MMO ACS 2

DOCUMENTATION

V1.0

Greetings and welcome in the official documentation of MMO Accounts & Characters System 2!

Thank you for buying this solution, I hope it will be a great booster for your game.

In this documentation I will cover **everything** about this asset, by describing every single element and by guiding you step-by-step on how to use and expand this system’s functionalities.

For every doubts, info or request related to this asset do not hesitate to send me an e-mail at “silvematt@libero.it”, you will be surely answered in less than 24hrs.

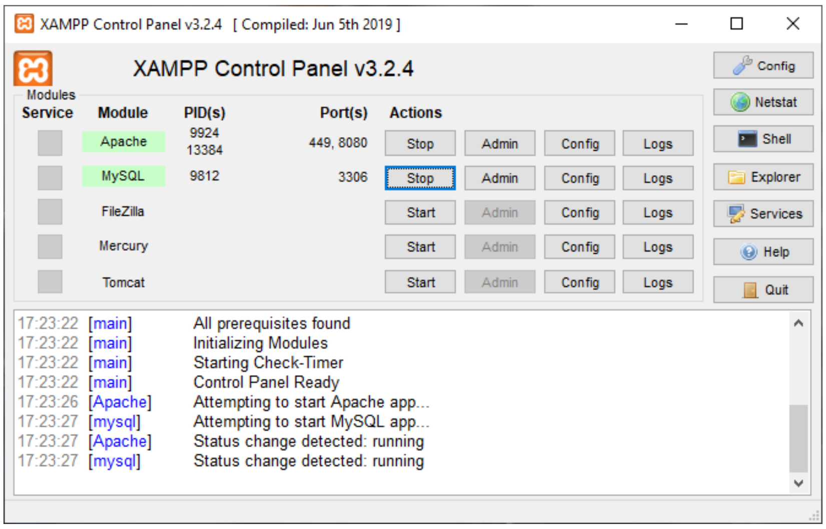
GETTING STARTED:

The first thing that we need to make MMO ACS 2 work is to setup a server and a database. You can use a real server just as I did with the demo, this will let all the people in the world to be able to play your game, or you could setup a local environment for developing and testing your project, which is the recommended solution if you're starting making your game.

In this section we'll use XAMPP, it allows us to have a virtual server in our machine with everything already setup. You just have to download and install it.

Link: https://www.apachefriends.org/index.html

Once you've installed and opened it, you should see something like this:


XAMPP Control Panel v3.2.4 [Compiled: Jun 5th 2019]
XAMPP Control Panel v3.2.4
Config
Netstat
Shell
Explorer
Services
Help
Quit
Modules

Service	Module	PID(s)	Port(s)	Actions			
	Apache	9924 13384	449, 8080	Stop	Admin	Config	Logs
	MySQL	9812	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
17:23:22  [main]    All prerequisites found
17:23:22  [main]    Initializing Modules
17:23:22  [main]    Starting Check-Timer
17:23:22  [main]    Control Panel Ready
17:23:26  [Apache]  Attempting to start Apache app...
17:23:27  [mysql]   Attempting to start MySQL app...
17:23:27  [Apache]  Status change detected: running
17:23:27  [mysql]   Status change detected: running
```

Start both 'Apache' and MySQL. The first thing that we'll do is to move the PHP scripts from the Unity Project to the local server.

Go in the installation directory of XAMPP and reach the folder '**htdocs**', here is where the server is locally hosted.

Create a new folder named 'MMO_ACS2' (or everything you like, a name without spaces is recommended), in the folder you've just created create another one named '**Scripts**'.

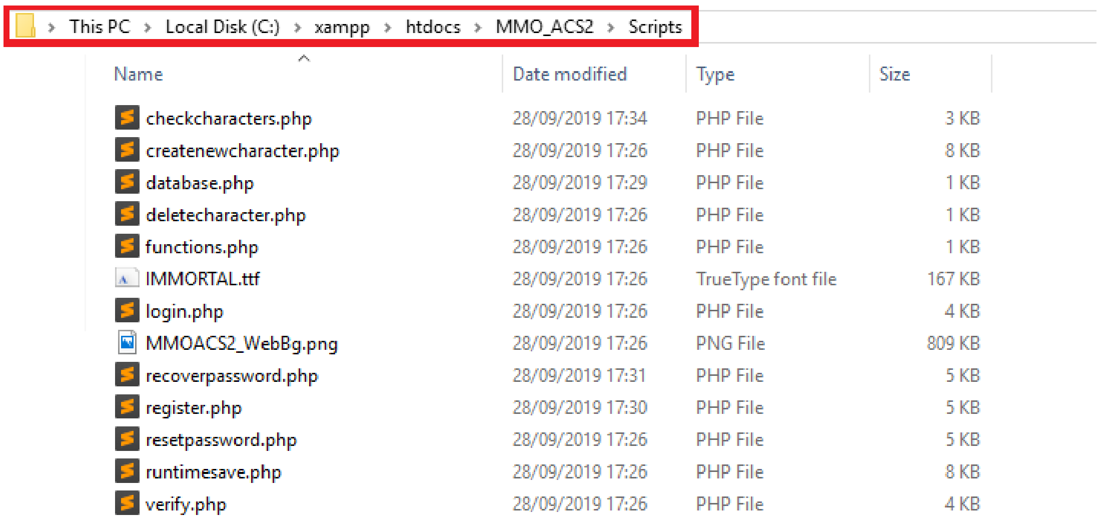
Copy and paste **all** the files from the folder

/YourUnityProject/Assets/MMO ACS 2/PHP/

To

/xampp/htdocs/MMO_ACS2/Scripts/

That's should be the situation:

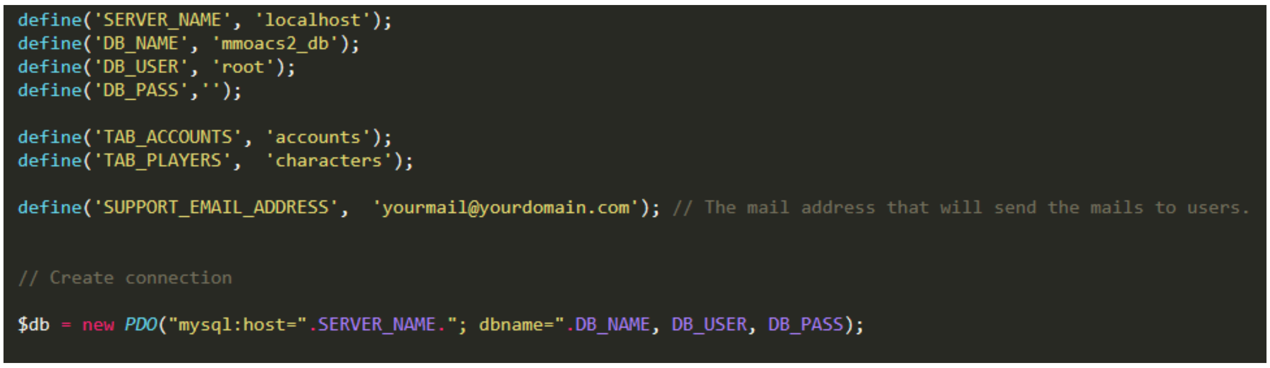


This PC > Local Disk (C:) > xampp > htdocs > MMO_ACS2 > Scripts

Name	Date modified	Type	Size
checkcharacters.php	28/09/2019 17:34	PHP File	3 KB
createnewcharacter.php	28/09/2019 17:26	PHP File	8 KB
database.php	28/09/2019 17:29	PHP File	1 KB
deletecharacter.php	28/09/2019 17:26	PHP File	1 KB
functions.php	28/09/2019 17:26	PHP File	1 KB
IMMORTAL.ttf	28/09/2019 17:26	TrueType font file	167 KB
login.php	28/09/2019 17:26	PHP File	4 KB
MMOACS2_WebBg.png	28/09/2019 17:26	PNG File	809 KB
recoverpassword.php	28/09/2019 17:31	PHP File	5 KB
register.php	28/09/2019 17:30	PHP File	5 KB
resetpassword.php	28/09/2019 17:26	PHP File	5 KB
runtimesave.php	28/09/2019 17:26	PHP File	8 KB
verify.php	28/09/2019 17:26	PHP File	4 KB

Once you did that, you're half ready!

Open up ‘**database.php**’:



```
define('SERVER_NAME', 'localhost');
define('DB_NAME', 'mmoacs2_db');
define('DB_USER', 'root');
define('DB_PASS','');

define('TAB_ACCOUNTS', 'accounts');
define('TAB_PLAYERS',  'characters');

define('SUPPORT_EMAIL_ADDRESS',  'yourmail@yourdomain.com'); // The mail address that will send the mails to users.


// Create connection

$db = new PDO("mysql:host=".SERVER_NAME."; dbname=".DB_NAME, DB_USER, DB_PASS);
```

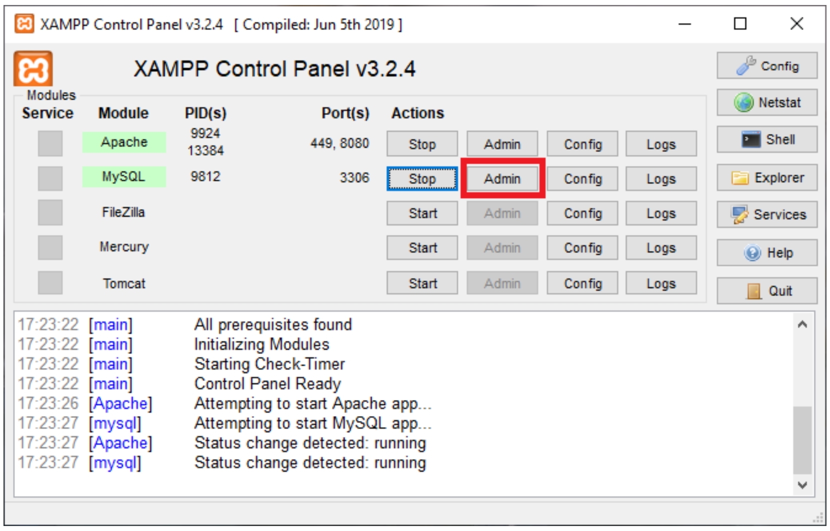
Here you will have to say where the database is and how to access to it. If you’re sticking with XAMPP and this guide just leave everything as it is, otherwise if you’re using a real server insert the host and database information here.

The only thing that you’ve to modify is the SUPPORT_EMAIL_ADDRESS, XAMPP requires a very little and fast configuration to let you send emails from localhost.

You can see a discussion on Stack Overflow regarding this here: https://stackoverflow.com/questions/15965376/how-to-configure-xampp-to-send-mail-from-localhost

The last thing that you have to do is to import the database.

Open the XAMPP panel and click on the ‘Admin’ button in line with MySQL:


XAMPP Control Panel v3.2.4 [Compiled: Jun 5th 2019]

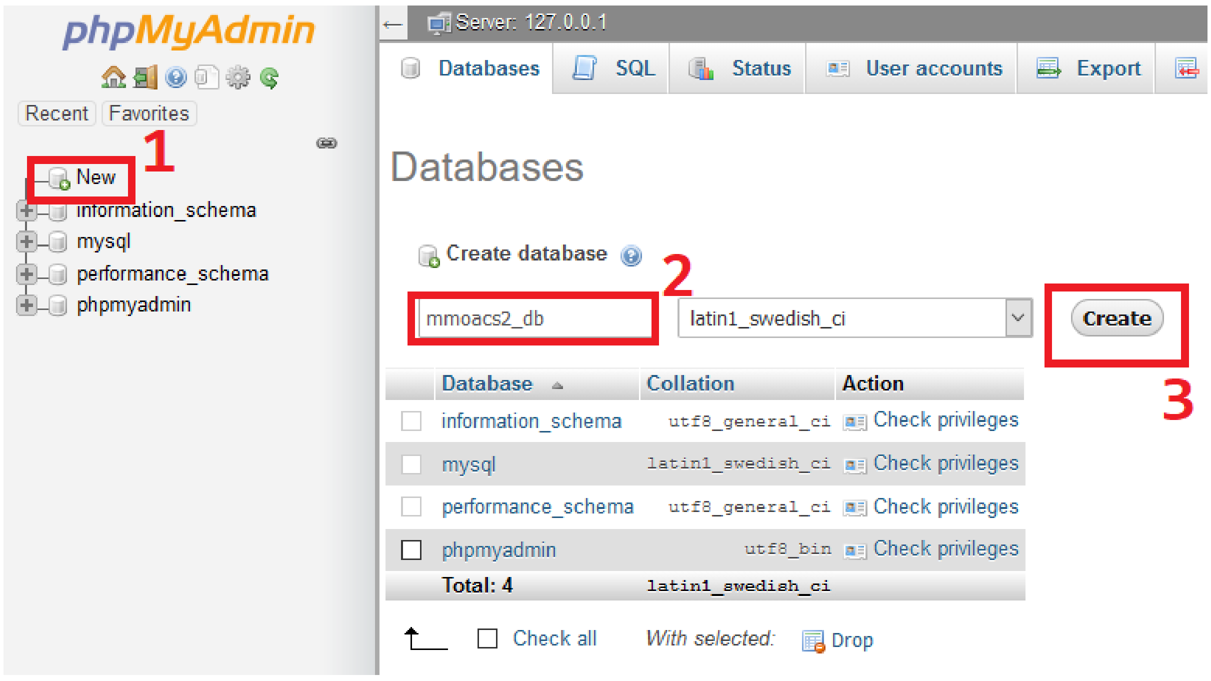
XAMPP Control Panel v3.2.4

Service	Module	PID(s)	Port(s)	Actions			
	Apache	9924 13384	449, 8080	Stop	Admin	Config	Logs
	MySQL	9812	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
17:23:22  [main]     All prerequisites found
17:23:22  [main]     Initializing Modules
17:23:22  [main]     Starting Check-Timer
17:23:22  [main]     Control Panel Ready
17:23:26  [Apache]   Attempting to start Apache app...
17:23:27  [mysql]    Attempting to start MySQL app...
17:23:27  [Apache]   Status change detected: running
17:23:27  [mysql]    Status change detected: running
```

This will open up PhpMyAdmin, a software that allows you to manage your databases:

From there create a new database **with the same name as the field ‘DB_NAME’ in the script ‘database.php’.**



Databases

Create database

mmoacs2_db | latin1_swedish_ci | Create

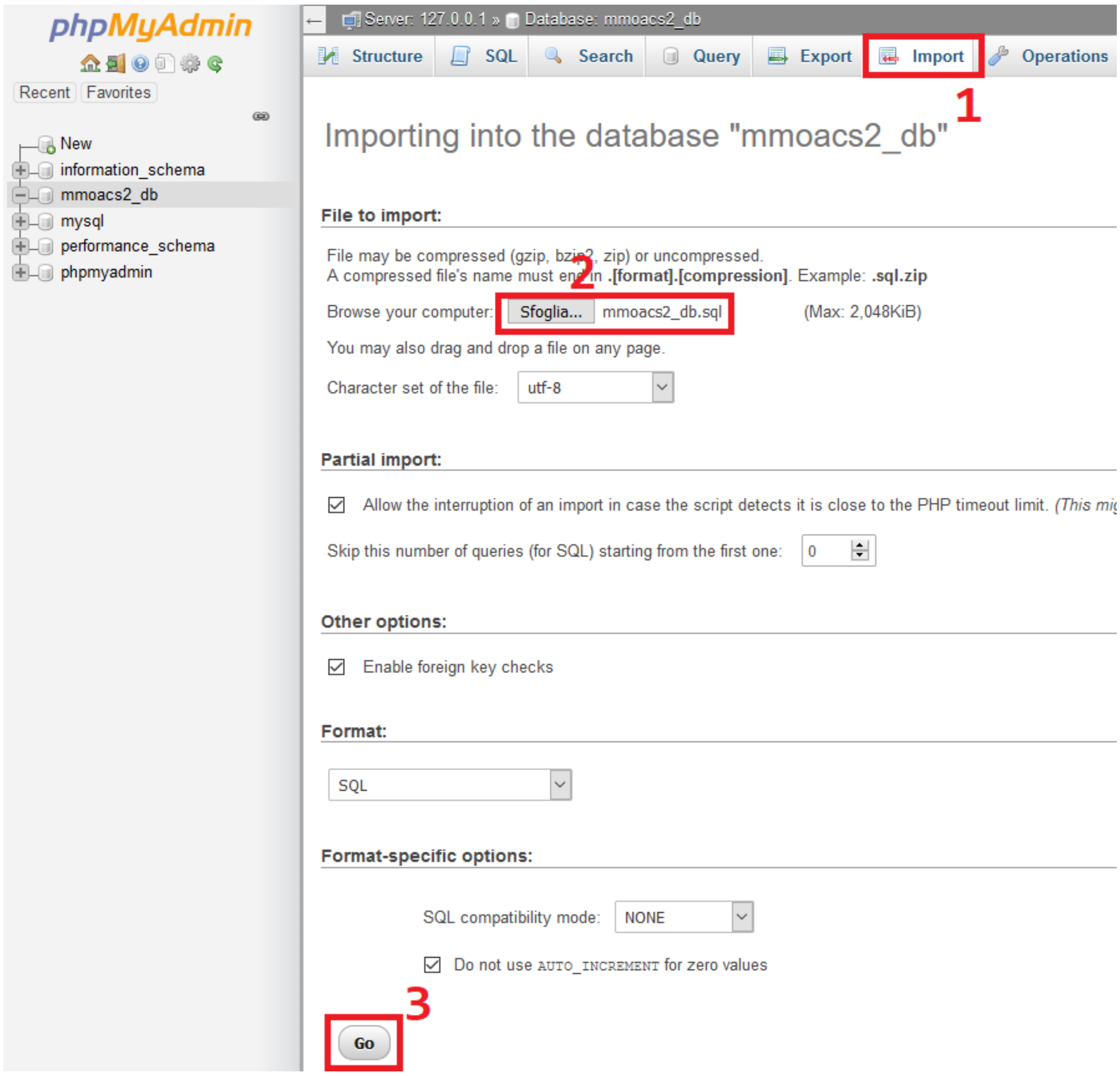
	Database	Collation	Action
☐	information_schema	utf8_general_ci	Check privileges
☐	mysql	latin1_swedish_ci	Check privileges
☐	performance_schema	utf8_general_ci	Check privileges
☐	phpmyadmin	utf8_bin	Check privileges
	Total: 4	latin1_swedish_ci	

Now all you have to do is to import the .SQL file present in your Unity Project:

/YourUnityProject/MMO ACS 2/Database/mmoacs2_db.sql

On PhpMyAdmin:

To do that:



phpMyAdmin

Recent Favorites

New
information_schema
mmoacs2_db
mysql
performance_schema
phpmyadmin

Server: 127.0.0.1 » Database: mmoacs2_db

Structure SQL Search Query Export Import Operations

1

Importing into the database "mmoacs2_db"

File to import:

File may be compressed (gzip, bzip2, zip) or uncompressed.
A compressed file's name must end in **.[format].[compression]**. Example: **.sql.zip**

2

Browse your computer: Sfoglia... mmoacs2_db.sql (Max: 2,048KiB)

You may also drag and drop a file on any page.

