

the GTA V

ped editing guide



about the author

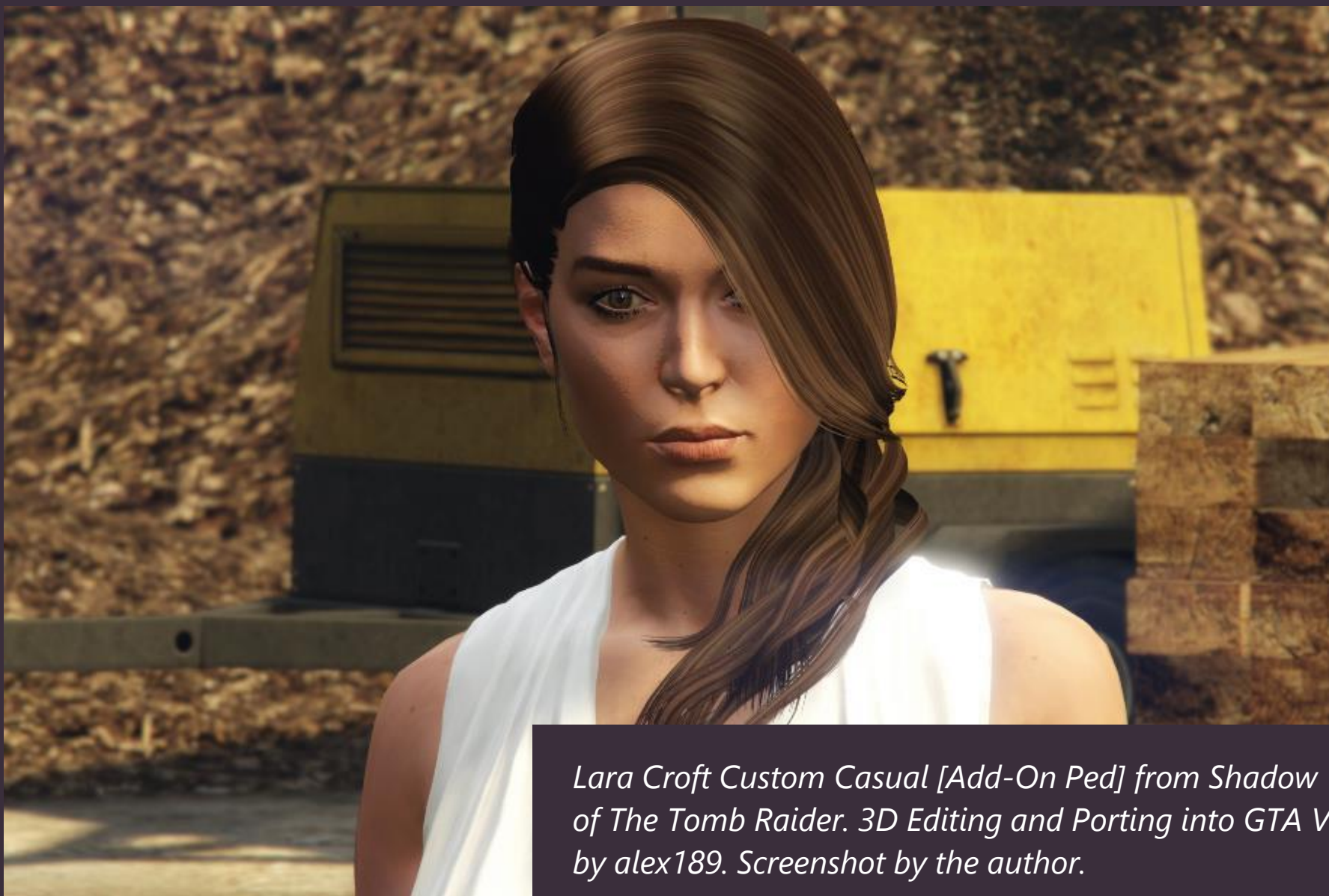


John From GWN

Content Strategist GTA V 911 Help Desk

PREFACE

The ultimate guide for GTA V modding



Lara Croft Custom Casual [Add-On Ped] from Shadow of The Tomb Raider. 3D Editing and Porting into GTA V by alex189. Screenshot by the author.

This guide will teach you how to customize peds without the need for 3D modeling software. It will give you all the background knowledge necessary to understand terms such as components, drawables, and textures. Don't worry about these for now, everything will be explained simply and clearly.

You will also be introduced, with easy to follow tutorials, to a new and powerful tool called the ymt editor. It greatly facilitates the process of editing ped files. The ped ymt files are files containing data about ped body parts and clothes.

Editing the ped files allows you to customize models, enabling you to change their physical features and clothing. Of course that's just one important part of the process.

The guide explains what's going on "under the hood" - how all the parts work together. Follow the articles and you'll become comfortable at making your peds unique to your own taste.

INTRODUCTION

In the sprawling, vibrant world of Grand Theft Auto V (GTA V), the term "peds" refers to "pedestrians," representing the diverse array of non-player characters (NPCs) that inhabit the game's urban and rural landscapes. These NPCs are crucial for creating a dynamic and immersive environment, contributing to the game's realism and variety. Peds in GTA V can be categorized into two main types: **streamed** and **non-streamed**.

Both types of peds are character models that populate the game world dynamically, appearing based on the player's location and actions. This system allows for a rich variety, from everyday citizens going about their lives to unique individuals with distinct appearances and behaviors – from hookers to construction workers and everything in between.

Within the world of GTA V, peds not only enrich the gameplay experience but also fall into various genres, including but not limited to, civilians, police officers, gang members, and characters central to missions and storylines.

Among the most significant peds are the protagonists: **Michael De Santa**, **Franklin Clinton**, and **Trevor Philips**. These three characters drive the narrative of GTA V, each with their unique backgrounds, motivations, and abilities that impact the story and gameplay.

In this guide, we will delve into the process of editing existing peds, whether they are streamed or non-streamed. We will explore how to utilize tools such as OpenIV, as well as simple text editors like Notepad++ for editing data files. Additionally, we will cover how to modify textures and visual elements with image editing software such as Photoshop, Paint.net, and GIMP.

It is important to note that this guide will not cover the full creation of 3D ped models from scratch using 3D modeling software. Instead, our focus will be on customizing existing models to suit your preferences. Whether you're looking for simple texture tweaks or complete makeovers, this guide will provide step-by-step instructions to navigate the editing process effectively.

PED TYPES

As mentioned in the introduction, GTA V peds are categorized as **streamed** and **non-streamed** (or component) peds. As we shall see, the term component is misleading as all peds are made up of components such as heads, hairs, torsos, and legs.

Non-streamed peds are easily recognizable because they are characterized by four files with these extensions: **ydd**, **ymt**, **ytd**, and **yft**. We will discuss these further, as well as other associated files, such as **yld** and **yed**. Most GTA V vanilla peds and downloaded ped mods are non-streamed peds. Modded peds are generally meant to replace existing peds, but they can easily be renamed to keep the original vanilla peds intact.

Streamed peds, on the other hand, are composed of one folder and two files. The folder contains the **ydd** and **ytd** files while the two files on their own are the **yft** and **ymt**. If the ped has a **yld** file, it will be in the folder as well. Examples are the **MP Freemode** male and female as well as the three player protagonists, Michael, Franklin, and Trevor.

In both streamed and non-streamed peds, the file and folder names are consistent and are the same names used to spawn the ped. For example, for the vanilla ped named **a_f_m_bevhills_02** all the files will have that name such as **a_f_m_bevhills_02.ytd**. However this isn't necessarily true for addon peds which are in **dlc.rpf** packages. As we shall see later these peds may have spawn names different from their folder names as is the case for vehicle mods.



Vanilla Peds

Vanilla peds can further be classified based on the role they play in the game. Generally speaking these are the categories:

- **Ambient males & females**
- **Animals**
- **Cutscene**
- **Gang males & females**
- **Multiplayer**
- **Scenario males & females**
- **Story**
- **Story scenario males & females**

PED NAMING CONVENTIONS

Peds are named according to their role in game, as well as their gender, race and age. The structure in general is: **Game Role_Gender_Age_Description_Number**

Regarding age, we have: **y=Young**, **m=Middle Aged**, **o=Older**

Ambient Peds

The peds you encounter around the GTA V map, called ambient peds, follow this structure:

a_f_m_beach_01. This can be deciphered as an ambient female, middle aged, likely to hang around the beach. She is also given a number as a suffix, even if she is one of her kind.

Animals

Peds of the animal variety, always start with **a_c** as in **a_c_chickenhawk**.

Cutscene

Cutscene peds are recognized by a **cs** prefix as in **cs_janet**

Gang

Gang peds start with the prefix **g** as in **g_f_y_vagos_01**

Multiplayer

These peds that start with the prefix **mp** as in **mp_m_fibsec_01**

Scenario

The peds that start with the prefix **s** as in the **s_f_y_hooker_01**

Story

The peds that start with the prefix **ig** (meaning in game) like **ig_Abigail** and also a few that start with the prefix **hc**, for example **hc_gunman**

Story Scenario

These final peds start with the **u** prefix as in **u_m_y_rsranger_01**



PED FILES

While we will get into more detail, let's take a quick look at the **Y files** which both streamed peds and non-streamed peds have in common. The naming conventions are those seen in OpenIV (pronounced Open 4 as in GTA 4).

The prefix Y is for the PC version of GTA V whereas GTA IV used W. For example the GTA V textures are in **ytd** versus **wtd** for GTA IV.

YFT – Fragment Object

The YFT file is the 3D model skeleton of the ped. It contains the geometry of the character that defines its shape and structure. This file is essential for rendering the character within the game environment.

Example for all peds: g_f_y_vagos_01.yft

In OpenIV, you can export this file as a .skel (skeleton) file and view it in a text editor. It can't be edited directly without 3D software.

YDD – Drawable Dictionary

The YDD file contains drawables, these are 3D meshes for the 12 ped components such as hair, uppers (torsos), and lowers (legs). A typical ped will have 3 meshes: high, medium, and low. These are levels of detail (LODs) and high is the most complex 3D geometry.

Non-streamed peds have YDD files with many components bundled together in the ydd file.

Example: g_f_y_vagos_01.ydd

Streamed peds have their YDD files in a separate folder and each YDD has only one component, for example a mesh for one hair style.

Example: hair_000_u_1.ydd

Both types of peds can be exported to what is called open formats as we will see later on. YDD files can have embedded textures, use YTD files, or have both embedded and YTD textures.

Finally, other GTA V models such as props or car components such as wheels are generally YDR files (single drawables).

PED FILES

YTD – Texture Dictionary

The YTD files store textures for peds that define how surfaces of 3D models appear. The terms diffuse, normal, and specular refer to different types of texture maps included in a YTD file.

Diffuse Map

This is the most basic texture and represents the base color of a surface. It determines what color the surface will appear and is vital for giving an object its overall visual characteristics. It doesn't account for any lighting, shadows, or surface details beyond color.

Example: `head_diff_000_a_whi`

Normal Map

A normal map is used to give textures a more detailed appearance without increasing the polygon count of a 3D model. It achieves this by simulating small surface details such as bumps, wrinkles, or grooves using RGB information. By altering the way light interacts with the surface, normal maps provide an illusion of depth and complexity.

Example: `head_normal_000`

Specular Map: This type of map defines how shiny or glossy a surface appears by affecting the specular reflection of light. It determines the highlights and shine on an object's surface, allowing for realistic rendering of materials like metals, plastics, or wet surfaces. The specular map can inform the rendering engine of which areas are more reflective and to what extent.

Example: `head_spec_000`

Non-streamed peds have all textures bundled together in the ytd file.

Example: `g_f_y_vagos_01.ytd`

Streamed peds have YTD files in a separate folder and each YTD has only the textures for the its ydd component, for example a mesh for one hair style.

