SIMPLE ONLINE ACCOUNTS KIT DOCUMENTATION

Thank you for taking an interest in the Simple Online Accounts system.

Intro:

I needed a system like this for a long time and when I went looking for it I found there are lots of people like me out there and there is a sea of information and that just confused me even more. More so the fact that there are such a myriad of possible combinations out there... I decided to create a system that is easy to use and just as easy to understand so that it can be easily expanded upon. Not everyone's needs are going to be the same. I have purposely left the real fancy stuff out of this product in an attempt to make it as simple as possible for people inexperienced with MySQL or PHP to pick up and learn. The goal here was to create something that works out of the box that could be easily modified to work the way the customer needed it to...

Requirements:

Hopefully you will have been aware of these limitations before you got the system so this should not be a surprise to you. This system was designed to work with MySQL and PHP. As such, you will need a website that allows you to run PHP scripts. You will also need to have MySQL server setup and running on your website, either by you or by your web host. Once you have PHP and MySQL setup on your website, you are ready to use this package. Unfortunately, getting hold of and setting up PHP and/or MySQL falls outside the scope of this product. Both products are free, though, and many web hosts include it with their packages so it is very likely that you may already have them installed on your website.
Contact your web host if you are not sure.

A word on databases:

This system made a couple of assumptions in terms of database needs and was written accordingly. I will explain what I have done and you can see for yourself if you want to add to that, remove from that or if you agree with my view on things. If you are completely new to working online, I would recommend just creating your databases to match mine as that will make this package real, real easy for you to use.

How to use this system:

Included in this system is 2 prefabs:

AccountPrefab

There is one setting you need to configure. That field is called "Website".
Change this value to point to the location where your PHP files are located on your website.
For instance, http://www.mybadstudios.com.
Take care to start the URL with HTTP and not to end it with /.
That is it. Now simply drag that prefab into your scene or instantiate it when you need it and all the rest is taken care of for you.

AccountUpdate

This prefab is intended to be called during runtime. It will again ask for username and password to make sure some random passerby doesn't get to change a person's account details. This prefab will retrieve a user's personal information and allow them to modify it

and save it back to your database.
This prefab also requires that you setup the Website field with the same value you used in the AccountPrefab prefab.

That is it. That is all you need to do on the Unity end of things. Simple, huh?

Preparations:

Unfortunately it is not that simple to get this entire system to work. You still need to setup the code on the server end. Simply drop all the included PHP files into an appropriate place on your website (the same URL you used in the prefabs). Before you do, though, you will have to open the file 'settings.php' and make a few changes to make it work on your website. This file contains a couple of variables that need to be set. Each one is explained in detail inside the file.

The tables:

To make this system work right out of the box, simply create your tables to have the same fields as mine and you are set to go. This, of course, is not the ideal situation which is why I kept everything as simple as possible so you can change what you want to without much fuss. So first, let me explain what fields I have in my tables and why

(p.s. I am going to write the field names in uppercase to highlight them, but in the actual tables, all fields are all lowercase with the exception of UID which is always uppercase. Please make sure your field names are created accordingly):

Table name: login
Table fields: UID, USERNAME, PASSW, STATUS, DATS, DATA

USERNAME
This is tested to be unique.

UID
This is the key between a user's account and all other data you are going to store.
As such, it must always be unique and the best method to insure that is by making the field automatically increment. I never change this value myself and you have to make sure you set it up to be an auto-increment field also.

STATUS Holds one of 3 values:
0 indicates this account was created but the user has not yet verified their email address
1 indicates this account has been suspended.
I never actually set this value but I prevent login in case this value is set
2 indicates that this account has been verified and is currently active

DATS and DATA
Determines when the account was created and last accessed respectively.
The latter would be a good indication of an inactive user in case you want to start clearing out unused accounts in your tables.

Table name : activate
Table fields : UID, EMAIL

Contains the same corresponding data to the Login file. When a person registers an account, they receive an email that prompts them to either verify or delete their account. Since that info is humanly readable I do not want to send that data directly to the relevant table so I store just those two fields in a separate file. When a person sends the request to cancel or delete the account, I check for that address in this file and then either change the status value inside the Login table or I delete the relevant records in the various tables, as determined by the UID field. This method eliminates the end users from ever knowing the private UID value of their account and it also serves as a method to avoid duplicate verifications etc. As soon as an account is verified or deleted, the entry in Activate is also deleted so either link can only ever be used once.

Table name: account
Table fields: UID, NAME, SURNAME, ADDR1, ADDR2, CITY, STATE, ZIP, COUNTRY, EMAIL, PAYPAL, NOTES

UID, again, this is the link between all tables and uniquely links a person's data to him/her
The rest of the fields are just standard text fields and represent the basics one might like to know about their customers/clients.
I do make NAME, SURNAME and EMAIL required fields during account creation as I do feel that is the minimum info one needs about a customer.
I need the email to get their consent to open the account and the names is merely common curtesy.

Technical:

Once you start using the scripts, you will notice that all you basically have to do on the Unity end is get the data from the user, send it to the PHP scripts and then deal with the response as appropriate. As such, you will most probably come up with new GUI's and ways of getting the info and displaying it, but in case you like the way I do it, allow me to give you a quick overview of what I do.

First, I define the various stages of the account creation and updates etc, as various modes. These could be "Waiting", which simply displays a "Please wait" message on screen while we wait for network messages to travel back and forth, removing any chance of the user clicking the button twice or three times out of impatience, or "Choice" which gives the user the chance to select wether they want to login or create a new account, etc. Each one of these "modes" have different requirements for how much info they will display and thus how much screen space they will need. I thus create a series of Rects in which you define the area you will use. I automatically centre the boxes in the middle of the screen so you only need to enter the width and height information. For example, to set the size of the Choice mode's screen space, look for a variable named choiceArea. The names are rather self explanatory.

Next, there is an array named Info. This array includes the data that will be displayed and saved during account creation or update, depending on the prefab in use. The class used in this array contains a selection of fields.:

_Label is the name that will be displayed next to the the input field.
_Value is the value you want to store in this field
_Field is the name of the relevant field inside the account table. Whatever you enter in _Value will be stored inside _Field in the database
_PasswordChar is the character to display if you want to hide the info that the user enters. Leave it empty otherwise
_Length is the maximum character length this _Value can hold
_FieldType defines the kind of validation tests that will be performed on this field. Values are:
Normal- Text is not validated and accepted as is
Required- This field must be filled in

Email- Certain tests are done to attempt to verify text as a valid email address.
RequiredEmail- This field must be filled in and it must pass the Email test.
Verify- In case one of the fields needs verification like a password or email address, set this value to the index of the field to be verified. Whatever you enter into this field must match the value in the field specified by the array index. Note that this field should NOT contain anything in _Field as all entries that have anything written in that field will have their values sent to the relevant PHP script.

I reckon that is about it... :)

Hope this speeds up your projects.
Have fun