Character set of the file: utf-8

Partial import:

☑ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. *(This mi*

Skip this number of queries (for SQL) starting from the first one: 0

Other options:

☑ Enable foreign key checks

Format:

SQL

Format-specific options:

SQL compatibility mode: NONE

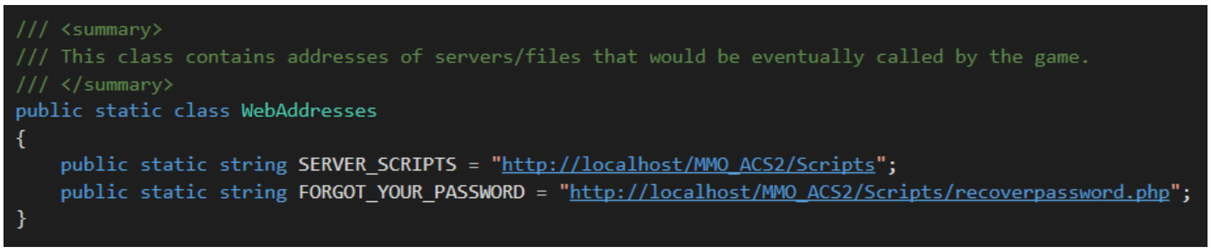
☑ Do not use `AUTO_INCREMENT` for zero values

3

Go

Now the database is imported, and you're ready to start using MMO-ACS 2!

The last thing you may want to setup is a class inside the project: "**WebAddresses.cs**":



```
/// <summary>
/// This class contains addresses of servers/files that would be eventually called by the game.
/// </summary>
public static class WebAddresses
{
    public static string SERVER_SCRIPTS = "http://localhost/MMO_ACS2/Scripts";
    public static string FORGOT_YOUR_PASSWORD = "http://localhost/MMO_ACS2/Scripts/recoverpassword.php";
}
```

Adjust those addresses and you're ready to start with MMO ACS 2!

LOGIN & REGISTRATION:

Client side the Login and Registration process is handled by ‘**LoginRegistrationManager.cs’**, attached to the GameObject ‘Login/Registration Manager’ in the Login scene.

This script is very simple, it holds some references to the GameObjects who contains all the UI for the Login and Register section, and of course every InputField.

The Login and Registration process is nothing more than a UnityWebRequest sent with a WWWForm.

If you don’t know those classes check the official documentation:

***UnityWebRequest**: https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html*

***WWWForm**: https://docs.unity3d.com/ScriptReference/WWWForm.html*

The request for the login is asked on the server to the script ‘**login.php**’. While for the registration we call ‘**register.php**’. You will find all the .php code commented.

Mainly what the ‘**login.php**’ script does is to check:

- If the received form is correct.
- If the values in the forms are safe for being processed.
- If the inserted account exists.
- If the inserted password is correct (after being encrypted).
- If the account is banned.
- If the account is active.

If all those conditions matches, the user will be able to login. Otherwise he will be prompted with the response for his case.

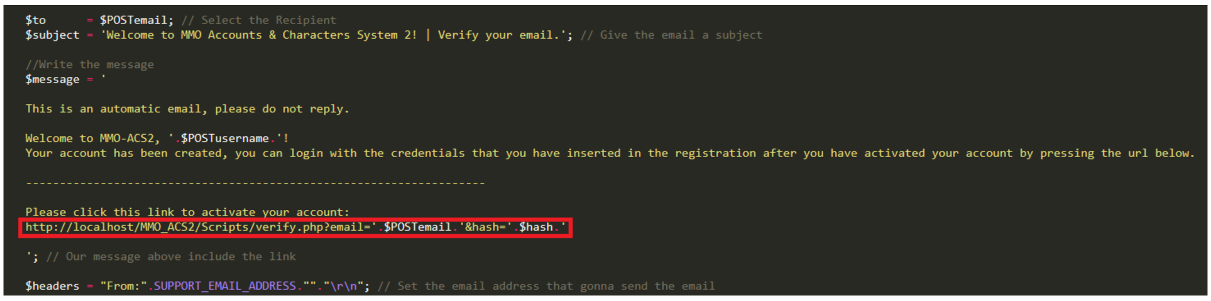
The ‘**register.php’** works almost the same:

- If the received form is correct.
- If the values in the forms are safe for being processed.
- If the inserted account name is not used yet.
- If the inserted email is not used yet.

If all those conditions match the last thing the script will do before terminating its execution and returning a message to the client is to send an email to activate the account on the address the user inserted while registering.

Otherwise he will be prompted with the response for his case.

So the register.php contains the activation email, and there’s where you should modify that thing:



```
$to      = $POSTemail; // Select the Recipient
$subject = 'Welcome to MMO Accounts & Characters System 2! | Verify your email.'; // Give the email a subject

//Write the message
$message = '

This is an automatic email, please do not reply.

Welcome to MMO-ACS2, '.$POSTusername.'!
Your account has been created, you can login with the credentials that you have inserted in the registration after you have activated your account by pressing the url below.

------------------------------------------------------------------

Please click this link to activate your account:
http://localhost/MMO_ACS2/Scripts/verify.php?email='.$POSTemail.'&hash='.$hash.'

'; // Our message above include the link

$headers = "From:".SUPPORT_EMAIL_ADDRESS.""."\r\n"; // Set the email address that gonna send the email
```

The highlighted line shows where the user will be redirected, and it’s to the script ‘**verify.php**’.

This script inside an HTML page will **get** the values from the URL (email='.$POSTemail.'&hash='.$hash.') and validate them.

If everything is correct the Account will be activated and the user will be able to login with his account.

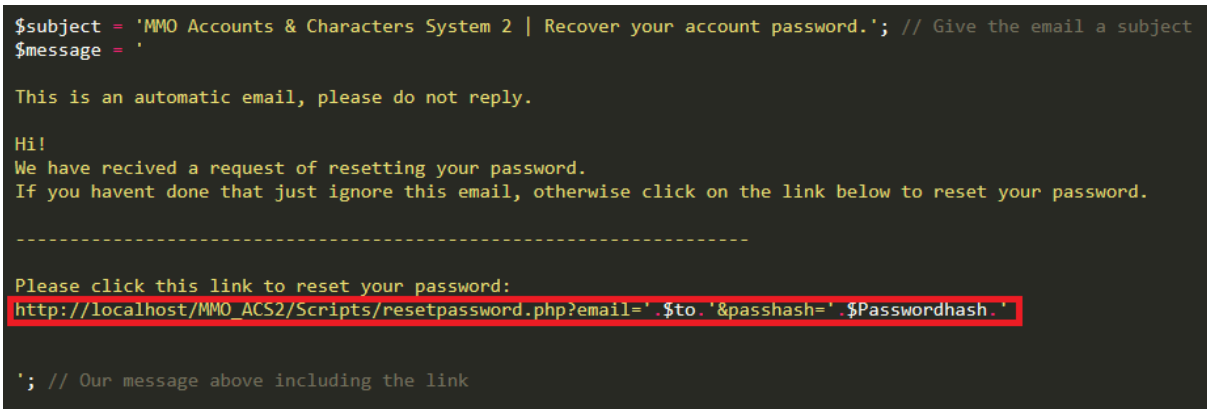
Otherwise if those values are not correct, the user has probably tried to manipulate the URL, in that case the script will simply stop the execution and prompt the user with a message.

The last thing you can do from the Login scene is to open the 'Forgot Your Password' link that must point to the script '**recoverpassword.php**.

That's another HTML page with some PHP in it.

From there as you may already saw in the demo, after inserting the email in the InputField, if the email is associated to an account in your database the script will send a mail to the account.

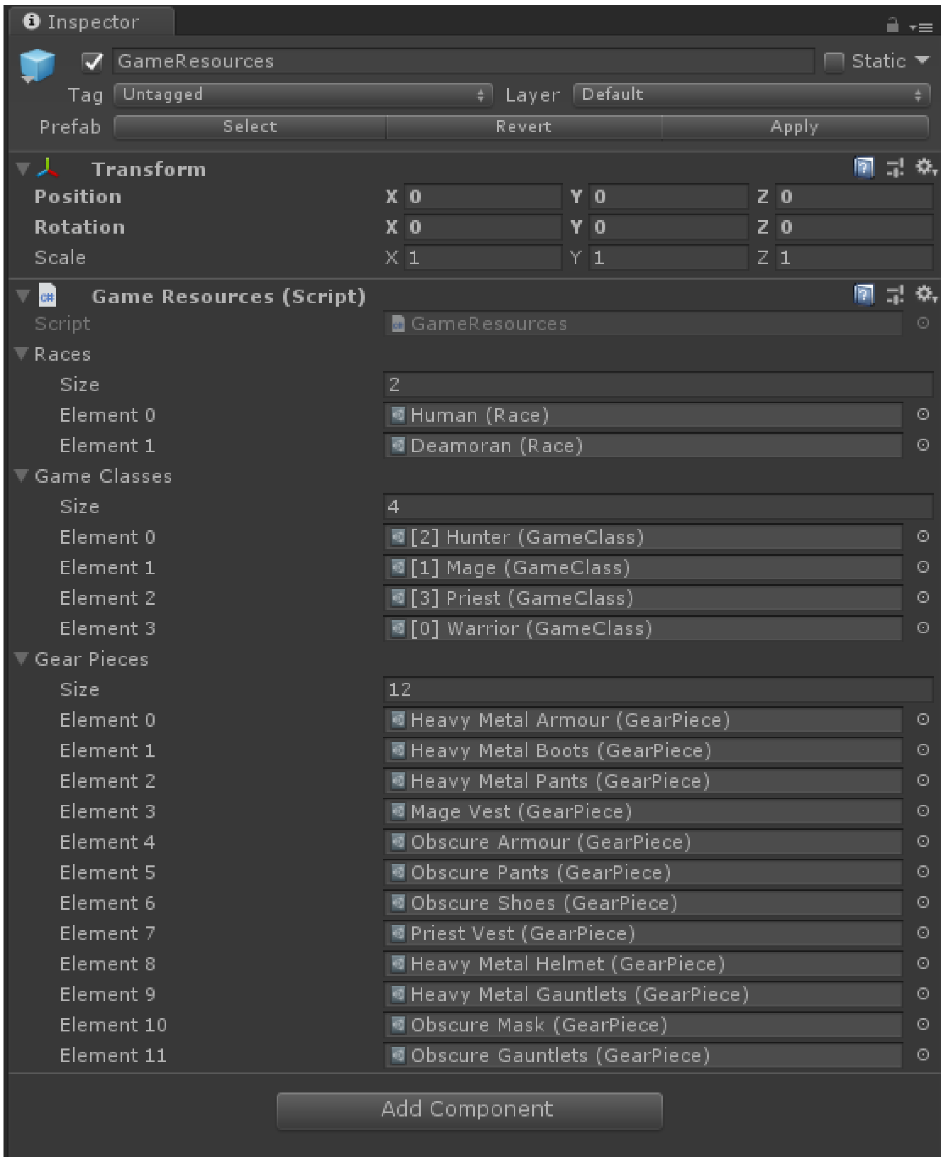
So the last link that you have to modify is in this script:



```
$subject = 'MMO Accounts & Characters System 2 | Recover your account password.'; // Give the email a subject
$message = '

This is an automatic email, please do not reply.

Hi!
We have recived a request of resetting your password.
If you havent done that just ignore this email, otherwise click on the link below to reset your password.

--------------------------------------------------------------------

Please click this link to reset your password:
http://localhost/MMO_ACS2/Scripts/resetpassword.php?email='.$to.'&passhash='.$Passwordhash.'


'; // Our message above including the link
```

The **resetpassword.php** is another HTML page that will let the user reset his password.

In the Login scene you may have noticed the “GameResources” GameObject, with its script attached.



GameResources
Tag: Untagged   Layer: Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Game Resources (Script)

Script	GameResources
Races	
Size	2
Element 0	Human (Race)
Element 1	Deamoran (Race)
Game Classes	
Size	4
Element 0	[2] Hunter (GameClass)
Element 1	[1] Mage (GameClass)
Element 2	[3] Priest (GameClass)
Element 3	[0] Warrior (GameClass)
Gear Pieces	
Size	12
Element 0	Heavy Metal Armour (GearPiece)
Element 1	Heavy Metal Boots (GearPiece)
Element 2	Heavy Metal Pants (GearPiece)
Element 3	Mage Vest (GearPiece)
Element 4	Obscure Armour (GearPiece)
Element 5	Obscure Pants (GearPiece)
Element 6	Obscure Shoes (GearPiece)
Element 7	Priest Vest (GearPiece)
Element 8	Heavy Metal Helmet (GearPiece)
Element 9	Heavy Metal Gauntlets (GearPiece)
Element 10	Obscure Mask (GearPiece)
Element 11	Obscure Gauntlets (GearPiece)

This script holds a collection of Assets used in the game.

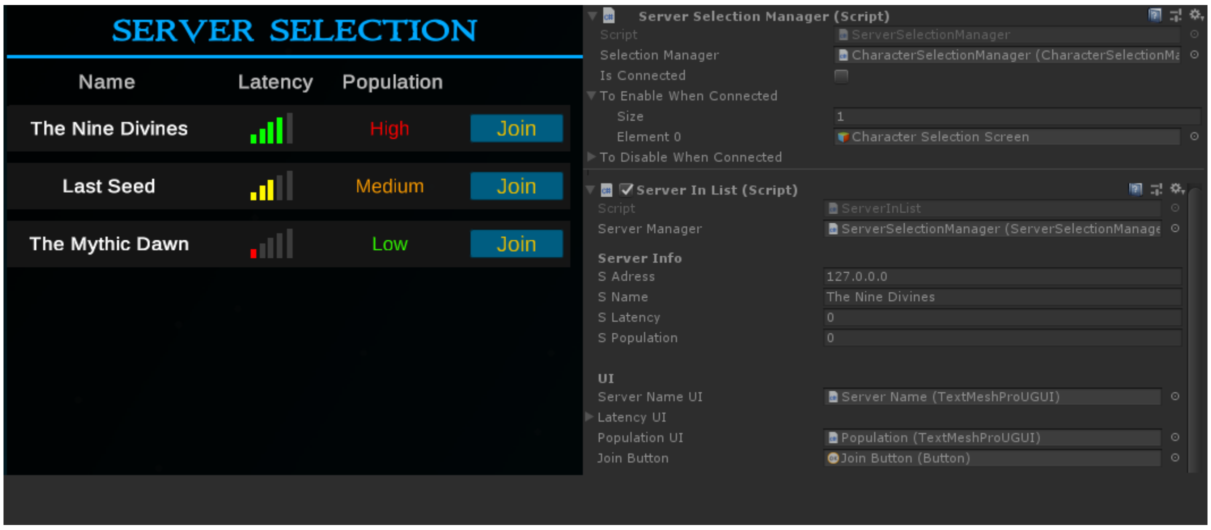
You can see this class like a replacement of the ‘Resources’ folder and its slow loading times. From this class you can load at runtime a Race, a GameClass or a GearPiece just by passing the ID. This class comes extremely useful when we have to convert the IDs from the database to actual game data (Race, GameClass, equipped Gear).

Still on a production I would recommend to use Assets Bundles instead.

SERVER SELECTION:

The first thing after a successfully login that he user will see is a list of servers. In the default case those servers point to nowhere.

The server selection is indeed just a template that you may or not may want to have in your game.



Name	Latency	Population	
The Nine Divines		High	Join
Last Seed		Medium	Join
The Mythic Dawn		Low	Join

```
Server Selection Manager (Script)
Script                       ServerSelectionManager
Selection Manager            CharacterSelectionManager (CharacterSelectionMa
Is Connected                 ☐
To Enable When Connected
  Size                       1
  Element 0                  Character Selection Screen
To Disable When Connected

☑ Server In List (Script)
Script                       ServerInList
Server Manager               ServerSelectionManager (ServerSelectionManage

Server Info
S Adress                     127.0.0.0
S Name                       The Nine Divines
S Latency                    0
S Population                 0

UI
Server Name UI               Server Name (TextMeshProUGUI)
Latency UI
Population UI                Population (TextMeshProUGUI)
Join Button                  Join Button (Button)
```

The Server Selection system is composed of 2 scripts:

- **ServerSelectionManager**: Manages the servers and will manage the connection to your servers.
- **ServerInList**: Represent a server and its information, like Latency and Population.