Example: `hair_000_u_1.ytd`

Textures can be imported, exported, renamed, deleted, etc. The native format is .dds (DirectDraw Surface) format. To edit textures you can use essentially any image format, OpenIV will convert them to dds but often dds will give the best results. We will discuss this further.

PED FILES

YTD – Texture Dictionary

You can view examples of diffuse (diff), normal, and specular (spec) below.

Example.ytd - OpenIV Texture editor

+

Import

Properties

Rename

Replace

Delete

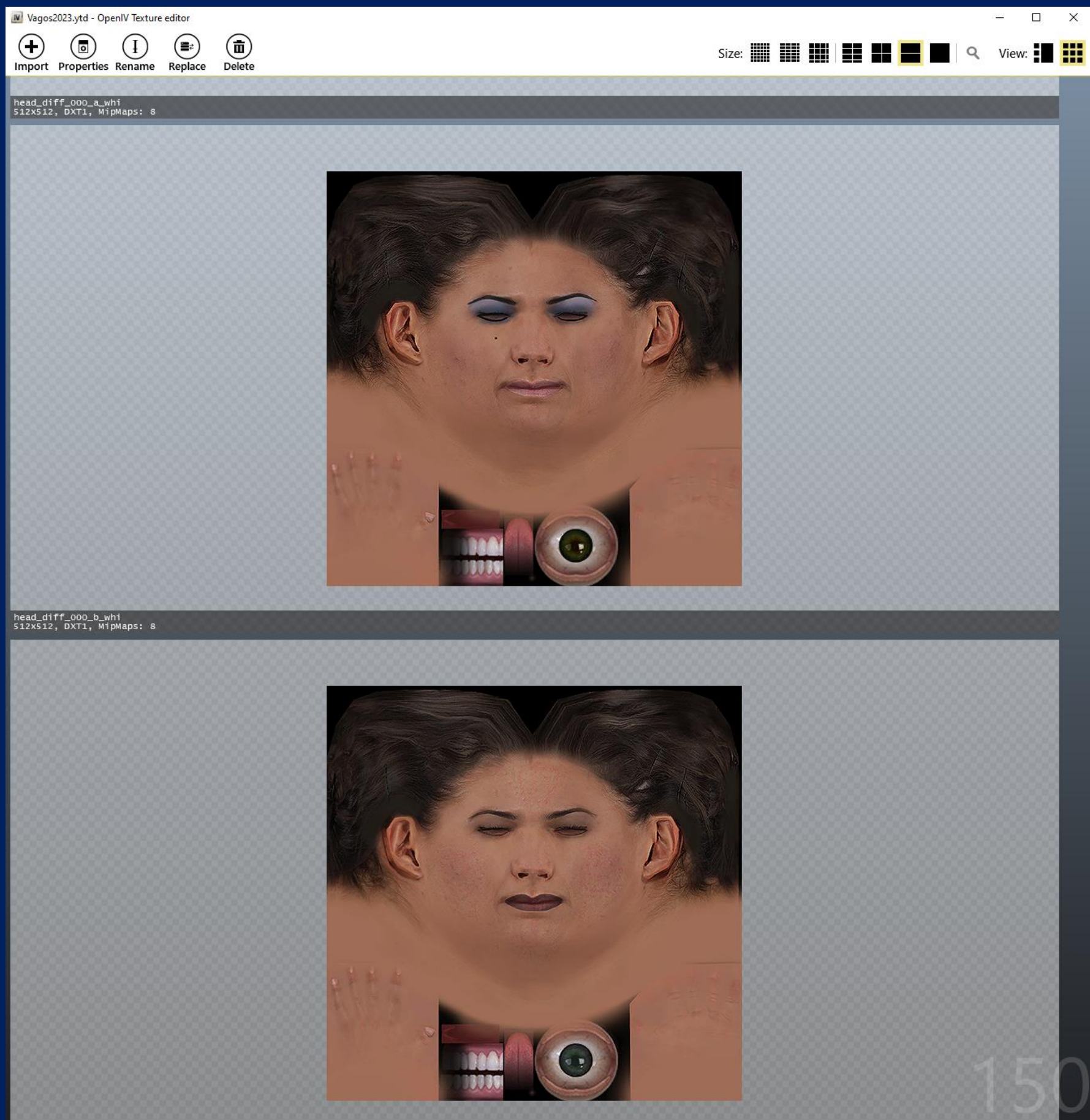
Size:

PED FILES

Anatomy of a Texture

The screenshot below is from OpenIV (Open 4) and is zoomed in to display two head images. The naming convention follows a specific structure. If we consider **head_diff_000_a_whi** we have:

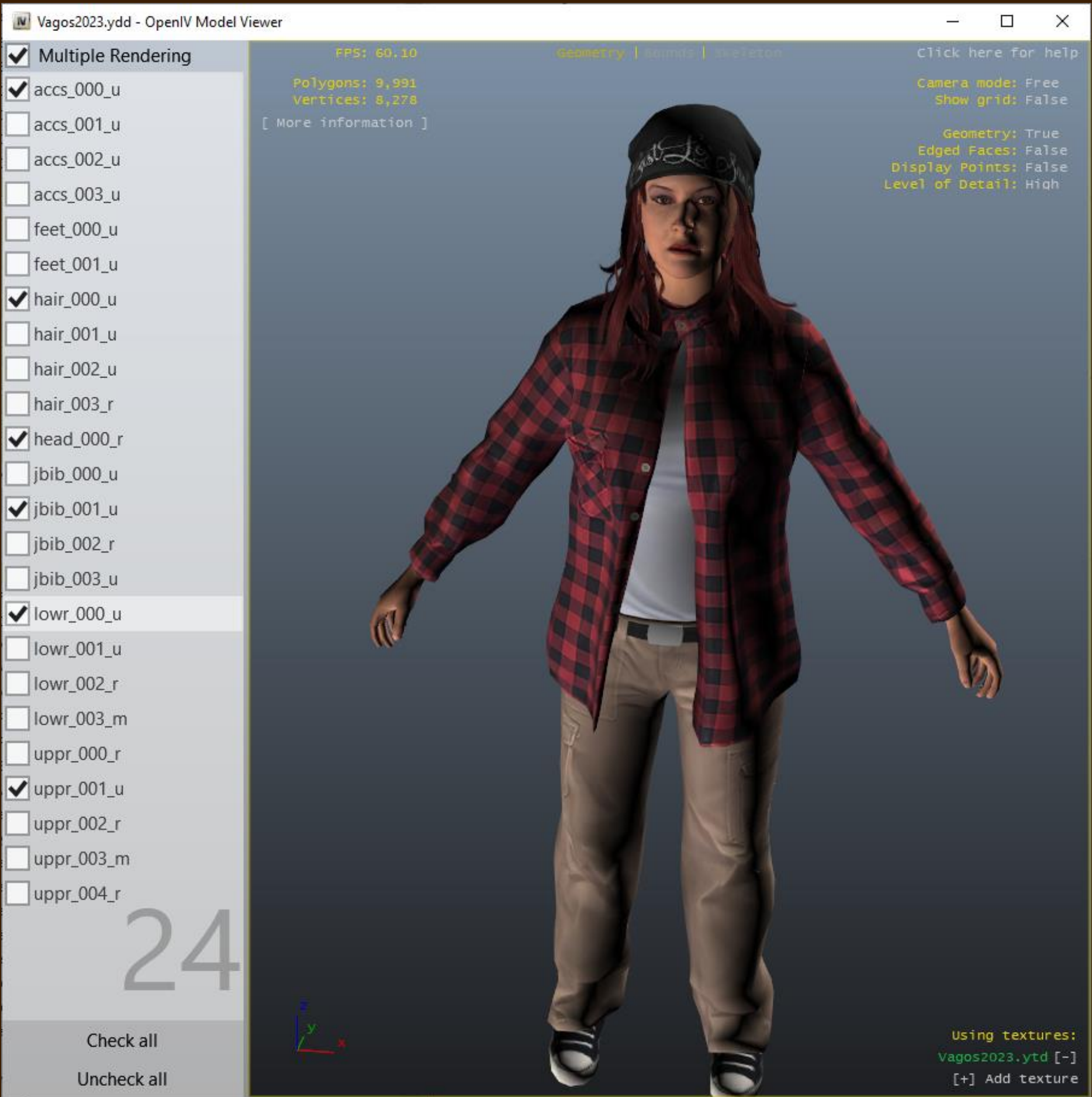
The **component** Head (slot 0), the **texture type** (diff = diffuse), the **drawable** (000 is the first, 001 would be second), the drawable first **texture variation** (ranges from a to z), and finally the race, which in this case is whi for white. You'll notice, if you look closely, that this particular ped texture not only has the head (face) but also the hands, teeth, and eyeballs.



PED FILES

YMT – Metadata

This file contains metadata for the character model. It is the most important file in this guide and it will be covered in depth. To examine this file, in the past, it needed to be exported as an xml file – allowing it to be viewed as text. Thankfully now there is a free tool mod called the ymt editor which greatly simplifies the process. The YMT file contains data for all 12 ped components and their textures. The screenshot below, from OpenIV, shows a typical non-streamed ped along with different components which we will discuss in the next section.



YMT AND COMPONENTS

So why exactly are ymt files that important for customization? The Ped ymt files are manifest files containing metadata about Peds. They allow you to customize models by changing their physical features and clothing where applicable. Not to be confused with the single file called peds.ymt.

Example for all peds: g_f_y_vagos_01.ymt

Components

There are 12 wardrobe components, as described in the table below. They are numbered from 0 to 11 and each component can have multiple drawables, often referred to as their slots. These are the same components with your ped outfits using Trainers like Menyoo.

Native Name	Description	Menyoo Name
PV_COMP_HEAD	0. Head	Head
PV_COMP_BERD	1. Masks	Beard/Mask
PV_COMP_HAIR	2. Hair Styles	Hair
PV_COMP_UPPR	3. Torsos	Torso
PV_COMP_LOWR	4. Legs	Legs
PV_COMP_HAND	5. Hands/Bags/Parachutes	Hands/Back
PV_COMP_FEET	6. Shoes	Shoes
PV_COMP_TEEF	7. Accessories	Teeth/Scarf/Necklace
PV_COMP_ACCS	8. Undershirts	Accessory/Tops
PV_COMP_TASK	9. Body Armor	Task/Armour
PV_COMP_DECL	10. Decals	Emblem
PV_COMP_JBIB	11. Tops	Tops2 (Outer)

YMT AND COMPONENTS

Drawables and Textures

If you consider the component called upper (torso), then its drawables would refer to clothing such as a shirt, or a sweater, or a t-shirt. For the component lower (legs), the drawables could be pants or a skirt. Remember that each component can have multiple drawables. Also, each drawable can have multiple textures (up to 26, from A to Z). A texture on the other hand could allow one t-shirt to be red or black or sport a cool logo. These different texture options are called texture variations. Together with drawables they are called component variations and apply to the player and all peds.



