WordPress LOGIN - Getting Started

Getting started with WordPress login is incredibly easy but once you get over how quickly you integrated the kit into your project, you will soon find that there is a lot of things going on underneath the hood that makes the kit super useful in a variety of ways.

STEP 1: Getting setup

The first thing to do is upload the wuss_login plugin to your website. If you do not know how to do that, please see the read_me document for step by step instructions with pictures to guide you along the way.

Please note that you only have to do this once and then you are all set for all your future games. The only other time you need to update this plugin is when there is an update.

> **NOTE:** It is recommended that you disable all other kits in the WUSS series before you disable/update this kit just to prevent the errors ugly-ing up your interface until you are gone.

Now comes the important part: Before you can use this kit inside of Unity you must first create a game on your website. There is a LOT of things that you can do with the game you create during this step once you are done with it but you can come back to this step as many times as you want later on. For now, all you have to do is:

1. Click on the Wuss Games link under the WUSS dashboard menu
2. Click on "Create New"
3. Enter your game's name into the Title field
4. Hit publish or safe it as a draft for now.

STEP 2: Getting the plugin ready for your project

Generally I tend to make my changes directly to the prefab itself but you are more than welcome to modify the instances after you dropped them in your scene. In order to get the asset ready for your game you need to do the following steps after you drag the relevant prefab into your scene:

1. Most importantly, you have to **tell it what game you are working on** since the plugin you installed on your website will serve all your future games and every one of my assets that you install after the WordPress Login asset will look to your initial setup to determine where to save it's data to. To do this simply click on the "Update Games List" and wait a few seconds for your game to appear in the drop down list (If it doesn't already). Simply select it from the list and this step is done…

2. Equally important is to **tell the prefab where your website can be found**. Unity's internet classes consider it an error if a website sends back a redirect header so to prevent most errors relating to login just make sure that you give the prefab the DIRECT URL to your website. For instance, if your http automatically redirects to https or your root redirects you to the "www" subdomain (or vice versa), give my prefab the final url just to avoid any chance of possible redirection errors.

   In the same vein as the above paragraph, the URL you must give the prefab is the path to where the "wp-content" folder is located. Most WordPress installations install their site into the root of their domain and run from there but WordPress does allow you to install it's files into a separate location than what your URL points to. I.e. your site could run from www.mysite.com but your WordPress installation is located at www.mysite.com/files/wordpress/ . Make sure to give the prefab the path to where you installed your website and do NOT include the trailing slash. An example valid URL would be: https://www.mysite.com

You can use both http and https without worries as long as the one you use doesn't redirect to the other. Also note that multi-sites use a number of different tables to store their data but they still install the files exactly the same as normal sites and thus, no special action is required on your side to make it work with multi-sites.

3. I give you the option of working on a local installation of your game and/or to work directly on your website during production by asking you what the URL is to both your websites. You only need one but I provide both just in case. Along with it I then give you a checkbox to **select which one of the two URL's to use**. Simply tick this box and everything will remain the same except it will work off of another URL. Be sure to repeat step 1 each time you change your mind.

Everything Is ready at this point and you can hit the play button and see the demo scene working with your website. Everything else is entirely optional.

STEP 3: OPTIONAL SETUP

There are a number of settings you can change on the prefab depending on what your needs are . To me, personally, the post method is the most important one since I don't like the idea of using long string containing all my data as my contact method with my website. It is purely a matter of preference but I prefer to use POST rather than GET.

Also highly recommended that you set is the "Security" field. It is entirely optional but highly recommended. With it blank, anyone who buys my kit can do anything they want on your website while they are inside Unity or if they have built a project that uses your website's url. You can set this to any string you choose but there are a few characters that you may not use. These are displayed on the dashboard where you have to enter the same value again so I recommend going to your dashboard and typing in your security string there, first. Save it then copy it and paste it into the prefab.

