Texture Modding

A comprehensive guide to creating texture mods for Doom Eternal, from basic to advanced techniques.

- **Beginner's Guide**
- **Intermediate Guide: Material2 & Custom Normals**
- **Advanced Guide: Advanced Customization & New Texture Creation**
- **Getting Started in Substance Painter**

Beginner's Guide

Tools You Will Need

- **EternalResourceExtractor by PowerBall253 - Download**
  It is essential that you extract all of the game's files when working on skins as doing so will allow you to see all of the true file extensions for texture files in the game's stream database (more on this later), have access to *all* decl files of all kinds, and be able to see any and every file path to get a greater understanding of where everything is located, which will save you hours of time down the road.

- **VEGA by DTZxPorter - Download**
  This is needed to extract both textures (for simpler mods) and models (for more complex ones).

- **Photoshop - Buy / GIMP - Download**
  You can primarily edit just the texture images if you choose, but even if you are doing more advanced work, you will need to modify and export normal maps through either of these programs before they will work in game.

- **Blender (optional) - App Download or Steam Download**
  If you are going to modify normal maps, you will need to properly 'bake' them in blender before injecting them back into the game. Not doing so will cause them to break the lighting and not function properly.

- **Substance Painter (optional) - Buy**
  To customize skins at maximum potential, Substance Painter is the route to go for sure. However, it can be expensive, so it will not be essential for making texture mods at base.

- **Auto Heckin' Texture Converter** by **PowerBall253** - **Download**
  All texture files will need to be converted to .TGA with this tool before they can be injected. Self explanatory.

> **BEFORE GOING ANY FURTHER:** Make sure you have uninstalled all mods by deleting them all from your Mods folder and then verifying the integrity of your game files in order to restore them to their vanilla forms. If you do not, then Eternal Extractor will extract the *modded versions* of any and all files injected instead of their vanilla versions, which is what you will need.

Extracting Resources

Extract the files from the Eternal Extractor zip file directly to your DOOMEternal root directory and run ‘Eternal Extractor.bat.’ Specify an output directory when prompted and it will extract every game file where you want. Done! Just know that the total file size of the files extracted will be *identical* to the size of your DOOMEternal installation directory and make sure enough disk space is available.

Running VEGA

Extract everything from the VEGA zip file also into your DOOMEternal root directory for the sake of organization. From here, you will need to know a few important folder paths for finding resource libraries to extract from. The most common ones will be gameresources, gameresources_patch1, gameresources_patch2, and warehouse. There are located in this folder path:

```
DOOMEternal/base
```

Nice and simple. However, if you need to extract files from individual *mission libraries*, that will be a little more complicated. Those are located here:

```
DOOMEternal/base/game/sp
```

There won’t be many reasons to do this simply for texture modding, but it’s good to know anyway.

Next, you should configure VEGA appropriately. Click on the Options button to set your export preferences. Make sure that VEGA is always looking for just Images and Models. Nothing else will be relevant. If you are working only in GIMP or Photoshop, you can select the DDS image format. You can edit DDS images directly in both apps and since you will eventually be converting DDS images to TGA for injection, doing so will save time. If you are working in Substance Painter, export them as PNG always.

Finally, you will need to be able to know what skin you’re looking for. Files for weapon skins are in folders with keyword identiications, such as “gold” or “astro.” However, all Slayer and Battlemode demon skins are identified by set number.

Slayer textures, regardless of resources library, will be in the following directories:

```
models/customization/characters/doomslayer/[set number]
models/customization/characters/doomslayer_1p/[set number]
```

Weapons textures, regardless of resources library, will be in the following directory:

```
art/weapons/[weapon name]/skins/[skin name]
```

All base campaign demon textures are located in:

```
gameresources/models/monsters
```

The overwhelming majority of skins are located in warehouse.resources, but there are still several that you may want to work with located throughout the three gameresources libraries, which is why running the Eternal Extractor will make it much easier for you to find what you're looking for. Make sure to download the text document from #resources that details all the set numbers before continuing (insert link here). Once extracted, VEGA will put all image files in the exported_files/images folder and all models in the exported_files/models folder. Now, let's go into a little more detail for each type of skin.

Slayer Skins

For Slayer skins, you will need a total of three sets of textures:

- All textures with _hq in their name from the appropriate folder.
- All textures *without* _hq in their name from the same folder.
- All 1st person textures from the appropriate doomslayer_1p set folder.

The textures with and without _hq in their name will be visually identical, but different in purpose. HQ textures are used specifically for cut scenes, whereas the corresponding textures without _hq in their names are used for Photo Mode. Both sets should be replaced.