The Wardrobe in Scripts

Just a side note as to how wardrobe components are changed in script mods. As you're aware, changing clothes and outfits is easily done with trainer mods like Menyoo and Simple Trainer. The trainers use what are called native functions to allow you to make these wardrobe changes in game. For example GTA 5 has the following native function:

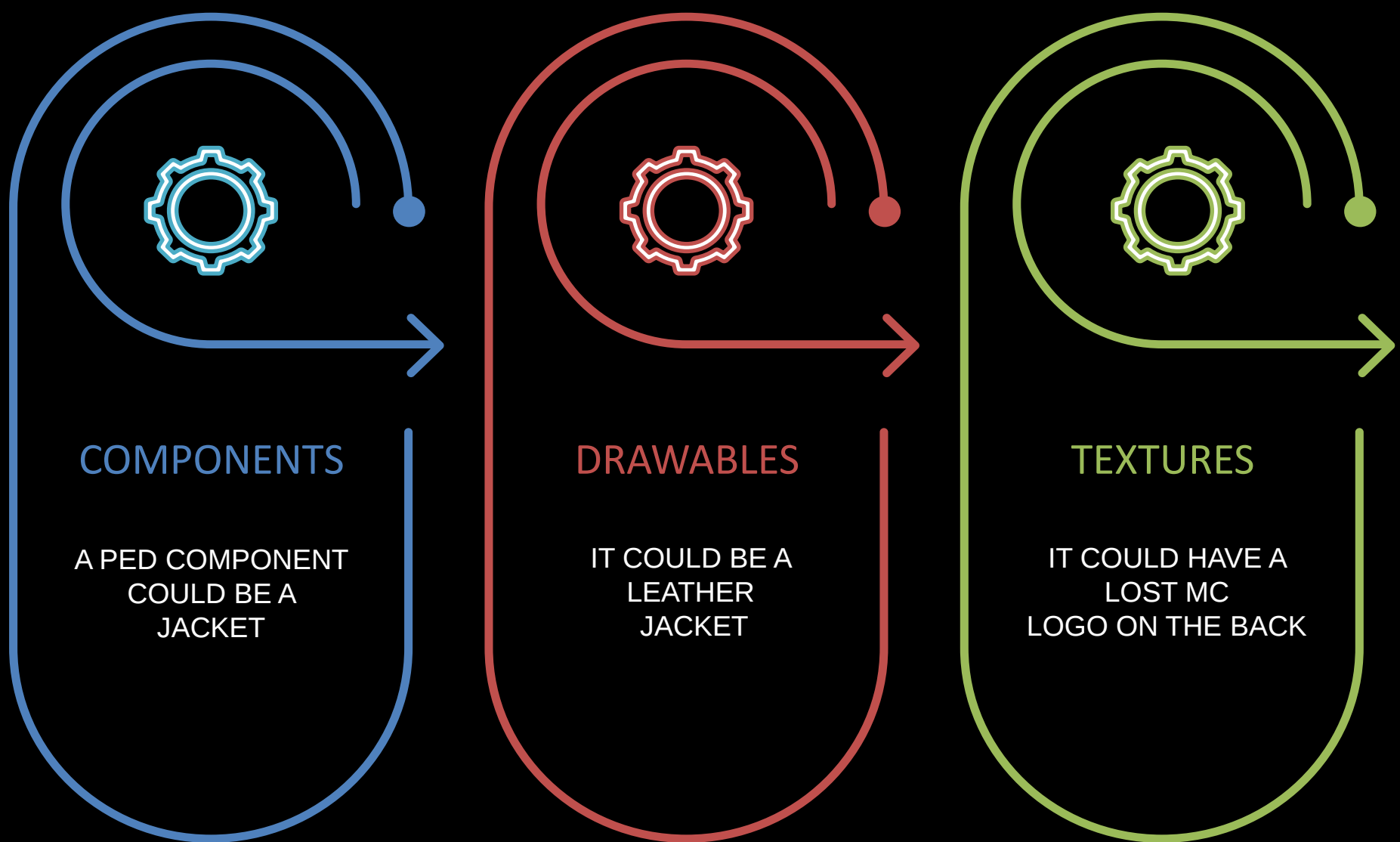
```
void SET_PED_COMPONENT_VARIATION(Ped ped, int componentId, int  
drawableId, int textureId, int palettId)
```

As an example, the function below would set the hair (component 2) to hairstyle 3 (ex. Long hair) and texture 2 (ex. Blonde hair). The last variable is always set to the value 2.

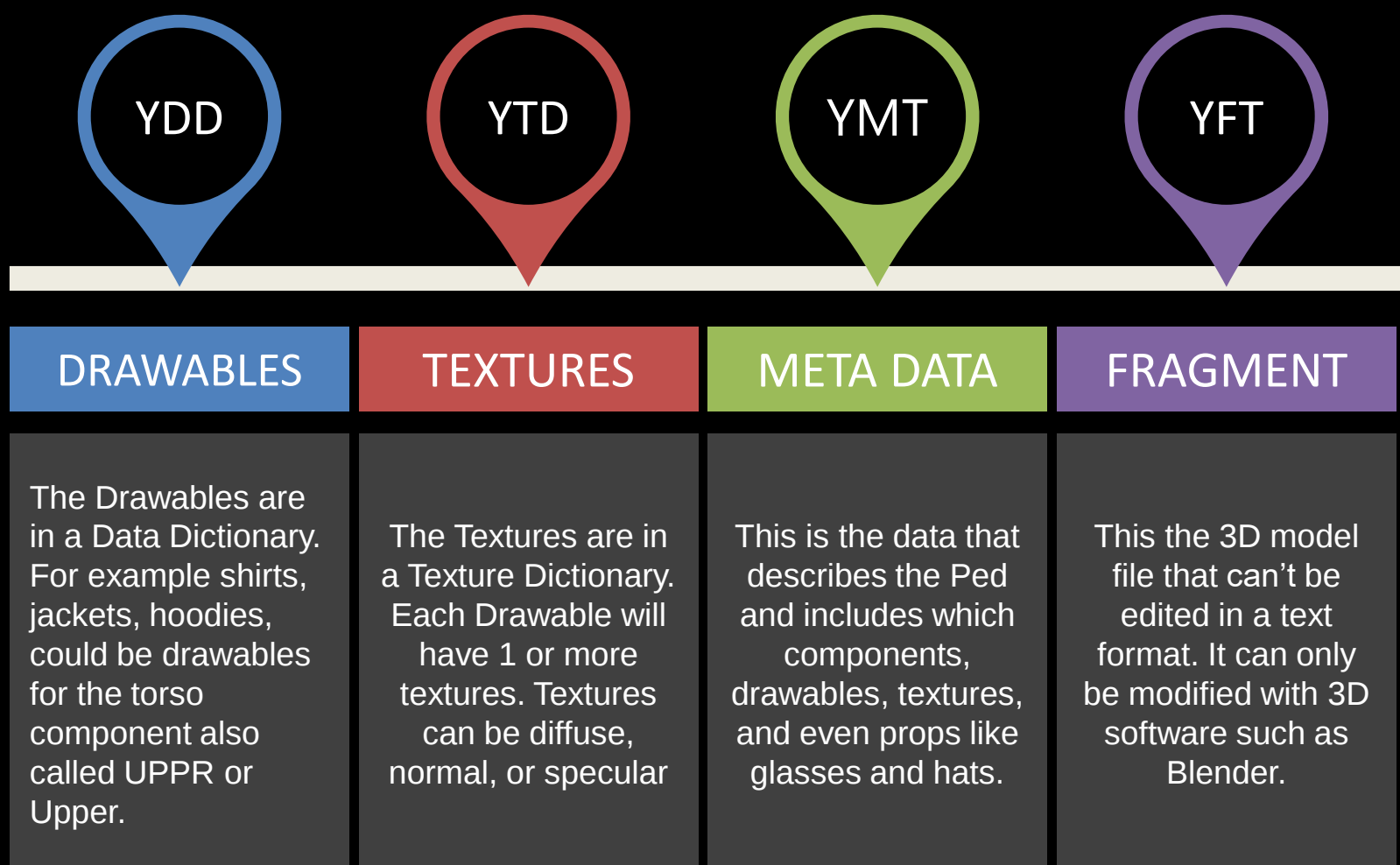
```
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, PedName, 2, 3, 2, 2);
```

So in summary, each component, for example a torso, has id's for the torso component itself as well as for the drawables and textures.

The Ped Wardrobe



The GTA V Ped Files



TUTORIAL

Add Drawables and Textures to Peds

In this tutorial you will learn, step by step, how to add drawables such as hairstyles or jackets or pants as well as different texture variations for those drawables. The process here applies to both non-streamed and streamed peds, the most important difference being non-streamed peds have drawables in one ydd file whereas streamed peds have individual drawables in their own ydr files.

What you will need:

1. Basic understanding of working with Windows files
2. OpenIV (Open 4 as in GTA 4)
3. YMT Editor
4. Code Walker RPF Explorer
5. A text editor (Notepad, Notepad++, Visual Studio Code, etc.)
6. Image Editor (Photoshop, GIMP, etc.). This is only required if modifying textures.

What you won't need:

1. Knowledge of 3D Modeling (although it can't hurt)
2. 3D Modelling Software (ZModeler3, Blender, 3ds Max, etc.) - this is optional

So the best part is all you need to know is how to edit text and of course an understanding of how components, drawables, and textures are the foundation of ped models.

Backup Files

You should backup your ped files in case you make a mistake or want to restore them. We will be working with only 3 of the 4 files present for most non-streamed Peds, but backup all four. The ones we will work with are: **ytd, ydd, and ymt**. We will not preoccupy ourselves with yft.

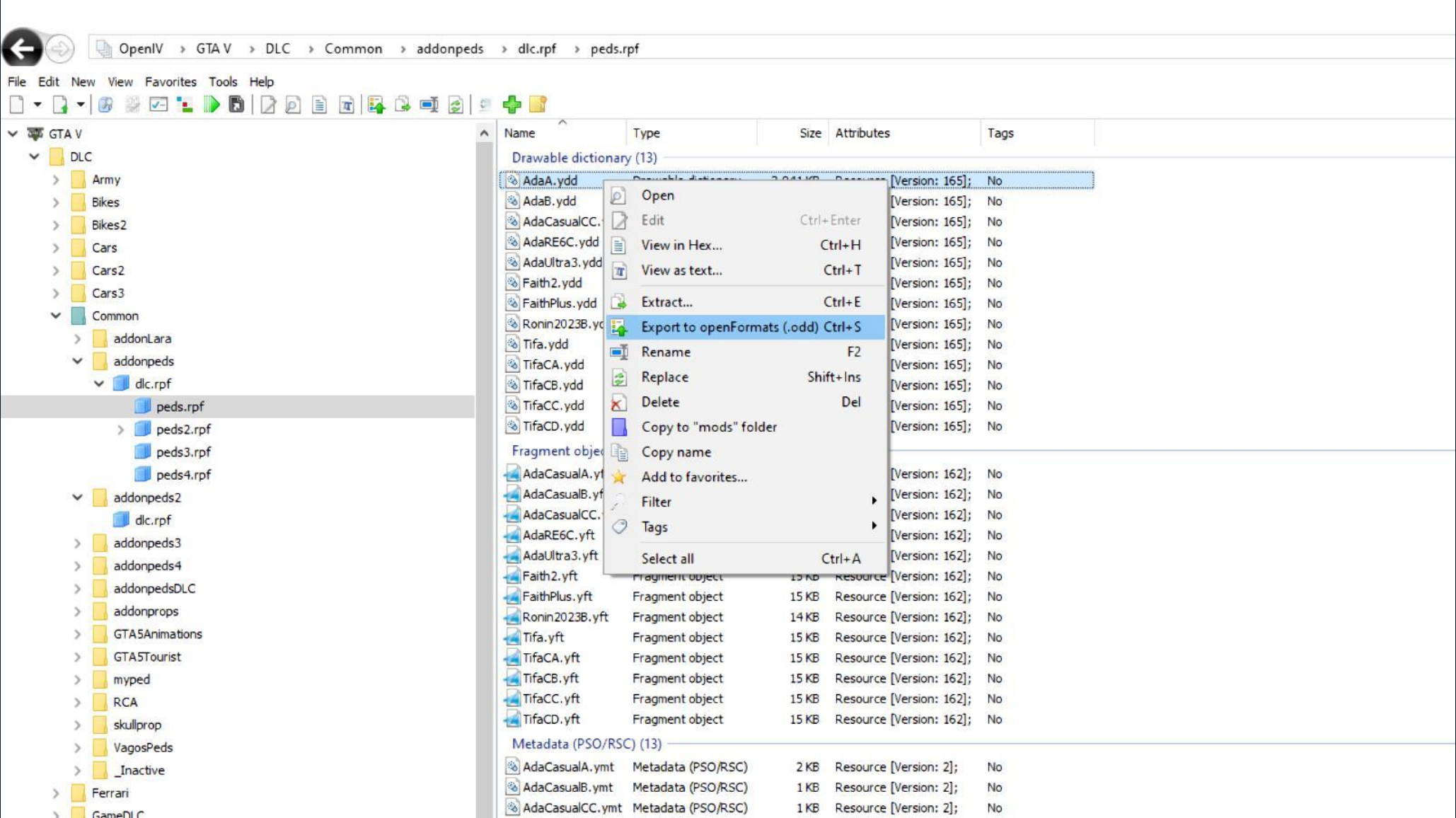
TUTORIAL

Step 1 - Export to Open Format

Although many of these steps can be taken in different chronological order, stick with this sequence for now until you're comfortable with the process. We will learn how to transfer hair drawables, in ydd files, from ped AdaB to ped AdaA.

