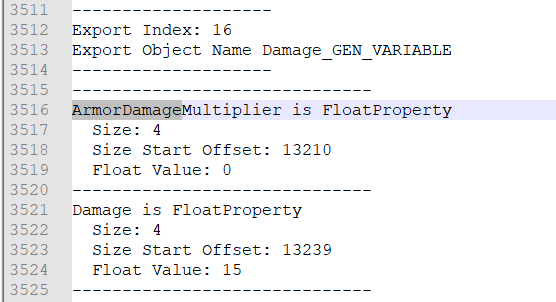
If we had no name def of type present in the file, we would have to add that too, with the same command.

Now we add a definition for the property.

ExportChange -a Damage\_GEN\_VARIABLE 0 “” “ArmorDamageMultiplier FloatProperty”.



From now after reparse you will have a yet another match for ArmorDamageMultiplier in the file.



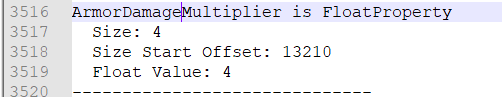
That part of file is “Export Contents”. See the Object match the ObjectName from an Export Definition?

You can use the good ol’ hex-editing to set the value we want. But why would we consider doing that if we can type one command more and be good to go?

ExportChange -r Damage\_GEN\_VARIABLE 0 ArmorDamageMultiplier/Float32/0 4



That’s it! Reparse to assure changed are done. And see following.



Pak and test.

Wait shi~ I don’t see that in the weapon stats in the terminal. Yes, because GSG lads used some hardcode here (and also shitcode IMHO, there are few worse cases with it).

Hover over the AB upgrade to reveal that change being applied.



## -ExportChange Command

Command to change properties of exports (objects defined in the files you currently operate).

-ExportChange -a/-r [export reference] [path] [generation/replacement params] [other args]

### Mode Key

-a stands for **a**dd, -r is **r**eplace.

### Export Reference

We used “Damage\_GEN\_VARIABLE 0” as export reference in both cases. These 2 words are object name (remember that pic with it being mentioned as an object name) and an augmentation number which can be found in that export definition after a colon at the object name.

Alternatively to name and aug number, you can also use the index to refer the export. Use “-i IndexNumber”, -i 16 for this case. Sometimes you can have multiple exports with exact the same name and aug so the only reliable way to point to a specific one is the index.

### Path

Path says where exactly you want to apply the changes. We had a property being a part of an object itself. Not anything nested inside its substructure or a collection. So we added a property at empty path “”.

Then we accessed the float value with path “ArmorDamageMultiplier/Float32/0”. It means that we go inside the property “ArmorDamageMultiplier” and we want to change a 4-byte (32-bit) float. 0 stands for the float being the first float over there (you know, in programming it is a common practice to start enumeration from zero). If ArmorDamageMultiplier was a vector and wanted to change its 2nd value, we would use Float32/1.

### Generation or Replacement Params

For generation we used “ArmorDamageBonus FloatProperty”. Which means we want to add a property called “ArmorDamageBonus” and it is a “FloatProperty”.

Note that we had the gen params in quotes. We have to do that because this is one argument, but it contains a space symbol. In order to pass a param with space inside, you quote it so the program understands that the space is not a separator between different params.

### Documentation

For deeper detail, see relevant [documentation](https://github.com/DarthPointer/DAUM/wiki/DAUM-Operations#export-change-exportchange). In particular, it describes the process of getting inside substructures, collections and [value type designations](https://github.com/DarthPointer/DAUM/wiki/DAUM-Operations#type-designations) used by DAUM.

## Exported Data

The second task introduces you to operating objects’ properties which are all located in uexp file. Properties can be of primitive types (ints, bools, floats), compound types (like Vector is float\*float\*float), have their own subproperties and be collections of elements and the elements can also be any of these.

## Summary

The guide tells how to add lacking property definitions so that you can change their values. You can also use the ExportChange -r command if the property is already present in the file and you only need to change the value, you only need that one command in such a case.

New commands:

ExportChange -a/-r, (**a**dd or **r**eplace) Adds properties and replaces their values.