The 1st person textures are, obviously, used in-game from the first person view while you play. On base Slayer-style skins, they will all appear visually different (unique UV wrapping) and will need to be modified separately. For certain skins, such as the Cultist Slayer or Maykr and their variants, most of these files will be identical to the others, but some will still need to be modified separately in order to have a complete mod.

The model files for the 3rd person/cut scene textures of slayer skins are obvious. They will always be named doomslayer_cine_[set number]. You can ignore the _3p variant as it's basically a redundant version of the same model but with less detail. However, the 1st person model is a bit different and confusing. All 1st person models, regardless of skin, will be called 'marine' instead of 'doomslayer.' They will not have 1p in the name. You will just need to remember that doomslayer means 3rd person and marine means 1st person. They will, instead, have the formatting marine_[set number]. If you are working in Substance Painter, make sure you extract both.

Best Normal Map Texture Sources:

3rd Person (Photo Mode & Cut Scenes): Same normal maps as the skin you're working on. Also, if working in Substance Painter, make sure you create them on the doom_marine_3p model as it has a small part of the helmet that the doomslayer_cine_3p model does not.

1st Person: Use arm textures from whatever set you are working on. For the Doomblade, Legs, and Torso, search gameresources.resources using SAMUEL for "doomslayer_cine" and extract the textures for that model (it will not have a set number). The Torso, Legs, and Doomblade normal maps for this model match the 1st person UV wraps, but they are much higher resolution.

Weapon Skins

For the weapons, each skin will have all the files specifically altered for that skin, which will not necessarily be every file. Instead, for the majority of skins, certain textures remain the same as default and will be located only in the default skin's original folder. This generally applies to:

- Shotgun Sticky Bomb Grenade and Full Auto Shells
- Heavy Cannon Precision Bolt Scope Emissive
- Plasma Rifle Glass
- Rocket Launcher Rocket
- Super Shotgun Shells and Hammers (named "mastery;" visually disabled by default)
- Ballista String
- Chaingun Bullets

There will be exceptions, such as the Hot Rod Shotgun, which includes its own orange Sticky Bomb grenade textures, but generally, if you can't find any necessary files located in the skin folder you wish to use, you will need to replace the ones in the default weapon folder:

```
gameresources/art/weapons/[weapon name]
```

Demon Skins

These are mostly straightforward, since almost every in-game demon skin is located in the single directory specified above. However, there are a few exceptions. The textures for the Doom Hunter and Dread Knight are pulled natively from each and every individual mission library for that specific mission. Additionally, all of the green textures for the Cyber Mancubus operate the same way. As such, since you will need to inject your modified textures into each and every mission resources library if you decide to create skins for those demons, having the full directories for them extracted with the Eternal Extractor will help immensely when creating your folder paths.

Preparing Your Files for Injection

Once you have your finished texture files in DDS format, drag and drop them onto the Auto Heckin' Texture Converter batch file and they will be automatically converted to .tga. You will then need to create your own clone file paths so that Mod Injector will inject the textures and replace the corresponding files in the correct location. Here are some examples:

Example: Plasma Rifle skin

I want my files to replace the textures for the Specter skin, which is in warehouse.resources. First, I make sure that all my new modified image files match the file names of the originals. Then, I create my own empty directory of folders that matches warehouse/art/weapons/plasma/skins/specter. I then put all my converted texture files in the specter folder for injection. I also decided to modify and tint the

glass textures, so I create another directory to match where those are located: gameresources/art/weapons/plasma and I put the .tga glass files there. I then take both of those folders, put them in a .zip file, and they're ready to test!

Example: Slayer Skin

I'm going to have my Slayer skin replace Galaxy Sprinkles, which is set58 and is located in warehouse.resources. Once all my images are converted to .tga, I'll create this directory:

```
warehouse/models/customization/characters/
```

In the 'characters' folder, I will make two more folders: doomslayer and doomslayer_1p. In both of those folders, I will create a set58 folder, so that within the characters folder, I will have these two file paths:

```
doomslayer/set58
doomslayer_1p/set58
```

I will then put all the _hq and photo mode images in the `doomslayer/set58` folder and all the 1p images in the doomslayer_1p/set58 folder. I will then make a zip file out of the warehouse folder and the mod will be ready to test!

This concludes the basic guide on how to extract, find, and then prepare images for mod injection. Continuing on, there will be a much more advanced guide on adding custom assets and how that process works, complete with a comprehensive explanation of so-called 'true' file formats work and which ones you will need to know. Additionally, there will soon be tutorial videos on how to use Substance Painter and Blender for creating custom skins. Thanks and good modding!