Whenever we press the 'Join' button on a server we'll call :

```
public void ConnectToServer(ServerInList server)
```

In this method you'll have to insert the code to connect to your servers.

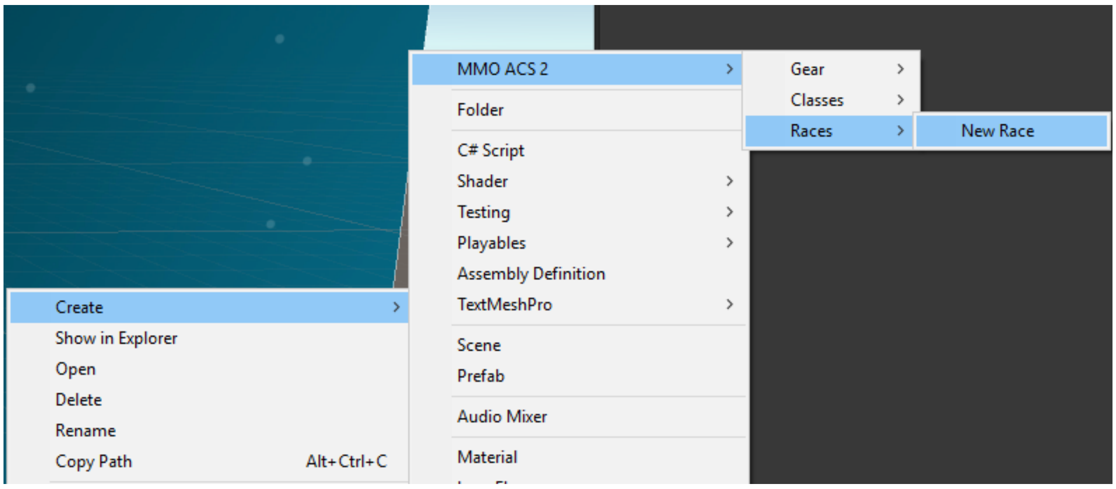
Nothing more. In the Demo the Game Server doesn't exist since we'll join in a local scene.

RACES:

Before getting into the actual Character Selection, Creation and GameScene, let's see how we are able to create the data of the game.

To create a new Race:

Right Click on the Project Window -> Create -> MMO ACS 2 -> Races -> New Race:



For creating a new race what you need to have is:

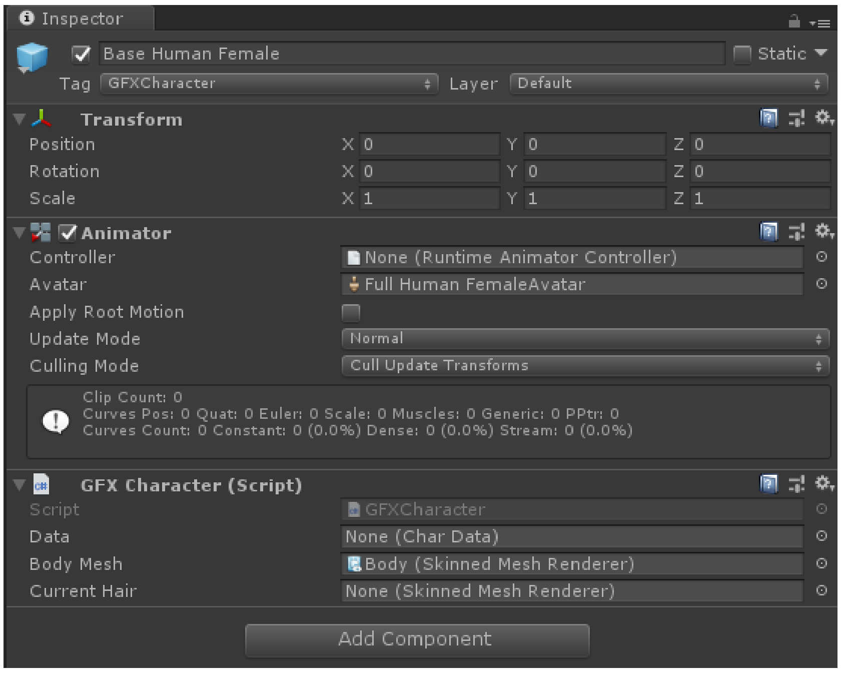
- A prefab of the Male Model of your Race.
- A prefab of the Female Model of your Race.

Those two prefabs are called 'Base *Race* Model'.

Those prefabs represent the 'naked' model, without any type of gear, hair or any other detail.

Every ‘Base *Race* Model’ must have those components attached:

- An Animator with the correct Avatar.
- GFXCharacter script with the ‘BodyMesh’ set.

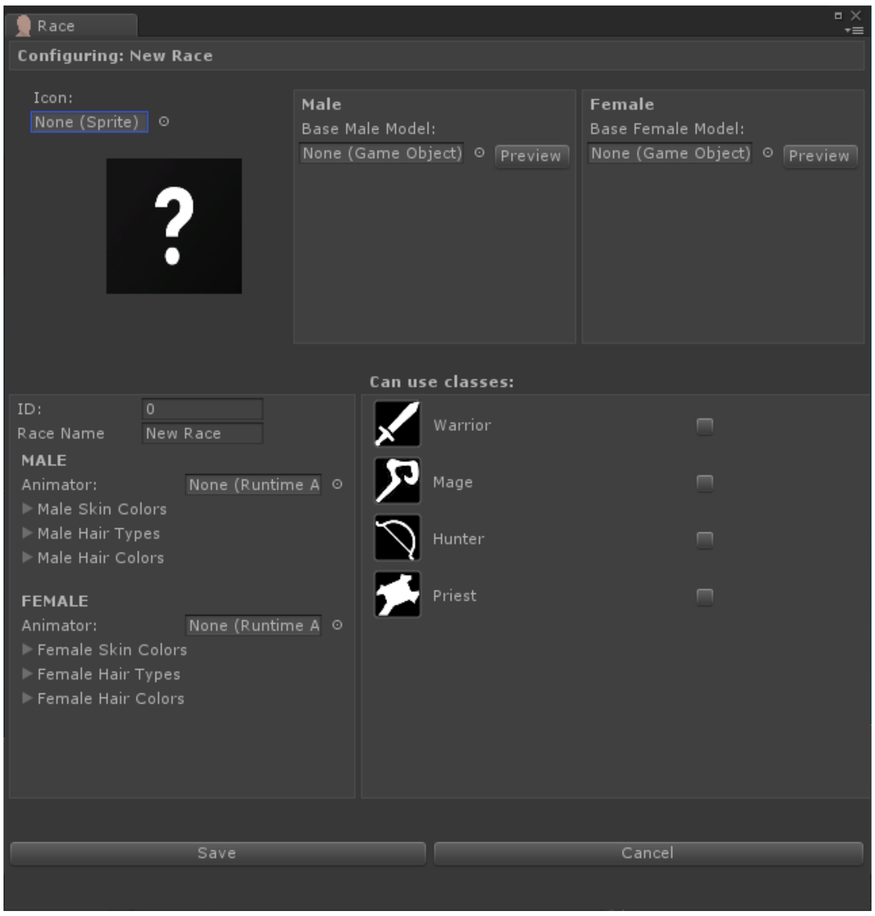


```
Inspector
[✓] Base Human Female                                  [ ] Static
Tag  GFXCharacter                   Layer  Default

Transform
Position              X 0        Y 0        Z 0
Rotation              X 0        Y 0        Z 0
Scale                 X 1        Y 1        Z 1

[✓] Animator
Controller            None (Runtime Animator Controller)
Avatar                Full Human FemaleAvatar
Apply Root Motion     [ ]
Update Mode           Normal
Culling Mode          Cull Update Transforms
    Clip Count: 0
    Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
    Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 0 (0.0%)

GFX Character (Script)
Script                GFXCharacter
Data                  None (Char Data)
Body Mesh             Body (Skinned Mesh Renderer)
Current Hair          None (Skinned Mesh Renderer)

Add Component
```

BodyMesh, in GFXCharacter is the SkinnedMeshRenderer of the model. This must be set manually while the other fields must remain empty since they’ll be set at runtime.

To work with the Equipment System, your Models should have the same BlendShapes of the ones in the project. But don’t worry. If you’re not planning to add those specifically BlendShapes, if you want to remove some or not use them at all you are able to modify that and I’ll explain how in this documentation.

Let’s take a look at the RaceCreationWindow:



Configuring: New Race

Icon: None (Sprite)

Male
Base Male Model: None (Game Object)

Female
Base Female Model: None (Game Object)

ID: 0
Race Name: New Race

MALE
Animator: None (Runtime A

FEMALE
Animator: None (Runtime A

Can use classes:
Warrior
Mage
Hunter
Priest

We have to setup some data here, like the Icon, the Base Models we’ve talked before, modifications for the Race and on the right what class can be used.

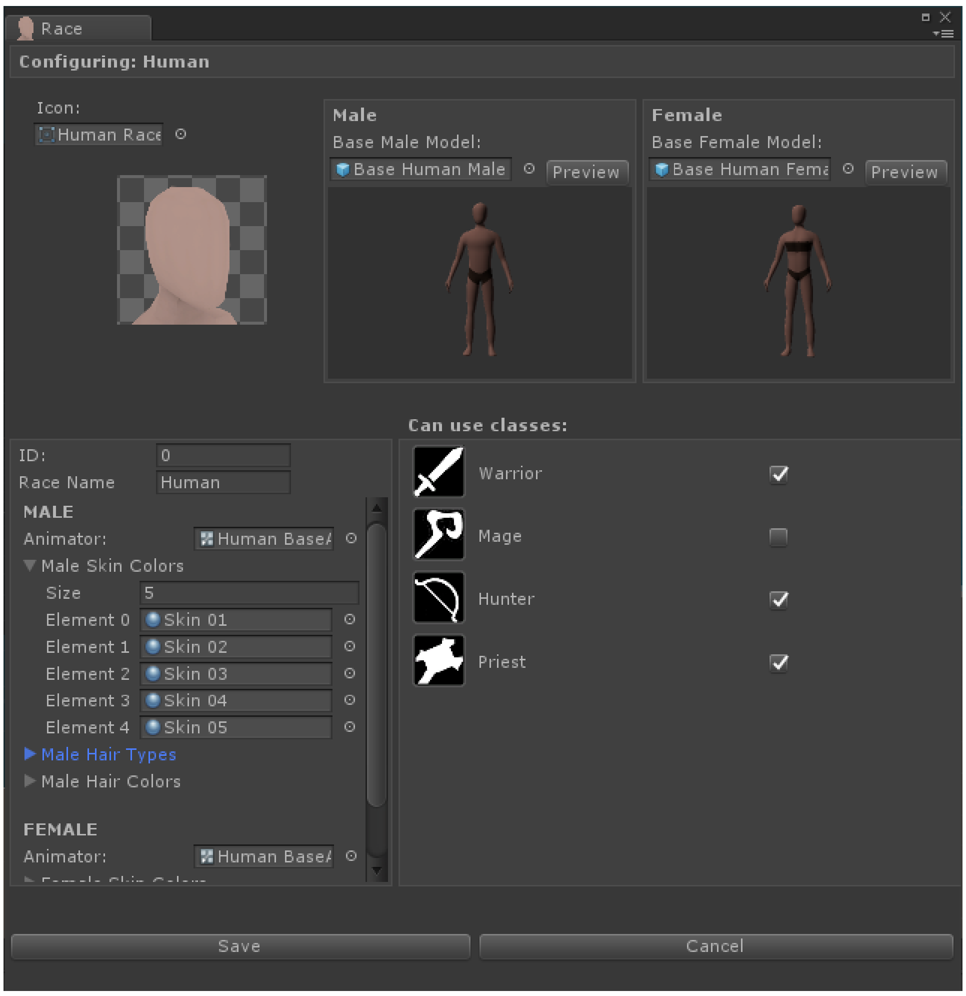
- **ID:** Identifies univocally a Race. **It’s extremely important that this value doesn’t change once it’s saved on the database**. When a character is saved on the database, his race will be identified by the ID. So if we create a ‘Human’ (ID:0) in the Game and then modify the ID of the ‘Human’ ScriptableObject with the value of 1, you’ll get an error. It’s also important if you’re not in the developing process to not modify the order of Races. If we swap the IDs of the two races in the Demo, who were a Human is now a Deamoran and who were a Deamoran is now a human. Also don’t assign the same ID to different races, you’re anyway not allowed to do that since the Window will prompt you an error.

- **Race Name:** The Name of the Race in the game, what will be displayed on the character selection and in many other situations.
- **Section 'Male' and 'Female':** Here you can assign the details of your Race, how he moves (Animator) and how he looks like. To add a new Skin Colour, a new HairType or HairColor you only have to add a new element to those arrays.

**** Remember that for the array 'MaleHairTypes' and 'FemaleHairTypes' the Element 0 must be null since the value 0 of the HairType says that the character is bald.***

On the section **'Can Use Classes'** you decide what classes will be able this Race will be able to play. This section is automatically built. That means if you add a New GameClass to the project it will appear in this section.

Here is a completed Race:



Configuring: Human

Icon: Human Race

Male
Base Male Model: Base Human Male

Female
Base Female Model: Base Human Fema

ID: 0
Race Name: Human
MALE
Animator: Human BaseA
Male Skin Colors
Size: 5
Element 0: Skin 01
Element 1: Skin 02
Element 2: Skin 03
Element 3: Skin 04
Element 4: Skin 05
Male Hair Types
Male Hair Colors
FEMALE
Animator: Human BaseA

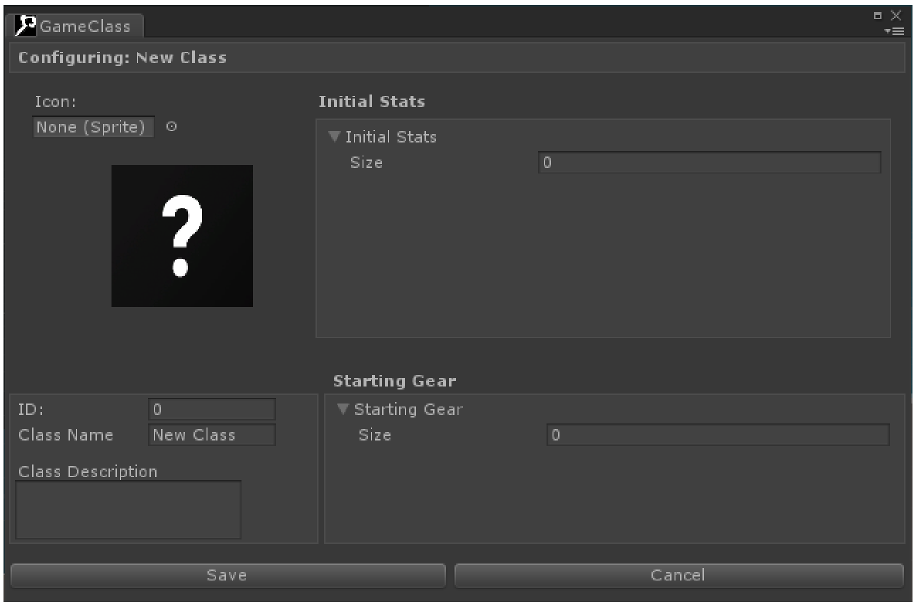
Can use classes:

Warrior	✓
Mage	
Hunter	✓
Priest	✓

GAME CLASSES:

To create a new GameClass:

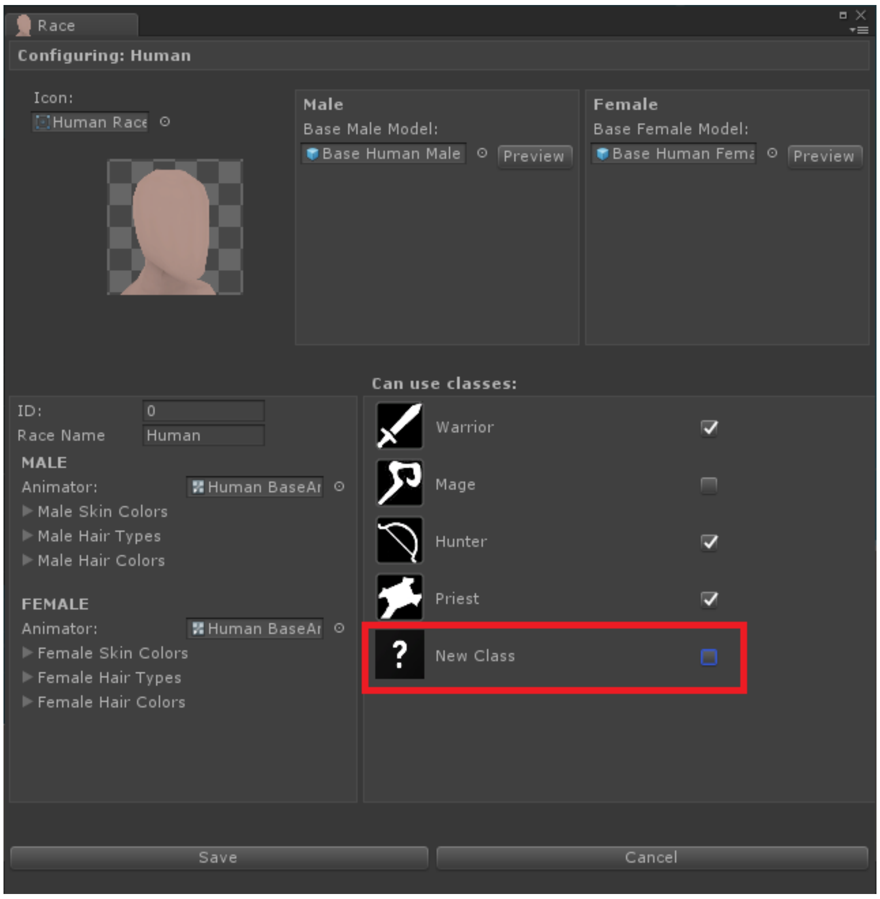
Right Click on the Project Window -> Create -> MMO ACS 2 -> GameClasses -> New GameClass:



For the ID it’s the same thing of the Races.

