Creating A Simplified Facebook Page Using PHP & MySQL

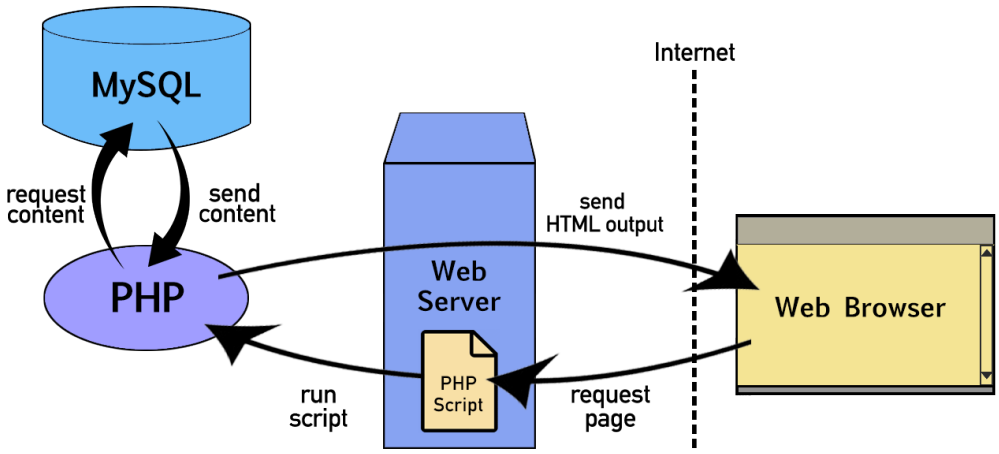
By Chad Jordan – April 22nd, 2009

Introduction

In this guide you will learn:

1. A brief overview and the general syntactic characteristics of PHP v5.2.9
2. How primitives, operations, and expressions work in PHP
3. File structure, form handling, and datatypes in PHP
4. A back-end implementation for building a simplified Facebook site using the vanilla coding style of PHP and MySQL

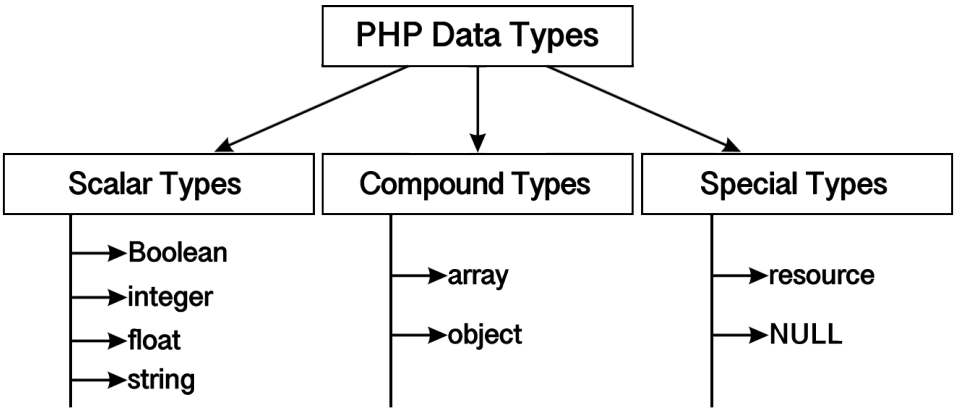
PHP *(PHP: Hypertext Preprocessor),* is a server-side XHTML-embedded and open source scripting language based on C. PHP scripts are either embedded in XHTML documents or are in files that are referenced by XHTML documents. PHP code is embedded in XHTML documents by enclosing it between the ***<?php*** and ***?>*** tags. As a server-side scripting language, PHP is naturally used for form handling and database access. Database access has been a prime focus of PHP development; as a result, it has driver support for 15 different database systems. PHP supports the common electronic mail protocols POP3, and IMAP. It also supports the distributed object architectures such as COM and CORBA *(Common Object Request Broker Architecture)*. Since PHP is a scripting language, it is an alternative to CGI, Microsoft's ASP *(Active Server Pages)* ASP.NET, Sun Microsystems' JSP *(Java Server Pages)* and Allaire's Coldfusion. In the sense of the way PHP's scripts are interpreted, it's related to client-side JavaScript. When a browser locates embedded JavaScript code in an XHTML doc, it calls the JavaScript interpreter to interpret the script. When a browser requests an XHTML doc that includes PHP script, the web server that provides the document calls the PHP processor. The server determines that a document includes PHP script by the filename extension. If it is .php, .php3, or phtml, it has embedded PHP. The purpose of using PHP & MySQL is to allow the content to be pulled dynamically from the database to create web pages for viewing in a regular browser. On one side of the system, you have a user accessing your site through a web browser to request a page. That browser expects to receive a standard HTML document in return. At the other end, you have the content of your site, which sits in one or more tables in a MySQL database that only understands how to respond to SQL query commands.