Intermediate Guide: Material2 & Custom Normals

Editing .decl Files

Required Tools:

- **Notepad++** - **Download**
  You will need this in order to edit decl files properly.

- **EternalResourceExtractor by PowerBall253 - Download**
  If you haven't already done so, it is essential that you use EternalResourceExtractor to extract all the game resources. This will give you access to *all* decl files of all kinds, and allow you to see any and every file path to get a greater understanding of where everything is located, which will save you hours of time down the road.

Continuing on from the previous guide, we're now going to talk about editing decl files, why you would do it, what to modify, and what not to modify. We're going to go through each set of files you can edit for your texture mods one by one, starting off with the most basic and ending with the most specialized. So here we go!

What are .decl files and what do they do?

Decl files **declare** definitions for the game to utilize in establishing functionality on boot. They do everything from telling the game how to make the Slayer animate properly down to simply what things are named in menus. They control particle colors and appearance, what models are assigned to what entities; basically how everything in the game functions is dictated by these files. They're extremely important, and, as a result, in addition to purely cosmetic purposes, can also be used to cheat.

.decl files and the new Mod Injector

As of Mod Injector 6.0, modifying certain .decl files that can be used to cheat in the game will disable Battlemode. It is up to you whether to use mods that alter those files or not and the Eternal Mod Manager will let you know if your mods are safe for multiplayer or not. Now, let us get to the most important .decl files you can use to improve and enhance your projects.

Material2 .decl files

Material2 files, at base, declare what textures are assigned to a given part of a model. They exist for all skins of all kinds (Slayer, guns, and demons). These can be modified to assign *any texture* to *any

model* and will match the names of the textures being assigned to that part of the model by default. However, assigning textures to a model not created for that model will result in stretching and wrapping in a way that doesn't match the shape at all, so instead, we're going to discuss proper usage for positive results.

Slayer files

As we discussed in the previous guide, Slayer skins have textures for every part of the model that they are assigned to. For the sake of simplicity, we're going to use the Maykr skin as an example. The parts of the models for texture assignment are:

```
doomslayer_arm_left
doomslayer_arm_right
doomslayer_helmet
doomslayer_knife
doomslayer_launcher
doomslayer_legs
doomslayer_torso
```

So where do you find these decl files? Regardless of what resource library you're looking in, all material2 decl files will be located in:

`generated/decls/material2`

Now, it's very clear what each of these model parts are and the only difference between the HQ and Photo Mode textures is the _hq suffix in the file name. If you were to replace the Fallen Angel (set41), your textures would be located in:

`warehouse/customization/characters/doomslayer/set41`
`warehouse/customization/characters/doomslayer_1p/set41`

Since this is set41, the textures will be named doomslayer_arm_left_set41_hq and so on. These are the same names as the material2 decl files and the file path for the textures will also be the same inside the material2 folder:

`warehouse/generated/decls/material2/models/customization/characters/doomslayer/set41`
`warehouse/generated/decls/material2/models/customization/characters/doomslayer_1p/set41`

If you'd like, you can take some time to locate these files, open them up, take a look at them so you can follow the next sections, and then continue on.

gameresources or warehouse?

Depending on the skin you have chosen to replace, those textures may be in gameresources, gameresources_patch1, gameresources_patch2, or warehouse. If the texture files are in any of the gameresources libraries, it is very likely that the corresponding material2 decl files will be in more than one of them. If this is the case, only the copies in the **highest priority .resources file** will be loaded by the game and actually affect the given skin. **Check this spreadsheet for current resource loading priority**.

For example, the material2 decl files for the default skins of every gun are in gameresources, gameresources_patch1, *and* gameresources_patch2. The only ones you want to modify and replace are the copies in **gameresoures_patch1**, because it is the highest on the resource priority list. If you have chosen a skin in warehouse, the corresponding material2 files will be there or in warehouse_patch1.