To accomplish this we will obviously need to have the peds properly installed and then we will use OpenIV to export both ydd files (from two similar peds) to the **openFormats (.odd)**. This operation will provide us access to all the drawable components such as hair, lowr, uppr, etc. In our demo we will be focusing on hair alone.

From OpenIV, select the AdaA.ydd, right click, and from the context menu select Export to openFormats. If you don't have AdaA installed (Ada Wong mod) use any non-streamed ped. Repeat this process for a second ped, in our example it will be AdaB.ydd.



TUTORIAL

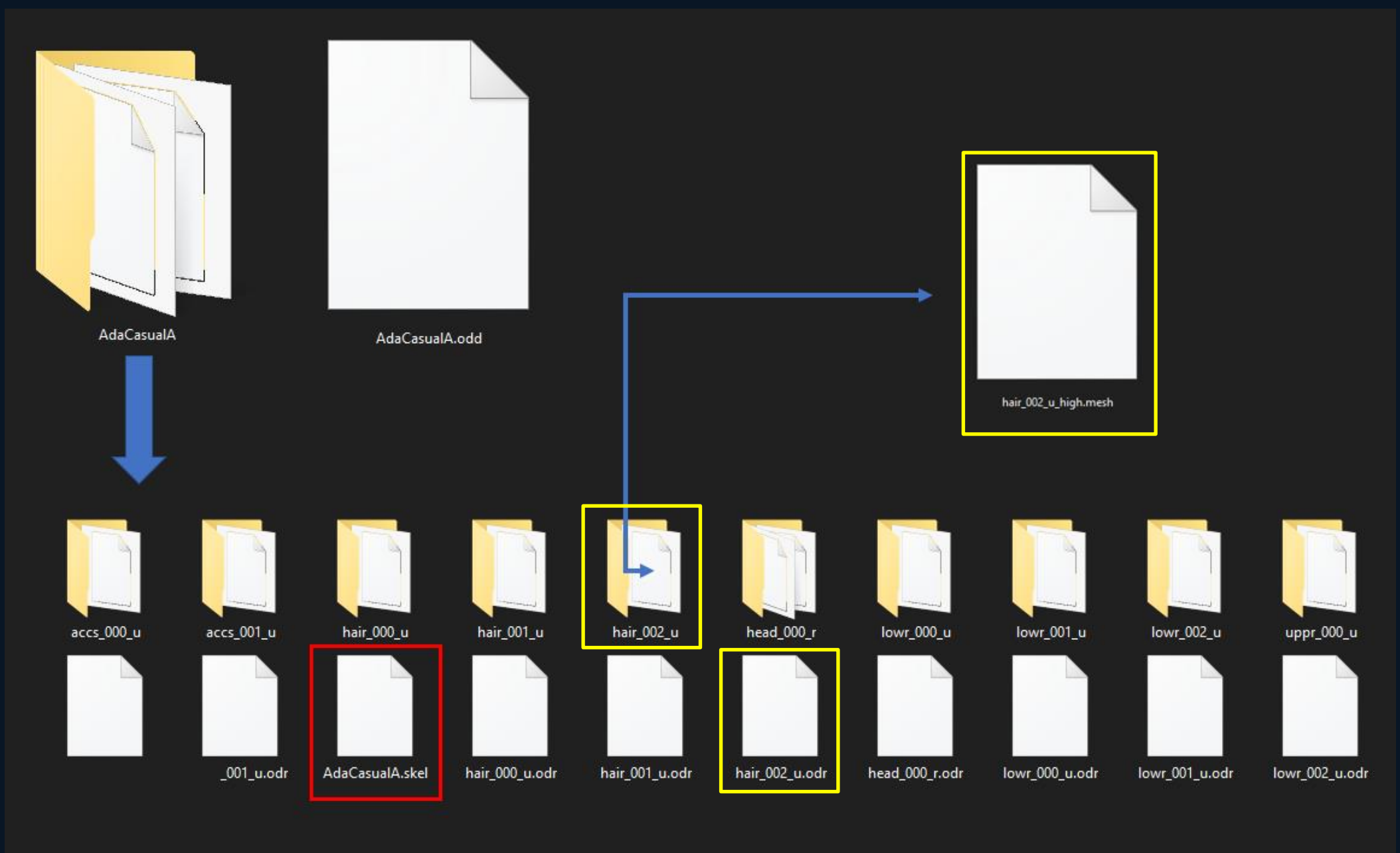
Step 1 - Continued

The Screenshot below is what you expect to see after converting a ydd from a non-streamed ped to the open format version called odd. In this example, we have one folder called **AdaCasualA** and one file called **AdaCasualA.odd** which is a plain text file. Inside the AdaCasualA folder we will find a series of folders, one for each drawable and a matching odr file.

For example the **folder hair_002_u** has a matching file called **hair_002_u.odr**. Inside the folder we will find the 3D model (mesh) for the hair called **hair_002_u_high.mesh**. Together these represent the third hair drawable ydr for our ped because the numbering starts at 0.

So an exported ydd (drawable) becomes and odd. The ydr files become odr files grouped together in one folder. Inside that folder are all individual folders for all drawables.

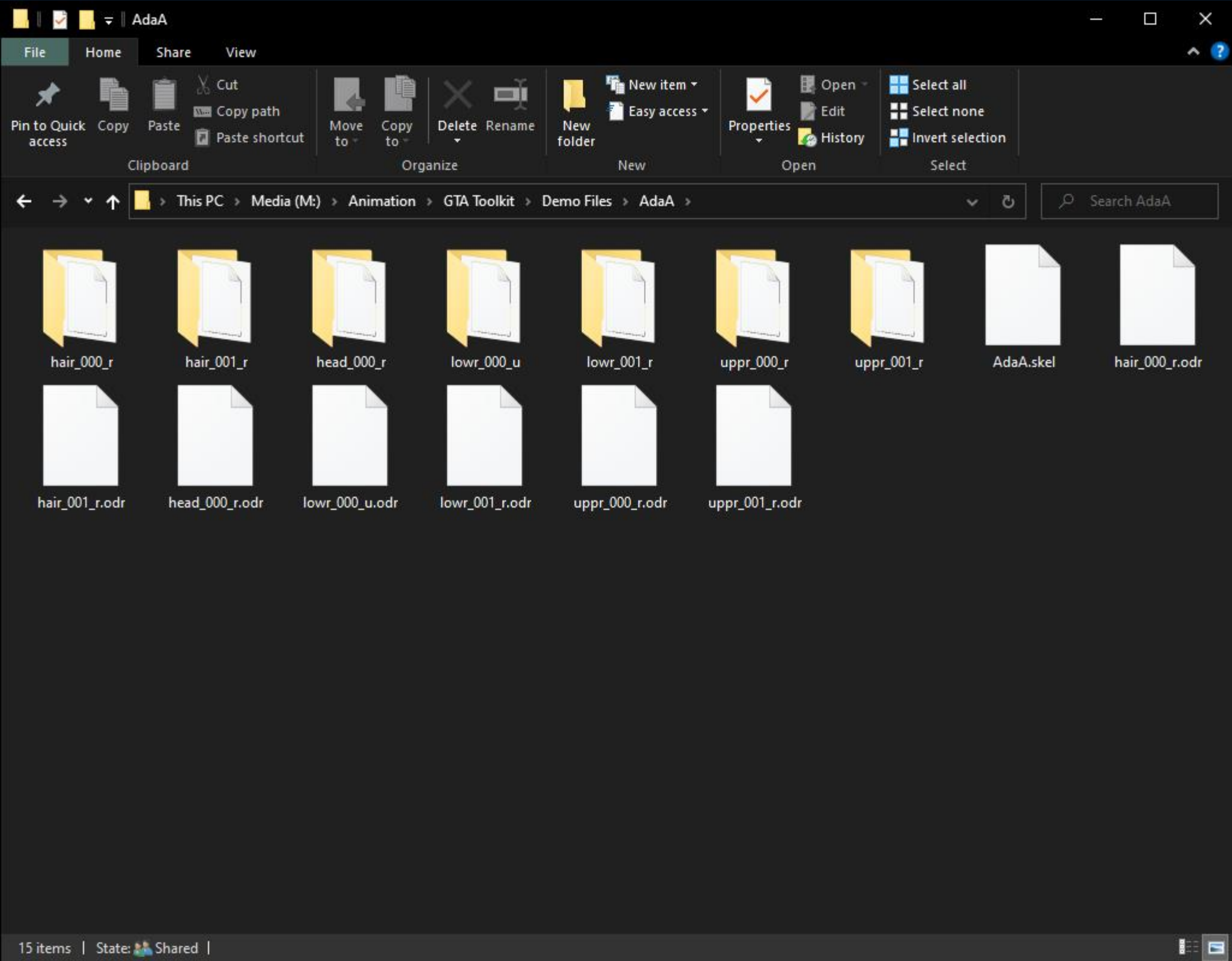
One final note, you will also notice there is a **.skel (skeleton)** file which we will not edit.



TUTORIAL

As we read previously, if you successfully exported the ydd file to openFormats you will end up with one .odr file and a folder. The folder contents will look like the screenshot below, depending on the model you customized. You need to do this for both your peds.

If the ydd file is locked, you will get an error message, when attempting to export to OpenIV. In that case you will need another step to easily unlock the ydd file. First close OpenIV so that the ydd file is free to use. Next use RPF Explorer, from CodeWalker, to export as xml and lastly import back as xml. This will unlock the file and restore it as an unlocked ydd.