- **Initial Stats**: Defines the initial stats of this class (upon creation).
- **Starting Gear**: Defines the starting gear of this class (upon creation).
- **Class Name**: The Name of the GameClass in the game, what will be displayed on the character selection and in many other situations.
- **Class Description:** The description of the GameClass that will be displayed on the Character Creation.

When a new class is created if you modify an existing or add a new race, you will be able to choose if that race can use the new class:



Configuring: Human

Icon: Human Race

Male
Base Male Model: Base Human Male

Female
Base Female Model: Base Human Fema

ID: 0
Race Name: Human

MALE
Animator: Human BaseAr
▶ Male Skin Colors
▶ Male Hair Types
▶ Male Hair Colors

FEMALE
Animator: Human BaseAr
▶ Female Skin Colors
▶ Female Hair Types
▶ Female Hair Colors

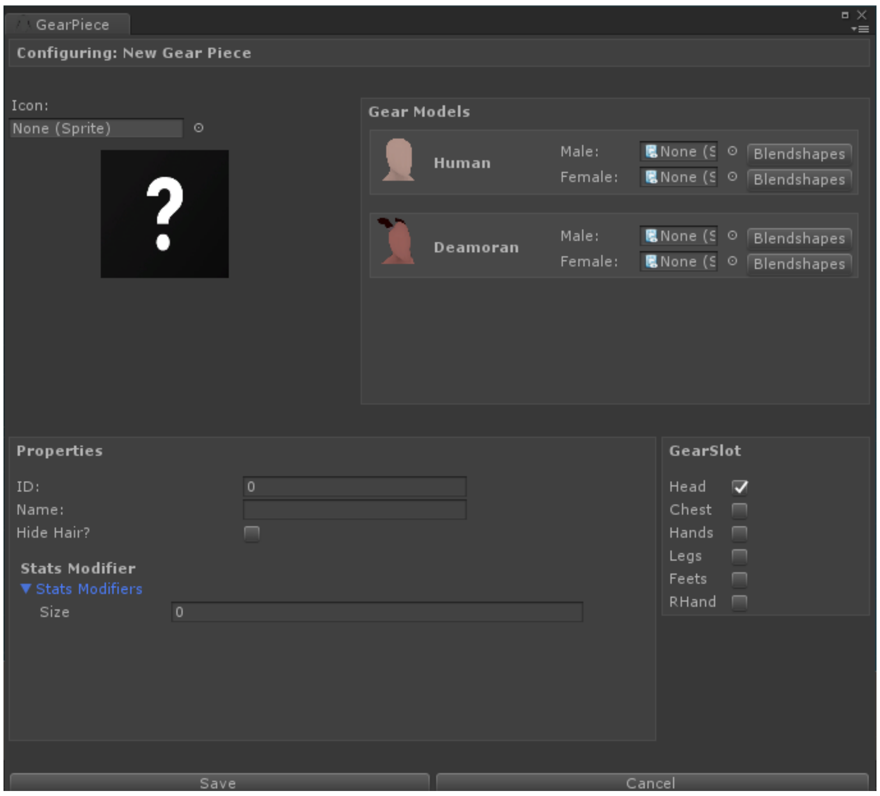
Can use classes:

Warrior	✓
Mage	☐
Hunter	✓
Priest	✓
New Class	☐

GEAR:

To create a new GearPiece:

Right Click on the Project Window -> Create -> MMO ACS 2 -> Gear -> New Gear Piece:



GearPiece

Configuring: New Gear Piece

Icon: None (Sprite)

Gear Models

Human	Male:	None (S	Blendshapes
	Female:	None (S	Blendshapes
Deamoran	Male:	None (S	Blendshapes
	Female:	None (S	Blendshapes

Properties

ID: 0

Name:

Hide Hair? ☐

Stats Modifier

▼ Stats Modifiers

Size 0

GearSlot

Head	✓
Chest	☐
Hands	☐
Legs	☐
Feets	☐
RHand	☐

For the ID it’s the same thing of the Races and GameClasses.

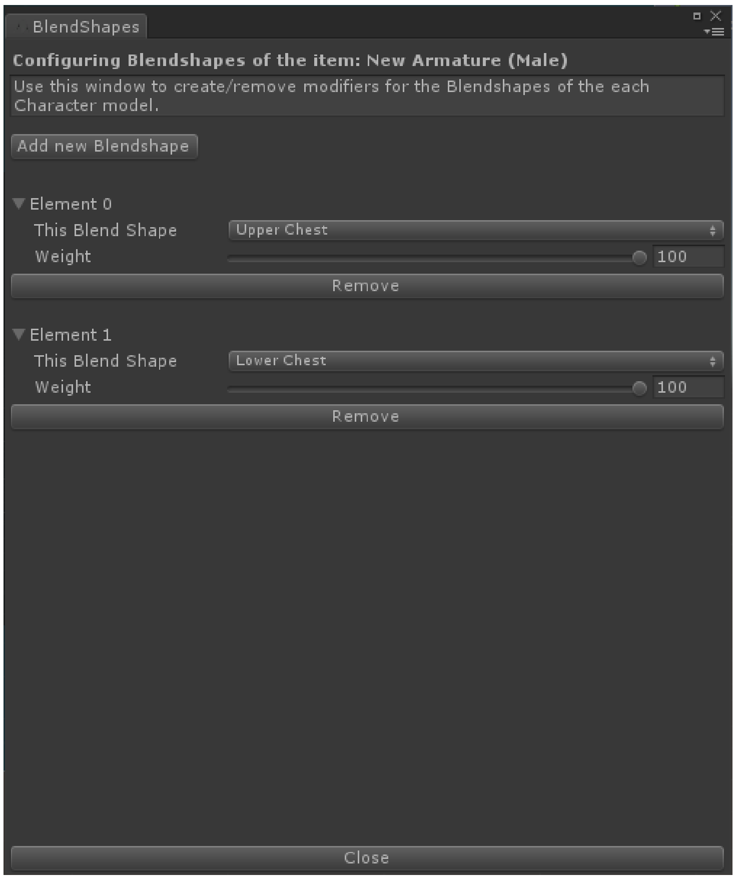
- **Name:** The Name of the GearPiece in the game.
- **Hide Hair?**: Should the hair disappear if this item is equipped? This is mainly used for helmets.
- **StatsModifier:** Define which stats this piece affect and how.
- **GearSlot:** Define the slot of this item.

On the upper-right you can see the ‘**Gear Models**’, here you have to assign for both genders the models of all the races.

Every Race created and present in the project will be displayed here. Assign the prefabbed SkinnedMeshRenderer for every Object field.

With the ‘BlendShapes’ button you can define the BlendShapes modifiers when this gear piece is equipped. If we’re equipping an armature we may want to shrink the chest of our character model to prevent clipping the body with the armature model.

If we click on the BlendShapes button this Window will appear:



BlendShapes

Configuring Blendshapes of the item: New Armature (Male)

Use this window to create/remove modifiers for the Blendshapes of the each Character model.

Element 0

This Blend Shape: Upper Chest

Weight: 100

Element 1

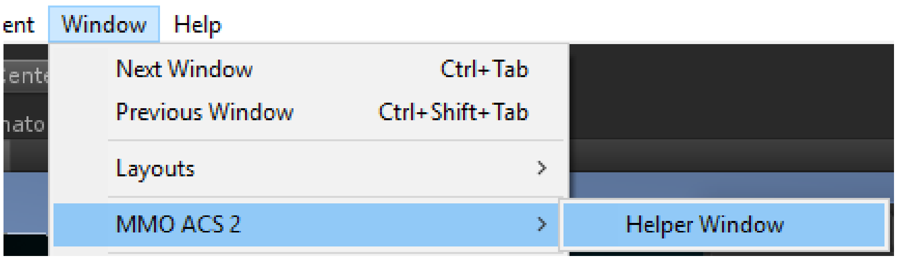
This Blend Shape: Lower Chest

Weight: 100

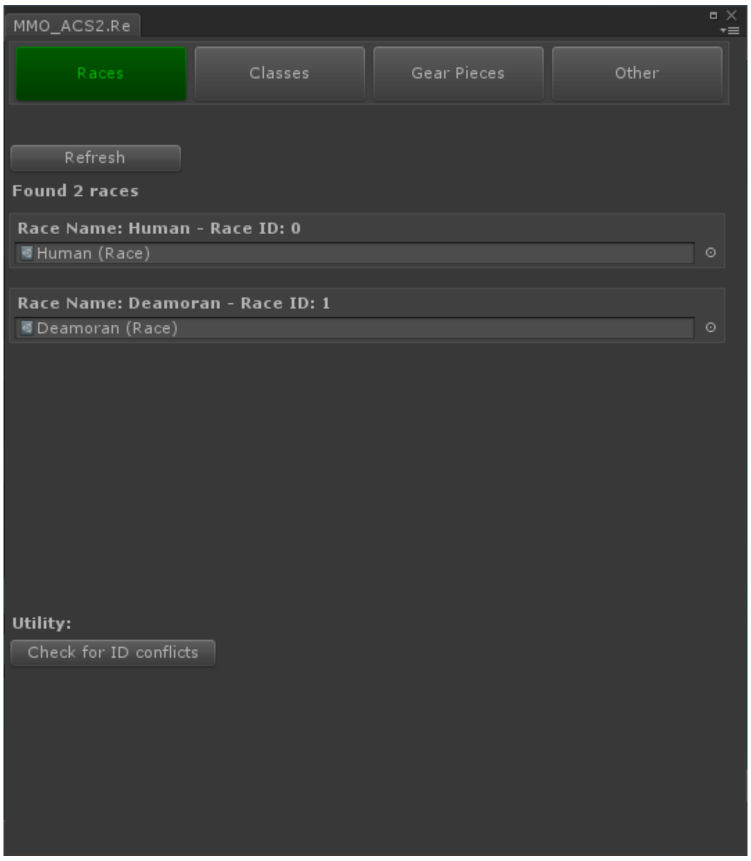
Since you may want to have different models for the gear piece you can define different SkinnedMeshRenderer and BlendShapes for each Race and both genders.

HELPER WINDOW:

As seen that Races, Classes and GearPieces utilizes IDs for identifying a specific thing in the project. A conflict or a mismatch of IDs will surely lead you to bugs.
Remember that every Race, GameClass and GearPiece must have unique ID.
You can open up the MMO ACS 2 Helper Window to help you with those checks from Windows -> MMO ACS 2 -> Helper Window:



The window will pop-up:



Found 2 races

Race Name: Human - Race ID: 0
Human (Race)

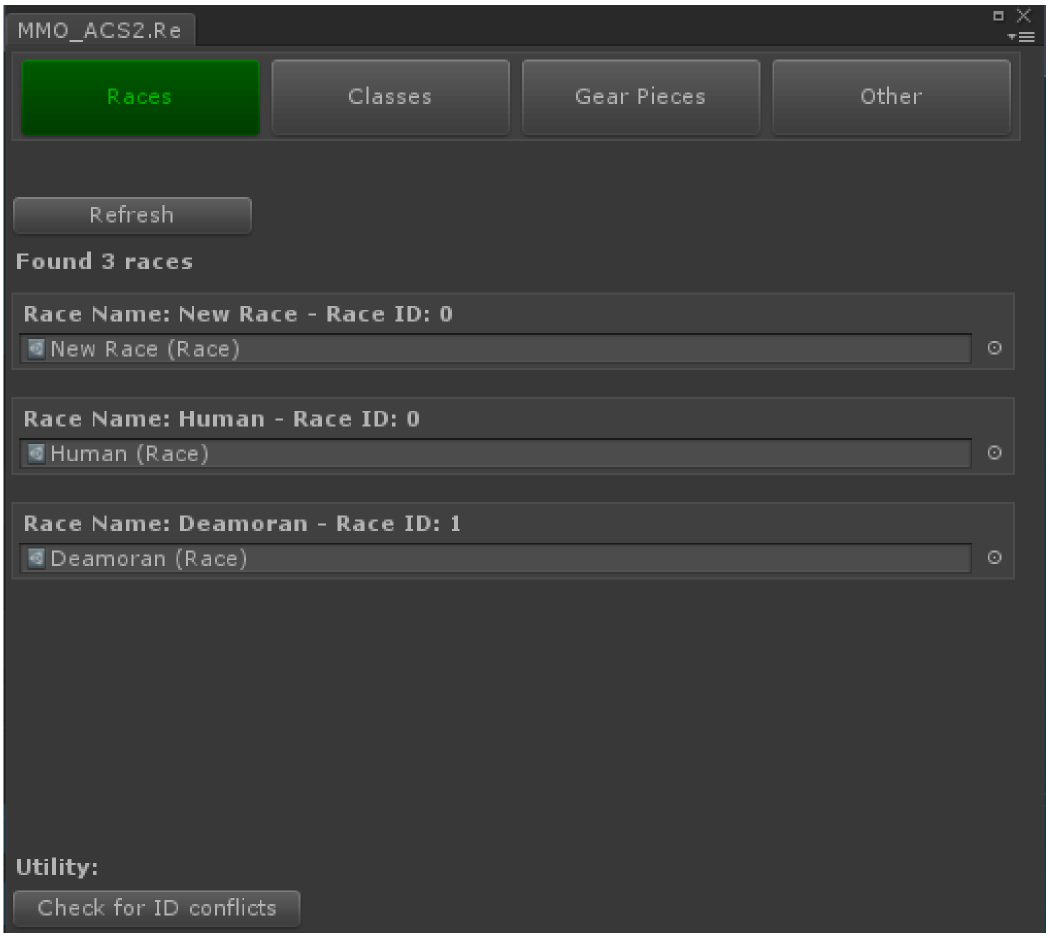
Race Name: Deamoran - Race ID: 1
Deamoran (Race)

It is organized in tabs:

- **Races**: List all the Races in the Project and allows you to run an 'ID Conflict Check' with them.
- **GameClass**: List all the GameClasses in the Project and allows you to run an 'ID Conflict Check' with them.
- **GearPieces**: List all the GearPieces in the Project and allows you to run an 'ID Conflict Check'.
- **Other**: Allows you to run general fixes if something in your project is wrong.

CONFLICT CHECK:

Let's look at this situation, we have two Races in the project with the same id (0):



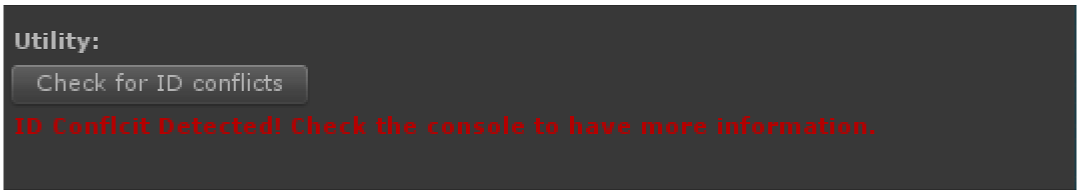
Found 3 races

Race Name: New Race - Race ID: 0
New Race (Race)

Race Name: Human - Race ID: 0
Human (Race)

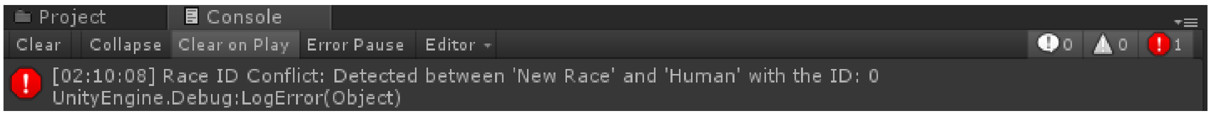
Race Name: Deamoran - Race ID: 1
Deamoran (Race)

If we run a ‘Check for ID conflicts’ this will be the result:



ID Conflict Detected! Check the console to have more information.

The console will provide us all the information we need in order to fix the issue:



```
[02:10:08] Race ID Conflict: Detected between 'New Race' and 'Human' with the ID: 0
UnityEngine.Debug:LogError(Object)
```

The other tabs will just display all the GameClasses/GearPieces in the Project and will let you run a ‘Conflict ID Check’.

Don’t forget to often use this tool!

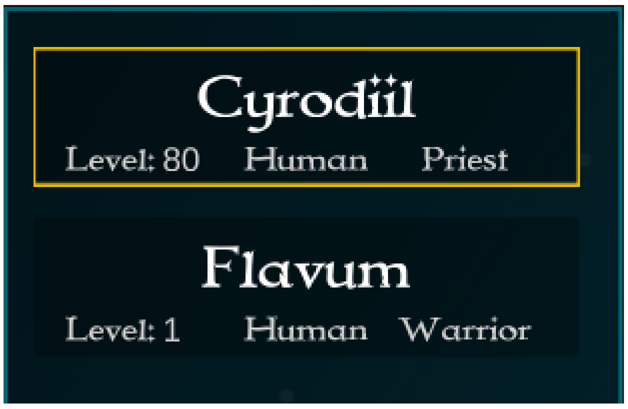
RETRIEVING CHARACTERS:

Let’s see how we can retrieve the data of a character from the database and show it in the game as a Character.

The “**CharacterSelectionManager**” class will do that. We’ll see how it works, it’s very important that you understand how we take and distribute Characters data from the database to the Character UI representation and then to the game.

For understanding the CharacterSelectionManager class we have to firstly take a look to other few scripts.

In the Character Selection we have a UI representation of the characters bounded to our account:



Those UI elements are instances of the prefab: ‘**CharacterInList**’. Those prefabs are composed of Texts and a ‘**CharData**’ GameObject. They also have a script attached that has same name.