NOTE: Once you have done this only your game can access your website so I will not be able to give you live support any more unless you remove the security string from your website or give me the string. Also note that the function that fetches your list of games (step 1 from earlier), that requires that the security string be empty inside the prefab when you call it. Once it has completed it's job you can paste the value back again.

Finally, there are the Do_not_destroy_this and Do_not_destroy_canvas options. The former will only work if the WPServer component is on a root object while the latter can also be used while the WPServer is a child of a game object with a Canvas component.

How you design your game is entirely up to you and the use of these two options are most likely going to be of no use to you but they are there just in case they can serve a purpose.

STEP 4: Integration with your game

The Login kit is pretty simple to setup and you can be interacting with your website in minutes but nowhere did you tell the game anything about your specific game or what to do after the user logged in. So now what? How do you make this kit do more than just log a person in to their account and then make them look at a blue screen?

Simple. The kit was designed to make it work with any project WITHOUT you ever having to change a single line of code in the kit. Every time you choose to do something the server will send you a response via an event so all you have to do is hook into that event and then take action whenever it occurs… That simple.

This means you can do anything you want in any game you make and you never have to touch the code that makes it all happen. You just worry about your code and that is that. So to get back to our earlier example of what to do after a person has logged in, well the event you are looking for is

called onLoggedIn. Over time most of the kit has become static and almost everything in WULogin is static nowadays so I is super easy to access anything at any time.

So, let's say for the sake of argument that you are in a scene with this prefab and when he user logs in successfully you want to start the game scene. All you need to do is create a new script and drop it on an empty game object in your scene. Here is an example of what you need to do. Be warned, it is a super hard script that you will have to create if you want to make my kit speak to your kit…

You might want to get some coffee before you continue…
Ready? Here we go:

```
using UnityEngine;
using MBS;
void MyClass : MonoBehaviour {
	void Start() => WULogin.onLoggedIn += StartTheGame;
	void StartTheGame(CML response) => Application.LoadScene("GameScene");
}
```

There we go. So in Start() we tell it to call StartTheGame when the player logs in (or is logged in automatically if he had "Remember me" ticked before). Next, we create the function StartTheGame with a single CML parameter which we don't really need in this example but it is required for all the server responses. In this function we tell it to load our scene and we are done. This script is done.

Now wasn't that hard? …wait a second… a class that contains only 2 lines of code and links my generic login kit with every game you will ever make… maybe not so hard after all. Agreed? It really is as simple as that so now you can flesh it out with extra functionality if you want but this is all it takes to go from my kit to your game…

ABOUT THE EVENTS

Whenever you contact the server the server will ALWAYS send back a response. Either it will send you what you asked for or it will tell you something went wrong.

When everything goes according to plan it will send the results to a function that takes only a single CML object as it's parameter. By convention I call it "response" so here is a valid server response function that can be called when everything went as expected:

```
void ParseTheResults(CML response) => print( response.ToString() );
```

When anything goes wrong (whether you made a typo in your website url or the user typed in a wrong password or even if your website is broken because of a third party plugin and can no longer function) the server will tell you and requires a function with a single cmlData object for a parameter. The actual error message will be contained inside a string called "message" within the server response. Here is a sample function to handle errors:

```
void PrintError(cmlData response) => print( response.String("message") );
```

The events you can register to can be found at the top of WULogin.cs and their names make it rather obvious when you would use them… Let me copy them here so you can see for yourself:

```
static public Action<CML>
    onRegistered,
    onReset,
    onLoggedIn,
    onLoggedOut,
    onAccountInfoReceived,
    onInfoUpdated,
```

```
    onPasswordChanged,
    onGameListFetched,
    onNewGameCreated;

static public Action<cmlData>
    onLoginFailed,
    onLogoutFailed,
    onRegistrationFailed,
    onResetFailed,
    onAccountInfoFetchFailed,
    onInfoUpdateFail,
    onPasswordChangeFail,
    onGameListFetchFailed,
    onNewGameCreationFailed;

static public Action<Texture2D>
    onGravatarTextureFetched;

static public Action<Sprite>
    onGravatarSpriteFetched;
```

ABOUT THE PREFABS

Many people are under the impression that the prefab that I am providing is the prefab you need to use in order to use this kit. This is greatest miss consumption you could possibly make.

