PHP Users Class

I got frustrated trying to find a class or library to authenticate user logins in PHP. The ones I found were either too simplistic, or required me to “join” some project just to have access to purportedly open source software.

So, I decided to dust off the neurons and see if I could build one. I decided to make it as flexible as possible, with only the very basics, but able to be enhanced via data calls. I also decided to make the data access independent of the class itself so data access classes could be written for tasks other than MySQL using the mysqli library.

Because of this, usersDataSource is an abstract class which can not be instantiated. Instead, you must extend the class, defining all of the abstract methods in the abstract. We've done this with the UsersDataSourceMySQLi class.

By itself, the users class (with a data access class usersDataSource like the included UsersDataSourceMySQLi class) handles basic login/logout/editing functions.

The class(es) were, however, designed for extensibility and customization in mind. Some design considerations were made with this in mind.

NOTE: This code uses the ternary and null coelescing shortcuts ?: and ??. I THINK these were introduced in PHP 5.3, but not sure. This code will not work on versions which do not have these shortcuts. See https://www.php.net/manual/en/language.operators.comparison.php

You can get a copy of this from our subversion repository

```
svn co http://svn.dailydata.net/svn/php_users/tags/stable php_users
```

My working copy is at http://svn.dailydata.net/svn/php_users/trunk but I recommend NOT using that as I use trunk as my personal playground and will commit broken code to it regularly

An extension of this basic class which adds boolean permissions is the UsersPermissions class. It is part of the same library.

Basic System

With no modification, the system will store username, password and two booleans, isAdmin and enabled. The default table is created as

```
CREATE OR REPLACE TABLE _users (
   _user_id    INT UNSIGNED NOT NULL AUTO_INCREMENT,
   login       VARCHAR(64), /* the login name */
   password    VARCHAR(128), /* encrypted form of the password */
   isAdmin     BOOLEAN DEFAULT 1, /* if true, user has full admin rights */
   enabled     BOOLEAN DEFAULT 1, /* if false, user can not log in */
   PRIMARY KEY (_user_id)
);
```

- login is filtered to alpha-numerics and the underscore character
- password is stored as a hash using PHP's password_hash
- users with isAdmin set will be able to add/edit other users
- users with enabled set to false (0) will not be able to log in

NOTE: the usersDataSourceMySQLi class has a public function, buildTable, which will build the table, so installation involves simply calling that function if you are using that data source class.

IMPORTANT: to allow the Users class to work with a wide variety of data types, it does no data access itself. It requires a data access class.

Basic use in a script involves instantiating a data access class object, then instantiating a Users class object.

Example:

```
<?php
   include_once( 'UsersDataSourceMySQLi.class.php' );
   include_once( 'Users.class.php' );
   session_start();
   $connection = new UsersDataSourceMySQLi(
         null,
         null,
         array( 'username' => 'test', 'password' => 'test', 'database' =>
'test' )
      );
   //$connection->buildTable( 'admin', 'admin' ); die;
   // ensure we always have a (possibly invalid) instance of user
   if ( ! isset( $_SESSION['user'] ) ) {
      $_SESSION['user'] = new Users( );
   }
?>
<html>
   <body>
      <div class="login">
         <?php
            if ( isset( $_SESSION['user'] ) )
               print $_SESSION['user']->HTML($connection);
         ?>
      </div>
   </body>
</html>
```

This example is using the UsersDataSourceMySQLi definition of data access (included)

If you run it the first time with

```
$connection->buildTable( 'admin', 'admin' ); die;
```

uncommented, it will build the table. Comment that line out on the next run and you will be presented with a login screen.

Class function HTML() displays various things to allow login, then quits displaying anything. Setting $_REQUEST['logout'] = 1 before calling HTML() will initiate a log out which will destroy the session variable

Full Functionality

Calling the class method admin and displaying the repeatedly will generate output allowing the user to change their username, password and any other fields which do not have the 'restrict' attribute set to true

If the user has admin rights set, it will also display a list of logins and allow you to select one to edit or, add a new user. Editing someone else shows all fields, whether or not the 'restrict' attribute is set to true.

CSS

I tried to not put any HTML layout into the code, relying instead on CSS (Thanks, Randell). Everything is supposed to have a class and be wrapped in a with a class. Following are the classes I have in the code (I THINK). * login_field = This is the class of all INPUT fields, and div's surrounding them * login_form = the class for all <form definitions * login_list = the class of the unsigned list (ul) which displays the links to edit other users for admin's * login = NOT in code, but used in example as a div around the div around the login area

Extended Functionality

First, everything is pretty basic. I tried to limit the number of fields to the absolute minimum, but also set it up to allow additional fields to be added programmatically. The example does this.

If you open the class source, you'll find the private member $dbDefinition, which defines everything in the code (I hope). When you create a new instance, you can pass the constructor an array which will be merged with this member, optionally increasing the number and definition of the fields.