The **CharData** holds all the data of a character. This data is filled while we're retrieving the characters from the database, they are filled from the script ‘**Character Selection Manager**’ and saved in the **CharData** script.

Open up the **CharData** script:

You see that in this class we’ve got all the variables that we’ve created in the database. As we said before the CharacterInList have a reference to the **CharData** of the current character, this will be used widely also later in the Demo scene.

Let’s return to the “**CharacterSelectionManager**” class.

This script manages the Character Selection; it achieves:

- Detecting Characters from the database and instantiate them as ‘**CharacterInList**’ prefabs.
- Sorting Characters in the list by ID.
- Delete characters.
- Allows to load the next scene.

So this is a really important part of MMO-ACS 2.

Let’s take a look on how we actually retrieve characters from the database and transform them into ‘CharacterInList’, from there once we’ve got the ‘**CharData**’ filled, we don’t have to do nothing more than spawn the correct Character Model, assign the correct Race and GameClass value and we’ll be ready to go.

In the ‘**CharacterSelectionManager.cs**’ jump to the IEnumerator ‘CheckCharacters’.

What we do in this IEnumerator, is to open the “checkcharacters.php” script from our server with a UnityWebRequest. We’ll send that request with a form that will contain the username we’ve used while we were logging in. Remember that the Account static class have a reference to the username, so that’s the parameter that we will send in the form:

```
form.AddField("user", Account.user);
```

Now take a look on the “checkcharacters.php” to see how it works. What it will simply do is to pick all the character of that username and send them back to Unity with a specific syntax. If no characters exist for that account it will simply return ‘**0**’ and from Unity we will ask to the user to create a new character.

That’s what you will receive from the “checkcharacters.php” if a character exists.

ID:3|NAME:Cyrodiil|GENDER:1|LEVEL:80|RACE:0|CLASS:3|SKIN:0|HAIRTYPE:1|HAIRCOLOR:4|STRENGHT:0|AGILITY:0|INTELLECT:0|SPIRIT:0|ARMOUR:0|CRITICALCHANCE:0|MELEEDAMAGE:0|SPELLDAMAGE:0|HEADEQUIPPED:-1|CHESTEQUIPPED:5|HANDSEQUIPPED:-1|LEGSEQUIPPED:-1|FEETSEQUIPPED:-1|RHANDEQUIPPED:-1;

It is that simple:

You have written in caps the **index** of a value (‘**ID**’), then you have ‘**:**’ (that means ‘equals’) and then it’s value (**3**).

After every values you will find the separator ‘|’, means that the following **index** and **value** will be of another variable.

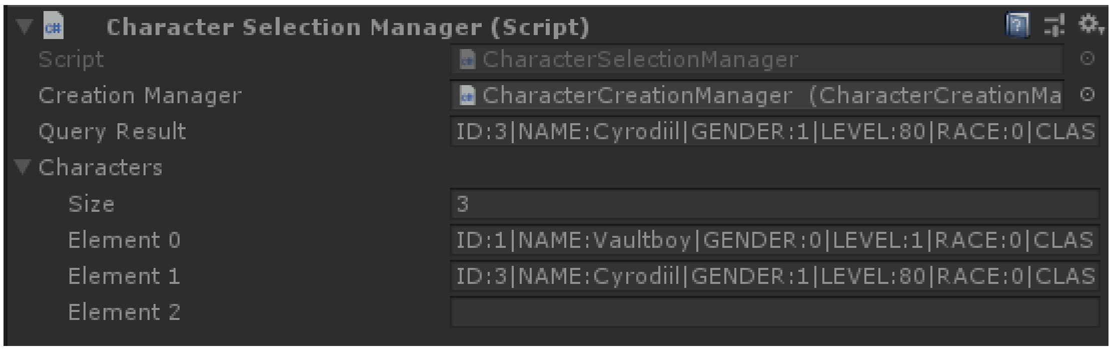
The last thing is the semicolon at the end ‘**;**’ that means all the data of the current character has been echoed.

So all the columns of the database will be inserted in a string like that and sent to Unity.

In the first example there was only the character named ‘Cyrodiil’ on the database, in the following one there will be 2 characters, notice where the ‘**;**’ separator applies.

ID:1|NAME:Vaultboy|GENDER:0|LEVEL:80|RACE:0|CLASS:0|SKIN:3|HAIRTYPE:1|HAIRCOLOR:0|STRENGHT:0|AGILITY:0|INTELLECT:0|SPIRIT:0|ARMOUR:0|CRITICALCHANCE:0|MELEEDAMAGE:0|SPELLDAMAGE:0|HEADEQUIPPED:11|CHESTEQUIPPED:6|HANDSEQUIPPED:12|LEGSEQUIPPED:7|FEETSEQUIPPED:8|RHANDEQUIPPED:-1;
ID:3|NAME:Cyrodiil|GENDER:1|LEVEL:80|RACE:0|CLASS:3|SKIN:0|HAIRTYPE:1|HAIRCOLOR:4|STRENGHT:0|AGILITY:0|INTELLECT:0|SPIRIT:0|ARMOUR:0|CRITICALCHANCE:0|MELEEDAMAGE:0|SPELLDAMAGE:0|HEADEQUIPPED:-1|CHESTEQUIPPED:5|HANDSEQUIPPED:-1|LEGSEQUIPPED:-1|FEETSEQUIPPED:-1|RHANDEQUIPPED:-1;

The class ‘CharacterSelectionManager’ will take all this data and manage it by itself by splitting characters and make life easier when it’s about transforming this data into actual game characters.



Character Selection Manager (Script)

Script	CharacterSelectionManager
Creation Manager	CharacterCreationManager (CharacterCreationMa
Query Result	ID:3|NAME:Cyrodiil|GENDER:1|LEVEL:80|RACE:0|CLAS
Characters	
Size	3
Element 0	ID:1|NAME:Vaultboy|GENDER:0|LEVEL:1|RACE:0|CLAS
Element 1	ID:3|NAME:Cyrodiil|GENDER:1|LEVEL:80|RACE:0|CLAS
Element 2	

- **Query result** is the data all together (like in the example above) while **Characters** is an array of strings that contains every character as an array element with its own data.

Note: you will always get an empty line at the end of the array.

Now we will perform some actions with those values, so how do we pick them form that string?

There is a function in the ‘**InputHelper**’ class that does that:

```
InputHelper.GetData(characters[0], "ID:")
```

Where characters[0] is the first element of the ‘Characters’ array we saw in the pick before and ‘**ID**’ is the **index** of the value that we want.

In this case this line of code will return “1” as a string.

You can use int.Parse, float.Parse etc to transform those strings to numbers.

Now there are some functions that are explained in the code that makes the magic happens, there are all as simple as loading a Material from an ID, ordering a list by a value, instantiating UI elements and spawning the character.

The InputHelper.GetData function will be widely used there to transform IDs to their respective value in terms of game data, like the GameClass, RaceData, Materials for the HairColor and Meshes for the HairTypes.

CREATING CHARACTERS:

Creating Characters is basically the opposite of Retrieving.

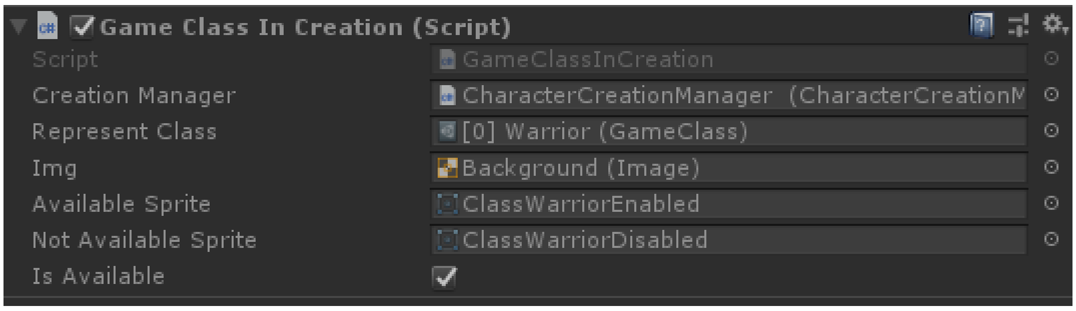
Instead of picking data from the database and get them to Unity we will do the inverse, setting values from Unity and send them to the database.

The process is handled by simple Toggles and Buttons. If you take a look at the functions that the Toggles and Buttons calls when they're clicked everything will be self-explained.

There is only one thing to mention about the Class selection, as you saw while creating a character you can select which classes can be used for a specific race.

To automate this process while creating a character, so to disable the toggles and the UI of a non-available class for the current race we use one script: 'GameClassInCreation.cs'.

Its usage is extremely simple; you only have to set those values in the inspector:



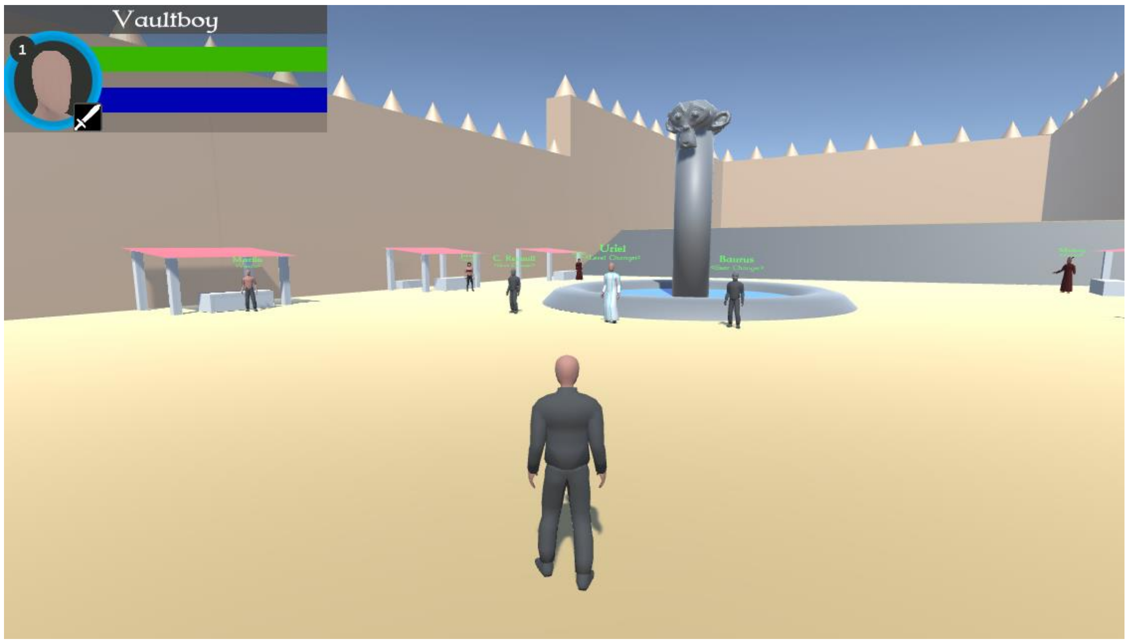
Game Class In Creation (Script)

Script	GameClassInCreation
Creation Manager	CharacterCreationManager (CharacterCreationM
Represent Class	[0] Warrior (GameClass)
Img	Background (Image)
Available Sprite	ClassWarriorEnabled
Not Available Sprite	ClassWarriorDisabled
Is Available	✓

Everything will be managed from the system. Errors like 'Character Name is Already In Use' or 'Invalid Name' will be managed server-side, when we press on "Create New Character" and send a form to the script "createcharacter.php" it will firstly validate all the data, then if everything is ok it will create a new record on the database with our new character.

Otherwise it will return an error message that will be managed from Unity.

GAME DEMO SCENE:



In the Asset a Demo of what you should do next on your project is included.

Here we will spawn our character model and attach it to a very simple Character Controller that will let us move him around.

There are some NPCs in the World and some interactions like changing gear and level is already scripted to give you an example on how to change and save data at runtime. This should let you start actually working on your project.

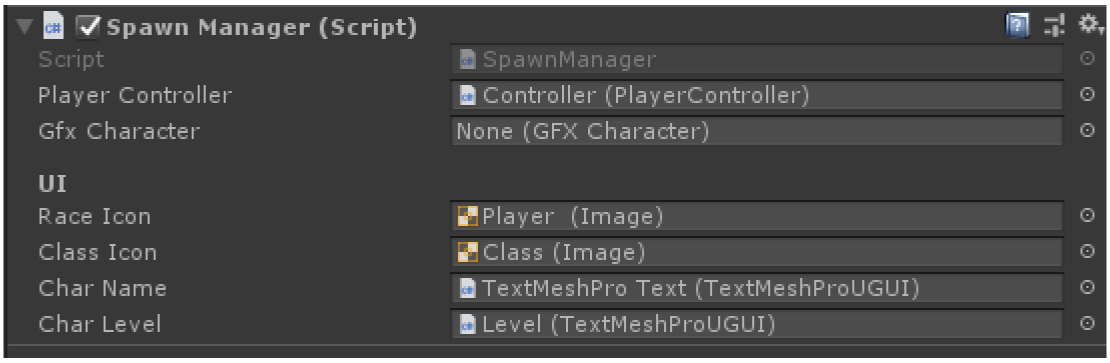
From the Character Selection Screen if we press on “Enter World” few things will happen:

- The parent of the Spawned Character Model will be cleared.
- The parent of the CharData of the selected character will be cleared.
- The parent of the CharData will be set to the Spawned Character Model.
- The Spawned Character Model will be flagged with ‘DontDestroyOnLoad’.
- The Loading Scene will be loaded.

The Loading Screen is managed by the script “FakeLoading.cs” that will fake the load by adding some extra seconds before loading the actual level where we can play (“3) World”).

The first thing that will happen into the World scene is in the script ‘SpawnManager.cs’. As the name may suggest this script will manage the spawning process of our character, including setting up the UI and parenting the Character Model to the Controller.

The SpawnManager is initialized in the scene like that:



Spawn Manager (Script)

Script	SpawnManager
Player Controller	Controller (PlayerController)
Gfx Character	None (GFX Character)
UI	
Race Icon	Player (Image)
Class Icon	Class (Image)
Char Name	TextMeshPro Text (TextMeshProUGUI)
Char Level	Level (TextMeshProUGUI)

It will perform those actions:

- Taking a reference to the Character Model by its tag ‘GFXCharacter’.
- Parenting the Character Model to the Controller and resetting its position.
- Setting reference to the Character Data and animator.
- Setting up the UI (Name, Image etc) thanks to the CharData.

With that done we have our character model able to walk around the world.

Other actions that we do in this level includes saving at runtime the state of our character to the database. Indeed, if you change the gear or the level of the character by interacting with the NPCs and then logging out, you will see that the changes will persist.

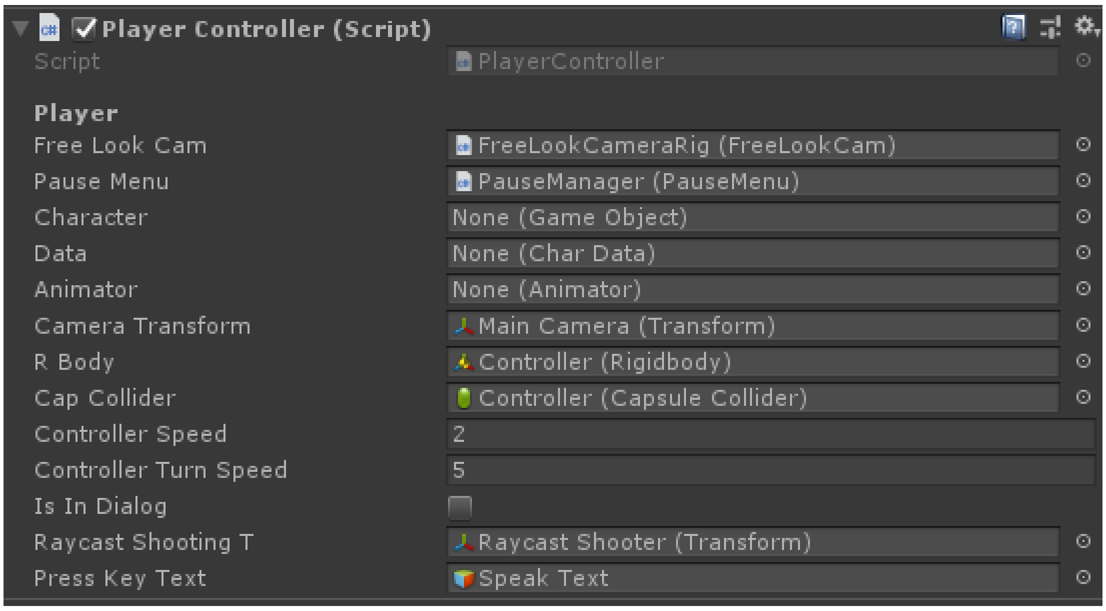
This is managed by the script ‘RuntimeSaveManager.cs’.

To save the state to a character you only have to call the following method from every script:

```
RuntimeSaveManager.instance.SaveCharData(characterData);
```

With this example you should be able to save at runtime any data you will want to have in your game. Still you can use any method you may prefer.

Let's now take a look at the PlayerController class and how we simply interact with the NPCs in the world:



Player Controller (Script)

Script	PlayerController
Player	
Free Look Cam	FreeLookCameraRig (FreeLookCam)
Pause Menu	PauseManager (PauseMenu)
Character	None (Game Object)
Data	None (Char Data)
Animator	None (Animator)
Camera Transform	Main Camera (Transform)
R Body	Controller (Rigidbody)
Cap Collider	Controller (Capsule Collider)
Controller Speed	2
Controller Turn Speed	5
Is In Dialog	☐
Raycast Shooting T	Raycast Shooter (Transform)
Press Key Text	Speak Text

You can read the code and the comments of this script, It's all explained.