Name -a (**a**dd), adds name definitions.

# Task 3a. Add Something Somewhere, ObjectProperty Case.

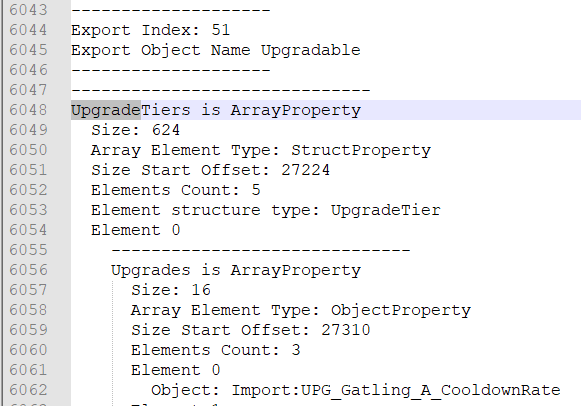
Now we will get back to upgrade trees. For this task we are going to add Subata’s T4 weakspot damage to Minigun’s T2.

## Research

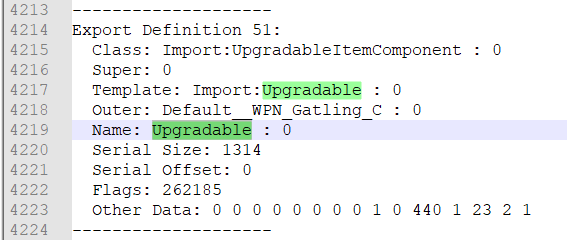
Actually, when you are doing “hex-modding” independently, you end up looking through different data sources in order to find what your modding subject is like inside the game files. And if you are trying to do something new, chances are there is no one to tell you for sure where to start from.

Let’s assume you know that upgrade tree is somewhere in the WPN file. We are going to alter it so we copy the minigun’s WPNs into our workspace in advance.

First step is to parse the files. Searching for “Upgrade”, about 80 matches in the file. And almost the last one is this:

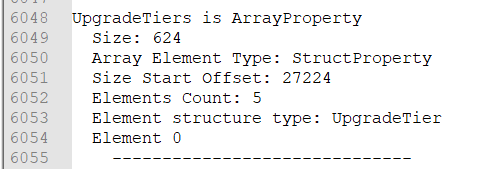


According to this Export Contents, we have an Export Definition with Name “Upgradable”. That’s it!

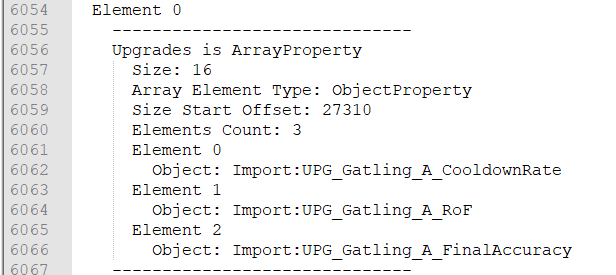


Conclusion: we will be working with export “Upgradable 0” or “-i 51”.

Get back to “UpgradeTiers”. We have an ArrayProperty called UpgradeTiers. Array element type is StructProperty, these structs are UpgradeTier’s and we have 5 of them.



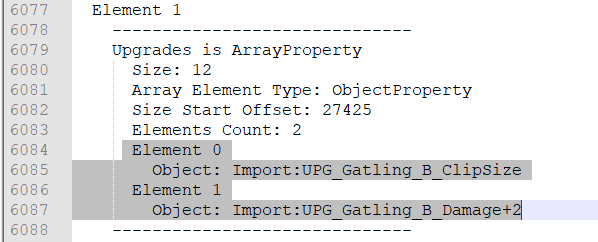
The first one (Element 0) starts with an Upgrades, an ArrayProperty. Upgrades contains 3 elements, these are ObjectProperties. And each object element refers one upgrade. If you check the ingame upgrades tree, you will see these match the upgrades and their location in the tree.

