WordPress User Data Documentation

BACKGROUND INFO

The easiest way to get to grips with WordPress User Data is to understand how it works and in reality it is very, very simple indeed.

WUData is built on top of CML and thus natively supports all the data types supported by CML. This includes:

- Int
- Long
- Float
- Rect
- Color
- Vector2
- Vector3
- Quaternion
- Bool
- String

Data can be stored and then fetched back in any of the above data types simply by calling the relevant function in the cmlData object.

> **NOTE**:
> *CML is included in MBSCore and is thus installed in your project. It is highly recommended that you become familiar with CML as soon as possible as it will open up a lot of extra doors for you. It is used by all my kits so it is a very good idea to get to know it well.*

WUData's purpose is to store your data and to fetch it back. Now you *could* just save every field one at a time and fetch them back one at a time and make many, many, *MANY* calls to the server but WUData offers you a better solution it calls "categories".

More on that next but first there is something very important I need to remind you off. All of the assets in this series always get a response back from the server. The response is either what you expected or it reports that something has gone wrong. You will either get an error or data but in all cases you always get a response. Every function offers you the option of responding to both cases but in mot cases either or both of those responses can be handled or ignored as you see fit. For now, just be aware that everything you do will always yield a response of some kind from the server.

CATEGORIES

Categories allow you to group together your data into whatever collection you choose or you can select to not use a category when you save your data. This data will then be considered part of the "Global" category. As such, all data that gets saved gets saved to a category and with that being the case, all data is saved using the UpdateCategory() function

To create a category and save your data, first create a cmlData object, then populate it with all the data you want to save under one category and finally, just call UpdateCategory and pick a name for your category. Example:

```
void SaveMyData()
{
	cmlData me = new cmlData();
	me.Set("name", "Ryunosuke");
	me.Seti("age', 30);
	me.Set("position", transform.position.ToString());
	me.Set("rotation", transform.rotation);
	WUData.UpdateCategory("Personal", me);
```

```
}
```

…or you could have said: WUData.UpdateCategory("", me);
What you call your category is entirely up to you.

RETRIEVING YOUR DATA

I will get into the various functions at your disposal in just a sec but first I need to give you a little lesson in CML. All your data will be returned in CML and everything you fetch will be returned as one or more categories… which is to say it's gonna be a cmlData object, to put it simply.

Firstly, your data will be returned as a CML file which is nothing more than an array of cmlData objects. The first one will always be a header and is safe to ignore. Whatever you fetch will always be in the second array entry or beyond.
cmlData contains functions to convert your data into any of the formats mentioned on top so all you need to know is:
What is the name of the variable and what type is that data supposed to be? And with that you can do something like this:

```
void ExtractMyDetails(CML response)
{
	cmlData me = response[1];
	string name = me.String("name");
	int age = me.Int("age");
	transform.position = me.Vector("position");
	transform.rotation = me.Quat("rotation");
}
```

So now you have seen how you can save your data and you have seen how you can extract your data once you get it back all that remains now is to learn how to fetch it back.

REQUESTING YOUR DATA IN THE FIRST PLACE

To fetch your data you can make use of any of the following functions:

- **FetchField**
  Specify the category and the field you want to fetch

- **FetchCategory**
  Specify only the category and fetch back all fields in one go

- **FetchCategoryLike**
  Same as FetchCategory except it fetches one or more categories and uses the category you specify as a partial category match. Example: To fetch the "Personal" category could simply specify "ers" or "P" or "al" etc

- **FetchGameInfo**
  Fetches all categories relating to the current game

- **FetchGlobalInfo**
  Fetches all categories stored outside of any specific game. This data can be shared between games. Global info must not be confused with the global category. Every game can have a global category but global data is supposed to be used for things like your pet's name, your favorite book etc. Stuff that stays constant between games

- **FetchEverything**
  As the name suggests, this function fetches back absolutely everything the system has on you. If fetches the global data as well as all categories of all games, sorted per game.

