Bridge : Data - Saving And Fetching Data

BACKGROUND INFO - CML

Data 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 and more…

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.
Example: if you want to return a variable as an int just use function Int(). To fetch a float just use function Float(), etc.

> **NOTE**:
> *CML is included in the base Bridge asset and should thus already be installed in your project. It is highly recommended that you become familiar with CML as this will expose a lot of additional functionality in our assets and in your own code also.*

The Bridge : Data extension's sole purpose is to store your data online and to fetch it back again. You *could* just save every field one at a time and fetch it back one at a time again making many, many, *MANY* calls to the server but the Bridge : Data extension offers you a better solution in the form of *categories*.

> **NOTE:**
> *Every time the Bridge or any of it's extensions contacts the server it will ALWAYS receive a response from the server. The response will either be what was expected or it will be a notification that something has gone wrong. In either event this leaves you with the option to handle either or both cases (or neither) as you prefer.*

CATEGORIES

Categories group data into a collection for easier retrieval of multiple variables. If you do not group data into a category it will be placed into a category with no name and be labelled "Global data". Since all data gets saved to a category (either with or without a category name) all data is saved using a single function[*1]: UpdateCategory()

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 the same category. Once you have done that, call UpdateCategory and pick a name for your category.

Example:

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

To store this as "Global data" you would have used: WUData.UpdateCategory("", me);
Category names can be anything you like as long as it follows the normal rules for choosing variable

names (i.e. it cannot start with a number or contain special characters etc.)

SAVING YOUR DATA

All data is stored using this single function:

UpdateCategory(string cat, CMLData fields, int gid = -1, Action<CML> response = null, Action<CMLData> errorResponse = null)
Updates a category with the fields provided. If the fields do not exist they are added. The following fields are NOT added or updated as they hold special meaning to the kit: id, gid, cat, wuss, action, unity
cat The name of the category you want to place these variables under **fields** A list of all the fields you wish to update / create along with their values **gid** The id of the game this data belongs to. 0 means global. -1 defaults to the value in WPServer.GameID **response** A function that will receive the response from the server

RETRIEVING YOUR DATA - PRIMER

All your data will be returned as CML which is basically a searchable array of CMLData objects (to put it very simply).
The first entry in every result will always be a header and is safe to ignore. Whatever you fetched from the server will always be in the second array entry or beyond.

CMLData contains functions to convert your data into a variety of data types so all you need to know is:

1. What is the name of the variable inside the category?
2. What data type should that value 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.Vector3("position");
	transform.rotation = me.Quaternion("rotation");
}
```

NOTE:

If you try to fetch data in an invalid format, CMLData will return a default value. For example: "80" can be returned as a string with a value of "80" but trying to return "Neo" as an int will return 0. Trying to return 18 as a Vector3 will return Vector3.zero. Thus, always be sure what type your variables are and retrieve it using the correct function !!

NOTE:

If you try to fetch a variable that doesn't exist, CMLData will return a default value for the selected data type. For example, if a user has no penalties and you try to fetch Int("penalties") it will return

0. If you want to test if a player has completed setup and call Bool("CompletedSetup") before you ever saved that variable it will return false

TIP:

You can skip setting/saving uninitialized fields (like the CompletedSetup example above). In addition, instead of setting a value to it's default value and saving that to your database, just delete the variable from your database instead. CML returns default values rather than throwing exceptions so uninitialized variables are not a problem!

RETRIEVING YOUR DATA - THE FUNCTIONS

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

FetchField(string field_name, string cat = "", Action<CML> response = null, int gid=-1, Action<CMLData> errorResponse = null)
Fetches a single field from the database
field_name The name of the field you wish to retrieve **cat** The name of the category containing this field **response** A function that will receive the response from the server **gid** The id of the game this data belongs to. 0 means global. -1 defaults to the value in WPServer.GameID

FetchCategory(string cat="", Action<CML> response = null, int gid=-1, Action<CMLData> errorResponse = null)
Fetches a single category from the database
cat The name of the category containing this field **response** A function that will receive the response from the server **gid** The id of the game this data belongs to. 0 means global. -1 defaults to the value in WPServer.GameID

FetchGameInfo(Action<CML> response = null, int gid=-1, Action<CMLData> errorResponse = null)
Fetches all categories of the specified game, including it's global category
response A function that will receive the response from the server **gid** The id of the game this data belongs to. 0 means global. -1 defaults to the value in WPServer.GameID

FetchGlobalInfo(Action<CML> response, Action<CMLData> errorResponse = null)

Fetches all categories of the non-game-specific global variables

response
A function that will receive the response from the server

FetchEverything(Action<CML> response, Action<CMLData> errorResponse = null)

Fetches absolutely everything for this player including all global variable categories as well as all info for all games.

response
A function that will receive the response from the server

RemoveField(string field_name, string cat="", Action<CML> response = null, int gid=-1, Action<CMLData> errorResponse = null)

Removes a single field from the database

field_name
The name of the field you wish to remove
cat
The name of the category containing this field
response
A function that will receive the response from the server
gid
The id of the game this data belongs to.
0 means global. -1 defaults to the value in WPServer.GameID

RemoveCategory(string cat="", Action<CML> response = null, int gid=-1, Action<CMLData> errorResponse = null)

Fetches a single field from the database

cat
The name of the category containing this field
response
A function that will receive the response from the server
gid
The id of the game this data belongs to.
0 means global. -1 defaults to the value in WPServer.GameID

RemoveGameData(int gid = -1, Action<CML> response = null, int gid=-1, Action<CMLData> errorResponse = null)

Removes all of the player's data for the game from the database

gid
The id of the game this data belongs to.
0 means global. -1 defaults to the value in WPServer.GameID
response
A function that will receive the response from the server