User Authentication With patUser (part 2)

By icarus
2003-05-01

Printed from DevShed.com
URL: http://www.devshed.com/Server_Side/PHP/patUser2/

Onwards and Upwards

In the first part of this article, I focused almost entirely on how patUser could simplify the task of adding authentication to your site. I explained the patUser database schema, ran you through the process of initializing a patUser object instance and linking it to a database and template engine, and showed you how built-in patUser methods could simplify the task of verifying user credentials and writing login and logout scripts.

However, patUser can do a lot more than just handle user authentication - the library also comes with a large number of methods designed to make the task of managing user data as simple and efficient as possible. Over the course of this second installment, I'm going to show you how these methods work, and how they can be used to quickly create scripts to view, add, edit and delete users (and user attributes) from your database. Keep reading!

Meeting The Family

Let's start with the basics - obtaining a list of current users from the database. If you've been following along, you already know that right now, the database has only a single user, "joe". Let's add a couple more:

```
INSERT INTO users ('uid', 'username', 'passwd') VALUES
('', 'sarah', 'sarah');
INSERT INTO users ('uid', 'username', 'passwd') VALUES ('', 'john',
'john');
```

You can verify the result with a quick SQL query to the "users" table:

```
mysql> SELECT uid, username FROM users;
+-----+----------+
| uid | username |
+-----+----------+
|   1 | joe      |
|   2 | sarah    |
|   3 | john     |
+-----+----------+
3 rows in set (0.00 sec)
```

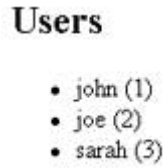
Now, let's try doing the same with patUser:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");
```

```
// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setAuthTable("users");

// get user list
$list = $u->getUsers(array("uid", "username"));

// uncomment this to see data structure
// print_r($list);

// print as list
echo "<h2>Users</h2>";
echo "<ul>";
foreach ($list as $l)
{
        echo "<li>" . $l['username'] . " (" . $l['uid'] . ")";
}
echo "</ul>";

?>
```

The first part of this script should be easily recognizable to you by now - it consists of the code needed to instantiate a patUser object instance and prepare it for use. Once the object has been prepared, the getUsers() function is used to retrieve a list of all the users in the database. The return value of the getUsers() method is an associative array containing user records, one element for each record.

The first argument to the getUsers() function is an array containing the names of the fields to be included in the result set - these field names appear as keys in the returned array, and can be used to access the corresponding values. Here's the output:



You're not restricted to using just the default fields built into the patUser database schema - you can just as easily add your own. Find out how, on the next page.

Asking For More

Now, let's make some additions to the patUser-provided schema for the "users" table. You might remember that the original schema looked like this:

```
#
# Table structure for table 'users'
```

```
#

CREATE TABLE users (
  uid int(10) unsigned NOT NULL auto_increment,
  username varchar(20) NOT NULL default '',
  passwd varchar(20) NOT NULL default '',
  email varchar(200) default NULL,
  nologin tinyint(1) NOT NULL default '0',
  first_login datetime default NULL,
  last_login datetime default NULL,
  count_logins int(10) unsigned NOT NULL default '0',
  count_pages int(10) unsigned NOT NULL default '0',
  time_online int(11) NOT NULL default '0',
  PRIMARY KEY  (uid),
  KEY username (username)
) TYPE=MyISAM;
```

Let's alter this to include some additional fields, for age, sex and telephone number:

```
CREATE TABLE users (
  uid int(10) unsigned NOT NULL auto_increment,
  username varchar(20) NOT NULL default '',
  passwd varchar(20) NOT NULL default '',
  email varchar(200) default NULL,
  age tinyint(4) NOT NULL default '0',
  sex char(1) NOT NULL default '',
  tel varchar(50) NOT NULL default '',
  nologin tinyint(1) NOT NULL default '0',
  first_login datetime default NULL,
  last_login datetime default NULL,
  count_logins int(10) unsigned NOT NULL default '0',
  count_pages int(10) unsigned NOT NULL default '0',
  time_online int(11) NOT NULL default '0',
  PRIMARY KEY  (uid),
  KEY username (username)
) TYPE=MyISAM;
```

While we're at it, let's also insert a few records into the new table:

```
INSERT INTO users (uid, username, passwd, email, age, sex, tel) VALUES
(1,
'joe', 'joe', NULL, 19, 'M', '123 4567'); INSERT INTO users (uid,
username,
passwd, email, age, sex, tel) VALUES (2, 'sarah', 'sarah', NULL, 35,
'F',
'543 18238'); INSERT INTO users (uid, username, passwd, email, age,
sex,
tel) VALUES (3, 'john', 'john', 'john@some.domain.com', 22, 'M', '853
2377'); INSERT INTO users (uid, username, passwd, email, age, sex, tel)
VALUES (4, 'william', 'william', 'william@somewhere.com', 31, 'M', '123
2372');
```

As before, I can retrieve this information via an SQL query,

```
mysql> SELECT uid, username, age, sex, tel FROM users;
+-----+----------+-----+-----+-----------+
| uid | username | age | sex | tel       |
```

```
+-----+----------+-----+-----+-----------+
|   1 | joe      |  19 | M   | 123 4567  |
|   2 | sarah    |  35 | F   | 543 18238 |
|   3 | john     |  22 | M   | 853 2377  |
|   4 | william  |  31 | M   | 123 2372  |
+-----+----------+-----+-----+-----------+
4 rows in set (0.00 sec)
```

---

or have patUser do it for me via its getUsers() function:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setAuthTable("users");

// get user list
$list = $u->getUsers(array("uid", "username", "age", "sex", "tel"));

// print as table
echo "<h2>Users</h2>";
echo "<table border=1>";
echo "<tr>";
echo "<td><b>UID</b></td>";
echo "<td><b>Username</b></td>";
echo "<td><b>Age</b></td>";
echo "<td><b>Sex</b></td>";
echo "<td><b>Tel</b></td>";
echo "</tr>";

// iterate over list
foreach ($list as $l)
{
        echo "<tr>";
        echo "<td>" . $l['uid'] . "</td>";
        echo "<td>" . $l['username'] . "</td>";
        echo "<td>" . $l['age'] . "</td>";
        echo "<td>" . $l['sex'] . "</td>";
        echo "<td>" . $l['tel'] . "</td>";
        echo "</tr>";
}

echo "</table>";
?>
```

In this case, I've told getUsers() to retrieve a few extra fields from the database - specifically, the "age", "sex" and "tel" fields. You'll remember these are not standard patUser fields, but have been added by me for illustrative purposes.

Here's the output:

Users

UID	Username	Age	Sex	Tel
1	joe	19	M	123 4567
2	sarah	35	F	543 18238
3	john	22	M	853 2377
4	william	31	M	123 2372

Drilling Deeper
You can also get fancy by adding selection criteria in your call to getUsers(), as a second argument to the method. Consider the following variant of the example above, which only returns records of those users who are male and between 18 and 25:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setAuthTable("users");

// get user list
$list = $u->getUsers(
                array("uid", "username", "age", "sex", "tel"),
                array(  array("field" => "sex", "value" => "M", "match"
=>
"exact"),
                array("field" => "age", "value" => array(18,25), "match"
=>
"between") ) );

// print as table
echo "<h2>Users</h2>";
echo "<table border=1>";
echo "<tr>";
echo "<td><b>UID</b></td>";
echo "<td><b>Username</b></td>";
```

```
echo "<td><b>Age</b></td>";
echo "<td><b>Sex</b></td>";
echo "<td><b>Tel</b></td>";
echo "</tr>";

// iterate over list
foreach ($list as $l)
{
        echo "<tr>";
        echo "<td>" . $l['uid'] . "</td>";
        echo "<td>" . $l['username'] . "</td>";
        echo "<td>" . $l['age'] . "</td>";
        echo "<td>" . $l['sex'] . "</td>";
        echo "<td>" . $l['tel'] . "</td>";
        echo "</tr>";
}

echo "</table>";