We will now discuss both what you can modify and what you *should* modify to get the job done.

material2 modification

The text of these decl will look similar to this:

```
{
inherit = "template/pbr_gun";
edit = {
	RenderLayers = {
		item[0] = {
			parms = {
				heightmap = {
					filePath = "textures/system/constant_color/grey_md.tga";
					options = {
						type = "TT_2D";
						filter = "TF_DEFAULT";
						repeat = "TR_REPEAT";
						format = "FMT_BC1";
						atlasPadding = 0;
						minMip = 0;
						fullScaleBias = false;
						noMips = false;
						fftBloom = false;
					}
				}
				smoothness = {
					filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_g.tga";
				}
```

```
□□□□normal = {
□□□□□filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_n.tga";
□□□□}
□□□□specular = {
□□□□□filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_s.tga";
□□□□}
□□□□albedo = {
□□□□□filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel.tga";
□□□□}
□□□}
□□}
□}
□Parms = {
□□wetnessdroptiling = 4;
□}
□SurfaceType = "SURFTYPE_METAL";
}
}
```

This is the combatshotgun_barrel material2 file for the Hot Rod Shotgun skin. The primary section to focus on first is the “parms” at the top, specifically the fields titled heightmap, smoothness, normal, specular, and albedo. You will notice that the primary purpose of this file is to assign the textures of this skin by *declaring* the file paths for each of them. These are, typically, what you will modify and what we will cover first.

Modifying File Paths

It is always recommended that you replace weapon skins in warehouse.resources. There are two reasons for this:

1. If you are declaring different file paths, it is pointless to do so for textures replaced in the gameresources libraries.
2. The combined file size of mods injected into warehouse.resources is far greater than that of the gameresources libraries. This is because all three libraries are actually all part of the same ‘mapresources’ library, common.mapresources. This means that common.mapresources has a far greater initial load and is far less tolerant of increased size. In contrast, warehouse.resources is the only resources library in the warehouse.mapresources library and will tolerate significantly more.

So why would you want to modify file paths? Well, the majority of skins located in warehouse are from event series and, thus, many players who recently purchased the game could very easily not have access to the skin you are replacing. As a result, you will want to change the file paths for a base campaign skin such as Crimson, Midnight, or even Default so that it will use the textures for said event skin and all players who have not unlocked that skin will be able to use your mod. (NOTE: If you are editing the Skullface skin, one of the Maykr skins, or any other skin that has a completely different model, you cannot reassign it to a campaign skin using this method. The method for doing so will be covered in the advanced guide if you want to skip this method and continue there. For all default-style skins, however, that method will typically not be necessary and you may use either based on your personal preference).

So, how is this done? Well, let's return to the Hot Rod Shotgun skin and use it here as well. If you return to the example, you will notice that the file paths declared always start with the names of folders within the resources libraries and do not begin with the name of the resource library itself. This is because if a folder path is not located in the resource library, it will search the others to locate it in order of load priority. **Check this spreadsheet for current resource loading priority**.

This load priority will be relevant in the advanced guide, but for now, you can ignore it. However, each given resource library will always search *itself* first, as you would expect. The Hot Rod skin is located in warehouse and, thus, the corresponding decl files will search warehouse, find the textures in the declared file paths, and assign them to the model. However, if you were to modify a decl file in gameresources_patch2 for a Shotgun skin located in common.mapresources to the file path for the Hot Rod skin, it would fail to find it in gameresources_patch2, eventually search warehouse, and load those textures. For the purposes of this guide, we will use the default Shotgun skin. The declared file paths of the default combatshotgun_barrel file (combatshotgun_barrel.decl located in gameresources_patch2/generated/decls/material2/art/weapons/combatshotgun) will look like this:

```
smoothness = {
	filePath = "art/weapons/combatshotgun/combatshotgun_barrel_g.tga";
}
normal = {
	filePath = "art/weapons/combatshotgun/combatshotgun_barrel_n.tga";
}
specular = {
	filePath = "art/weapons/combatshotgun/combatshotgun_barrel_s.tga";
}
albedo = {
	filePath = "art/weapons/combatshotgun/combatshotgun_barrel.tga";
}
```

"Smoothness" will correspond to your "glossiness" layer in Substance Painter if you are using it for your textures (_g). Normal and Specular are self-explanatory, but "albedo" refers to your Diffuse layer. So, we will now modify the parms to use the Hot Rod textures instead, which will make them identical to the parms of the Hot Rod decl file:

```
smoothness = {
	filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_g.tga";
}
normal = {
	filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_n.tga";
}
specular = {
	filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_s.tga";
}
albedo = {
	filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel.tga";
}
```

The main drawback to this method is that your textures will replace *both* the textures for Hot Rod *and* Default. However, since you are doing this for the benefit of those that do not have the Hot Rod skin, it can be entirely forgiven. To inject these decl files, you will create the file path gameresources_patch2/generated/decls/material2/art/weapons/combatshotgun, place the files in this new combatshotgun folder, add the gameresources_patch2 and inject! Your skin will now replace the default skin and be usable for everyone! For most skins, this is all that will be required. For this method, there is no need whatsoever to ever inject custom normal maps. The vanilla normals will work just fine for your purposes. However, if you are using Substance Painter and you have added additional detail from the skin you have created, you may want to use your own custom normal maps. For those of you to whom this applies, continue on to the second half of this guide.