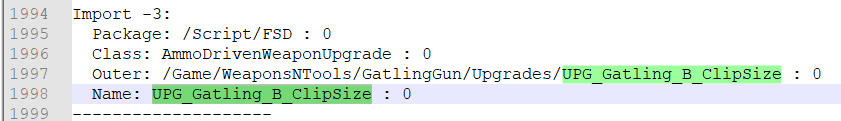




Minigun upgrade tier A, nuff said.

We want to extend tier B so we need element 1 of the “outer” array. And we will need to add an element into its “inner” array. Scroll down a bit.

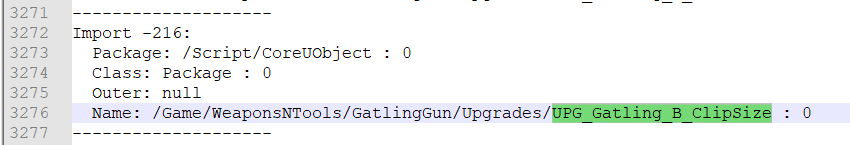


Now we should track down everything that happens with present upgrades in the file. See that “Import:UPG\_Gatling\_B\_ClipSize”? Search for “UPG\_Gatling\_B\_ClipSize” mathces is the parse. Apart from 2 name definitions you should know about from the Task 1, there are 2 more matches. Start with the following one: 

This is an “Import Definition”. It allows us use that mysterious “UPG\_Gatling\_B\_ClipSize” as an ObjectProperty value. This is not the only way to “allow” an object, so the parser hints it is “allowed” as an import.

The import definition contains Class Package, Class Name, Outer and Object Name.

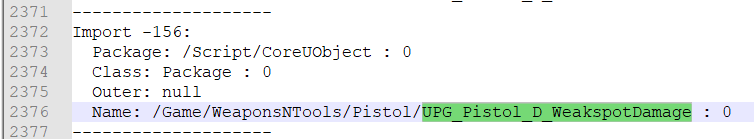
Class Package and Name are names from name map (list of name definitions), object name also is a name from that map. And the Outer is a reference to a different import definition. Under the hood it is a negative number. Why? Because imports use negative enumeration. It starts with -1 and goes further down. See the import numbers in the parse. DAUM replaces it with Name of the referred import, if any is linked.

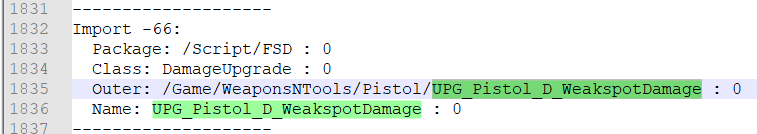


This import definition defines an import with a “filename” name. The name is exactly the same as the outer of the previous import we looked at.

There are no other mathces for “UPG\_...”, outer is “null”.

In order to find out about Subata’s upgrade relevant name, we can see the parses of its files. Our target turns out to be UPG\_Pistol\_D\_WeakspotDamage.





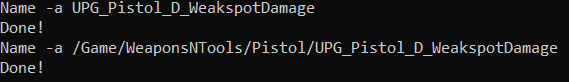
## Taking Action

In order to add an upgrade to T2.C, we have to extend tier 2 array and place there a new upgrade object property. Object property in its turn needs an import definition that refers a yet another import definition (hopefully, the second definition does not refer anything else).

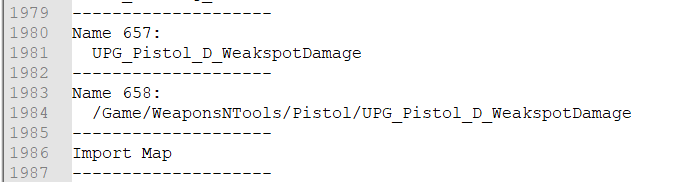
And these import definitions needs names for object names (remember them being manipulated in Task 1?). So let’s add them. Open the minigun’s WPN and follow the commands.

Name -a UPG\_Pistol\_D\_WeakspotDamage

Name -a /Game/WeaponsNTools/Pistol/UPG\_Pistol\_D\_WeakspotDamage



Following will appear in the parse if you update it:

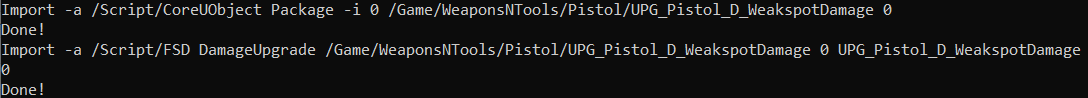


Now the import definitions. We better start with the one that has zero outer index, because it has all its “dependencies” already here.