?>
```

In this case, the second argument to the getUsers() method - a series of nested arrays containing selection criteria - provides patUser with a list of additional conditions to be satisfied when querying the database. Only those user records that match the specified conditions will be included in the result.

Here's the output:

Users

UID	Username	Age	Sex	Tel
1	joe	19	M	123 4567
3	john	22	M	853 2377

All For One, And One For All

patUser also allows you to organize users into named collections or groups. You might remember, from the discussion of the patUser database schema in the previous segment of this article, that there are two tables designed specifically for this task, the "groups" table (which maintains a list of available groups) and the "usergroups" table (which maintains a list of which users belong to which group).

Here's the schema for the "groups" tables:

```
#
# Table structure for table 'groups'
#

CREATE TABLE groups (
   gid int(11) NOT NULL auto_increment,
   name varchar(50),
   UNIQUE gid (gid)
```

```
);
```

Let's now add a couple of groups to the system,

```
INSERT INTO groups (gid, name) VALUES (1, 'Accounts');
INSERT INTO groups (gid, name) VALUES (2, 'Administration'); INSERT
INTO
groups (gid, name) VALUES (3, 'Operations');
```

and verify that they have been added correctly via an SQL query:

```
mysql> SELECT * FROM groups;
+-----+----------------+
| gid | name           |
+-----+----------------+
|   1 | Accounts       |
|   2 | Administration |
|   3 | Operations     |
+-----+----------------+
3 rows in set (0.04 sec)
```

Obviously, we can perform the same task using patUser's getGroups() function, which works in much the same way as the getUsers() function discussed on the previous page. The following code listing demonstrates:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set tables
$u->setAuthTable("users");
$u->setGroupTable("groups");

// get group list
$list = $u->getGroups(array("gid", "name"));

// uncomment this to see data structure
// print_r($list);

// print as list
echo "<h2>Groups</h2>";
echo "<ul>";
foreach ($list as $l)
{
        echo "<li>" . $l['name'] . " (" . $l['gid'] . ")";
```

```
}
echo "</ul>";

?>
```

Here's the output:

Note the introduction of a new method in the script above, setGroupTable(). This method, together with the setGroupFields()method, allows you to configure patUser to use a different set of tables than its default by remapping the default table and column references to custom values.

Accounting For Change
The default patUser-supplied database schema for the "groups" table includes only two group attributes: its name, and a unique group ID. Most of the time, this is sufficient; however, in case you need to add more attributes, patUser can be easily tweaked to work with this extra data.

Let's alter the default "groups" table to include some additional fields, for description and location:

```
#
# Table structure for table `groups`
#

CREATE TABLE groups (
  gid int(11) NOT NULL auto_increment,
  name varchar(50) default NULL,
  dsc varchar(255) NOT NULL default '',
  loc varchar(255) NOT NULL default '',
  UNIQUE KEY gid (gid)
) TYPE=MyISAM;
```

While we're at it, let's also insert a few records into the new table:

```
INSERT INTO groups (gid, name, dsc, loc) VALUES (1, 'Accounts',
'Billing and
finance activities', 'AZ'); INSERT INTO groups (gid, name, dsc, loc)
VALUES
(2, 'Administration', 'Administrative activities', 'CA'); INSERT INTO
groups
(gid, name, dsc, loc) VALUES (3, 'Operations', 'Logistics and
operations',
'CA');
```

As before, I can retrieve this information via an SQL query,

```
mysql> SELECT * FROM groups;
+-----+----------------+---------------------------------+-----+
| gid | name           | dsc                             | loc |
+-----+----------------+---------------------------------+-----+
|   1 | Accounts       | Billing and finance activities | AZ  |
```

```
|   2 | Administration | Administrative activities      | CA  |
|   3 | Operations     | Logistics and operations       | CA  |
+-----+----------------+--------------------------------+-----+
3 rows in set (0.00 sec)
```