At this point, Facebook has been out for several years and I've signed up so that I can have a more concise understanding of the site layout, and data flow between actions in order to prepare my file structure for my Facebook Lite assignment. This guide demonstrates the process I followed for writing my own Facebook Lite page in PHP with MySQL. As usual, my written code snippets will be demonstrated using the Vim code editor in Linux.

Primitives, Operations, and Expressions

PHP has four scalar types—**Boolean**, **integer**, **float**, and **string**; two compound types—**array** and **object**; and two special types—**resource** and **NULL**. Because PHP is dynamically typed, it has no type declarations. In fact, there is no way or need to ever declare the type of a variable. The type of a variable is set every time the variable is assigned a value. An unassigned variable, sometimes called an unbound variable, has the value NULL, which is the only value of the NULL type. If an unbound variable is used in an expression, NULL is coerced to a value that is dictated by the context of the use. If the context specifies a number, NULL is coerced to 0; if the context specifies a string, NULL is coerced to the empty string. A variable can be tested to determine whether it currently has a value. The test is carried out with the **IsSet** function, which takes the variable's name as its parameter and returns a Boolean value. For example, **IsSet($fruit)** returns TRUE if $fruit currently has a non-NULL value, FALSE otherwise. A variable that has been assigned a value retains that value until either it is assigned a new value or it is set back to the unassigned state, which is done with the unset function.



PHP has a single integer type, named integer. This type corresponds to the long type of C and its successors, which means its size is that of the word size of the machine on which the program is run. In most cases, this is 32 bits, or a bit less (not fewer) than 10 decimal digits. PHP's double type corresponds to the double type of C and its successors. Double literals can include a decimal point, an exponent, or both. The exponent has the usual form of an E or an e, followed by a possibly signed integer literal. There is no requirement for any digits before or after the decimal point, so both `.345` and `345.` are legal double literals. Characters in PHP are single bytes; UNICODE is not supported. There is no character type. A single character data value is represented as a string of length 1. String literals are defined with either single-quote (') or double-quote (") delimiters. In single-quoted string literals, escape sequences, such as \n, are not recognized as anything special and the values of embedded variables are not substituted for their names. This substitution is referred to as interpolation. In double-quoted string literals, escape sequences are recognized and embedded variables are replaced by their current values. For example, the value of

```
'The sum is: $sum'
```

is exactly as it is typed. However, if the current value of $sum is 10.2, then the value of

```
"The sum is: $sum"
```

is

```
The sum is: 10.2
```

If a double-quoted string literal includes a variable name, but you do not want it interpolated, precede the first character of the name (the dollar sign) with a backslash (\). If the name of a variable that is not set to a value is embedded in a double-quoted string literal, the name is replaced by the empty string. Double-quoted strings can include embedded newline characters that are created with the Enter key. Such characters are exactly like those that result from typing \n in the string. The length of a string is limited only by the memory available on the computer. The only two possible values for the Boolean type are TRUE and FALSE, both of which are case insensitive. Although Boolean is a data type in the same sense as integer, expressions of other types can be used in a Boolean context. If a non-Boolean expression appears in a Boolean context, the programmer obviously must know how it will be interpreted. If an integer expression is used in a Boolean context, it evaluates to FALSE if it is zero; otherwise, it is TRUE. If a string expression is used in a Boolean context, it evaluates to FALSE if it is either the empty string or the string "0"; otherwise, it is TRUE. This implies that the string "0.0" evaluates to TRUE. The only double value that is interpreted as FALSE is exactly 0.0.

Let's consider **scalar-type conversions**. PHP, like most other programming languages, includes both implicit and explicit type conversions. Implicit type conversions are called coercions. In most cases, the context of an expression determines the type that is expected or required. The context can cause a coercion of the type of the value of the expression. Some of the coercions that take place between the integer and double types and between Boolean and other scalar types have already been discussed. There are also frequent coercions between numeric and string types. Whenever a numeric value appears in a string context, the numeric value is coerced to a string. Likewise, whenever a string value appears in a numeric context, the string value is coerced to a numeric value. If the string contains a period, an e, or an E, it is converted to double; otherwise, it is converted to an integer. If the string does not begin with a sign or a digit, the conversion fails and zero is used.