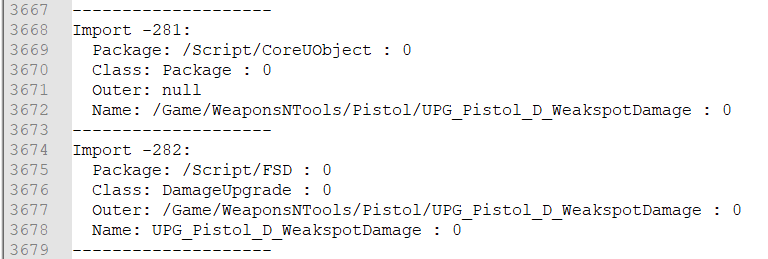
Import -a /Script/CoreUObject Package -i 0 /Game/WeaponsNTools/Pistol/UPG\_Pistol\_D\_WeakspotDamage 0

And then the import with a link to the previous import.

Import -a /Script/FSD DamageUpgrade /Game/WeaponsNTools/Pistol/UPG\_Pistol\_D\_WeakspotDamage 0 UPG\_Pistol\_D\_WeakspotDamage 0



Result, new matches for the “UPG\_...” if you reparse the files:



Where did we take the /Script/CoreUObject, Package, /Script/FSD and DamageUpgrade from? We take them from what we see for existing upgrades.

Now we are ready to extend the array. Remember, we need an export called Upgradable : 0, alternatively export by index 51.

ExportChange -a Upgradable 0 UpgradeTiers/Array/0/1/Upgrades/Array/0 “”

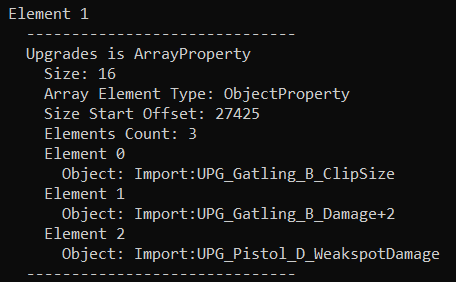


ExportChange -r Upgradable 0 UpgradeTiers/Array/0/1/Upgrades/Array/0/2/ObjectIndex/0 “UPG\_Pistol\_D\_WeakspotDamage 0”



Alternatively to full reparse, you can use “ExportRead” command with same arguments to refer the export as we used before.





Good to test.



And here we go. The weakspot damage is not displayed because of the HarDcOdE, but it should work.

## Import Map

Import Map has something in common with Name Map. It is a sequence of elements, each element is an import. Import is a declaration of “something” that is defined in a different file and it lets you refer that “something” in this file.

Each import element has package name, class name and object name. All these names are taken from Name Map and are referred via indices. So that if you change the corresponding name definition in the Name Map, you also change that name in the import (and anywhere else if referred).

Outer is a different import in the Import Map.

Under the hood all of these are numbers. Package, Class and Name take 8 bytes each. First 4 bytes there refer to the name itself while the last 4 are “name augmentation”. In our case all the name augmentations are zero (numbers after colon). Outer is just 4 bytes with a negative integer that refers number of an export. Or is zero if no export is referred.

## Import Add

“Import -a” command used in the task adds a new import definition into the file. It needs you to specify 4 things: Package Name, Class Name, Outer and Object Name. Package Name and Class Name are simply names defined in the name list.

You usually type name strings to determine class, package and object names as in the “task guide”, the tool will find a matching definition and use its index. Alternatively, you can use “-i index”.

For Outer you can either use the target import object name string and augmentation (see the task’s example) or “-i index”. If outer has to refer nothing, you can only use “-i 0”.

## Export Change

Task 3 shows you how to use the Export Change command to work with arrays. You use Array/ArrayNumber/ElemenNumber to access specific element, or just Array/ArrayNumber with empty generation params (“”) when adding a new array element.