---

or have patUser do it for me via its getGroups() function:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set tables
$u->setAuthTable("users");
$u->setGroupTable("groups");

// get group list
$list = $u->getGroups(array("gid", "name", "dsc", "loc"));

// print as table
echo "<h2>Groups</h2>";
echo "<table border=1>";
echo "<tr>";
echo "<td><b>GID</b></td>";
echo "<td><b>Name</b></td>";
echo "<td><b>Description</b></td>";
echo "<td><b>Location</b></td>";
echo "</tr>";

// iterate over list
foreach ($list as $l)
{
        echo "<tr>";
        echo "<td>" . $l['gid'] . "</td>";
        echo "<td>" . $l['name'] . "</td>";
        echo "<td>" . $l['dsc'] . "</td>";
        echo "<td>" . $l['loc'] . "</td>";
        echo "</tr>";
}

echo "</table>";

?>
```

California Calling

You can also apply selection criteria in your call to getGroups(), so as to return only a specific subset of records. Consider the following variant of the example above, which only returns those groups located in California:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set tables
$u->setAuthTable("users");
$u->setGroupTable("groups");

// get group list
$list = $u->getGroups(
                      array("gid", "name", "dsc", "loc"),
                      array(  array("field" => "loc", "value" => "CA",
"match" => "exact")  )
               );

// print as table
echo "<h2>Groups</h2>";
echo "<table border=1>";
echo "<tr>";
echo "<td><b>GID</b></td>";
echo "<td><b>Name</b></td>";
echo "<td><b>Description</b></td>";
echo "<td><b>Location</b></td>";
echo "</tr>";

// iterate over list
foreach ($list as $l)
{
        echo "<tr>";
        echo "<td>" . $l['gid'] . "</td>";
        echo "<td>" . $l['name'] . "</td>";
        echo "<td>" . $l['dsc'] . "</td>";
        echo "<td>" . $l['loc'] . "</td>";
        echo "</tr>";
}

echo "</table>";

?>
```

Here's the output:

Groups

GID	Name	Description	Location
2	Administration	Administrative activities	CA
3	Operations	Logistics and operations	CA

Making New Friends
It's just as simple to add new users and groups to the system - all you need to do is use patUser's addUser() and addGroup() methods, which accept an array of field-value pairs and uses them to create a new user or group record in the database. Both methods return the ID of the newly-created user or group if successful. Consider the following example, which demonstrates how to add new users:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setAuthTable("users");

// add user
$uid = $u->addUser( array("username" => "tom", "passwd" => "tom") );

// check to see if user added and display message
if ($uid)
{
        echo "User successfully added with UID $uid";
}
else
{
        echo "User could not be added!";
}

?>
```

By default, patUser automatically logs the newly-created user into the system. You can avoid this by specifying a second argument to addUser(), as in the example below:

```
<?php
```

```
$u->addUser( array("username" => "tom", "passwd" => "tom"), false );

?>
```

In a similar manner, a new group may be added - here's how:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setGroupTable("groups");

// add group
$gid = $u->addGroup( array("name" => "Human Resources") );

// check to see if group added and display message
if ($gid)
{
        echo "Group successfully added with GID $gid";
}
else
{
        echo "Group could not be added!";
}

?>
```

A Fast Edit

Once a user is entered into the system, patUser allows you to add new data to the user record, or make changes to the existing data, with its modifyUser() method. This method accepts two primary arguments: an array containing the data to be inserted, and an array containing the user ID of the record to edited and related options.