What this script do:

- Transform the Input of the Axis (Vertical and Horizontal) into forces applied to our rigidbody.
- Makes the Animator function properly.
- Shoots Raycasts to detect NPCs.
- Open Dialogs (by simply enabling UI GameObjects) if a NPC is near and a key is pressed.

The Dialogs are really simple, they’re a collection of Buttons that does more or less the same action.

Those actions are written in the ‘**ChangesAtRuntime.cs’** script. Here you can see how to modify CharData at runtime, like the character Level and how to equip and unequip gear and spawn the correct model with the correct Blendshapes.

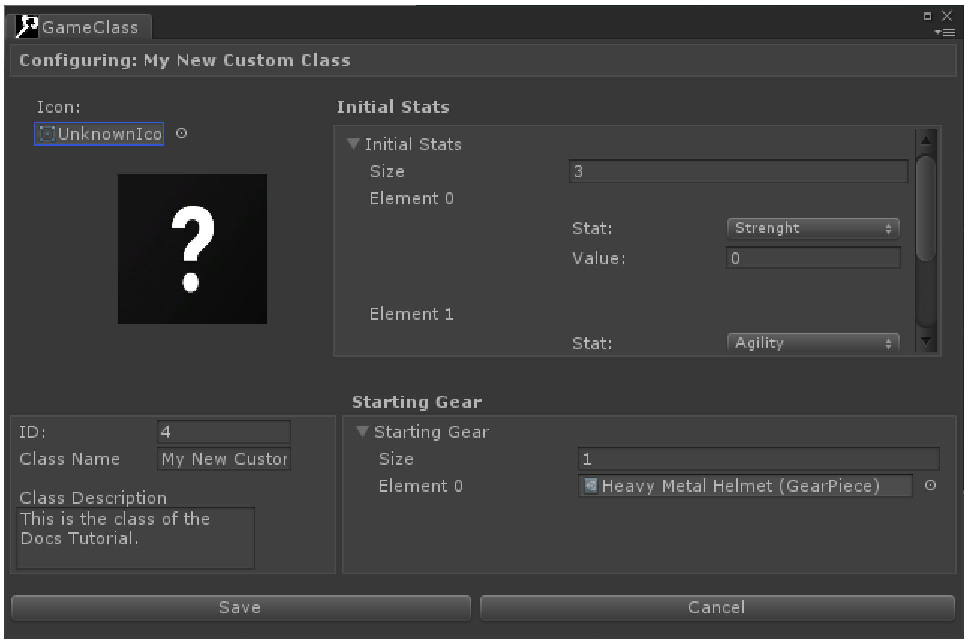
That’s all you have to know to kick-start your project.

In the next sections I will show step by step how to add new data to the database and back, how to add new races, new classes and new gear.

CREATING NEW CLASS:

Create a New Class and configure it from the Inspector.

Be sure that the ID is not the same as another class.



GameClass

Configuring: My New Custom Class

Icon: UnknownIco

Initial Stats

Initial Stats			
Size	3		
Element 0			
	Stat:	Strenght	
	Value:	0	
Element 1			
	Stat:	Agility	

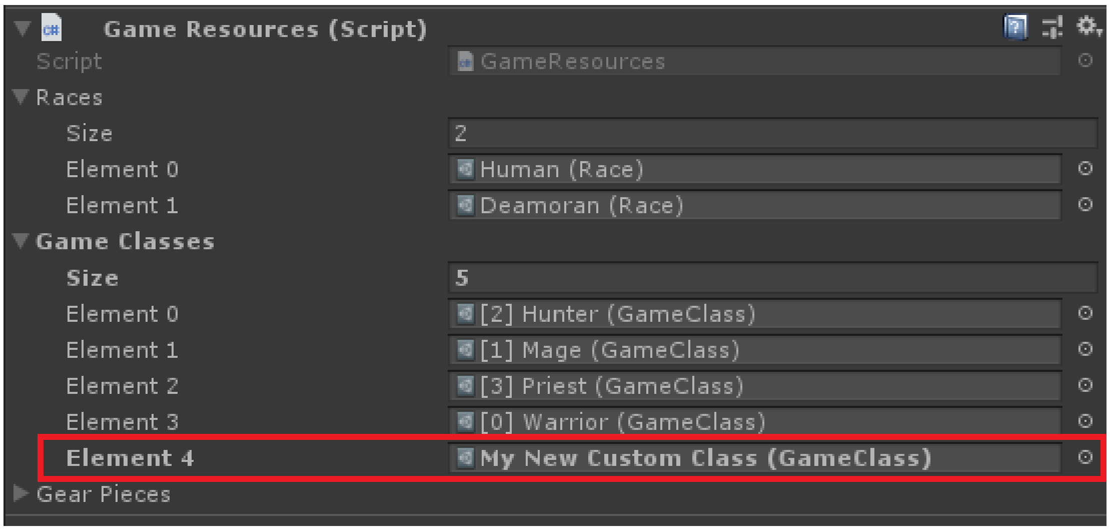
ID: 4
Class Name: My New Custor
Class Description: This is the class of the Docs Tutorial.

Starting Gear

Starting Gear	
Size	1
Element 0	Heavy Metal Helmet (GearPiece)

Save | Cancel

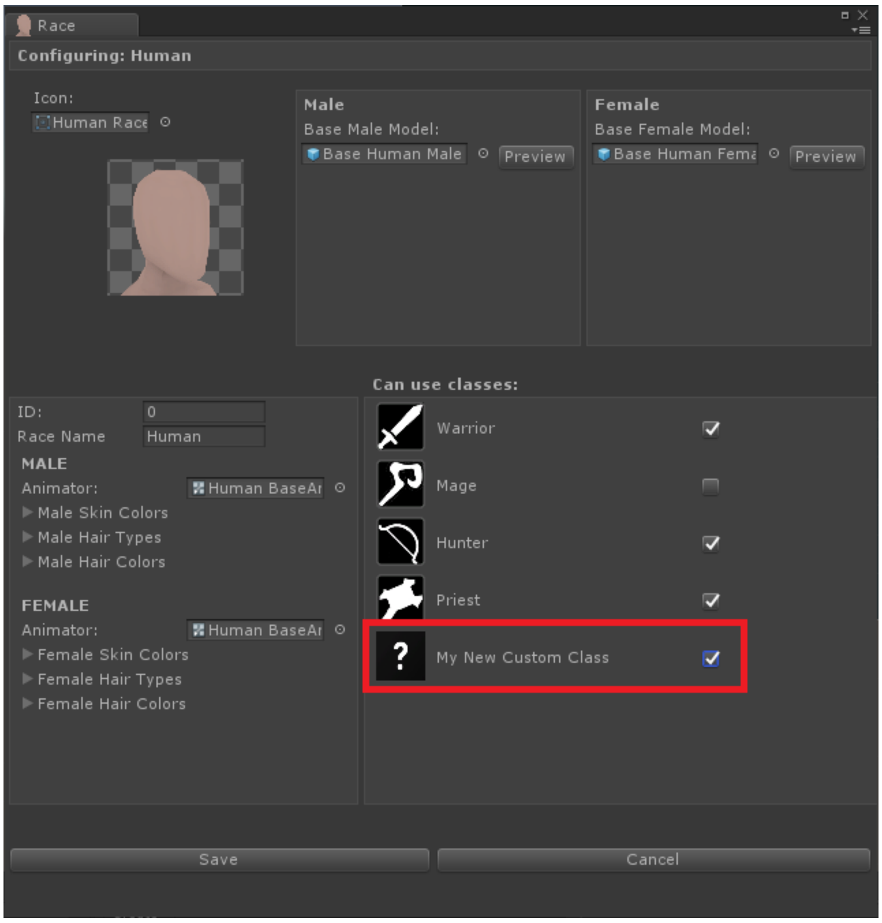
Insert the New Class in the correct GameResources array:



Game Resources (Script)

Script	GameResources
Races	
Size	2
Element 0	Human (Race)
Element 1	Deamoran (Race)
Game Classes	
Size	5
Element 0	[2] Hunter (GameClass)
Element 1	[1] Mage (GameClass)
Element 2	[3] Priest (GameClass)
Element 3	[0] Warrior (GameClass)
Element 4	My New Custom Class (GameClass)
Gear Pieces	

Open a Race will be able to use this New Class and by configuring it, flag it as usable class:



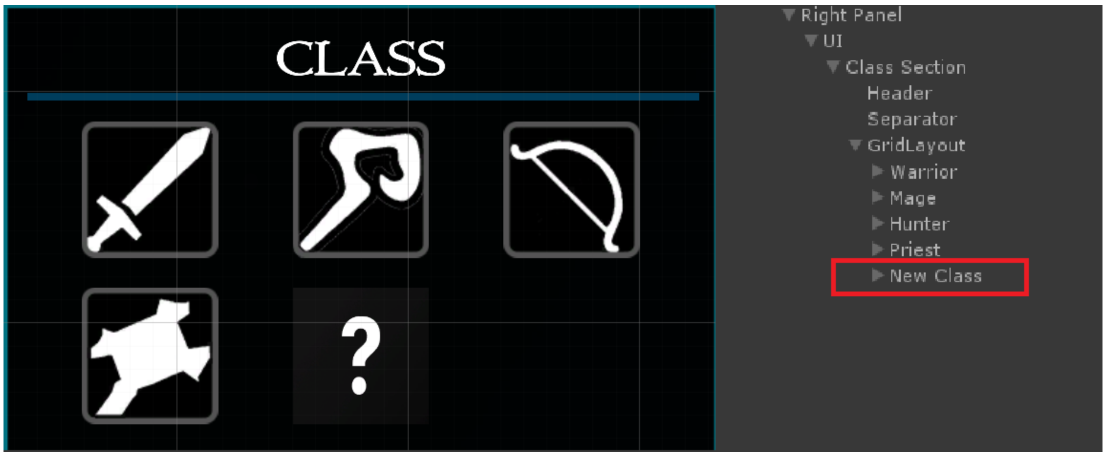
ID: 0
Race Name: Human

MALE
Animator: Human BaseAr
Male Skin Colors
Male Hair Types
Male Hair Colors

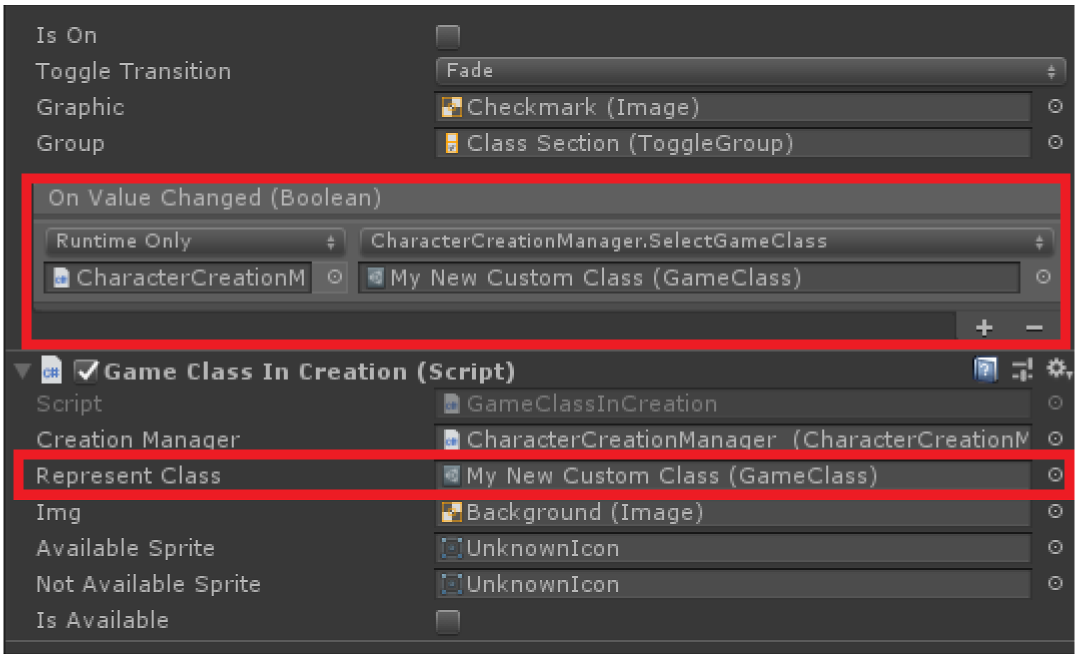
FEMALE
Animator: Human BaseAr
Female Skin Colors
Female Hair Types
Female Hair Colors

Warrior	✓
Mage	
Hunter	✓
Priest	✓
My New Custom Class	✓

Open the “1) Character Selection” scene and modify the Creation UI by duplicating an existing class and modify the icon:



The last thing to modify is the **OnValueChanged** of the Toggle and the fields of the script ‘**GameClassInCreation**’.

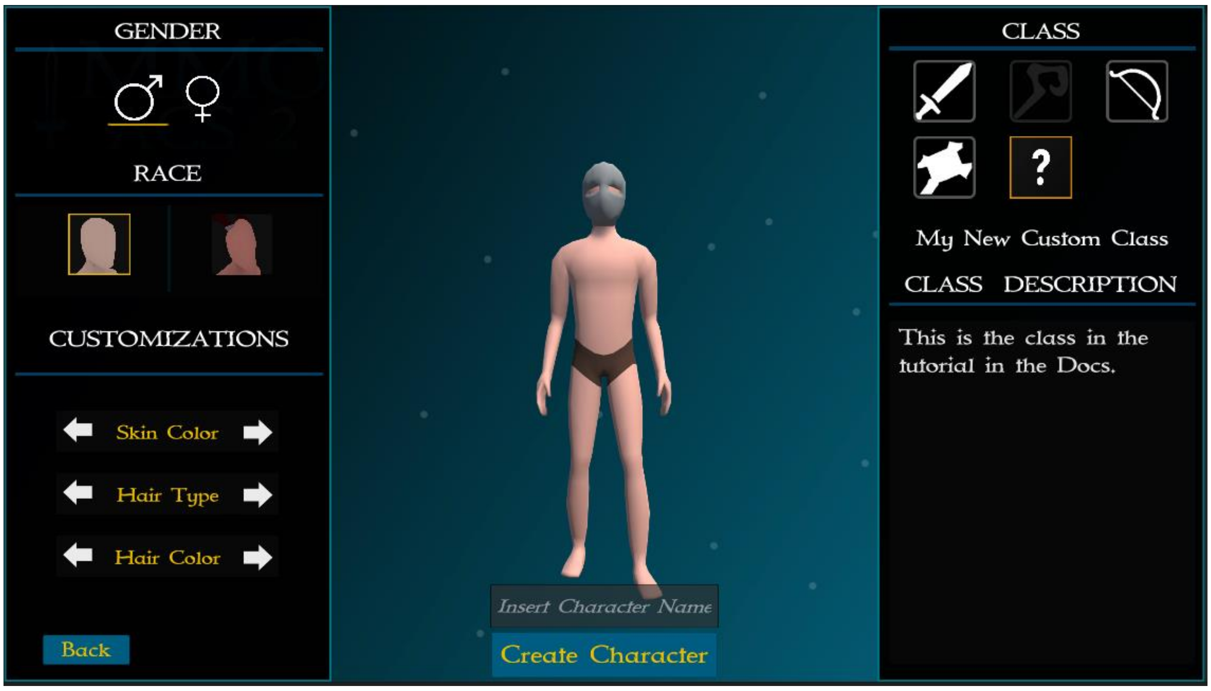


Is On ☐
Toggle Transition: Fade
Graphic: Checkmark (Image)
Group: Class Section (ToggleGroup)

On Value Changed (Boolean)
Runtime Only — CharacterCreationManager.SelectGameClass
CharacterCreationM — My New Custom Class (GameClass)

Game Class In Creation (Script)
Script: GameClassInCreation
Creation Manager: CharacterCreationManager (CharacterCreationM
Represent Class: My New Custom Class (GameClass)
Img: Background (Image)
Available Sprite: UnknownIcon
Not Available Sprite: UnknownIcon
Is Available ☐

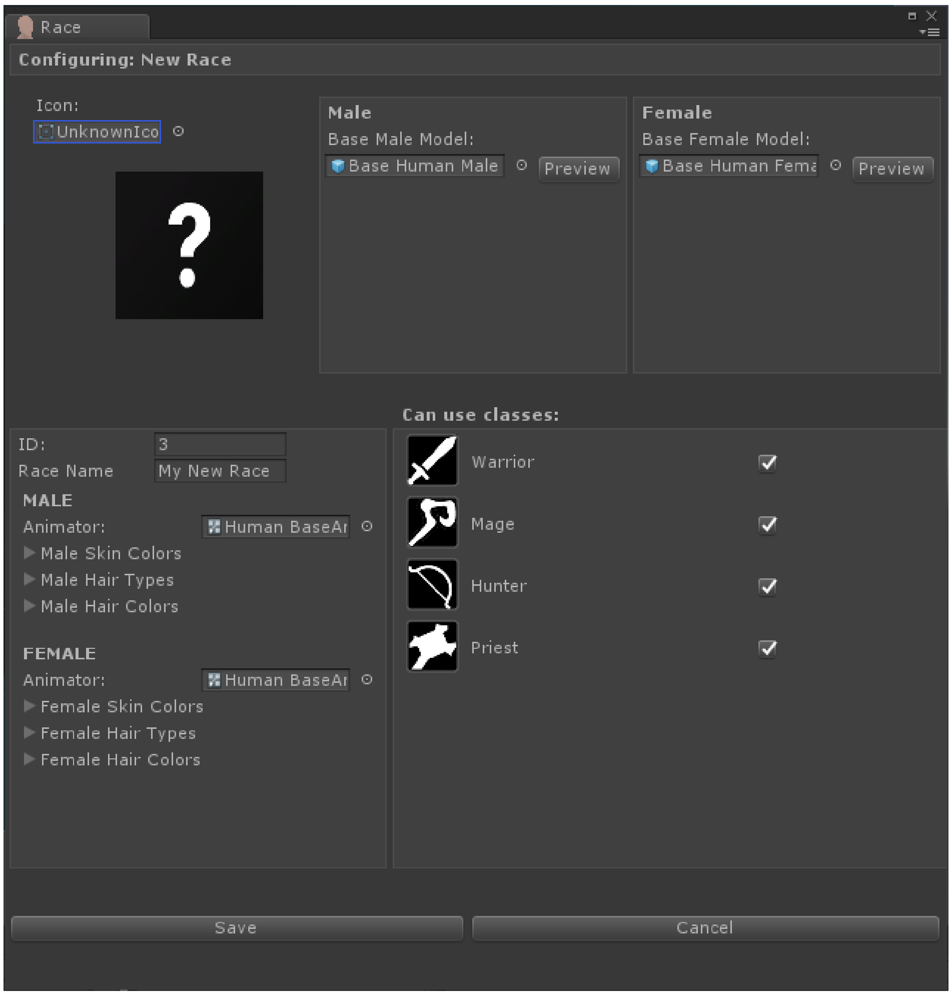
After this, your new class is ready to be used.