The kit started as an OnGUI prefab then evolved to a UGUI prefab. A number of developers have discarded the GUI altogether and use the kit with NGUI instead. Over time the prefab became 4 prefabs and in one demo scene I even demonstrate the use of the kit without any prefab at all.

All the prefabs that I include in the kit are there to give you a head start and help you hit the ground running, nothing more. Since my prefabs are rather full featured and are likely all you will ever need I have taken care to make the kit as easy as possible to skin or theme to your project. For instance, all elements that contain text have the word "text" in their name so that you can search for "text" in the search bar, select everything that is found and then, in one drag of a font to the inspector, the entire prefab now uses a font of your choosing.

This is just one example of how I attempt to help you make use of my prefabs as your one-stop-login-prefab but even so, for all my efforts and good intentions, your game design might simply need something else and my prefabs are not suited. You will be very happy to hear that the code that drives everything you bought when you bought this kit is entirely standalone and 100% separate from the GUI code.

What this means to you is that you can delete all the prefabs I gave you and start from scratch to build the perfect prefab for your setup. All you need to do is derive one of your scripts from WULogin and you are ready to get to work. All the functionality of the Login kit is now 1005 at your disposal so go forth and create away!

There is one other thing to keep in mind, though… This kit can be separated into not 2 standalone components, but 3.

1. The first is WULogin which contains all the login functionality you need to do everything that my prefabs do
2. The GUI code I use for the sample prefabs
3. The WPServer component. This class is the heart of not this kit, but all the kits in this series

ABOUT WPSERVER

WPServer would ideally have been a static class if not for the few fields you needed to set in the inspector during the Project Preparation section listed at the start. All kits in the series require that WordPress Login be installed first but the Login kit and all the others do everything they do through accessing the WPServer class.

It is for this reason that WPServer must always be available in every scene in your game if you intend to contact your website for any reason. Fortunately, even though everything depends on it, as stated before, this is an entirely self contained class so if you want to redesign the gui completely from scratch, just make sure to derive one of your scripts from WULogin and make sure to drop WPServer on a game object and you are good to go.

ABOUT WULogin AND THE SAMPLE PREFAB(s)

WULogin.cs is the heart of the login system and the WUUGLoginGUI.cs is the default sample that shows you what WULogin has to offer. WULogin is the heart of the entire WUSS system which means that some functionality cannot simply be contained within the individual kits themselves because it would overcomplicate life for you, the users thereof.

To that end WULogin is forever growing to include features that become available as you install the relevant kits that contain them. With the demo prefab mirroring what is going on inside WULogin you will often see that after you installed a new WordPress plugin there are now new options available in the prefabs. I will explain those new features per kit in future updates of this supplemental documentation. For now, let's focus on the defaults…

Every WordPress account has some data that it allocates to all it's users. Even odd ones you might never have heard of before like Yim (or is it Jim? I don't know because I've never heard of it before, myself :P). In addition to that WordPress can save an unending number of additional pieces of info as more kits are installed or as the user uses the website. All that info gets stored in the usermeta table.

So the first thing you will see in the prefab is a bunch of tick boxes corresponding to the values in the Users table, followed by 'fetch_roles' which was often requested, fetch_gravatar to turn on or off default loading of the gravatar upon login and finally "fetch_meta_info" in which you can specify absolutely any piece of data that is located in the usermeta table. All the fields you ticked and every piece of info you specified in the array (if found on the website) as well as email and the user's account ID(returned by default) will be returned to you immediately upon successful login.

These values are all available as static values within WULogin so no linking to any files are required to get access to your data. Everything fetched from the user meta table is available to you via WULogin.fetched_info so as long as you have a basic understanding of CML (included in MBSCore and in WordPress Login Standalone) and you know the data type of the field you fetched you can access all your data as int, long, float, Vector2, Vector3, Rect, Color, Quaternion, bool and of course, String… all statically via that one variable.