Consider the following example, which illustrates the process of editing a user record and entering new information into it:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");
```

```
// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setAuthTable("users");

// modify record
$u->modifyUser( array (
                        "username" => "tom",
                        "passwd" => "tom",
                        "email" => "tom@some.domain.com",
                        "age" => 31,
                        "sex" => "M",
                        "tel" => "759 3539"
                        ),

                        array (
                        "mode" => "update",
                        "uid" => 15
                        )
               );

?>
```

Note that in the example above, all changes will be made to the record with user ID 15. If no user ID is provided, the currently logged-in user's ID is used instead.

You can also modify group data with the corresponding modifyGroup() method. I'll leave that to you to experiment with.

Here Today, Gone Tomorrow
Just as you can add and edit users, patUser() also comes with - obviously! - a deleteUser() method. This method accepts, as input, a user ID, and removes the corresponding user record from the database. If no user ID is provided, the currently logged-in user's ID is used instead. The following example demonstrates:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);
```

```
// set table
$u->setAuthTable("users");

// delete user
$u->deleteUser( array ("uid" => 15) );

?>
```

Similarly, the deleteGroup() method allows you to remove a group from the group database.

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setGroupTable("groups");

// delete group
$u->deleteGroup( array("gid" => 4) );

?>
```

Connecting The Dots
Once you've got your users and groups created, you can begin organizing users into groups, via the addUserToGroup() and removeUserFromGroup() methods.

Given the following users and groups,

```
mysql> SELECT uid, username FROM users;
+-----+----------+
| uid | username |
+-----+----------+
|   2 | joe      |
|   3 | sarah    |
|   4 | john     |
|   5 | tom      |
+-----+----------+
4 rows in set (0.00 sec)

mysql> SELECT gid, name FROM groups;
+-----+-----------------+
| gid | name            |
+-----+-----------------+
```

```
|   1 | Accounts        |
|   2 | Administration  |
|   3 | Operations      |
|   4 | Human Resources |
+-----+-----------------+
4 rows in set (0.00 sec)
```

---

it's fairly easy to, say, add "joe" and "sarah" to the "Accounts" and "Operations" group,

---

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set table
$u->setAuthTable("users");
$u->setGroupTable("groups"); $u->setGroupRelTable("usergroups");

// add joe to Accounts
$u->addUserToGroup( array("uid" => 2, "gid" => 1) );

// add joe to Operations
$u->addUserToGroup( array("uid" => 2, "gid" => 3) );

// add sarah to Accounts
$u->addUserToGroup( array("uid" => 3, "gid" => 1) );

// add sarah to Operations
$u->addUserToGroup( array("uid" => 3, "gid" => 3) );

?>
```

---

or remove "sarah" from "Operations" and move her into "Human Reources".

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
```

```
$u->setAuthDbc($db);

// set tables
$u->setAuthTable("users");
$u->setGroupTable("groups"); $u->setGroupRelTable("usergroups");

// remove sarah from Operations
$u->removeUserFromGroup( array("uid" => 3, "gid" => 3) );

// add sarah to Human Resources
$u->addUserToGroup( array("uid" => 3, "gid" => 4) );

?>
```

Once you've got your users organized the way you want them, patUser also offers the following three utility functions to help you make sense of all the relationships:

getJoinedGroups() - returns a list of the groups the named user belongs to, accepts user ID as input

getUsersInGroup() - returns a list of all the users in a named group, accepts group ID and list of required user attributes as input

isMemberOfGroup() - returns a Boolean value indicating whether or not the named user belongs to the named group, accepts user ID and group ID as input

The following example illustrates how these work:

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database
$u->setAuthDbc($db);

// set tables
$u->setAuthTable("users");
$u->setGroupTable("groups"); $u->setGroupRelTable("usergroups");

// add joe to Accounts
$u->addUserToGroup( array("uid" => 2, "gid" => 1) );

// add joe to Operations
$u->addUserToGroup( array("uid" => 2, "gid" => 3) );

// add sarah to Human Resources
$u->addUserToGroup( array("uid" => 3, "gid" => 4) );
```

```
// add john to Operations
$u->addUserToGroup( array("uid" => 4, "gid" => 3) );


// which groups is joe a member of?
$foo = $u->getJoinedGroups( array("uid" => 2) );
print_r($foo);


// who belongs to the Operations group?
$foo = $u->getUsersInGroup( array("uid", "username"), array("gid" => 3)
);
print_r($foo);


// does joe belong to Operations?
// returns Boolean true (1)
$foo = $u->isMemberOfGroup(2, 3);
print_r($foo);

?>
```