TUTORIAL

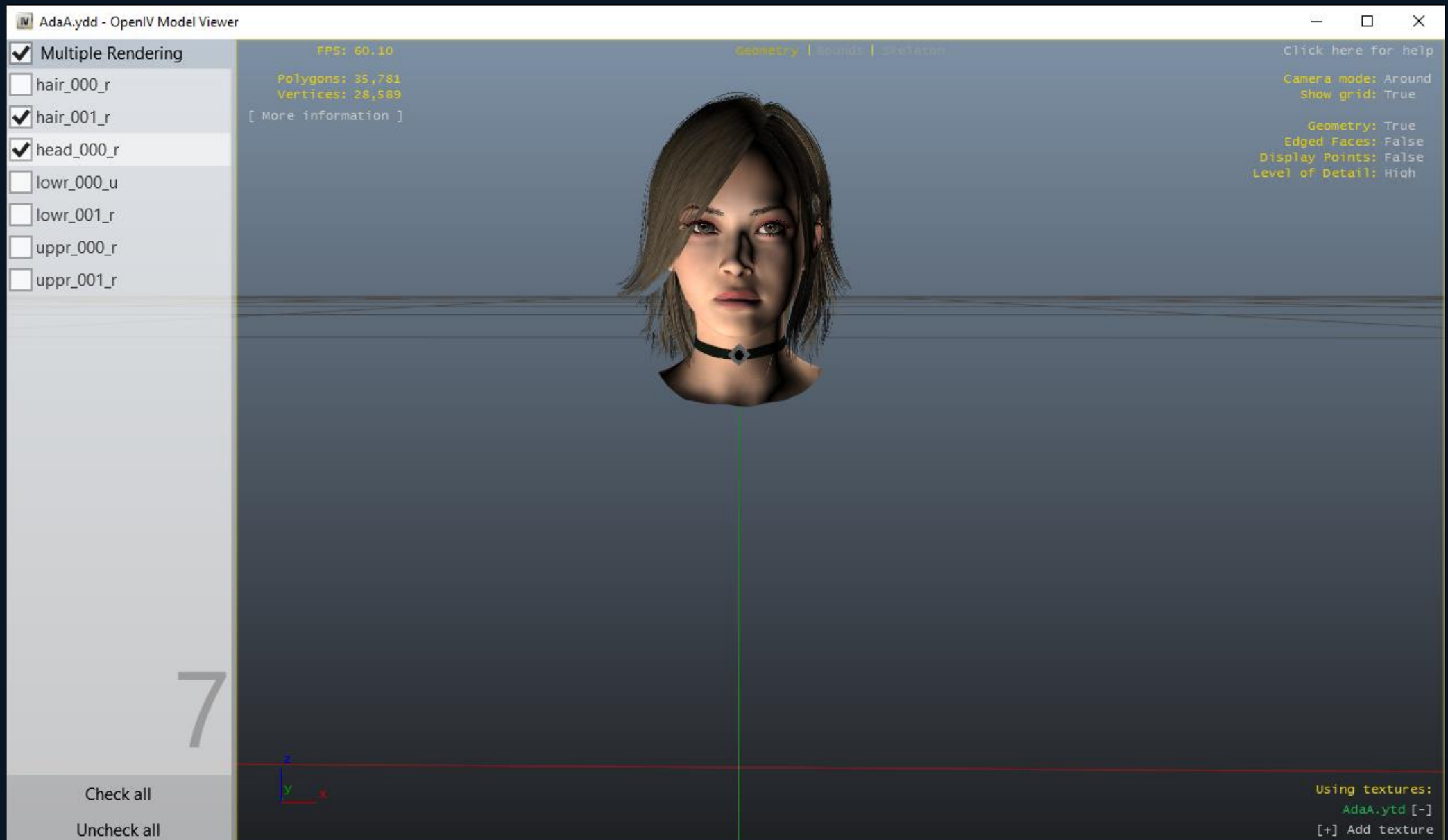
Step 2 - Editing the YMT file

While YMT files can be edited manually with tools that convert the file to editable text, a recent mod called the YMTEditor simplifies the process and reduces the risk of errors. In passing you should note, that we can't edit YMT files directly from OpenIV.

Now that we've successfully exported drawables in the openformat, this next step consists of editing the YMT file (metadata) to add a new drawable. Once more, in our demo this is a new hairstyle. We are going to take a hairstyle from Ped AdaB and add it to Ped AdaA.

AdaA already has 2 hairstyles (drawables), one numbered 000 and the other numbered 001. The hairstyle we will add will be numbered 002. Before we can add the hairstyle from one ped to the other, we need to edit AdaA so that she can receive the new hairstyle.

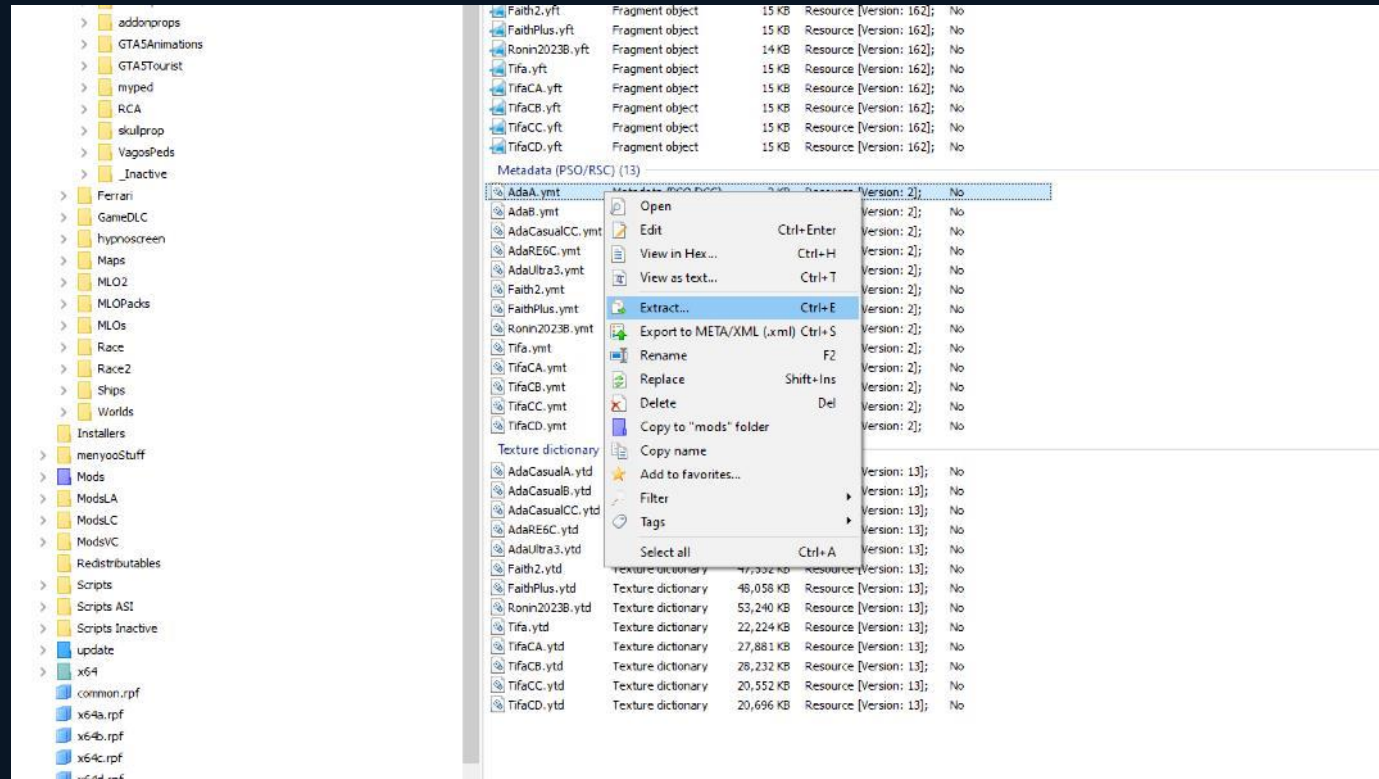
From OpenIV, by clicking on the YDD (Drawable Dictionary) you can see the drawables for your ped model such as the head, hair, uppers, and lowers. Notice hair 000 and hair 001 in the screenshot below from OpenIV.



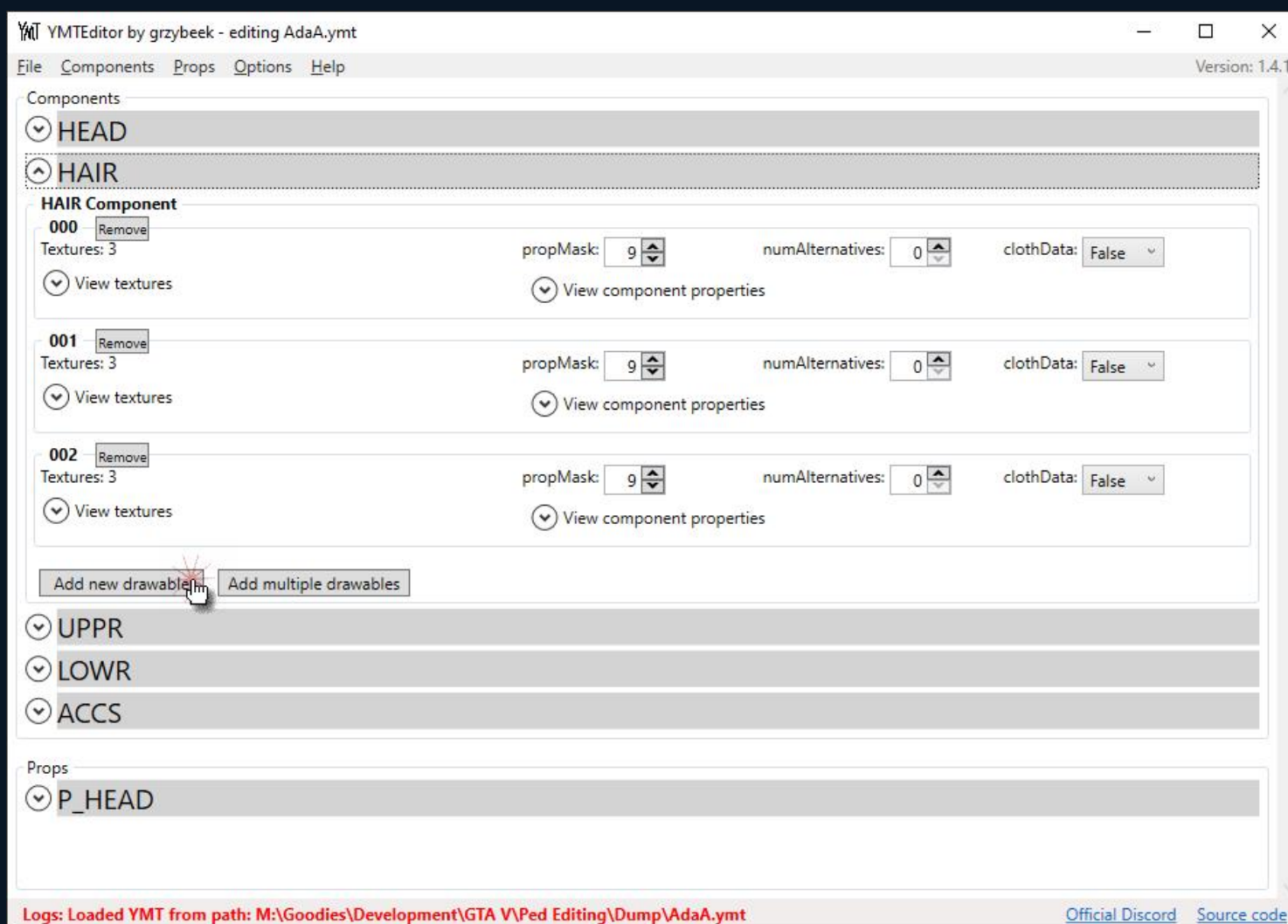
TUTORIAL

Step 2 - Continued

We have to extract our ymt to a folder on our computer. This is done from OpenIV and we only need to do this for one ped, AdaA.ymt in our example.



We can then drag our extracted ymt onto the YMTEditor. Next we select, **HAIR** from the components and then **Add New drawable** which will be numbered automatically as 002. We don't need to add a texture entry as there is one already by default. Finally we save our modified ymt file and drag it back into OpenIV to replace the original one.