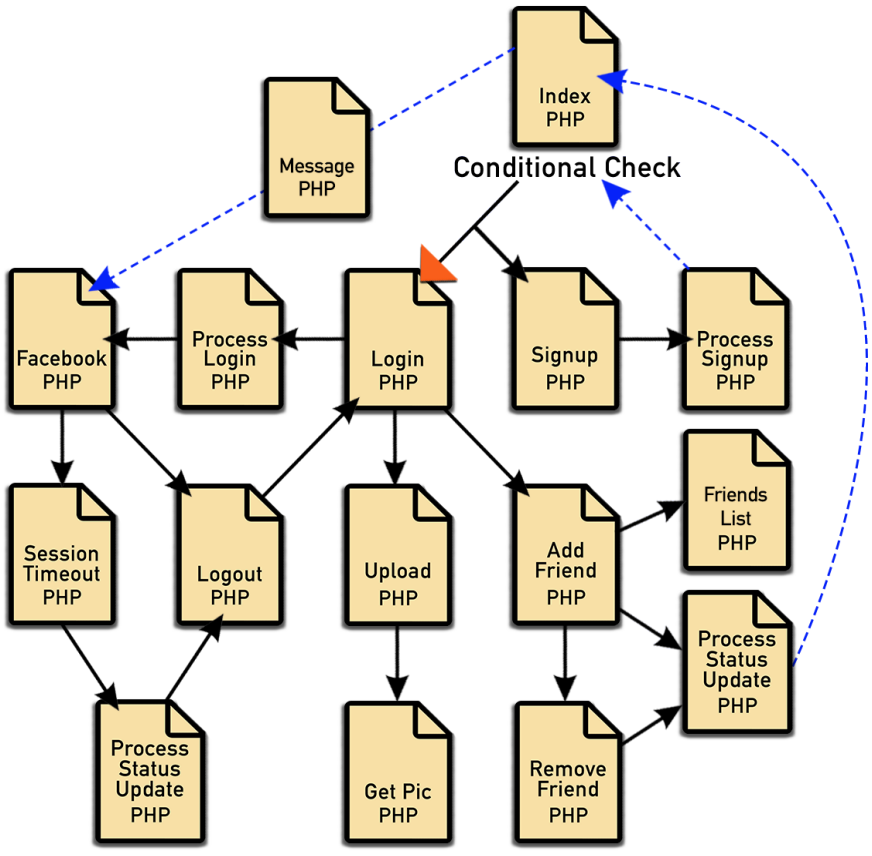
Non-numeric characters following the number in the string are ignored. PHP uses the eight **relational operators** of JavaScript. The usual six **(>, <, >=, <=, !=, and ==)** have the usual meanings. PHP also has **===**, which produces TRUE only if both operands are the same type and have the same value, and **!==**, the opposite of **===**. If the types of the operands of the other six relational operators are not the same, one is coerced to the type of the other. If a string is compared with a number and the string can be converted to a number *(if it is in fact a string version of a number—for example, "42")*, the string will be converted and a numeric comparison will be done. If the string cannot be converted to a number, the numeric operand will be converted to a string and a string comparison will be done. If both operands are strings that can be converted to numbers, both will be converted and a numeric comparison will be done. This is often not what is desired. To avoid it and similar problems associated with string-to-number coercions, if either or both operands are strings that could be converted to numbers, the `strcmp` function should be used rather than one of the comparison operators. One common way for a browser user to interact with a Web server is through forms. A form is presented to the user, who is invited to fill in the text boxes and click the buttons of the form. The user submits the form to the server by clicking the form's Submit button.

The contents of the form are encoded and transmitted to the server, which must use a program to decode the contents, perform whatever computation is necessary on the data, and produce output. When PHP is used to process form data, it implicitly decodes the data. It may seem strange, but when PHP is used for form handling, the PHP script is embedded in an XHTML document, as it is with other uses of PHP. Although it is possible to have a PHP script handle form data in the same XHTML document that defines the form, it is perhaps clearer to use two separate documents. For this latter case, the document that defines the form specifies the document that handles the form data in the action attribute of its <form> tag. PHP can be configured so that form data values are directly available as implicit variables whose names match the names of the corresponding form elements. However, this implicit access is not allowed in many Web servers (through the configuration of PHP), because it creates a security risk. The recommended approach is to use the implicit arrays **$_POST** and **$_GET** for form values.

