

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 gameltem line from the Hot Rod warehouseitem file and paste it over the gameltem 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 red 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 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=bloommask",  
        "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](#)

Revision #14

Created 19 September 2021 14:31:21 by Will Ryan

Updated 12 September 2026 09:36:46 by SamPT