Here's the output:

```
Array
(
    [0] => Array
        (
            [gid] => 1
            [name] => Accounts
        )

    [1] => Array
        (
            [gid] => 3
            [name] => Operations
        )

)

Array
(
    [0] => Array
        (
            [uid] => 2
            [username] => joe
        )

    [1] => Array
        (
            [uid] => 4
            [username] => john
        )

)

1
```

Needless to say, these utility functions come in very handy when you need to find out which users belong to which groups, or if you need to alter the various user and group configurations. You'll see this in action in the composite example on the next page.

A Well-Formed Plan

Now, how about a couple of examples to put all this in context? This first example demonstrates how the various user and group manipulation methods discussed in this article can be used to rapidly build a user administration module for a Web application or Web site. Consider the following script, which is designed to allow administrators to add new users to the system.

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize patUser
$u = new patUser(true);

// connect patUser to database/template engines $u->setAuthDbc($db);

// set tables
$u->setAuthTable("users");
$u->setGroupTable("groups"); $u->setGroupRelTable("usergroups");

?>

<html>
<head>
<basefont face="Arial">
</head>

<body>

<?php

// display initial form
if (!$_POST['submit'])
{
?>

        <h2>New User</h2>
        <form action="<?=$ME?>" method="post">

        First name
        <br>
        <input type="text" name="fname" size="10">

        <p>

        Last name
        <br>
```

```
        <input type="text" name="lname" size="10">

        <p>

        Username
        <br>
        <input type="text" name="username" size="10">

        <p>

        Password
        <br>
        <input type="password" name="passwd" size="10">

        <p>

        Email address
        <br>
        <input type="text" name="email" size="25">

        <p>

        Department
        <br>
        <select name="gid[]" multiple>

<?php
        // get group list
        // display as multi-select box
        $groups = $u->getGroups( array("gid", "name") );
        foreach ($groups as $g)
        {
?>
        <option value="<?=$g['gid']?>"><?=$g['name']?></option>
<?php
        }
?>
        </select>

        <p>
        <input type="submit" name="submit" value="submit">

        </form>

<?
}
else
{
        // if form submitted
        // validate form data

        // ideally you would want more stringent validation rules here!
        if (!$_POST['fname']) { echo "First name not entered!"; die; }

        if (!$_POST['lname']) { echo "Last name not entered!"; die; }

        if (!$_POST['username']) { echo "Username not entered!"; die; }
```

```
        if (!$_POST['passwd']) { echo "Password not entered!"; die; }

        if (!$_POST['email']) { echo "Email address not entered!"; die;
}

        if (sizeof($_POST['gid']) == 0) { echo "Department not
selected!";
die; }

        // if data OK, add user
        $uid = $u->addUser( array(
                                "username" => $_POST['username'],
                                "passwd" => $_POST['passwd']
                                ) );

        // get UID
        if ($uid)
        {
                // add other user data
                $u->modifyUser( array(
                                "username" => $_POST['username'],
                                "passwd" => $_POST['passwd'],
                                "fname" => $_POST['fname'],
                                "lname" => $_POST['lname'],
                                "email" => $_POST['email'],
                                                ) );

                // add user to groups
                foreach ($_POST['gid'] as $g)
                {
                        $u->addUserToGroup( array("uid" => $uid, "gid" =>
$g) );
                }

                // display status
                echo "User successfully added!";

        }
        else
        {
                // else display error
                echo "User could not be added!";
        }