## Summary

One of the trivial cases of referring objects includes 2 names (“object name” and “file name”), 2 imports (a “file import” and an “object import” that refers to the “file import”). Sometimes the hierarchy is more complicated. Just see what matches you have for stuff that is already in place.

# Task 3b. Adding Something Somewhere, SoftObjectProperty Case.

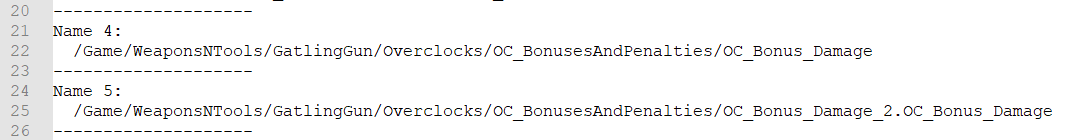
For this one we will add an extra effect to an existing overclock. Let’s say we want Minigun’s Compact Feed Mechanism to share a damage bonus used for Exhaust Vectoring.

WARNING: Task 3b is not an alternative to 3a. Only approach it after completing the 3a as it contains info that you are considered to know here in advance.

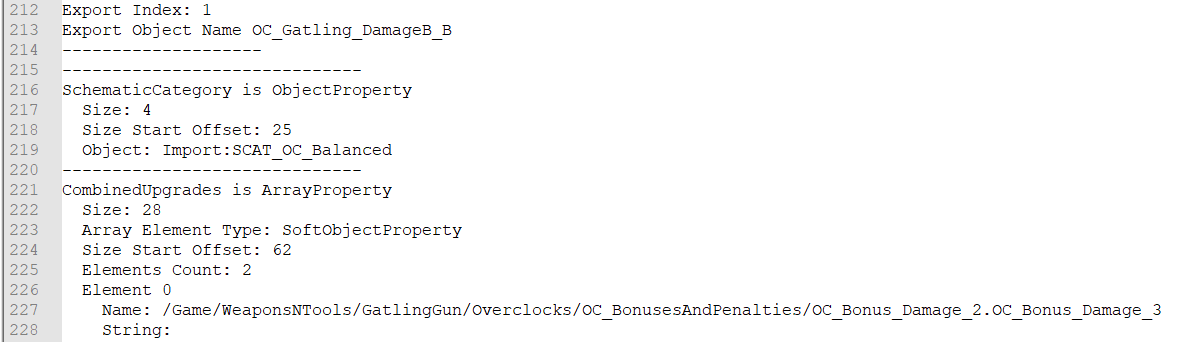
## Research

I’m not going to tell much detail here because the ideas behind finding the stuff to change are all the same, only the “route” changes.

We go to EV files, parse them and see following:



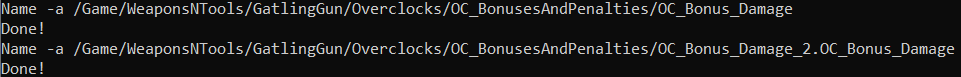
2 names. Slightly different from what we have in the 3a task. Now they are all “long”.

Then we see this 

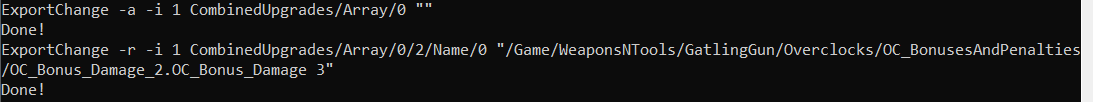
See that \_3 at the end of the first name element? There was no such a thing in its name definition, so 3 is not a part of a name, but rather a “name augmentation”. We will refer the name as “name 3” and not “name\_3 0” in our commands.