Following that are options for what type of gravatar to fetch and what size to return the image as. This, like everything else, is available statically from WULogin once fetched.

The remainder of the fields in the default prefab are private and function purely on the prefab itself. As such, if you delete the sample GUI and create your own, all of the will be destroyed along with it.

Optional Extras

1. If you purchased the WordPress Money kit then you will also see an array of strings called "currencies". Since WUMoney allows you to create as many virtual currencies as you see fit per game, here you can tell it what currencies you use and it will fetch the respective balances

during login. You will then also have access to static functions inside WULogin to interact with these balances.

2. If you purchased WordPress Scoring then you will see a field called fetch_highscore. WULogin already knows what field to fetch in the usermeta so all it wants to know from you is whether or not to fetch it during login. Will you want to HOW the high score during the game or doe you only care about scores during the high scores screen? Tick the box and access your high score statically from within WULogin.

3. If you purchased WordPress serials you will see a field called "check_for_serial". If true It will test if the game was legally purchased and if the person has a right to play it. Tick it not and you can do that test manually later via the Serials kit. Secondly you will be asked if a serial is required before a person can play. This allows you to enforce a pay-before-you-play system or allow for demo levels. Finally, it will ask you if you want to retrieve the serial number for display in game, if found. Of course, all of this is available statically inside of WULogin.

PICKING YOUR PREFAB

As I said before, there are now 4 prefabs to choose from where before there was only 1. What's going on? What are all these prefabs?

Well, simply put, as I said before, you can design your game any way you want to and depending on how your game works the default prefab might work just fine or it might not… All the tools are there for you to use but some people find it hard to remove a component from a prefab and add it to a new game object so instead of spending days on support calls trying to explain how to remove a component here and how to add a component there, I just went ahead and did it for you so now yo have choices. It might still not be perfect but it should cover most use cases.

1. Let's start with the original prefab: **WULoginPrefab**
   This prefab contains WPServer and the GUI code on the same prefab. If you have a scene that already contains a canvas and you want to add the login kit to that then this is the prefab for you. When it is active it shows it's screens and when you are playing your game it displays nothing, thus letting you play without knowing it is even there.

   If you are employing a game development choice of building your entire game in a single scene (like some of my clients have in past) then this prefab is perfect for you. Drag it into your existing canvas and you are ready to rock.

   It is on this prefab that it would be useful to click on Do_not_destroy_canvas in case you want to take the canvas with you into other scenes.

2. **WULoginCanvas**
   Some customers started to complain that they don't like having game objects as complex as this prefab permanently loaded in their scene even if they are not showing anything or doing anything at all. The simple solution would be disable the entire prefab when not in use but if they do that then they also disable the WPServer component which will mean everything is broken. The same is more true for deleting the prefab when not in use.

   The simple solution here is to move the WPServer onto the Canvas and remove it from the prefab. This way you can disable the prefab without affecting the WPServer component and simple enable it again whenever you want. Also, now Do_not_destroy_this and Do_not_destroy_canvas can come in handy in you want to take the canvas to another scene with you.

   Even though it is a super simple thing to just remove the WPServer from them prefab and then add WPServer to your canvas, some people simply could not figure out how to do that. So created a prefab to show them:

See? WPServer here. The prefab as a child. There you go

On the down side this now means you have to use my canvas in addition to your own canvas(es) but on the up side, this is the most useful prefab and I tend to use it the most. One drag and I'm done :)

3. **WPServer**
Now, when I design scenes for my clients it makes sense to have the login scene as a separate scene from your main game menu so that you can return to the menu as often as you want without loading a duplicate login prefab. I make login a scene on it's own and then I go to the game menu after successful login… and return to game menu, also, whenever I need to when still logged in. Only when a player signs out do I return to the login scene where they or someone else can sign in again.