CREATING NEW RACES:

Create a New Race and configure it from the Inspector.

Be sure that the ID is not the same as another race.



Race

Configuring: New Race

Icon: UnknownIco

Male
Base Male Model: Base Human Male [Preview]

Female
Base Female Model: Base Human Fema [Preview]

ID: 3
Race Name: My New Race

MALE
Animator: Human BaseAr
▶ Male Skin Colors
▶ Male Hair Types
▶ Male Hair Colors

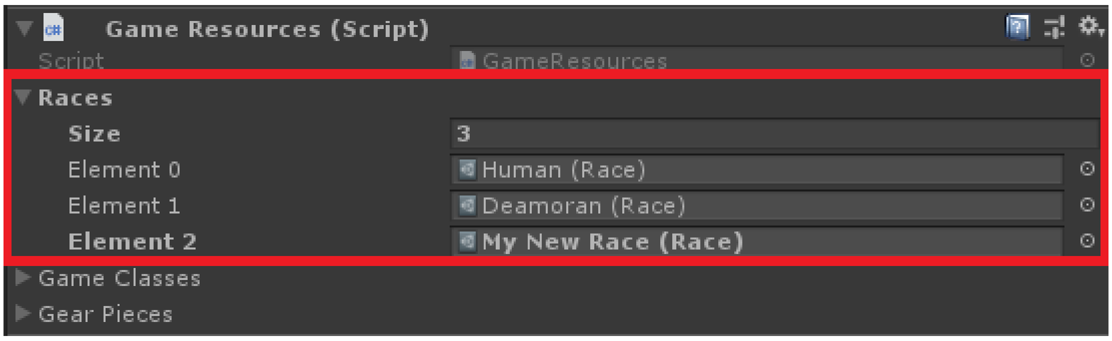
FEMALE
Animator: Human BaseAr
▶ Female Skin Colors
▶ Female Hair Types
▶ Female Hair Colors

Can use classes:

Warrior	✓
Mage	✓
Hunter	✓
Priest	✓

[Save] [Cancel]

Insert the New Race in the correct GameResources array:



Game Resources (Script)

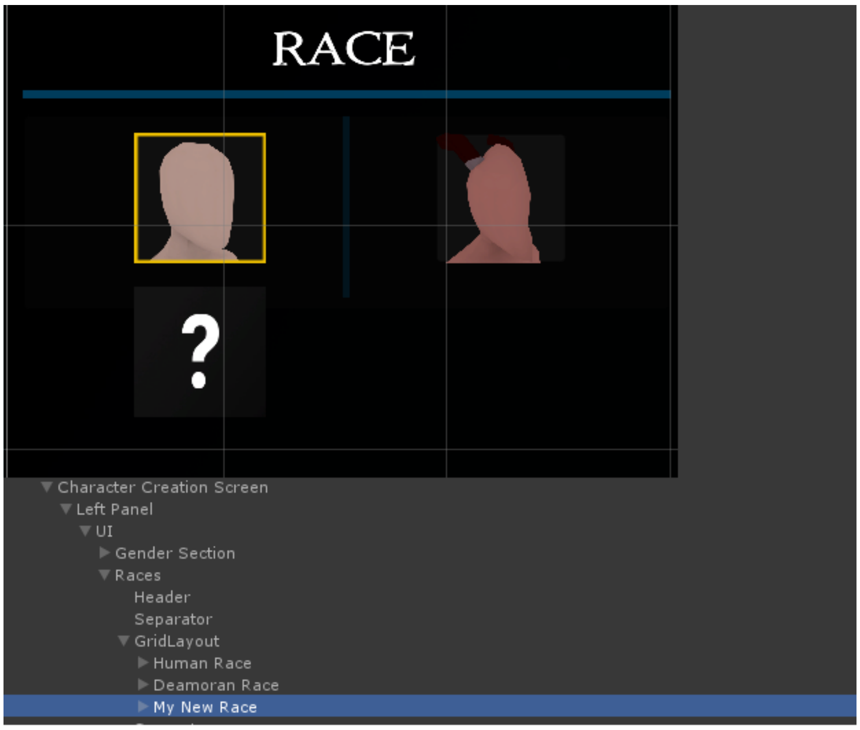
Script: GameResources

▼ Races

Size	3
Element 0	Human (Race)
Element 1	Deamoran (Race)
Element 2	My New Race (Race)

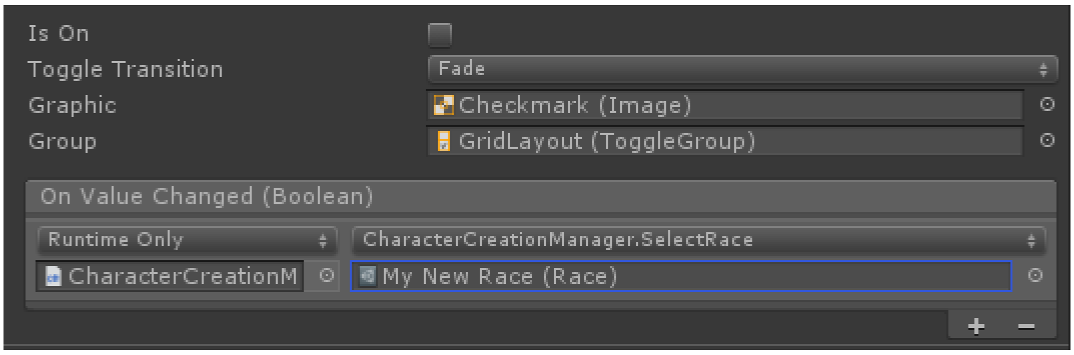
▶ Game Classes
▶ Gear Pieces

Open the “1) Character Selection” scene and modify the Creation UI by duplicating an existing Race and modify the icon:



```
▼ Character Creation Screen
  ▼ Left Panel
    ▼ UI
      ▶ Gender Section
      ▼ Races
          Header
          Separator
        ▼ GridLayout
          ▶ Human Race
          ▶ Deamoran Race
          ▶ My New Race
```

The last thing to modify is the **OnValueChanged** of the Toggle.



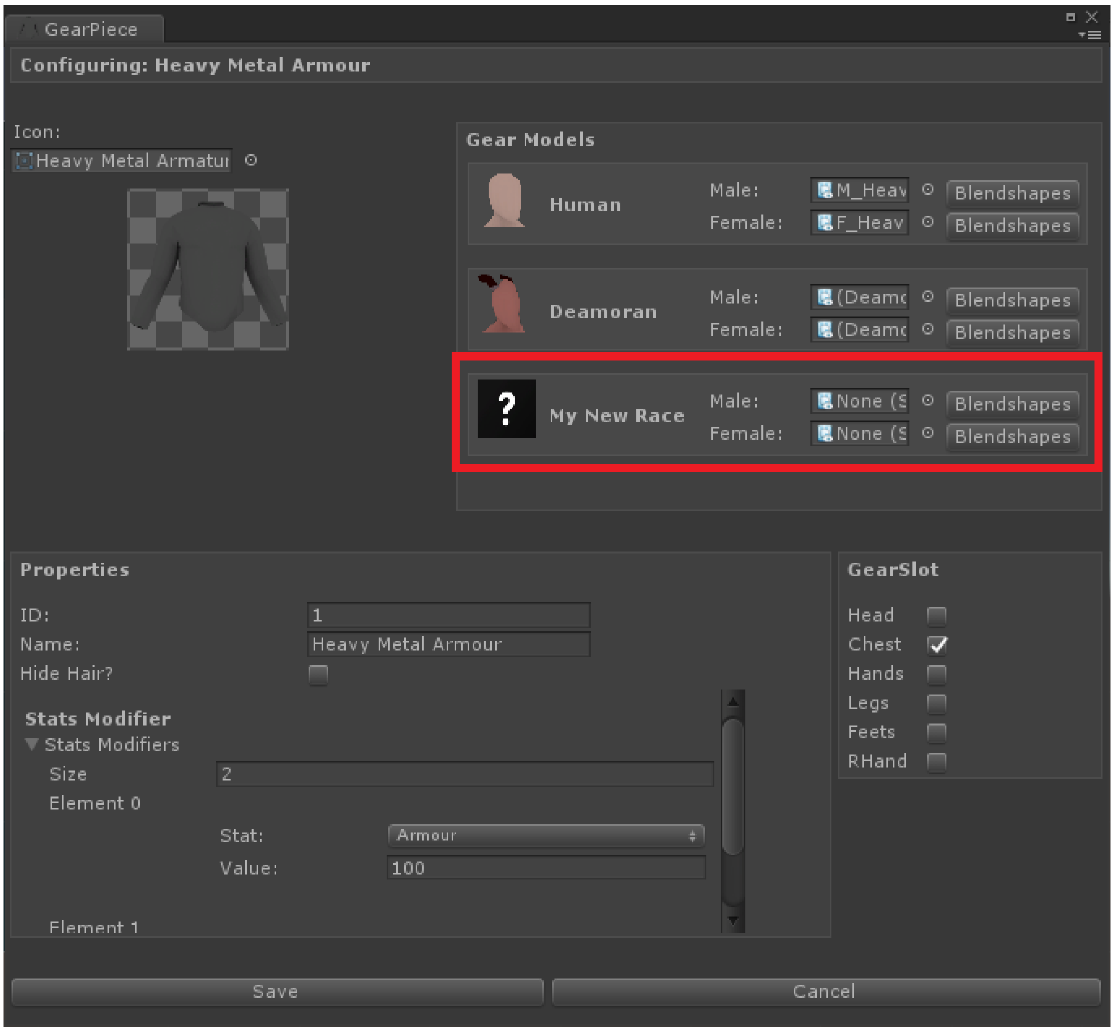
Is On	☐
Toggle Transition	Fade
Graphic	Checkmark (Image)
Group	GridLayout (ToggleGroup)

On Value Changed (Boolean)

Runtime Only	CharacterCreationManager.SelectRace
CharacterCreationM	My New Race (Race)

And your race is now ready! In this example it will look exactly like the Human race, you should import new models, animations and customizations.

Remember that all the GearPieces that are now present in the Project will need to be uploaded since they need the meshes for the new race:



Configuring: Heavy Metal Armour

Icon: Heavy Metal Armatur

Gear Models

Human	Male:	M_Heav
	Female:	F_Heav
Deamoran	Male:	(Deamc
	Female:	(Deamc
My New Race	Male:	None (S
	Female:	None (S

Properties

ID: 1
Name: Heavy Metal Armour
Hide Hair? ☐

Stats Modifier
Stats Modifiers
Size 2
Element 0
Stat: Armour
Value: 100
Element 1

GearSlot

Head ☐
Chest ✓
Hands ☐
Legs ☐
Feets ☐
RHand ☐

If you don’t add meshes to the new races in every gear piece you will encounter error when trying to equip a piece.

CREATING NEW GEAR:

You can create a new Gear Piece just like you’ve created the Race and the GameClass.

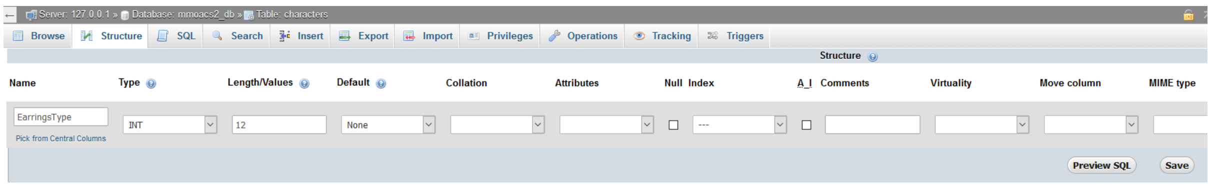
Create a New Gear Piece, configure it and assign it into the ‘GameResources’ ‘Gear Pieces’ array and it is ready.

ADDING NEW DATA:

There are few scripts to modify to add a new variable that will be used in the system.

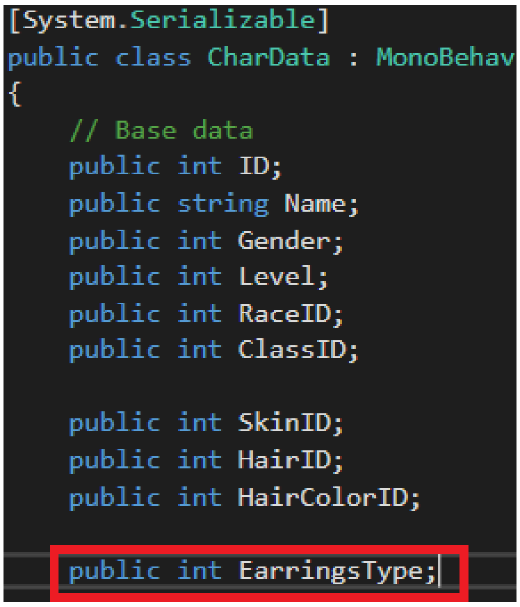
In my case I will create a variable named ‘EarringsType’. It will be an integer that will represent the ID of the equipped earring. You can make it work as a GearPiece or just as a mesh. The purpose of this example is to show you how, what and where you have to write that.

The first thing I will do is to create a new column in the database on the ‘Characters’ table:



Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I	Comments	Virtuality	Move column	MIME type
EarringsType	INT	12	None			☐	---	☐				

Then I’ll open the script ‘CharData’ from the Unity Project, and add a new variable with the same name and type of that we’ve made in the database:

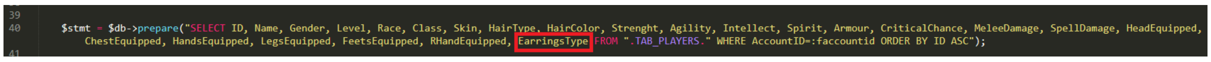


```
[System.Serializable]
public class CharData : MonoBehav
{
    // Base data
    public int ID;
    public string Name;
    public int Gender;
    public int Level;
    public int RaceID;
    public int ClassID;

    public int SkinID;
    public int HairID;
    public int HairColorID;

    public int EarringsType;
```

Now we have to modify the query we do to the database when we ask for all the columns values and when we create new rows, specifically we also want to include that new column we created earlier:

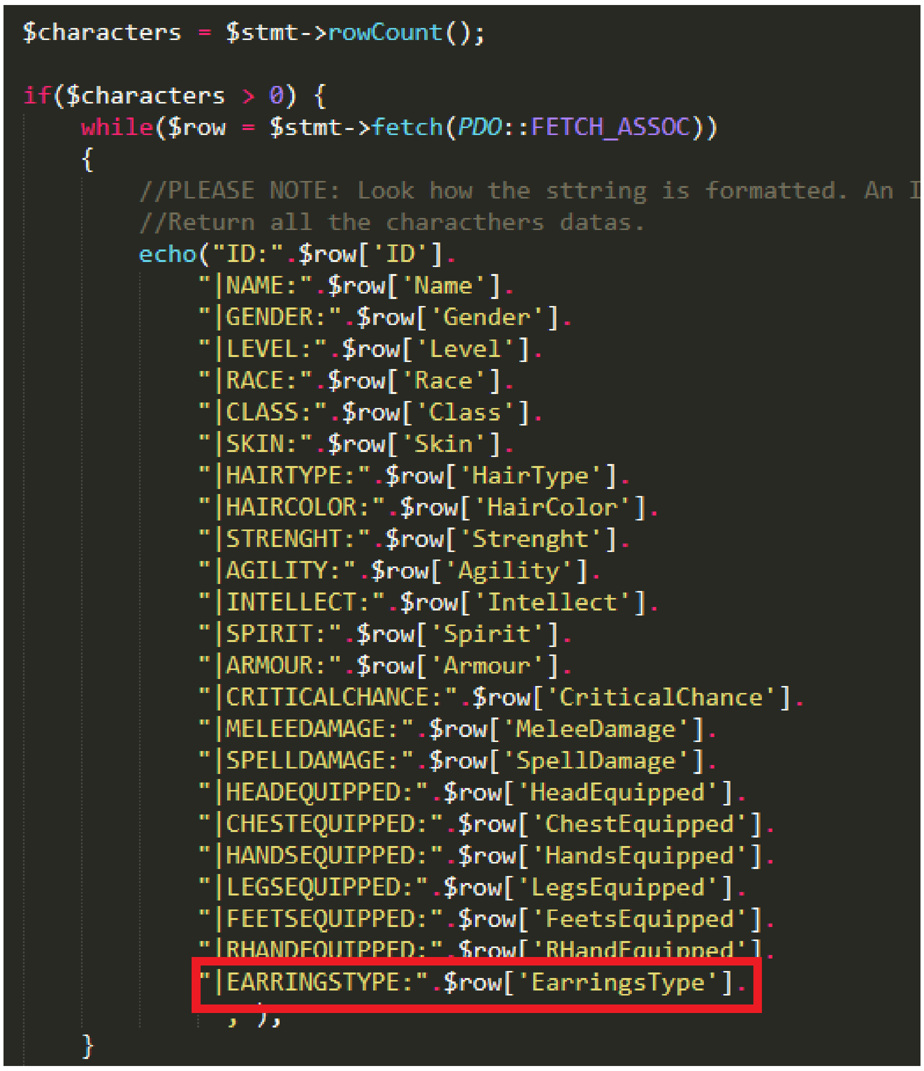
Open the PHP script “checkcharacters.php”:

Add the new column in the SQL Command:



```php
$stmt = $db->prepare("SELECT ID, Name, Gender, Level, Race, Class, Skin, HairType, HairColor, Strenght, Agility, Intellect, Spirit, Armour, CriticalChance, MeleeDamage, SpellDamage, HeadEquipped,
    ChestEquipped, HandsEquipped, LegsEquipped, FeetsEquipped, RHandEquipped, EarringsType FROM ".TAB_PLAYERS." WHERE AccountID=:faccountid ORDER BY ID ASC");
```

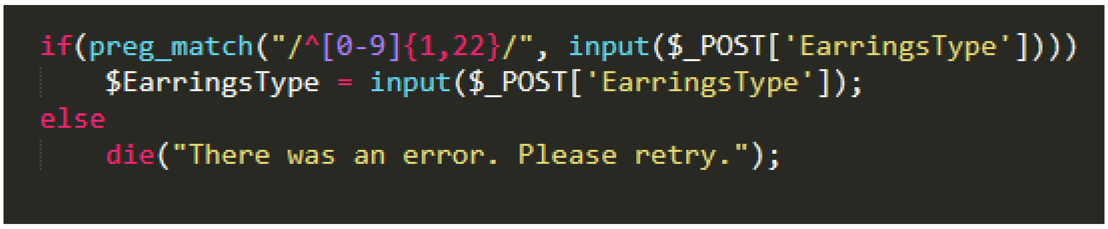
Last thing here is to modify what will be sent to Unity, the long ‘echo’ function will do that, add the new column with the value took from the database:



```php
$characters = $stmt->rowCount();

if($characters > 0) {
    while($row = $stmt->fetch(PDO::FETCH_ASSOC))
    {
        //PLEASE NOTE: Look how the sttring is formatted. An I
        //Return all the characthers datas.
        echo("ID:".$row['ID'].
            "|NAME:".$row['Name'].
            "|GENDER:".$row['Gender'].
            "|LEVEL:".$row['Level'].
            "|RACE:".$row['Race'].
            "|CLASS:".$row['Class'].
            "|SKIN:".$row['Skin'].
            "|HAIRTYPE:".$row['HairType'].
            "|HAIRCOLOR:".$row['HairColor'].
            "|STRENGHT:".$row['Strenght'].
            "|AGILITY:".$row['Agility'].
            "|INTELLECT:".$row['Intellect'].
            "|SPIRIT:".$row['Spirit'].
            "|ARMOUR:".$row['Armour'].
            "|CRITICALCHANCE:".$row['CriticalChance'].
            "|MELEEDAMAGE:".$row['MeleeDamage'].
            "|SPELLDAMAGE:".$row['SpellDamage'].
            "|HEADEQUIPPED:".$row['HeadEquipped'].
            "|CHESTEQUIPPED:".$row['ChestEquipped'].
            "|HANDSEQUIPPED:".$row['HandsEquipped'].
            "|LEGSEQUIPPED:".$row['LegsEquipped'].
            "|FEETSEQUIPPED:".$row['FeetsEquipped'].
            "|RHANDEQUIPPED:".$row['RHandEquipped'].
            "|EARRINGSTYPE:".$row['EarringsType'].
            ";");
    }
```

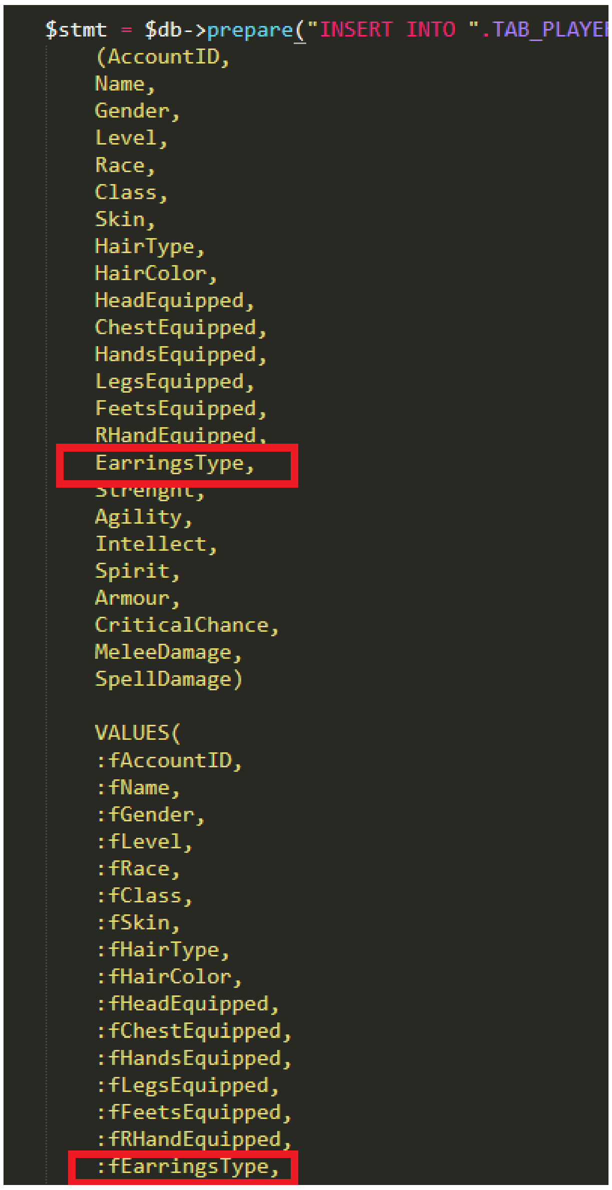
You may also want to send this variable when you create a character, let’s say this will be a customization. In this case you may also want to modify the ‘createnewcharacter.php’:

Add a new variable that equals what’s sent in the POST form at index ‘EarringsType’.



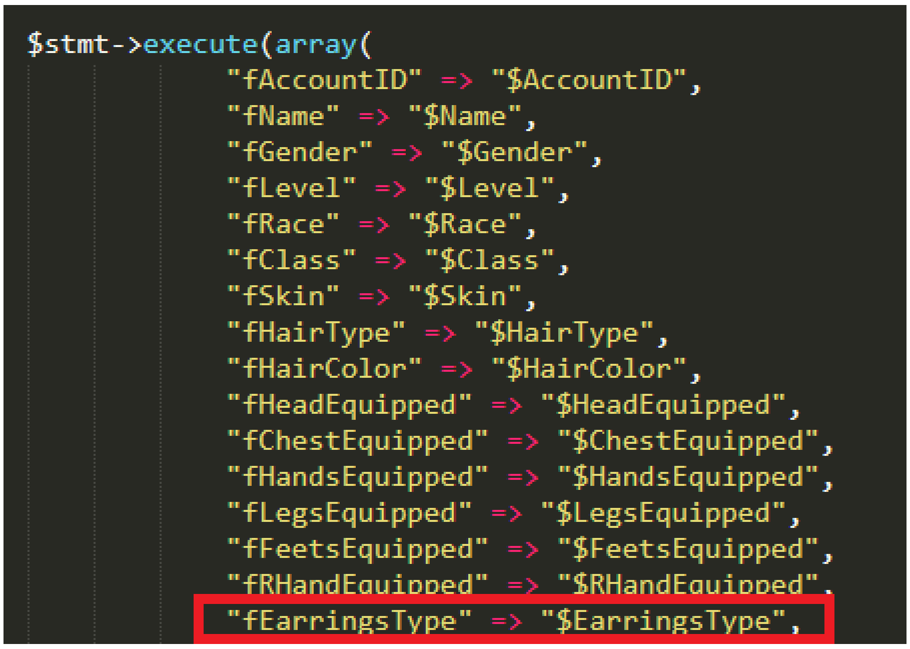
```
if(preg_match("/^[0-9]{1,22}/", input($_POST['EarringsType'])))
    $EarringsType = input($_POST['EarringsType']);
else
    die("There was an error. Please retry.");
```

Modify the query that insert a new record to also include the new variable:



```
$stmt = $db->prepare("INSERT INTO ".TAB_PLAYE
    (AccountID,
    Name,
    Gender,
    Level,
    Race,
    Class,
    Skin,
    HairType,
    HairColor,
    HeadEquipped,
    ChestEquipped,
    HandsEquipped,
    LegsEquipped,
    FeetsEquipped,
    RHandEquipped,
    EarringsType,
    Strength,
    Agility,
    Intellect,
    Spirit,
    Armour,
    CriticalChance,
    MeleeDamage,
    SpellDamage)

    VALUES(
    :fAccountID,
    :fName,
    :fGender,
    :fLevel,
    :fRace,
    :fClass,
    :fSkin,
    :fHairType,
    :fHairColor,
    :fHeadEquipped,
    :fChestEquipped,
    :fHandsEquipped,
    :fLegsEquipped,
    :fFeetsEquipped,
    :fRHandEquipped,
    :fEarringsType,
```

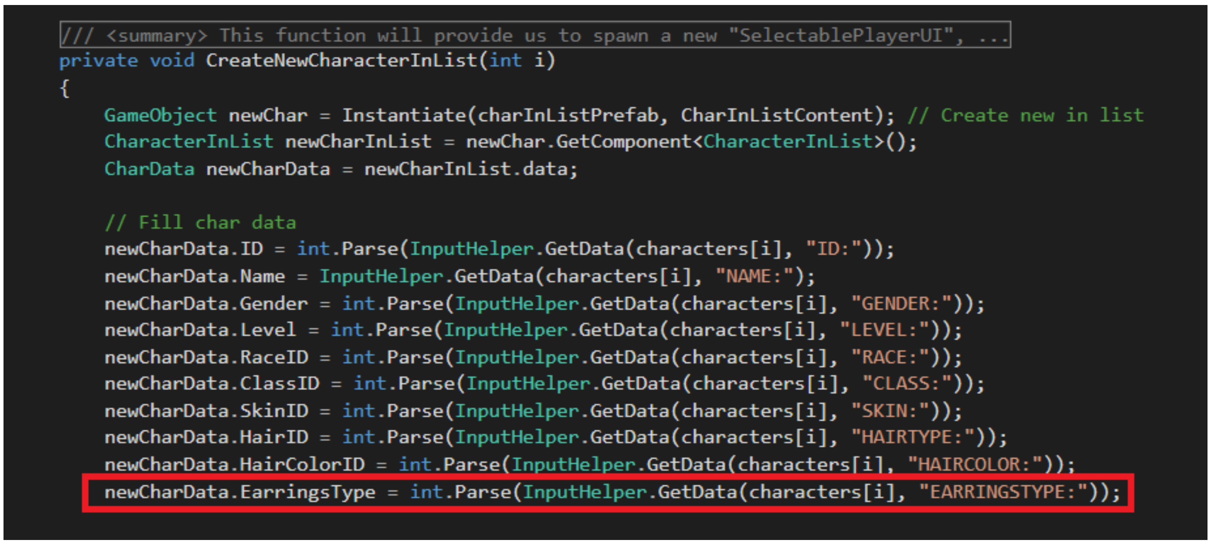
At the time we execute the query, we also need to bind the new created parameter (:fEarringsType) to the value we've got from the POST form before:



```
$stmt->execute(array(
                "fAccountID" => "$AccountID",
                "fName" => "$Name",
                "fGender" => "$Gender",
                "fLevel" => "$Level",
                "fRace" => "$Race",
                "fClass" => "$Class",
                "fSkin" => "$Skin",
                "fHairType" => "$HairType",
                "fHairColor" => "$HairColor",
                "fHeadEquipped" => "$HeadEquipped",
                "fChestEquipped" => "$ChestEquipped",
                "fHandsEquipped" => "$HandsEquipped",
                "fLegsEquipped" => "$LegsEquipped",
                "fFeetsEquipped" => "$FeetsEquipped",
                "fRHandEquipped" => "$RHandEquipped",
                "fEarringsType" => "$EarringsType",
```

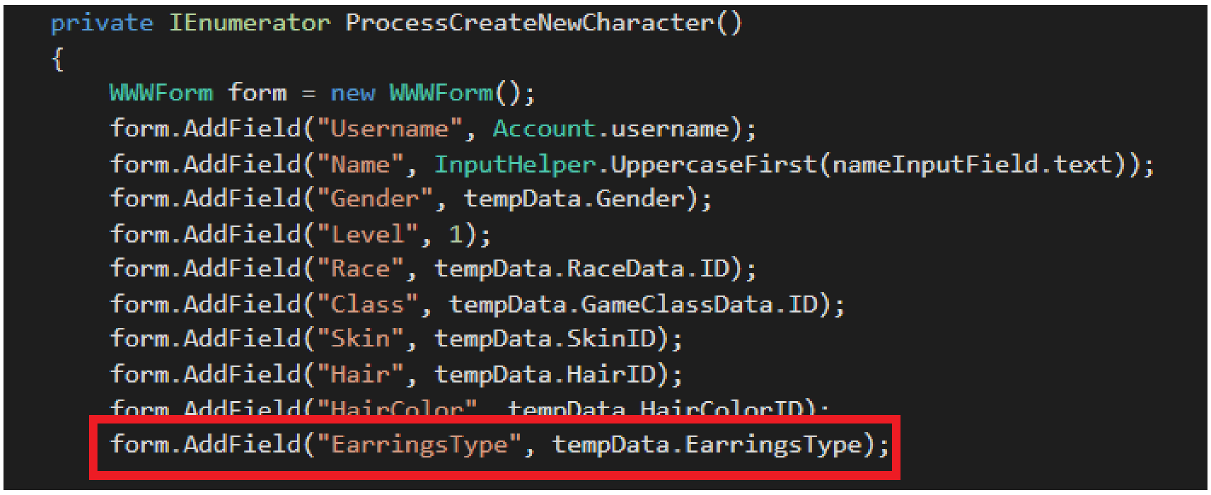
Last two steps and all will be ready!

Open up the script '**CharacterSelectionManager**', like said before in this script we will load and transform the data we've got from the database to actual player data. Reach the 'CreateNewCharacterUI' function and set the newCharData.EarringsType value to what we've found on the database:



```
/// <summary> This function will provide us to spawn a new "SelectablePlayerUI", ...
private void CreateNewCharacterInList(int i)
{
    GameObject newChar = Instantiate(charInListPrefab, CharInListContent); // Create new in list
    CharacterInList newCharInList = newChar.GetComponent<CharacterInList>();
    CharData newCharData = newCharInList.data;

    // Fill char data
    newCharData.ID = int.Parse(InputHelper.GetData(characters[i], "ID:"));
    newCharData.Name = InputHelper.GetData(characters[i], "NAME:");
    newCharData.Gender = int.Parse(InputHelper.GetData(characters[i], "GENDER:"));
    newCharData.Level = int.Parse(InputHelper.GetData(characters[i], "LEVEL:"));
    newCharData.RaceID = int.Parse(InputHelper.GetData(characters[i], "RACE:"));
    newCharData.ClassID = int.Parse(InputHelper.GetData(characters[i], "CLASS:"));
    newCharData.SkinID = int.Parse(InputHelper.GetData(characters[i], "SKIN:"));
    newCharData.HairID = int.Parse(InputHelper.GetData(characters[i], "HAIRTYPE:"));
    newCharData.HairColorID = int.Parse(InputHelper.GetData(characters[i], "HAIRCOLOR:"));
    newCharData.EarringsType = int.Parse(InputHelper.GetData(characters[i], "EARRINGSTYPE:"));
```

The very last thing is to update the ‘**CharacterCreationManager**’ script and make the FORM sent to the script ‘createnewcharacter.php’ to also include the new variable:



```
private IEnumerator ProcessCreateNewCharacter()
{
    WWWForm form = new WWWForm();
    form.AddField("Username", Account.username);
    form.AddField("Name", InputHelper.UppercaseFirst(nameInputField.text));
    form.AddField("Gender", tempData.Gender);
    form.AddField("Level", 1);
    form.AddField("Race", tempData.RaceData.ID);
    form.AddField("Class", tempData.GameClassData.ID);
    form.AddField("Skin", tempData.SkinID);
    form.AddField("Hair", tempData.HairID);
    form.AddField("HairColor", tempData.HairColorID);
    form.AddField("EarringsType", tempData.EarringsType);
```

You’re all done at this point!

You may want to actually make this variable do something like spawn a mesh or modify a colour, just like every other variable acts.

You only have to define what have to happen if that value changes, you already have the variable sent to the database at the moment of creation of the new character and when an existing one has been retrieved.

That means if you code that if ‘EarringsType’ will define mesh spawned on a character all you have to do, like the HairType, is to spawn a mesh while the character creation screen (maybe with UI buttons) and spawn it when the character is retrieved.

I suggest you to read a bit of code specially where the HairType is handled, it’s really about few lines on instantiating meshes.

CONCLUSIONS:

Now you should have learnt all the elements of MMO Accounts & Characters System 2 and you're definitely ready to start building your MMO, after you've practiced a bit with all those elements.

For every type of problem or questions you have regarding the Quests Creation Kit do not hesitate me to send me an email at "silvematt@libero.it", I'll do my best to help you!

I really wish you all the best for your project, thank you for using this tool, I hope it will be a great booster and a resource to learn from!