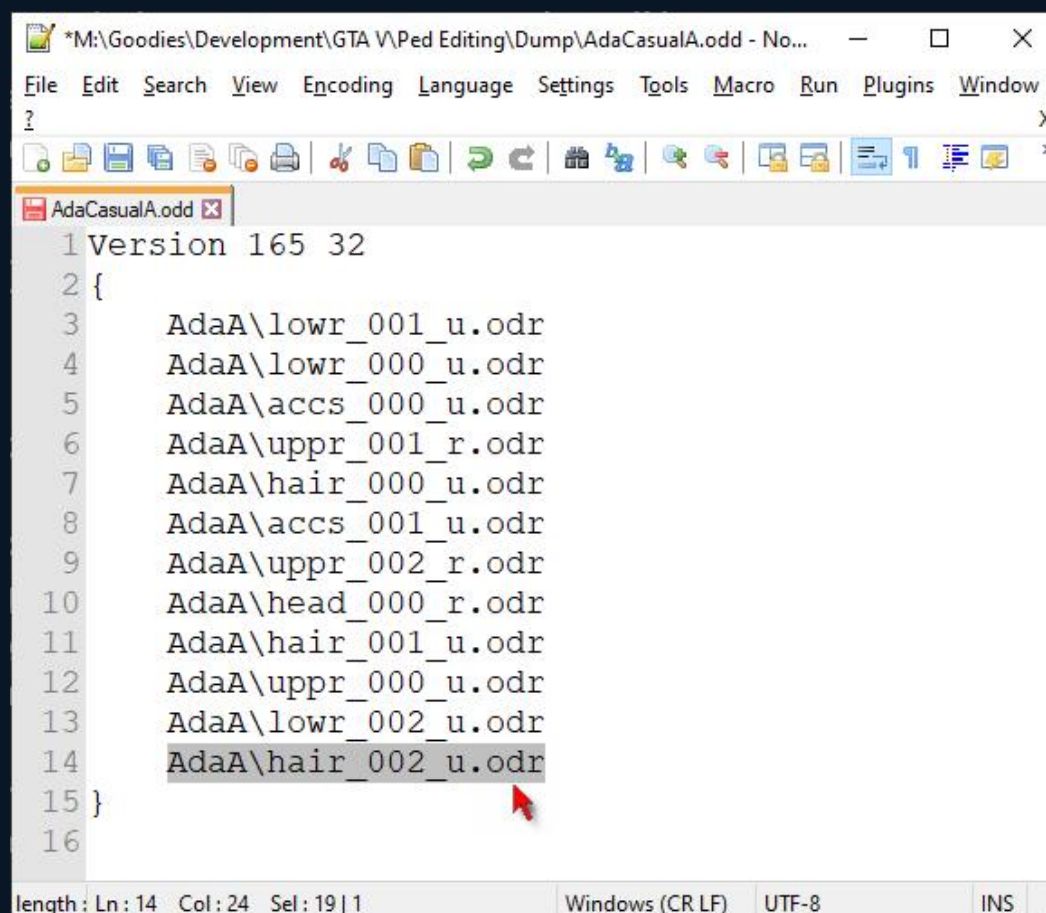
TUTORIAL

Step 3

In this step we add 1 hairstyle drawable from AdaB to AdaA, renaming it 002. We then bring the odd file back into OpenIV to create a new YDD file for our ped Ada A. This is simply done, once we've completed our renaming/edits, by dragging and dropping the .odd file back into OpenIV to create the ydd. We don't worry about any folders, OpenIV will take care of the updating process.

The files that need renaming are:

1. The Folder (AdaA) containing subfolders with meshes. For example, hair_000_u.
2. Inside hair_000_u, which we will rename hair_002_u, there will up to 3 meshes. We will rename these as well. For example, **hair_000_u_high.mesh** to **hair_002_u_high.mesh**.
3. We will also rename the odr file, for example hair_000_u.odr to hair_002_u.odr
4. With a text editor such as Notepad (or Notepad ++), we now do edits inside the odr file. As examples, **DiffuseSampler hair_diff_000_a_uni** will be renamed **DiffuseSampler hair_diff_002_a_uni**. Also, near the bottom of this file we will rename **hair_000_u\hair_000_u_high.mesh 0** to **hair_002_u\hair_002_u_high.mesh 0**
5. Finally, we will edit the odd file that is outside the AdaA folder. We will make an entry for the new hair drawable as in screenshot below. And as a last action, we will drag that odd file, and only that file, into OpenIV where it will make all the adjustments necessary. An odd file is shown below.

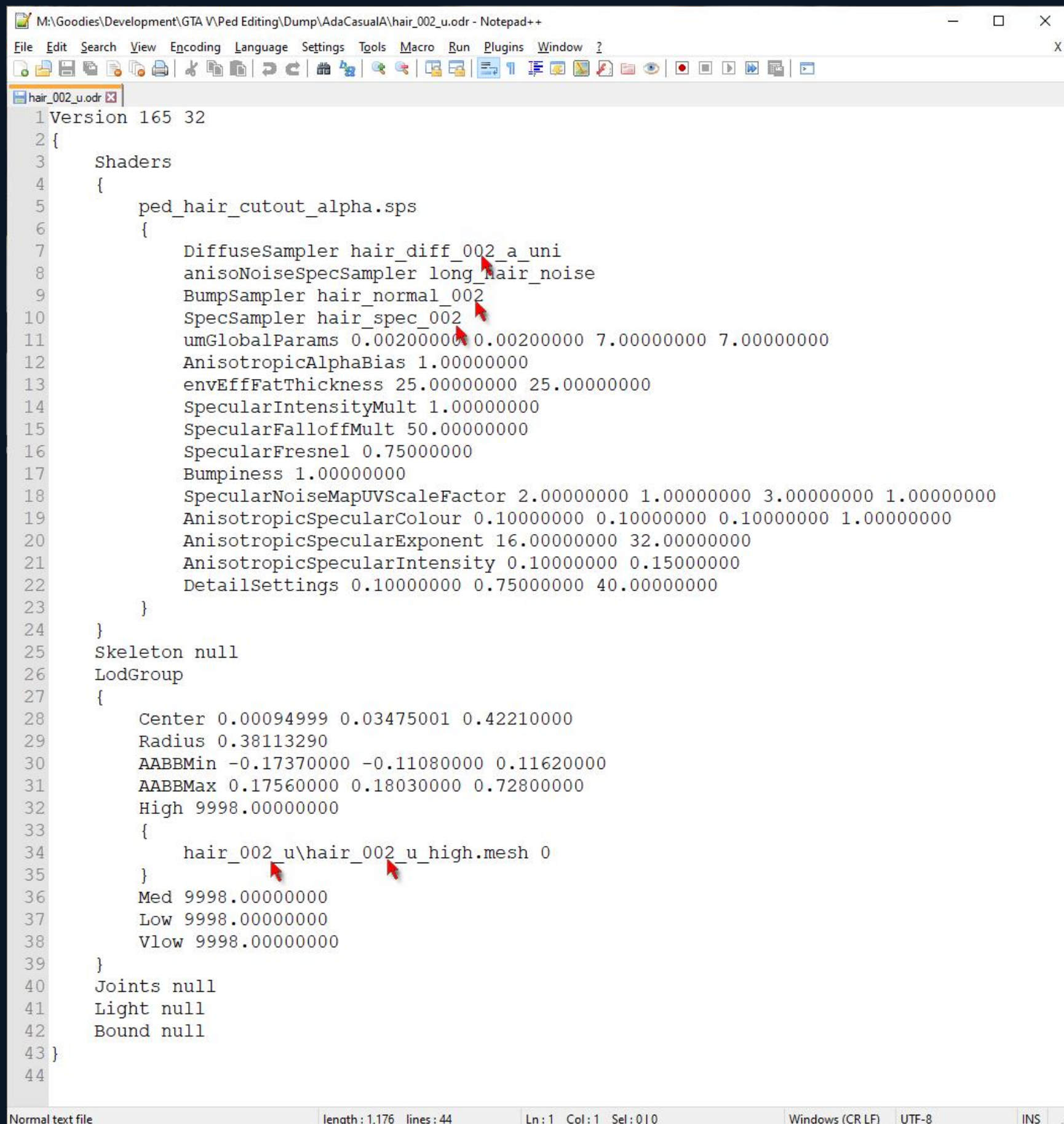


```
*M:\Goodies\Development\GTA V\Ped Editing\Dump\AdaCasualA.odd - No...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window
?
AdaCasualA.odd x
1 Version 165 32
2 {
3     AdaA\lowr_001_u.odr
4     AdaA\lowr_000_u.odr
5     AdaA\accs_000_u.odr
6     AdaA\uppr_001_r.odr
7     AdaA\hair_000_u.odr
8     AdaA\accs_001_u.odr
9     AdaA\uppr_002_r.odr
10    AdaA\head_000_r.odr
11    AdaA\hair_001_u.odr
12    AdaA\uppr_000_u.odr
13    AdaA\lowr_002_u.odr
14    AdaA\hair_002_u.odr
15 }
16
length: Ln: 14 Col: 24 Sel: 19|1 Windows (CR LF) UTF-8 INS
```

TUTORIAL

Step 3 – Continued – ODR/YDR

As we saw earlier, drawables such as head, hair, uppers, and lowers are in **ydr files** which OpenIV converts to **the open format .odr**. For non-streamed peds, ydr files are grouped together in a drawable dictionary called a **ydd file** - which OpenIV converts to **.odd**. We saw on the previous page that our odd file lists all its odr files. In the screenshot below we see a sample odr file (open format of a ydr) and you can see how the textures (**diffuse, normal, and spec**) are properly named to reflect our changes as is the mesh and the folder that contains the mesh (or meshes). This is the 3rd hair drawable for this ped.

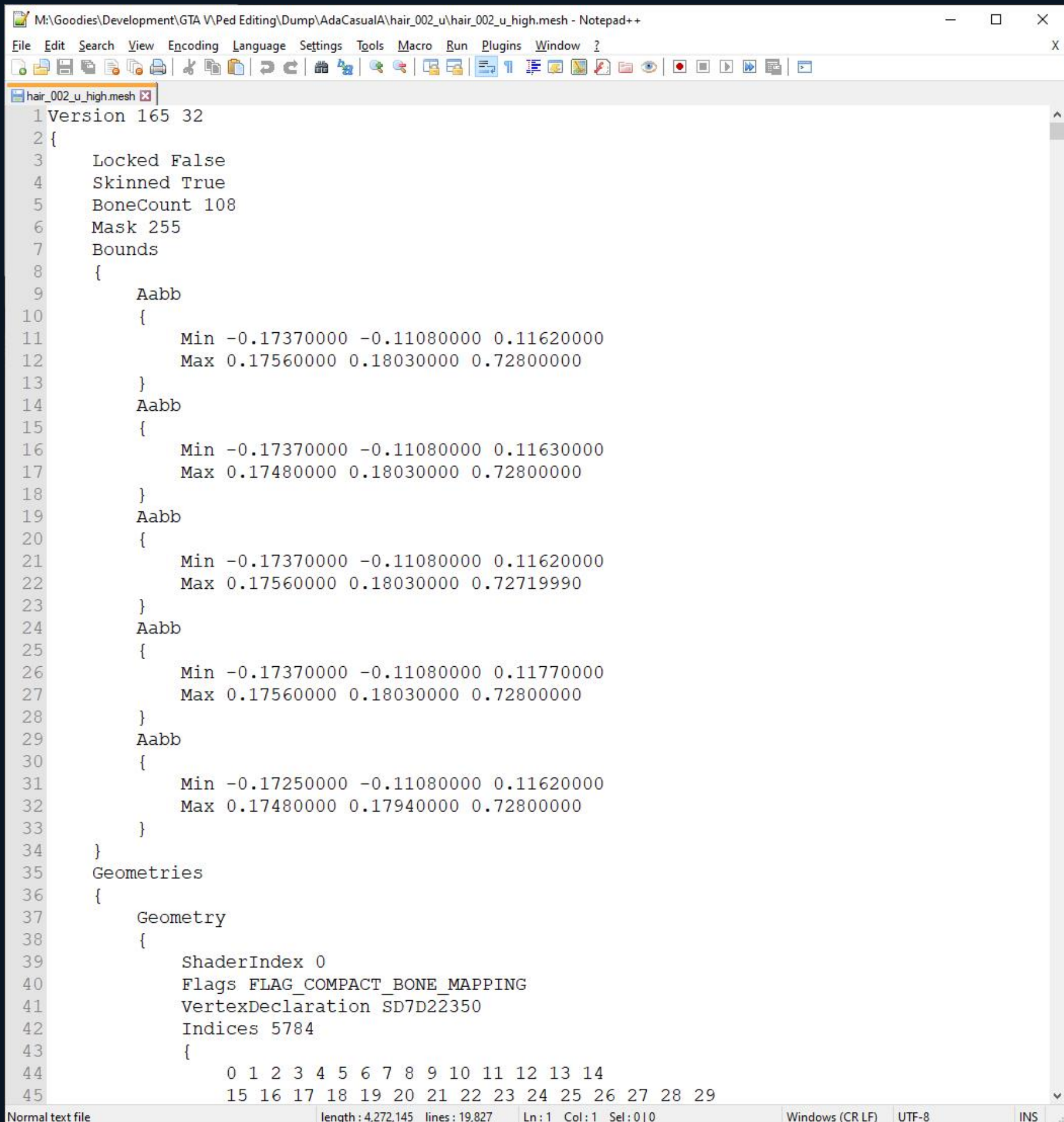


```
1Version 165 32
2{
3  Shaders
4  {
5    ped_hair_cutout_alpha.sps
6    {
7      DiffuseSampler hair_diff_002_a_uni
8      anisoNoiseSpecSampler long_hair_noise
9      BumpSampler hair_normal_002
10     SpecSampler hair_spec_002
11     umGlobalParams 0.00200000 0.00200000 7.00000000 7.00000000
12     AnisotropicAlphaBias 1.00000000
13     envEffFatThickness 25.00000000 25.00000000
14     SpecularIntensityMult 1.00000000
15     SpecularFalloffMult 50.00000000
16     SpecularFresnel 0.75000000
17     Bumpiness 1.00000000
18     SpecularNoiseMapUVScaleFactor 2.00000000 1.00000000 3.00000000 1.00000000
19     AnisotropicSpecularColour 0.10000000 0.10000000 0.10000000 1.00000000
20     AnisotropicSpecularExponent 16.00000000 32.00000000
21     AnisotropicSpecularIntensity 0.10000000 0.15000000
22     DetailSettings 0.10000000 0.75000000 40.00000000
23   }
24 }
25 Skeleton null
26 LodGroup
27 {
28   Center 0.00094999 0.03475001 0.42210000
29   Radius 0.38113290
30   AABBBMin -0.17370000 -0.11080000 0.11620000
31   AABBBMax 0.17560000 0.18030000 0.72800000
32   High 9998.00000000
33   {
34     hair_002_u\hair_002_u_high.mesh 0
35   }
36   Med 9998.00000000
37   Low 9998.00000000
38   Vlow 9998.00000000
39 }
40 Joints null
41 Light null
42 Bound null
43 }
44
```