In these cases I just use WULoginCanvas but in some cases (like the BadDreams game you can play here: http://guild.site/bad-dreams/) I combine the login prefab and the main menu into one scene since loading a separate scene would break the immersion.

In this case I can't just move WPServer from the prefab onto the canvas like I demonstrated before. Instead, I need to move the WPServer off the canvas entirely and onto it's own game object. This way, when I move to the game scene I can destroy the entire canvas and still have WPServer alive and well. I tend to do this a lot so I created the WPServer prefab for this purpose. Saves me a few clicks and saves me from having to name the game object every time… it's just a tad more convenient for me, personally.

The main problem with this is that even though it was super, super, super simple to just remove WPServer from the canvas and add it to a new game object, the bigger problem here is that I need to keep WPServer alive and do so via Do_not_destroy_this but now every time I return to the scene I get a duplicate WPServer object and I don't want that.

Fortunately, anybody who has any experience with Unity will know that all you need to do is drop this prefab into a new scene which you load right at the start of your game while the Unity logo is still showing. You load all your game objects that must stay alive forever in this scene and just let it load your ACTUAL first scene immediately during Start(). Problem solved. So this is what I did and for this purpose this prefab comes in handy as it saves me that extra 20 seconds of repetitive work every time.

Of course, you COULD drop this prefab in the bootstrap scene and then in the menu scene just remove the WPServer component from WULoginCanvas and you are set… but you could just as easily load WULoginCanvas during Bootstrap instead. That approach, though, requires that I have an empty menu scene and write a controller function to make it show the canvas when the scene loads. It's an option but the former is just far easier… Place WPServer in bootstrap and remove WPServer from WULoginCanvas in my menu scene. Job done.

As I said, I give you all the tools so it just depends on how you design your game. It may work as is or you might need to move the odd script around but now this prefab exists so now you can save 20 seconds per project! :)

4. **WULoginCanvasNoServer**
In the previous example I spoke of how you place WPServer in one scene with Do_not_destroy_this turned on and then in the main menu scene I just remove WPServer from WULoginCanvas. To my great surprise, I found that even experienced developers find it hard to do this. In some cases it has taken me (literally) days of repeatedly saying over and over again "Place WPServer in the bootstrap scene and remove WPServer from the prefab in the menu scene" and still they don't get it.

Eventually I've had to do it for them and email them the sample before they finally understood.
In an effort to save myself days worth of support in future I now present you with the WULoginCanvasNoServer prefab. It is 100% identical to WULoginCanvas except I removed the WPServer component from it.

This prefab and WPServer now go hand in hand: Load one in your bootstrap scene and load the other in your menu scene. All confusion should now be completely removed from the equation.

> **NOTE:** After the surprising discovery of how difficult developers find this process, I created a demo to show how to go from bootstrap to menu to game and back to menu without causing duplication of the object loaded in the bootstrap scene. I also wrote a 150 line article *explaining* it in detail to help even the most green of developers use this kit like a boss. You can find the article and the demo in the demos folder. Look for the MultiSceneUse folder

So there you have it. An in depth explanation of the various prefabs and how they differ from one another. Hopefully one of these will fit your needs but if not, all the tools are there to help you get the perfect setup for your project.

WORDPRSS WEBSITE PORTAL

By installing the wuss_login plugin on your website you gain access to a wealth of widgets and shortcodes that do everything from generate entire pages to just showing basic info about the game or user on your website. You can even display your WebGL games inside your WordPress page using a single shortcode and, as always, everything comes with a host of customization options.

Simplicity and ease of use first… but options galore for the more adventurous.

In addition to all the website related shortcodes, wuss_login also enables the WUSS configuration panel. Every additional wuss kit you install on the site will show it's configuration settings here. Unfortunately there is just way too much to cover here and the content is forever changing so please see the link below…

To learn more about what is on offer please check out the website at:
http://mybadstudios.com/wordpress-systems/wuss-portal/