}
?>

</body>
</html>
```

This script is split into two parts. The first part is a simple HTML form containing fields for user information (including the critical username and password fields) and group membership. The list of groups is obtained from the system itself, via a call to the getGroups() method. An administrator may fill up this form with the components of a user record, and also attach a user to one or more of the named groups.

Once the form has been submitted and the data within it validated, the addUser() method is invoked to add the user to the system. The unique user ID returned by the addUser() method can then be used by the modifyUser() and addUserToGroup() methods to set up the rest of the user record and group memberships.

You'll notice that, unlike traditional scripts of this nature, there are no SQL queries in the code above. This is because patUser does most of the heavy lifting for you, encapsulating all needed queries within its user management API; all you need to do is invoke the appropriate method and pass it the data that needs to be entered into the database. Integrating patUser, therefore, can substantially reduce the time you spend on building such utility scripts.

Slice And Dice

How about one more? Once the user has been registered in the system (perhaps using a script like the one on the previous page), it becomes possible to slice and dice that user information for a variety of different purposes. In this next script, different sections of a single Web page are hidden or displayed on the basis of a user's credentials and group membership.

```
<?php

// include classes
include("../include/patDbc.php");
include("../include/patUser.php");
include("../include/patTemplate.php");

// initialize database layer
$db = new patMySqlDbc("localhost", "db211", "us111", "secret");

// initialize template engine
$tmpl = new patTemplate();
$tmpl->setBasedir("../templates");

// initialize patUser
$u = new patUser(true);

// connect patUser to database/template engines
$u->setAuthDbc($db);
$u->setTemplate($tmpl);

// set tables
$u->setAuthTable("users");
$u->setGroupTable("groups");
$u->setGroupRelTable("usergroups");

// check authentication
$u->requireAuthentication("displayLogin");

?>

<html>
<head>
<basefont face="Arial">
</head>

<body>
```

```
<!-- this section only displayed to authenticated users --> <hr> You
have
been authenticated. Welcome.

<!-- this section only displayed to members of the Operations group -->
<?php

// get GID for Operations group
$data = $u->getGroups(array( "gid" ),
                      array( array( "field" => "name",
                                    "value" => "Operations",
                                    "match" => "contains" ) ) );
$opsGid = $data[0]['gid'];

// check to see if user is member and display section if so
if ($u->isMemberOfGroup($u->getUid(), $opsGid))
{
?>

<hr>
Latest news from Operations: All systems up and running normally.

<?php
}

// get GID for Administration group
$data = $u->getGroups(array( "gid" ),
                      array( array( "field" => "name",
                                    "value" => "Administration",
                                    "match" => "contains" ) ) );
$adminGid = $data[0]['gid'];

// check to see if user is member of both Administration and Operations
// and display page if so
if ($u->isMemberOfGroup($u->getUid(), $opsGid) &&
$u->isMemberOfGroup($u->getUid(), $adminGid)) { ?>

<!-- this section only displayed to members of both Operations and
Administrations group --> <hr> <a href="#">Click here to modify system
configuration settings.</a>

<?php
}
?>

</body>
</html>
```

Most of this should be fairly easy to understand. The page is divided into three sections, with the first one available to all authenticated users and the remaining two turned on only if the user is a member of the appropriate groups. The isMemberOfGroup() method discussed on the previous page is used to test whether the user belongs to the group or not, while the the getGroups() method is used, this time with additional selection criteria, to obtain the group ID of the various groups involved.

Business logic similar to that above can be used to create Web pages that are sensitive to user

credentials and privileges, and that can dynamically change so that only the appropriate information is presented to each user. And with patUser again doing most of the work, adding this business logic to a Web page is a snap.

And that's about it for the moment. This article was a little longer than the previous one, but it also covered a lot more ground. You should now have a better understanding of patUser's concepts of users, groups and group membership, and of the user and group management API available in the patUser library. In this article, I demonstrated most of the important components of this API, showing you how to list, add, edit and modify users and groups, and how to organize users into groups. Finally, I wrapped things up with two examples of how these API calls can be used in real-world situations: a script for administrators to add new users via a Web-based interface, and a Web page that altered its visible content based on the logged-in user's permissions.

In the third (and concluding) article in this series, I will be briefly looking at a number of hard-to-categorize-yet-very-useful methods in the patUser library. These include methods to handle errors, track user movement, collect statistics and identify users and groups using different criteria. Make sure you don't miss that one!

Note: Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!

This article copyright Melonfire 2000-2002. All rights reserved.