Custom Normal Maps

Required Tools:

- **SAMUEL by SamPT - Download**
  To ensure the best possible results, you will want to use SAMUEL to specifically extract Normal Maps. This is due to encoding that will be explained briefly. For all other textures, VEGA will do the trick, but you may elect to use SAMUEL for all your textures and simplify the process.

- **VEGA by DTZxPorter (optional) - Download**
  See above.

- **GameImageUtil by Scobalula - Download**
  You will use this program on normal maps extracted by SAMUEL to make them usable in Substance Painter.

- **Substance Painter/Photoshop**
  You will use either one of these programs or both to create the modified Normal maps.

- **Auto Heckin' Texture Converter by PowerBall253 - Download**
  You will use this to convert your Normal maps from Substance Painter to work properly in game.

Before we get started, I will give a brief history of modifying normal maps for Doom Eternal, as it will help you better understand file formats, which is helpful information for best understanding the whole process. At the beginning of texture modding, we found that textures (aside from Normal Maps), need to be encoded as "BC1a" DDS files before using Divinity Machine (now a part of the Texture Converter) to make the TGA files for injection. As a result, VEGA extracted all image files (including Normal Maps) as BC1a and Divinity Machine encoded them as BC1a for injection as well. Normal Maps produced by this process looked visually identical to the Normal Maps from the game in terms of the dark red and green hues, but they would break in game, resulting in terrible, inconsistent lighting.

To compensate for this undesired outcome, a time-consuming, complicated process was devised involving Blender and Photoshop to brighten the hues of red and green, which actually caused them to work in game. So why was this the case?

The red and green in Normal Maps are not processed as colors. Instead, a complex algorithm converts them to information that is used by the game to simulate lighting and reflections. This is why the encoding format and hues of the Normal Map are critical. The process we used for the majority of a year, however, created a problem of its own: Compressed Normal Maps that produced pixelation and loss of information, making shiny surfaces look choppy and wavy; a complete mess. We spent months trying to fix this issue to no avail. However, a solution was eventually discovered and, as it turned out, that solution was impossible for us to find where we were looking because it happened at the beginning before our process of extraction even began.

As mentioned at the beginning, VEGA was extracting Normal Maps as BC1a because it was deduced that they must be BC1a because all other textures required BC1a encoding. This was, however, completely wrong. The normal maps, in fact, required BC5 encoding to work properly and, as of now, SAMUEL is the only app that will give this proper encoding. With it, there is no more damage to the Normals or loss of information at the outset and, using the process which I will now outline, will not be lost in the end product you see after injection.

Creating Your Custom Normal Maps

1. Use SAMUEL to extract your desired Normal Maps. You will notice that these normal maps are in only red and green, which is what Doom Eternal uses. However, they will also require blue in order for Substance Painter to use them properly and, later, export them in a usable state (to understand how to properly utilize and export normal maps in Substance Painter, make sure to watch the tutorial video also posted on this wiki).

2. Run GameImageUtil. In the drop down at the top, select the last option: “XY.” Drag and drop your DDS BC5 Normal Map onto the app window and it will export a BC5 PNG with blue added in the same folder as your DDS image.

3. Run Substance Painter. Import your Normal Maps, create your skin, and then export your customized Normal Maps (PNG).

4. Open your PNG file in Photoshop and save it with DDS encoding. Whether you are using a simple Intel plugin or the NVIDIA encoder, select “8 bit BC5” from your list of options and save the file with the appropriate name (e.g., combatshotgun_barrel_n.dds for the example in the previous section).

5. Drag and drop your DDS file onto the Auto Heckin’ Texture Conerter .bat file and it will convert it to a BC5 .tga file ready for injection. Congratulations! You’ve just created your own custom Normal Map and, when paired with the material2 replacement method, it’s now available for everyone to use!

Coming up in the Advanced Guide, we will discuss a more complex method for assigning event textures to campaign skins and taken on the most complicated process of all: Adding new custom assets. See you there!

Advanced Guide: Advanced Customization & New Texture Creation

Tools You Will Need

- **Substance Painter - Buy (30-day trial)**
  The most common custom textures created for gun and Slayer skins are emissives, which can only properly be created in Substance Painter.

- **Notepad++ - Download**
  You will need this for both decl modification and JSON creation, which will both be covered.