- **StoreImage - NEW**
  Starting from version 4.1 WordPress User Data allows for image uploads. **NOTE:** Images present at design time must be marked as readable in the inspector. The function takes a Texture2D as input to ensure the data being uploaded to your site is in fact an image, asks you if you want to upload it as a JPG or PNG and then saves it to the WordPress default uploads folder under a folder named after the user. Delete it from the dashboard just like any other data.

The format of the data that gets returned depends on which function you called. As mentioned earlier, the first entry is always going to be a header so you can count on that. Next, if you requested one field or one category then the second entry (or NODE, as it's called in CML) will be your data. See the ExtractMyDetails example function in the earlier section and notice that I simply fetched node 1 and started extracting my details from it.

You can do that when you know what you are getting but when you are requesting multiple categories or multiple games you are going to have more than 1 node and must first find the node you want before you can start extracting details from it.

In CML a node is denoted by enclosing it inside < and > and fields inside the node are enclosed within [and].
The CML returned to the ExtractMyDetails function from above would thus have looked something like this:

```
<DATA>
    <_CATEGORY_>
        [category] Personal
        [name] Ryunosuke
        [age] 30
        [position] (0,0,0)
        [rotation] (0,0,0,0)
```

Depending on which function you called the results can differ greatly:

When fetching a single field:

```
<DATA>
    [success] true
    <_CATEGORY_>
        [category] name
        [requested field] value
```

When fetching a single category:

```
<DATA>
    [success] true
    <_CATEGORY_>
        [category] name
        [field 1] value
        [field n] value
        ...
```

When fetching a single game:

```
<DATA>
<_GAME_>
    [gid] value
    <_CATEGORY_>
        [category] name
        [field 1] value
        [field n] value
        ...
    <_CATEGORY_>
```

...

When fetching everything:

```
<DATA>
<_GAME_>
    [gid] value
    <_CATEGORY_>
        [category] name
        [field 1] value
        [field n] value
        ...
    <_CATEGORY_>
        ...
<_GAME_>
    ...
```

CML offers you a plethora of functions to search for and filter content. As long as you are familiar with how the data is returned you should have no problems getting to your data.

QUICKSTART

For those of you who are not familiar with CML and don't want to go learn that first before you can get started with this kit, for you I created wrapper functions that call the core CML functions via easy to remember names. I always recommend to people that they learn CML because of it's immense usefulness (especially since it is the driving force behind all my assets) but if you simply don't want to then the wrapper functions will do the job just fine also.

Please see the WUData_readme.html for an in depth explanation of those functions.

DELETING STUFF AND ERROR HANDLING

You have the following options available to you when it comes to deleting content:

- RemoveField
  Specify the category and the name of the field to remove

- RemoveCategory
  Specify only the category to remove

- RemoveGameInfo
  This removes all categories for the specified game

- DeleteImage - NEW
  This will remove the URL from the table and delete the image from the user's uploads folder

When removing anything the response from the server is only the header node with a bool field called "success".

When this or any function that you called fails the response is not a CML file but instead only a cmlData object.
This object will contain exactly 2 fields:

1. A bool called "success" with is always false
2. A string called "message" which tells you what problem occurred.

ADVANCE USE: SHARED DATA / DEVELOPER FUNCTIONS

Up to this point every function that you called always operated only on the currently logged in player's data. All players data was isolated from one another and nobody had any access to any data of any kind belonging to other players.

A feature that was requested quite often was the ability to have data that you can share between players. This would be useful for creating game settings for example. As soon as you make the

change it is immediately live for all players so if you make the difficulty for level 3 extra hard, everybody gets that same value.

For this reason it makes absolutely no sense at all to let the players modify shared data and this should be considered developer only settings and, as hinted at above, can be used to tweak gameplay post-publishing without requiring people to download the game again.

The functions are identical to the ones described above so if you have become familiar with their use then the use of these will feel very natural and come with no learning curve. There are only 2 things that set the shared functions apart from the functions that operate on user's own data:

1. There is no FetchGlobalInfo or FetchEverything functions since that would make no sense
2. The function names have the word "Shared" in them to make their use clear.