My File Structure for Facebook Lite

We know that Facebook was primarily structured around using PHP to build the site. The following diagram demonstrates a complete version of the file structure for everything I'll be implementing on my Facebook Lite page:

1) A required Index file
2) A messaging system
3) The ability to Signup
4) A data file for processing the Signup
5) A Login page
6) A data file for processing the login
7) A generic Facebook file for storing the login cache
8) A logout data file
9) A session timeout for an idle user
10) A data file for processing the status update for a logout
11) The ability to upload a photo
12) A data file for processing the photo
13) The ability to add or remove a friend
14) See the status of friends



This file structure is what I needed to create a lightweight version of the Facebook website. This process is also how we consider a single webpage version of Facebook to work. The data flow between files is fairly straightforward, and creating a visual diagram/flow chart can help you see what you will need to start implementing your page. Like most websites, the vast majority of data is handled within the index file. The index file will contain every aspect of the assignment requirements, including additional functionality for a messaging system for communicating with other students in the same class. The webpage had to function just like the functionality of the existing Facebook website. Just like other assignments, the professor has a database set up for us to access and store the files that need to communicate with the files that we are uploading. Unlike HTML, PHP is a server-side scripting language, therefore the content of PHP files can only be seen once uploaded to a server. Considering this information, it's time to jump into the implementation of my Facebook Lite page.

Implementing the Code

Following the file structure that I created, like an html file, I can start with the index.php file. It honestly doesn't matter what order the user starts in. The elements of it are created specifying and using XHTML code, but we save the file as a PHP extension. While the main file is the index file, the signup.php file is a logical starting point prior to creating the index page since we need to provide credentials to get into the index file, and the signup form is where we create said credentials. The following example is how I created the signup form.

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

  <head>
        <link rel="stylesheet" type="text/css" href="style.css" />
        <title>Chad's FaceBook Page</title>
  </head>
  <body>
  <div id="container">
  <h1>Start a New Account</h1>
  <hr/>
  <form action="processSignup.php">
                <p>What is your desired screenname?</p>
                <input type="text" id="screenname" name="screenname"/>
                <p>Desired Password</p>
                <input type="password" name="password" id="password"/>
                <input type="submit" name="submit" id="submit"/>
                <br/>
  </form>
  </div>
</body>
</html>
```

This is a basic HTML setup, but the file must be saved as signup.php. The signup form is created using plain HTML, but just like any online form, we have to write the back-end code to provide the required functionality to the form. PHP scans files for parsing, and it looks for an opening

and closing tags to gain access to the file. We use the **<?php** open tag to act as a door so the file can be accessed and all code between the opening and closing tags is interpolated. Starting on **line 2** the dollar sign declares a variable call to screenname and then set a request to screenname as an operator. The same is done for the password as well. Next on **Line 4**, I make a MySQL database connection request for my screenname and my password. Starting on **line 9** I make a function call to select the database that has been supplied on the department servers.

```
 1 <?php
 2 $screenName=$_REQUEST["screenname"];
 3 $password=$_REQUEST["password"];
 4 $con = mysql_connect("mysql","cjordan","phapaigo");
 5 if (!$con)
 6 {
 7         die('Could not connect: ' . mysql_error());
 8 }
 9 mysql_select_db("cos264s09", $con);
10 $query="INSERT INTO Accounts(ScreenName,Password)
11                     VALUES('$screenname', '$password');";
12 mysql_query($query);
13
14 mysql_close($con);
15 header("Location:login.php");
16 ?>
```

On **line 15** I declare a header call to login.php as a reference point for the signup, and then type a closing tag to seal the code. This covers the signup portion of the program, but for anyone who has already created an account, we need to give people a way to log in as a returning user. Creating the login is essentially the same process as creating the signup form, just with slightly varied verbiage indicating a normal login. This is provided in the following example:

```
 1 <!DOCTYPE html
 2 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 5
 6   <head>
 7         <link rel="stylesheet" type="text/css" href="style.css" />
 8         <title>Chad's FaceBook Page</title>
 9   </head>
10   <body>
11   <div id="container">
12   <h1>Login</h1>
13   <hr/>
14   <form action="processLogin.php">
15                 <p>Screenname</p>
16                 <input type="text" id="screenname" name="screenname"/>
17                 <p>Desired Password</p>
18                 <input type="password" name="password" id="password"/>
19                 <input type="submit" name="submit" id="submit" value ="submit"/>
20                 <br/>
21   </form>
22   <a href="signup.php">Signup</a>
23   </div>
24 </body>
25 </html>
```

At this stage, most of this is the same as the previous *processSignup* file, except that this one is checking against the status of current user, and fetches the result for the existing condition and if there is a mismatch in either the id or password, this processing file throws an error.