## Taking Action

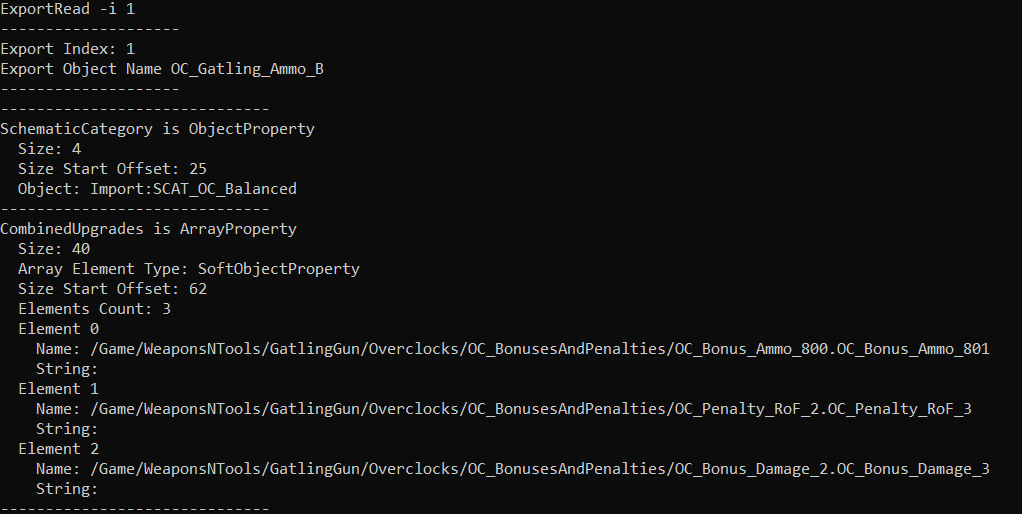
Switch to your copies of CFM OC files, add the names.

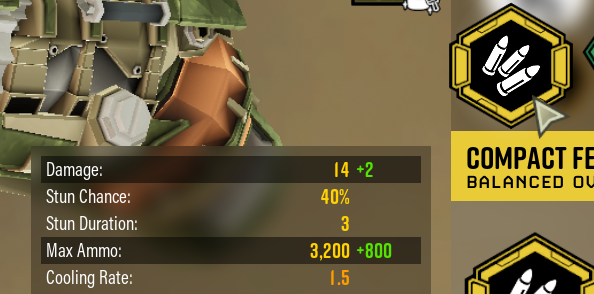


Extend the array and fill the value in.



And see it in the ExportRead results.





## Export Change

For this case (soft object property, which was the array element type) we used “Name” to fill the desired value in. In fact, SoftObjectProperty is a Name+SPNTS, SPNTS = Size-Prefixed Null-Terminated String. There are rare occasions when that string is not empty. And we didn’t have to fill it in because DAUM creates the SPNTS in an empty state.

If you are interested in contents of different properties and relevant primitive types keywords, you can check pattern folders of the DAUM tool and open files in there as regular text files.

Or you may visit the tool wiki if you want to discover/revise the list of type keywords.

That thing with name being defined with a string and a number is absolutely the same as with export references. String is taken from the name definition and then you have an extra number stored locally, nearby the name “reference index”.

## Summary

It is supposed that this task was a piece of cake for you. And that you had no issues understanding logical connections between the steps as you could fill these “gaps” with help of your Task 3a experience.

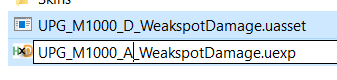
# Task 4. Reusing and Duplicating.

Sometimes you can’t achieve your design target within the limit of vanilla only having certain number of asset instances. You may want to add new vanities, upgrades, weapons or any other stuff, based on existing assets.

Let’s get some practice with the easiest case – making a new upgrade. Our target will be duplicating M1K’s T4 weakspot upgrade and putting its copy into T1. For better distinction, we will double the effect of our copy. That way we will be assured it is a brand new upgrade when we check stats in upgrades terminal.

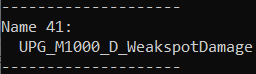
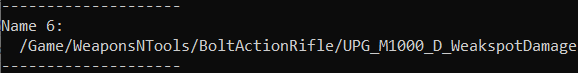
## Taking Action

Our “sample” asset is UPG\_M1000\_D\_WeakspotDamage. We want make a new upgrade instance of upgrade class with help of it. Copy its .uasset and .uexp into your working folder. Then rename the files (select both files with ctrl, F2 and type new name, the other file with update its name too). I’d suggest using new name UPG\_M1000\_A\_WeakspotDamage.