- FetchSharedField
- FetchSharedCategory
- FetchSharedCategoryLike
- FetchAllSharedInfo
- RemoveSharedField
- RemoveSharedCategory
- DeleteSharedImage **- *NEW***
- UpdateSharedCategory
- StoreSharedImage **- *NEW***

AUTHORITATIVE NETWORK UPDATE

For years I have been asked for these functions and for years I flat out refused with the reason being:

There is no reason for any one user to ever need to know anything about another player's data

Recently I tried to help a friend set up an authoritative server and have it make use of WUData. I then realized that at the end of each match each player was responsible for uploading his own high scores and stats and kill ratings etc and I finally found myself agreeing with those people who asked me for this feature all those many times…

As of WordPress User Data v4.0 I now allow you access to other player's data. since it is intended to be used primarily or even exclusively during networked multiplayer games I have dubbed this the Authoritative Network Update functions.

Just like before, the functions are nearly identical to those you use on your own data except not all functions exist (for example, after a match has ended and you need to upload a point to a user there is no reason why you need to fetch all the info of all his games and all his personal info also just to assign him that point… so that function was omitted), just like you used "Shared" in the developer functions you now use "User" instead and, lastly, the first parameter of these functions are now the id of whoever's data you are updating.

- FetchUserField
- FetchUserCategory
- FetchUserCategoryLike
- FetchUserGameInfo
- FetchUserGlobalInfo
- RemoveUserField
- RemoveUserCategory
- DeleteUserImage **- *NEW***
- UpdateUserCategory
- StoreUserImage **- *NEW***

IMAGE HANDLING

Images are uploaded individually, not as a collection like categories, but can still be placed under

categories just like any other data. Every image that gets uploaded stores 2 things:

1. It saves the image (JPG / PNG) under the user's upload folder under the WordPress default uploads folder
2. It saves an entry into the WUData table listing the absolute path to the image

To fetch back your image inside your game is a two step process:

1. Get the URL using the normal FetchField or FetchCategory functions like you would any other info
2. Start a Coroutine and fetch the image using Unity's WWW class.

Example:

```
Texture2D image;
IEnumerator FetchImage(string url)
{
	WWW w = new WWW(url);
	yield return w;
	image = w.texture;
}
```

Because the image is stored on your website as an image, not as encrypted data, it means you can display the image on your website like any other image.

WUDATA PRO : FOR ADVANCED USERS ONLY

If you are familiar with CML at all then you will know how powerful that class really is and what functions you have available to you and how intuitive it makes CML files to work with. If you have that knowledge then working with WUData's very limited node names (only GAME and CATEGORY and that's it….) it might frustrate you when you work with more than one category at a time. It sure as heck frustrates me to no end…

One of the most annoying workarounds I am forced to use with this kit is using:

```
FindNodeWithField("category", "Personal")
```

instead of the proper CML format for finding nodes:

```
FirstNodeOfType("Personal")
```

As of version WordPress User Data v4 there is now a PRO mode included for advanced users. Enabling PRO mode will break all the High Level wrapper functions so use this only if you are comfortable with CML. To enable PRO mode simply set the static bool WUData.WUDataPro to true and from now on all functions will be passed in PRO mode.

What does this mean? This means that all _CATEGORY_ nodes will now be replaced by their proper names and the "category" field will no longer be contained inside of it. Also, categories now come with closing tags just for good measure.

Example normal data:

```
<DATA>
	<_CATEGORY_>
		[category] Personal
		[name] Ryunosuke
		[age] 30
		[position] (0,0,0)
		[rotation] (0,0,0,0)
```

Example PRO data:

```
<DATA>
	<Personal>
		[name] Ryunosuke
```

```
        [age] 30
        [position] (0,0,0)
        [rotation] (0,0,0,0)
    </Personal>
```

IN CONCLUSION

Enjoy :D