- **EternalResourceExtractor by PowerBall253 - Download**
  If you have not already run this, it is *absolutely essential* that you do so now so that you will be able to read the "true" file extensions of the .tga files you will need to use for custom textures.

- **Resource Priority List - View on Google Sheets**
  This file lists all the .resources files in the game, *in order of priority.* When a file exists in multiple .resources, it will be overwritten by the one highest on this list. This is important when we discuss how and where to inject custom textures.

Advanced Skin Replacement

In the previous guide, we covered how to replace skins using material2 decl files, but now we will cover the advanced method for doing so, which is cleaner, simpler, and much more effective. Returning to the Hot Rod skin from earlier, we will ignore the material2 files under the assumption that, for now, you are merely replacing pre-existing textures.

Instead, go to this file in your extracted resources:

warehouse/generated/decls/warehouseitem/weapon_skin/combatshotgun/combatshotgun_skin_hotrod.decl

This is what we will call a 'warehouseitem' decl after the name of the folder. What does this do? It declares to the game what 'gameitem' decl file it will use for each skin *listing* on the customization menu. What is a gameitem file? A gameitem file declares to the game what model--enumerated for each individual part of the model--it will use for the skin that corresponds to that gameitem. It pairs with the md6def decl, which declares what each model is. However, it is not necessary to modify the gameitem or md6def files for this purpose. Via this new method, only the warehouseitem file will be necessary. Returning to the Hot Rod warehouseitem decl file, opening it in Notepad++ will show you

this text:

```
{
	edit = {
		warehouseItemClass = "WIC_WEAPON_SKIN";
		clientAwarded = true;
		qualityTier = "QUALITY_TIER_1";
		displayName = "#str_decl_cosmetic_slayer_skin_praetor_name_GHOST70738";
		description = "#str_decl_cosmetic_customization_milestone_item_description_GHOST67788";
		icon = "textures/guis/icons/weapons/simple/shotgun";
		targetWeapon = "weapon/player/shotgun";
		gameItem = "weapon_skin/combatshotgun/combatshotgun_skin_hotrod";
	}
}
```

Unlike the material2 decls, almost all of these lines are relevant, including the most advanced touches for really giving your skins a unique flair. For now, however, we will focus on the final 'gameItem' line. Using this method, you can replace a campaign unlock skin in either warehouse *or* common, but for the sake of consistency, we will replace the Shotgun Praetor skin, which is located here:

gameresources/generated/decls/warehouseitem/weapon_skin/weapon_skin_milestone_master_combat_shotgun.decl

Opening this file, if you look at the same gameItem line, you will see this:

```
gameItem = "weapon_skin/weapon_skin_milestone_master_combat_shotgun";
```

The easiest way of thinking about this is that the gameItem line essentially assigns which skin (and model) to which to each menu entry. So, with your custom skin already replacing the textures for Hot Rod, you simply need to copy the gameItem line from the Praetor warehouseitem decl and paste it into the Hot Rod warehouseitem decl, resulting in this:

```
{
	edit = {
		warehouseItemClass = "WIC_WEAPON_SKIN";
		clientAwarded = true;
		qualityTier = "QUALITY_TIER_1";
		displayName = "#str_decl_cosmetic_slayer_skin_praetor_name_GHOST70738";
		description = "#str_decl_cosmetic_customization_milestone_item_description_GHOST67788";
		icon = "textures/guis/icons/weapons/simple/shotgun";
		targetWeapon = "weapon/player/shotgun";
		gameItem = "weapon_skin/weapon_skin_milestone_master_combat_shotgun";
	}
```

```
}
```

Add this file and its folder path listed above to your mod and, when injected, you will have the same result as before in a much simpler and more streamlined fashion. However, your skin will still be replacing both Hot Rod *and* Praetor. Unlike before, however, there is a solution to this when using this method. All you have to do is copy the gameItem line from the Hot Rod warehouseitem file and paste it over the gameItem line of the Praetor warehouseitem file. Adding this decl file and its folder path to your Mod will now cause the Praetor skin to replace Hot Rod, which will result in the player having both the Praetor skin *and* your custom skin. So why, if this method is much simpler, was it saved for the Advanced Guide? Well, that's because there is a lot more you can do with the warehouseitem decl file, which we will now discuss.

Custom Skin Names and Quality Tiers

Let's say you made a really cool glowing read and orange glowing Shotgun. You spent a lot of time getting the metal textures and the emissives just right so that the whole thing looks like superheated steel. After all that effort, you've decided to call this the "Forge Steel Shotgun." But what if, instead of just listing it on your mod page with that name and naming the zip file as such, you could actually have it listed in the game as the Forge Steel Shotgun? Well, that's exactly what you *can* do with the method we'll now go through.

