PHP Users Permissions Class

The Users Permissions Class (UsersPermissions.class.php containing the class UsersPermissions) is an extension to the Users Class which adds a list of boolean values and code to retrieve and maniuplate them.

As with the Users class, there is a separate data access class that is used (usersPermissionsDataSourceMySQLi.php containing the class usersPermissionsDataSourceMySQLi for MySQL).

NOTE: a user with admin rights (Users::isAdmin() == true) has all permissions regardless of what permissions are set in the additional Permissions list. Thus, a user can be promoted to admin, then demoted at a later date and have the same permissions they had before their promotion.

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 we use trunk as our personal playground and will commit broken code to it regularly

Basic System

When administrators edit a user, a list of permissions with their values shows up below the other fields. The admin can turn on/off a permission at this point.

The main function needed is isAuthorized($permission) which returns a true or false depending on the value and whether the user in question is an administrator (admins have full access).

Permissions are a member of a Permission Category, though the category is only used for display purposes. In the Edit screen if permissions are displayed, they are grouped by the Permission Category.

Permissions have a short name (name), display name (description), category and default value. When a user is added, the default value is used to populate the initial screen. Additionally, when a permission is added to the system, all users will receive the default value.

The short name is designed to be used by the calling program, and recommended to be something meaningful programmatically. The short name must be unique across all categories.

Display name is used to label the permissions in the edit screen.

Users and Permissions are joined through the UsersPermissions table in the MySQLi access class along with the current value.

The default structure in the MySQLi access class is:

```
CREATE TABLE _permissions_categories (
   _permissions_categories_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
   name VARCHAR(16), /* display name of the category */
   PRIMARY KEY (_permissions_categories_id)
);

CREATE TABLE _permissions (
   permissions_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   name           VARCHAR(16), /* internal unique name for this programming
*/
   description    VARCHAR(64), /* display name */
   _permission_category_id INT UNSIGNED REFERENCES
_permission_category(_permission_category_id),
   default_value  BOOLEAN, /* default value for any user */
   PRIMARY KEY (permissions_id)
);

CREATE TABLE _users_permissions (
   _users_permissions_id INT UNSIGNED NOT NULL AUTO_INCREMENT, /* not used
*/
   _user_id INT UNSIGNED NOT NULL, /* user's id */
   _permission_id INT UNSIGNED NOT NULL, /* permission id */
   VALUE       BOOLEAN, /* the value for this user */
   CONSTRAINT UNIQUE (_user_id,_permission_id) /* do not allow duplicate
entries */
   PRIMARY KEY (_users_permissions_id)
);
```

Note that all of the above table and column names can be overridden by the calling program by passing a correctly formed $customFields, just like in the Users class.

There is also a view, based on _users, _permissions, _permissions_categories and _users_permissions. It is called (by default) _view_users_permissions and is created as

```
CREATE OR REPLACE VIEW _view_users_permissions AS
   SELECT
      _users._users_id users_id,      /* users id */
      _users.login USER,          /* users login */
      _permissions._permissions_id permission_id, /* permissions id */
      _permissions.name permission,    /* permissions name */
      _permissions.description description,   /* permissions description */
      _permissions_categories.name category,      /* permissions_categories
name */
      ifnull(_users_permissions.value,0)  VALUE /* actual value */
   FROM
      _users /* users */
      JOIN _permissions   /* permissions (permissions_id */
      LEFT JOIN _users_permissions USING (_user_id,_permission_id)   /*
users_permissions (users_id) */
      JOIN _permissions_categories USING (_permissions_categories_id)   /*
permissions_categories( permissions_categories_id) */"
```

By using a left join into _users_permissions, a false can be indicated by a value of 0, or by a lack of an entry, allowing us to not actually enter a row for a false value.

NOTE: the usersDataSource class has a public function, buildTable, which will build the table, so installation involves simply calling that function. This correctly adds the new tables.

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

Example, assuming your Users instance is stored in $_SESSION['user'] and you want to build a menu from an array containing menu names as the key and the permission name as the value:

```
<div class='menu'>
   <?php
      print '<ul class="menu">';
      foreach ( $menu as $display => $permissionName ) {
         if ( $_SESSION['user']->isAuthorized( $permissionName ) ) P
            print "<li>$display</li>";
         } // if
      } // foreach
      print '</ul>';
   ?>
</div>
```

Full Functionality

Your software project can dynamically add/remove permissions. An example would be a base system which adds/removed modules. When a module is activated, the activation code can dynamically add new permissions (probably in a new permissions category). To add a new permission to allow a user to see the menu option for something called 'New Module 1', you might do something like this.

```
   // allow access to the menu by default
   $_SESSION['user']->addPermission ( $connection, 'New Module 1',
'mnuMod1', 'Show Menu', 1 );
   // allow users to edit an existing entry by default
   $_SESSION['user']->addPermission ( $connection, 'New Module 1',
'newmod1_edit', 'Edit Entry', 1 );
   // do not allow users to create a new entry unless they are an admin
   $_SESSION['user']->addPermission ( $connection, 'New Module 1',
'newmod1_add', 'Create Entry', 0 );
   // do not allow users to delete an entry unless they are an admin
   $_SESSION['user']->addPermission ( $connection, 'New Module 1',
'newmod1_del', 'Delete Entry', 0 );
```

This would create four permissions in the category 'New Module 1' and set all existing users to the default permission. It will create the category 'New Module 1' on the first call if it doesn't already exist.

NOTE: In some cases, you might want the first permission (Show Menu) to actually be in an existing category, maybe 'Menu'. Again, the categories are only there to make it easier to find a permission if

your list grows.

CSS

Again, we are using CSS for all formatting. In this case, each category of permissions will be in a div of class 'category'. Inside that div will be an <h3> with the name of the category, and each permission is a checkbox.