TUTORIAL

Step 3 – Continued – Mesh Files

The 3D mesh files, up to 3 for low, medium, and high, are contained in a folder. For example the folder called **hair_002_u** would contain these meshes and the high mesh contents are displayed in the screenshot below. It is important to note that we are just renaming the folder and the mesh, we are in no way attempting to edit the actual 3D model in the mesh file even though it is presented as plain text. In our example the mesh file is called **hair_002_u_high.mesh**.



```
1Version 165 32
2{
3    Locked False
4    Skinned True
5    BoneCount 108
6    Mask 255
7    Bounds
8    {
9        Aabb
10       {
11           Min -0.17370000 -0.11080000 0.11620000
12           Max 0.17560000 0.18030000 0.72800000
13       }
14       Aabb
15       {
16           Min -0.17370000 -0.11080000 0.11630000
17           Max 0.17480000 0.18030000 0.72800000
18       }
19       Aabb
20       {
21           Min -0.17370000 -0.11080000 0.11620000
22           Max 0.17560000 0.18030000 0.72719990
23       }
24       Aabb
25       {
26           Min -0.17370000 -0.11080000 0.11770000
27           Max 0.17560000 0.18030000 0.72800000
28       }
29       Aabb
30       {
31           Min -0.17250000 -0.11080000 0.11620000
32           Max 0.17480000 0.17940000 0.72800000
33       }
34     }
35     Geometries
36     {
37         Geometry
38         {
39             ShaderIndex 0
40             Flags FLAG_COMPACT_BONE_MAPPING
41             VertexDeclaration SD7D22350
42             Indices 5784
43             {
44                 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
45                 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
```

TUTORIAL

Step 4 Add Textures

In our final step, we will now import the textures for the hairstyle drawable we just added. The textures will also include the specular (spec) and normal maps. These last steps will be entirely done in OpenIV with the ytd files. Of course you can create your own textures using an image editor like Photoshop or GIMP or even Microsoft Paint.

Once more we will use OpenIV. We will export the 3 textures above from AdaB, rename them, and Import them into Ada A - it is important to rename the textures to match our new hair drawable which is 002 (**numbering starts at 000**).

Final Notes

In our process we ensured we were consistent in renaming all our files correctly. As you gain experience you will notice that you have some flexibility in naming files and folders. This is not really a good thing and you will find some lazy and sloppy developers abuse this. You will find some ped mods with heads named feet, legs named torsos, and numbering systems that don't make any sense. Even Rockstar vanilla peds often don't follow best practices.

A few more points about ped terminology. Components have a `_u` suffix for universal or `_r` for race. You will also see `_m` which might mean mixed, that the ped model may have white and black components for example. Textures have the extension `dds`: the image file type, DirectDraw Surface. And what about ped props?

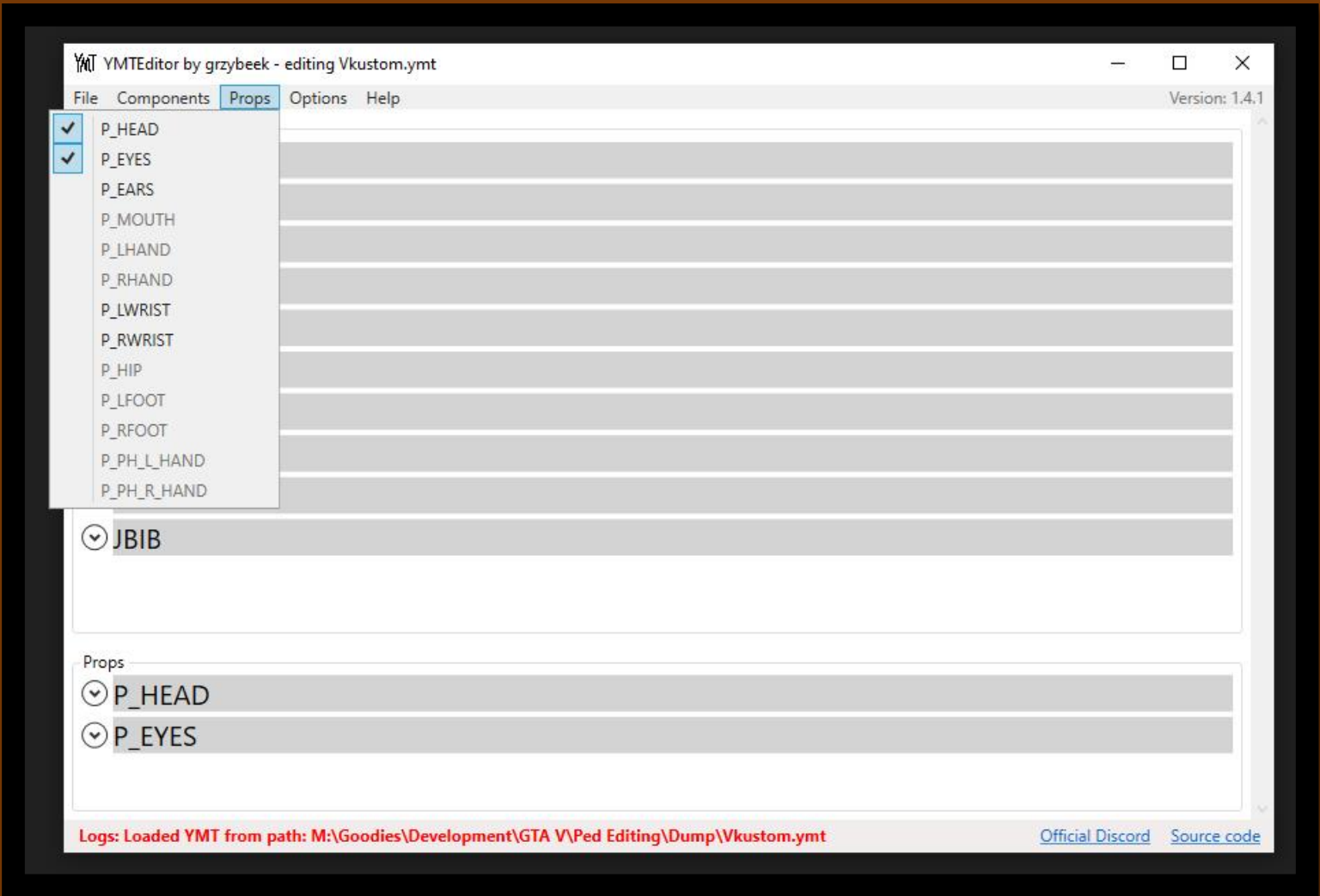
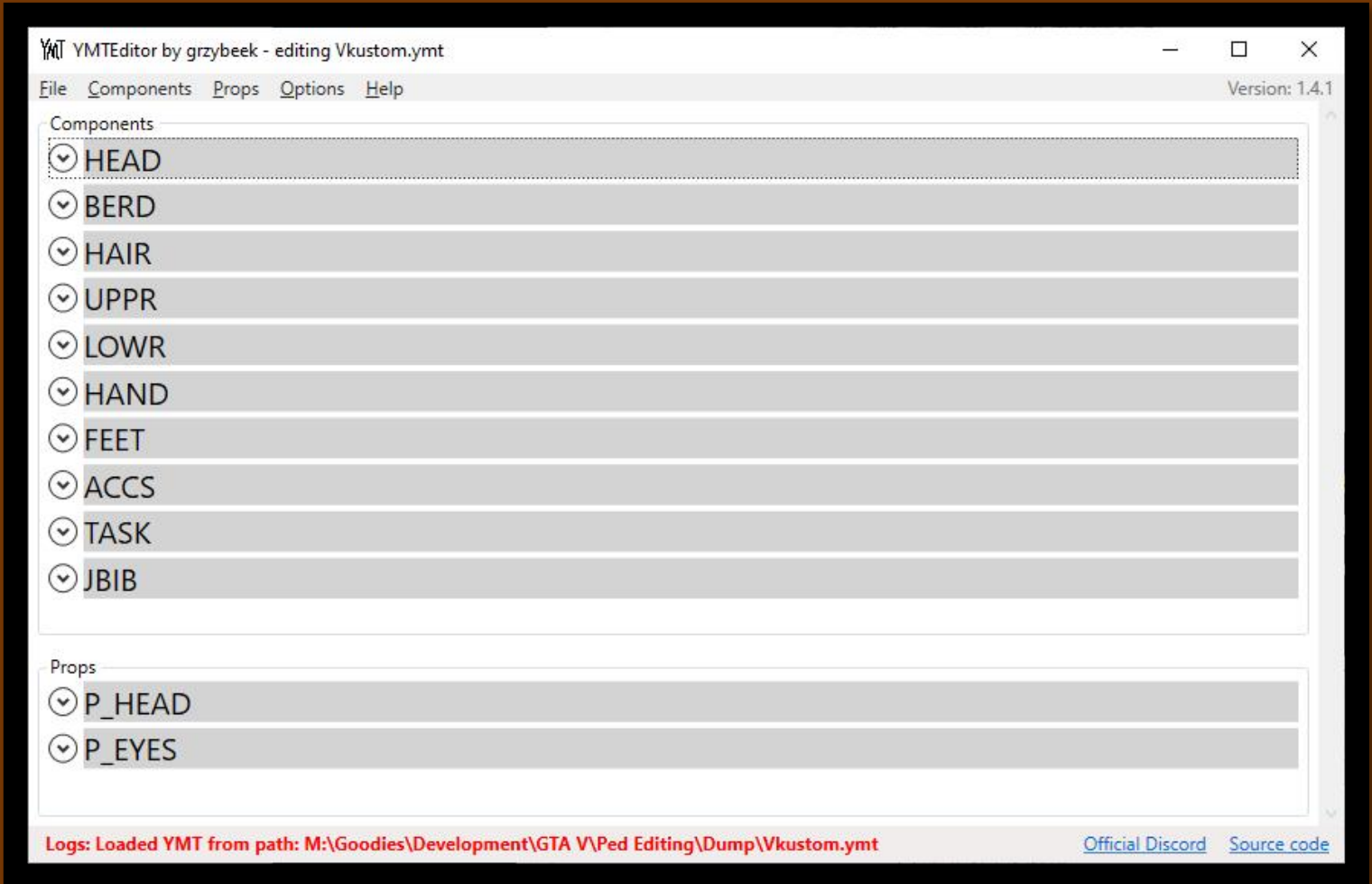
If you take a peek with Menyoo or a similar trainer, you will see 5 ped props are used in GTA5: Hats (helmets), Glasses, Ear Pieces, Watches, and Bangles. Or, respectively, `p_head`, `p_eyes`, `p_ears`, `p_lwrist`, `p_rwrist`. Watches will be worn on the left wrist (`p_lwrist`) while bangles/bracelets will be worn on the right (`p_rwrist`). In the YMTEditor you will see 13 prop items and 10 are listed in SHVDN, for those who write their own scripts, however in all three cases only 5 are in use.