Referring back to the Praetor warehouseitem files, take a look at these two lines:

```
displayName = "#str_decl_cosmetic_slayer_skin_praetor_name_GHOST70738";
description = "#str_decl_cosmetic_customization_milestone_item_description_GHOST67788";
```

It is immediately obvious that these two lines are declaring the name and description of the Praetor skin in the Customization menu, but what exactly is going on with the actual text? Well, those items starting with # are *strings.* Strings, as those familiar with code will already know, are lines of text. Specifically, #str_decl_cosmetic_slayer_skin_praetor_name_GHOST70738 contains the text "Praetor," which is why that is the name displayed on the menu in game. So what are we supposed to do? Export that string and change the text? No. That would be very complicated and time consuming. Instead, we're just going to make up our own strings with these easy steps:

1) Look at the load priority list. As of the current patch (6.66 r-1), the library with top priority is gameresources_patch1, so this is where you will put your strings. Regardless of what what library has top priority, you will need to put your strings in whatever it is at the top of the list. If not, the game will not load them. If you do not already have a gameresources_patch1 folder as part of your Mod, create the folder, then, in gameresources_patch1, create the following path: `EternalMod/strings`

2) Open Notepad++ and create a new file. Click Language>J>JSON. This is where we will create our new strings.

In the JSON file, create this template:

```
{
"strings":
    [
{{
"name": "";
"text": "";
}}
]
}
```

Save the file as "string template.json" so that you can customize it again and again going forward. This clause will be used for each individual string you create. So, let's do what we mentioned earlier and rename your awesome Forge Steel Shotgun. We do it just like this:

```
{
"strings":
    [
{{
"name": "#str_forge_name";
"text": "Forge Steel Shotgun";
}},
{{
"name": "#str_forge_desc";
"text": "A Shotgun Forged From Hardened Darksteel";
}}
]
}
```

Make sure you always put a comma after the closing bracket for each string if you have more that follow (reference: },) or Mod Injector will produce an error and not use the strings. Now save this file in gameresources_patch1/EternalMod/strings as "english.json." If you want to check to make sure you don't have any errors, you can use **https://jsonlint.com/** by pasting your code into the blank in the middle of the page and clicking "Validate JSON." Sometimes, it will identify errors that aren't actually errors because of the specialized code we use for Doom Eternal, but you will learn to identify what errors are real and which ones are false. To test these things out, You can inject the file and if errors appear, test it then.

Returning to the warehouseitem file, you will change those two lines we mentioned earlier to use the string names from your JSON file like this:

```
displayName = "#str_forge_name";
description = "#str_forge_desc";
```

Now, with these changes in place, your textures that replace the Hot Rod skin will show up as the Praetor skin and by modifying the name and description of the Praetor skin, it will be called "Forge Steel Shotgun" and on the right side of the screen, instead of saying that it was unlocked for milestone completion, it'll tell everybody how cool it really is.

Let's be honest, though. The Praetor skin is Quality Tier 1, bronze. Your skin is not a third place medal. It's your most impressive skin yet. It deserves the highest tier, doesn't it? Return to this line:

```
qualityTier = "QUALITY_TIER_1";
```

Changing it to 2 will make it silver, and changing it to 3 (which is what you're going to do) makes it gold. And there you go. You're all set... or are you?

If you like, you can use the ^1^7 name color codes used for assigning colors to your username in any number of modern video games and add them to your name text. Reference the codes in the **String Customization** page.

You'd probably go with yellow or red, making the new name ^1Forge Steel Shotgun^7 or ^3Forge Steel Shotgun^7. And that's everything to do with customizing your skin for your best appearance in game. Now, finally, we get to the biggest means of creating your own skin really *your own:* Brand new textures of your own creation.

Custom Textures

This is where you absolutely must run the most recent version of the Eternal Extractor, which no longer simply extracts "texture" files from the game as *.tga files, but with, as I've mentioned several times now, their *true* file extensions. So what are those file extensions and why are they necessary? After all, when you inject your files to replace textures, you just have to convert them to .tga files, right? Well, this gets down to how Mod Injector actually works. As it turns out, Mod Injector locates the file with the same name in the folder path you create within your zip file and uses .tga as shorthand because that's all Mod Injector needs to *replace textures that already exist.* Adding new textures that don't have anything to replace is a different story. How different? Dramatically.