WARNING: if you add new fields to the Users class, you must also add them to class usersDataSource. See below

Let's add a new field, say we want to store the users e-mail address. In our PHP, create an array as follows:

```
$customFields = array(
   'tables' => array(
      'users' => array(
         'fields' => array(
            'email' => array(
                  // == For Users class ==
                  // this will be the display label on the form
                  'label'  => 'E-Mail',
                  // the input type to use for data entry
                  'html type' => 'text',
```

```
                    // you can only edit this if an admin and changing
someone
                    // else' record
                    'restrict' => false,
                    // will be displayed on a hover in HTML5 (ie, title=)
                    'instructions' => 'Enter your e-mail address (WARNING,
not verified)',
                    // this is entered in an empty box, ie placeholder=
                    'hint'     => 'E-Mail Address',
                    // a regex to run it against to verify it is ok
                    'filter'   => '/^[-_a-z0-9.]+@[_a-z0-9]+\.[a-
z0-9]+$/i',

                    // == for Data Source ==
                    'dbColumn'  =>  'email',
                    // actual mySQL column type
                    'type'      => 'varchar(64)',
                    // set it to not null if we build the table ourselves
                    'required'  => false
                    )
                )
            )
        )
    );
 ?>
```

Now, when we instantiate a new object of class Users AND class UsersDataSourceMySQLi, we simply pass this array in.

```
    $connection = new UsersDataSourceMySQLi(
        null,
        $customFields,
        array( 'username' => 'test', 'password' => 'test', 'database' =>
'test' )
    );
  if ( ! isset( $_SESSION['user'] ) ) {
     $_SESSION['user'] = new Users( $customFields );
  }
```

Note that since we replicated the basic structure of $dbDefinition in Users and usersDataSource, we can use the same hash to pass into both; they will store, but ignore values not relevant to them.

When the usersDataSource and Users objects are created, $customFields will be merged, with duplicates overwritten by the $customFields value.

This is not limited to adding new columns; you can modify the display definitions also, ie how the information is stored on the screen, to some extent.

New Column Definitions

Note: If a new column is defined with the name (see below) of 'last password change', anytime a password is changed, it will be updated to the current date/time in YYYYMMDDHHMMSS format, which can be directly plugged into MySQL. The code is in there, but to not overpopulate the tables with columns that may not be necessary, I did not include this column in the default table structure.

The structure of a new column only requires a value for

- type
- html type
- dbColumn

Defaults (generally empty strings) will be used for anything else.

key name

The key is used throughout the program to identify what column we are working on. It can be any value that can be used as a PHP hash key.

html type (required)

This determines how the field is displayed and processed during editing. Accepted values are:

- text - standard text input field, corresponds to SQL varchar or char field
- textarea - multi-line strings of arbitrary lenght, corresponds to mysql text field
- boolean - displayed as checkbox, corresponds to mysql bool and/or tinyint (ie, 0 and 1 only)
- password - displayed as a type='password' field, not pre-populated or displayed when typing. Uses varchar(128) in mysql for the length of the hash created

Any value not in the list above will probably result in weird errors.

dbColumn (required)

This is the column name in the database for this field, and all sql uses this value when accessing a column.

type (required for creating tables)

this is a valid MySQL database type used by the buildTable function to create the table, ie varchar, char,

label

If set, will be the label for the input screens. If not defined, the label defaults to the key.

instructions

Replaces the html TITLE attribute for an INPUT, which is displayed by default on a hover. If not set, defaults to an empty string.

hint

Placed in the PLACEHODER attribute for an INPUT. Displayed in an empty text field most of the time. If not set, defaults to an empty string.

restrict

If set to true, will not be displayed when a user is editing their own record (so, not updateable by a user). Examples would be admin and enabled, which would not be something a user should change themselves. If an admin is editing a different user, these fields are available.

filter

If set, this is assumed to be a regular express. The result of the input is checked against the regex. If it does not match the regex, the update is declined and an error message displayed. By default, the username can only be alpha-numeric and an underscore, so the regex '/^[a-zA-Z0-9_]+$/' is set as the filter. If a user puts a period in their password, it will be rejected. Validity of the regex is not checked.

size

For database creation, if set, will create a column (like varchar) with this size, ie size='128' in a varchar field will result in varchar(128)

required

For database creation, sets the NOT NULL attribute to the column

default

For database creation, sets the DEFAULT attribute for the column

usersDataSource

This is our data access class. As stated earlier, it is an abstract class, with UsersDataSourceMySQLi a class built on it.

This code accesses the data (duh), and is consistently called $connection in the Users class. The only requirement is that it must be able to implement the following functions

getPassword($username) returns encrypted password getRecord($username) returns array containing the values for a user getAllUsers() returns an array of all user id's and names getARecord(array of key value pairs to limit what is retrieved) returns all values update(array of key value pairs) returns true/false on success/failure

It is also nice is it can A) handle new columns and B) create and initialize the necessary storage

I separated this out from the Users class because not all programs need database access. For instance, the favorites_urls app uses file based storage, so by writing a new access class for it, we will hopefully be able to get the same functionality, but with a different storage back end.