As an example, the ped `g_m_y_korean_01` has its prop textures in **`g_m_y_korean_01.p.ytd`**. Opening that file we would find a texture called **`p_eyes_diff_000.a.dds`**, as an example, for glasses. Notice the **`p` suffix** for the ytd file, and **`p` prefix** for the texture dds file.

YMTEditor – Up Close

Components

In the screenshot below, from the YMTEditor, all the components for a ped will be displayed. In this example 10 out of a possible 12 are included for this particular ped. Only the DECL and TEEF aren't being used. You can also see this ped has 2 props – for the Head and for the Eyes. The screenshot below will show you 5 props, the balance are disabled (greyed out).



YMT Files – Advanced Topics

YMT Files – in Depth

As we saw previously, a relatively new tool called the YMTEditor allows you to easily modify ymt files. It is perfect for beginners and even expert modders. However there are times where you might need to get your hands dirty and modify the ymt file without this tool.

There is an issue that needs to be addressed before you do any editing. YMT files need to be decrypted and exported as xml (text) files before they can be modified or edited. There are two principal tools that will allow you to do this: The Meta Toolkit and CodeWalker's RPF Explorer. Both have their advantages and disadvantages although RPF Explorer is much more user friendly than its older counterpart.

Exporting YMT to XML

As stated above, ymt files can't be edited directly, they need to be exported into the xml format which is a text format. In summary, this is how the two tools work.

Meta Toolkit. Export the ymt file to your desktop or any folder from OpenIV. Then just drag the ymt file on top of the executable MetaTool.exe and it will automatically generate your xml file.

RPF Explorer. Run the explorer and let it scan your game installation. When ready, right click your file and select export to xml.

Terminology

The two tools use different terminology, one is more cryptic than the other but you will quickly familiarize yourself with either tool as you edit your exported xml file.

YMT Components	Meta Toolkit	RPF Explorer
Components	<hash_E2489C4F>	CPVComponentData
Drawables	<hash_68AC8351>	CPVDrawblData
Texture Variations	<hash_4A92222A>	CPVTextureData

YMT Files

YMT Files – Manual Edits

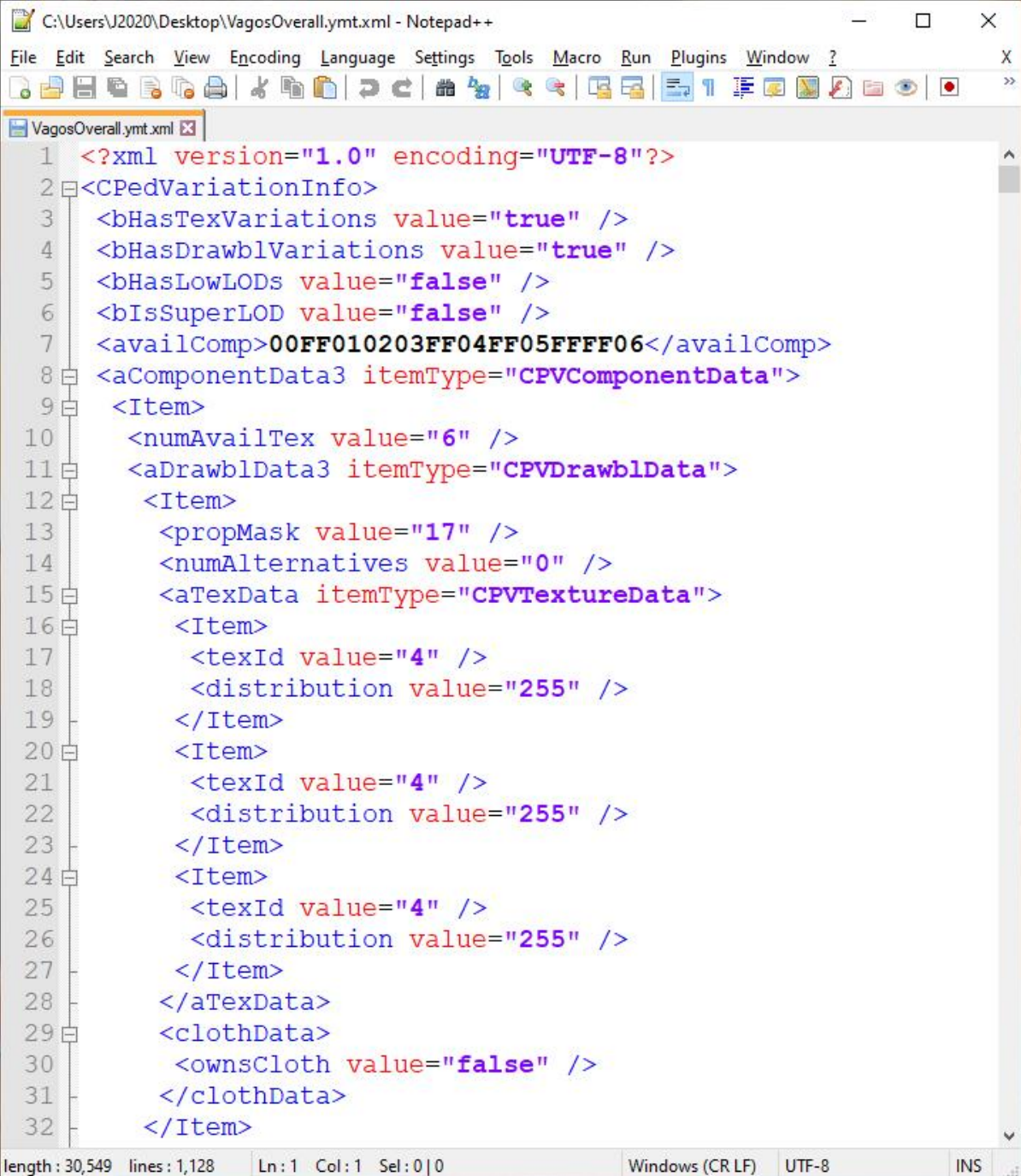
With our two tools, the available component items will also use different terminology.

Meta Toolkit: `<hash_B29BE228>0 255 1 2 3 4 255 5 6 255 255 255</hash_B29BE228>`

RFP Explorer: `<availComp>00FF01020304FF0506FFFFFF</availComp>`

Of course `CPVTextureData` is much easier to understand and relate to than `<hash_4A92222A>`, but you will definitely get used to the jargon in no time at all. When 255 appears it means that item is not being used by the model, RFP Explorer uses the hex FF instead. The single digits in Meta Toolkit refer to the item numbers, 0 being the head and 1 the hair. RFP Explorer uses 00 and 01 for the same items respectively. Notice that in this example hair is numbered 1 rather than 2 because component 1 isn't being used in that particular model.

Editing the ymt file can allow you, depending on the models, to add both drawables (a new jacket for example) and new textures. While drawables are in ydd files, textures will generally be found in the YTD file. Textures can also be embedded in ydd files. The screenshot below displays a section of a ymt file that has been exported to xml with RFP Explorer.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <CPedVariationInfo>
3   <bHasTexVariations value="true" />
4   <bHasDrawblVariations value="true" />
5   <bHasLowLODs value="false" />
6   <bIsSuperLOD value="false" />
7   <availComp>00FF010203FF04FF05FFFF06</availComp>
8   <aComponentData3 itemType="CPVComponentData">
9     <Item>
10      <numAvailTex value="6" />
11      <aDrawblData3 itemType="CPVDrawblData">
12        <Item>
13          <propMask value="17" />
14          <numAlternatives value="0" />
15          <aTexData itemType="CPVTextureData">
16            <Item>
17              <texId value="4" />
18              <distribution value="255" />
19            </Item>
20            <Item>
21              <texId value="4" />
22              <distribution value="255" />
23            </Item>
24            <Item>
25              <texId value="4" />
26              <distribution value="255" />
27            </Item>
28          </aTexData>
29          <clothData>
30            <ownsCloth value="false" />
31          </clothData>
32        </Item>
```

For Developers

Native Database

The Native Database lists 14 component variables rather than 12.

Additions, just for documentation purposes are:

PV_COMP_INVALID = -1 //easy to understand as it will return nothing or an error.

PV_COMP_MAX // not documented, max is always 12 so unsure why it's there.

Ped Component Types in SHVDN3

Just to make things even more complicated, here is how scripthookvdotnet treats the 12 components, these are displayed in alphabetic order:

Native Database	SHVDN
<pre>componentId: enum ePedVarComp { PV_COMP_INVALID = -1, PV_COMP_HEAD, PV_COMP_BERD, PV_COMP_HAIR, PV_COMP_UPPR, PV_COMP_LOWR, PV_COMP_HAND, PV_COMP_FEET, PV_COMP_TEEF, PV_COMP_ACCS, PV_COMP_TASK, PV_COMP_DECL, PV_COMP_JBIB, PV_COMP_MAX };</pre>	<pre>namespace GTA { public enum PedComponentType { Face, Head, Hair, Torso, Legs, Hands, Shoes, Special1, Special2, Special3, Textures, Torso2, } }</pre>

For Developers

Native Function Example Code

```
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 0, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 1, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 2, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 3, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 4, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 5, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 6, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 7, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 8, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 9, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 10, 0, 0, 2);
Function.Call(Hash.SET_PED_COMPONENT_VARIATION, CurrentPed, 11, 0, 0, 2);
}
```

SHVDN Example Code

```
CurrentPed.Style[PedComponentType.Face].SetVariation(0, 0);
CurrentPed.Style[PedComponentType.Head].SetVariation(0, 0);
CurrentPed.Style[PedComponentType.Hair].SetVariation(2, 0);
CurrentPed.Style[PedComponentType.Torso].SetVariation(15, 0);
CurrentPed.Style[PedComponentType.Legs].SetVariation(21, 0);
CurrentPed.Style[PedComponentType.Hands].SetVariation(0, 0);
CurrentPed.Style[PedComponentType.Shoes].SetVariation(1, 0);
CurrentPed.Style[PedComponentType.Special1].SetVariation(0, 0);
CurrentPed.Style[PedComponentType.Special2].SetVariation(0, 0);
CurrentPed.Style[PedComponentType.Special3].SetVariation(0, 0);
CurrentPed.Style[PedComponentType.Textures].SetVariation(0, 0);
CurrentPed.Style[PedComponentType.Torso2].SetVariation(0, 0);
```