Going back to the Hot Rod Shotgun file name for an example, these are the files you would most commonly replace for any given weapon skin:

```
combatshotgun_barrel.tga
combatshotgun_barrel_e.tga
combatshotgun_barrel_g.tga
combatshotgun_barrel_n.tga
combatshotgun_barrel_s.tga
```

The *true* file extensions are:

```
combatshotgun_barrel.tga$streamed$mtlkind=albedo
combatshotgun_barrel_e.tga$bc1srgb$streamed$mtlkind=bloommask
combatshotgun_barrel_n.tga$streamed$mtlkind=normal
combatshotgun_barrel_s.tga$streamed$mtlkind=specular
```

There's one file missing though, and that's combatshotgun_barrel_g.tga. Well, that's because *this* is the true file extension for the "smoothness" layer:

combatshotgun_barrel_g.tga$smoothnessnormal=art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_n.tga$streamed$mtlkind=smoothness

This is brought up simply because it's comical; you will probably never have to use it. However, it's important to understand the format of these files. The reason these files have such esoteric extensions is because the vanilla files aren't actual images; they're essentially 'shortcuts' that direct the game to the *actual* textures, which are resized to fit your video settings and streamed live from the streamdb libraries, which you view alongside these resource libraries in the DOOMEternal/base folder.

Now, the most common file you will be creating and adding to the game is emissive textures for parts of models where they don't already exist. So, how do we do this? Well, after you've exported your Emissive texture from Substance Painter (for the Hot Rod Shotgun Barrel, obviously), you want to open in it in Photoshop or GIMP and save it as a DDS with the proper name:

`combatshotgun_barrel_e.dds`

You'll then use the Texture Converter to make combatshotgun_barrel_e.tga. At this point, you need to make sure you put it in the right place, so where is that, exactly? You will want to inject your textures in the highest patch number of the library your skin is in. Currently, that is gameresources_patch1 and warehouse_patch1.

For our current example, create a clone of your Hot Rod skin path in warehouse_patch1: (art/weapons/combatshotgun/skins/hotrod) and in the EternalMod folder, create an 'assetsinfo' folder.

Amend the file extension appropriately:

`combatshotgun_barrel_e.tga$bc1srgb$streamed$mtlkind=bloommask`

Create another new JSON file in Notepad++ and, once again, create the following template and save it as "image template.json":

```
{
"assets":
    [
{
"resourceType": "image",
"version": 21,
"name": "",
```

```
□□□"mapResourceType": "image"
□□}
□]
}
```

Then modify it so that it will add your emissive texture properly:

```
{
□"assets":
    [
□□{
□□□"resourceType": "image",
□□□"version": 21,
□□□"name":
"art/weapons/combatshotgun/skins/hotrod/combatshotgun_e.tga$bc1srgb$streamed$mtlkind=bloommas
k",
□□□"mapResourceType": "image"
□□}
□]
}
```

If you are adding multiple emissive textures, you'll do the same thing as with strings by adding a comma to the closing bracket of the asset (reference: },) and then type the next one. Save this as "warehouse.json" in the warehouse_patch1/EternalMod/assetsinfo folder. The reason why you need to use the full file extension here in the JSON file (and, subsequently, why your texture file needs the full extension) is because it needs to be added to the game's "table" of image textures so that it can be identified by decl files and then assigned to the actual skin. This brings us to the final step.

Acquire warehouse/generated/decls/material2/art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel.decl and open it. Above the 'smoothness' item parm, you'll want to add a 'bloommaskmap' item parm to assign your new emissives like this:

```
bloommaskmap = {
□filePath = "art/weapons/combatshotgun/skins/hotrod/combatshotgun_barrel_e.tga";
}
```

When the game fails to find the texture in warehouse, it will immediately find it in warehouse_patch1 and you'll be all set. Save the file in your own created copy of the file path specified above and, as always, put all your folders in a zip and inject! If you've made any errors in the JSON or decl files, you'll need to make edits to clear them up. If a JSON file has an error, it'll show up in Mod Injector and if there's an error in a decl file, you'll get an error message when the game tries to launch telling you exactly what decl file it is. Don't worry; this is a pretty common mistake. However, once everything is working, you've got your perfect skin modeled exactly how you want it for everyone to admire and

enjoy!

Congratulations! You've finished the entire guide and you should be all set to make skins! If you haven't already, make sure to watch the tutorial video on Substance Painter that accompanies these guides so that you can be ready to do everything in between all these technical steps. Thanks and happy modding!

See Also

- **String Customization**

Getting Started in Substance Painter

https://www.youtube.com/embed/H7EPoIQadFg

In this video, I outline how to get your models into Substance Painter, use Normal Maps, bake your Mesh Maps, and everything else you need to get your project off on the right foot!