Now we need to change the copy’s **consciousness according to its new name.**

**Remember these 2 names we added in task 3a? The names to link an import object. These names are also present in the asset they refer to. Use ReadNames command and see these:**

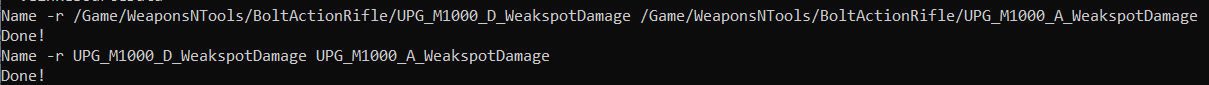
****

Now we replace them to reflect new name.

Name -r /Game/WeaponsNTools/BoltActionRifle/UPG\_M1000\_D\_WeakspotDamage /Game/WeaponsNTools/BoltActionRifle/UPG\_M1000\_A\_WeakspotDamage

And

Name -r UPG\_M1000\_D\_WeakspotDamage UPG\_M1000\_A\_WeakspotDamage



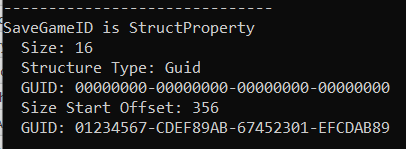
Is it all we need to make a completely new upgrade? Not yet. Upgrades crafting and selection systems use upgrades’ SaveGameID’s. It means we should give our upgrade a different, distinct GUID for that property. In general GUID is a sequence of 4 UINT32’s. However DAUM uses a special GUID pattern element to change them. One of the available input formats is a string of 32 hexadecimal numbers. For education purposes we could use something dumb like 0123456789ABCDEF0123456789ABCDEF. USE PROPER RANDOM GENERATION FOR REAL MODDING.

ExportChange -r -i 1 SaveGameID/GUID/1 0123456789ABCDEF0123456789ABCDEF



We are omitting the reason to use -i 1 as a target export. It is a common thing for upgrades and many other asset types to only have 1 export per file. However you should be careful and check if Export 1 is really your target or not.

Notice that now we use /1 at the end. The reason for it to be not /0 is that GUID properties are in fact a subtype of StructProperty. And each StructProperty has a GUID inside its definition. Since that the GUID value we want to change is the second one.



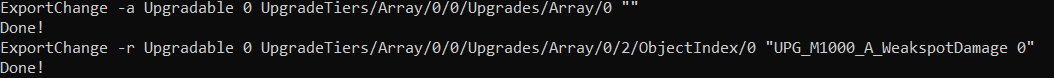
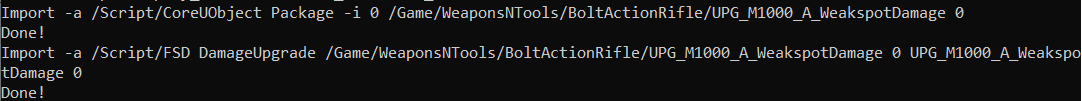
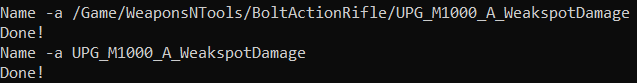
The last change to make the effects distinct.

ExportChange -r -i 1 Amount/Float32/0 0.4



Don’t focus on me using comma as a decimal delimeter, that simply is regional formats cancer. Gotta do something with it so that all scripts could be freely used across culture contexts.

And now we are done with this file. Just slap this upgrade into the upgrade tree of M1K. The same way as we used to change the WPN file in task 3a so I’m not going to walk you through that part here.



Done!



## SaveGameID

All selection pools and progression items are managed by these GUIDs. If you have issues with selecting or obtaining your new item, chances are you have encountered a GUID collision. Generate a different one.  
Also, keep the save game IDs same for your items between mod updates. In order to prevent people losing them from their progression and having to buy and select them once again.

## Summary

Task 4 covers the easiest case of asset duplication. Different assets may have more complex. Like the M1K weapon file has more than 2 names with M1000 substring. Which ones should you change? That is a subject to trial and testing.