```
 1 <?php
 2 $FormScreenName=$_REQUEST["screenname"];
 3 $FormPassword=$_REQUEST["password"];
 4 $con = mysql_connect("mysql","cjordan","phapaigo");
 5 if (!$con)
 6 {
 7         die('Could not connect: ' . mysql_error());
 8 }
 9 mysql_select_db("cos264s09", $con);
10 $query="SELECT * FROM Accounts WHERE ScreenName='$FormScreenName';";
11 $result=mysql_query($query);
12 if (!$result) {
13         print "Error - the query could not be executed ";
14         $error = mysql_error();
15         print "<php>" . $error . "</p>";
16         exit;
17 }
18 if (mysql_num_rows($result) > 0){
19         $row = mysql_fetch_array($result);
20         if ($FormPassword==$row["Password"]){
21                 session_start();
22                 $_SESSION["id"]=$row["id"];
23                 $_SESSION["screenname"]=$row["ScreenName"];
24                 $_SESSION["status"]=$row["Status"];
25                 $_SESSION["picture"]=$row["Picture"];
26                 header("Location:index.php");
27         }else print("incorrect password");
28 }else print("screenname unknown");
29
30 mysql_close($con);
31 header("Location:index.php")
32 ?>
```

On **line 10** I begin a MySQL query with a `SELECT` statement to instruct the database of the desired data to be retrieved, in this case, locating the data for the variable *FormScreenName* taking the result, and then checking the status of screenname and password with an if statement. At the end just like the previous files, we link header as index.php. Continuing with the index portion of the program, we have to create the main page *(index)* file to display our Facebook page. Just as before, I link the previous page to reference the file data required to tie into each page function and then begin declaring my standard html code.

```
 1 <?php
 2 session_start();
 3 if (!isset($_SESSION['id'])){
 4         header('Location:login.php');
 5 }
 6 ?>
 7 <!DOCTYPE html
 8 PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 9 "http://www.w3.org/TR/xhtml11/DTD/xhtml11-strict.dtd">
10 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

This next portion of the code contains the standard HTML structure and then a few other elements that will be used on the main page of my Facebook Lite page.

```
12  <head>
13        <link rel="stylesheet" type="text/css" href="style.css" />
14        <style type="text/css">
15        body {background-color: #0099FF}
16        </style>
17        <title>Facebook Lite</title>
18        </script>
19  </head>
20  <body>
21  <div id="top-container">
22  <div class="topNav">
23              <input type="text" alt="float:left" placeholder="Search...">
24  </div>
25  <div id="fblogo">
26       <img src="fblogo.png" alt="image" class="center" />
27       <h2 align="center">The new place to exploit your friends face</h2>
28  <hr/>
29        <div style="float:right">
30        <p style="font-size:8px; font-family: Arial, Verdana;">Friends List</p>
31        <a href="friendsList.php" style="float:right">friends</a>
32        <p style="font-size:8px; font-family: Arial, Verdana;">Logout</p>
33        <a href="logout.php" style="float:right">logout</a>
```

Starting on **line 34** I create a table tag to hold the next piece of content and the PHP is written to handle the data for the variable id. Just as before, starting on **line 38** I run my conditional check to open the database again, and this time starting on **line 44** I run a MySQL query to `SELECT` the screenname and id. The `WHERE` clause is used to filter records, extracting only records that fulfill specific conditions in this case, accessing the *id* attribute for Accounts.

```
34  <table>
35  <?php
36        $id=$_SESSION["id"];
37        $con = mysql_connect("mysql","cjordan","phapaigo");
38        if (!$con)
39        {
40              die('Could not connect: ' . mysql_error());
41        }
42        mysql_select_db("cos264s09", $con);
43        $Search=$_REQUEST["search"];
44        $query="SELECT Accounts.ScreenName, Accounts.id
45                  FROM Accounts, Friends
46                      WHERE
47                      Accounts.id=Friends.Second
48                      AND Friends.First=$id OR
49                      Accounts.id=Friends.First
50                      AND Friends.Second=$id;";
51        $result=mysql_query($query);
52        if (!$result) {
53              print "Error - the query could not be executed ";
54              $error = mysql_error();
55              print "<php>" . $error . "</p>";
56              exit;
57        }
```

The if conditional on **line 58** will print off a 'no search results' found message if the conclusion of the table data in the search is equal to **0**. **Line 61** loops through and returns the result of *ScreenName* in the associative array. The rest of the while loop echoes *(displays)* the output of the specified parameters from **lines 62** to **69** for removing a friend.

```
58          if (mysql_num_rows($result)==0){
59                  print "<tr><td> No Results Found </td></tr>";
60          }
61          while($row=mysql_fetch_assoc($result)){
62                  echo '<tr><td>';
63                  echo $row['ScreenName'];
64                  echo  '</td>';
65                  echo '<td> <form action="removefriend.php">
66                          <input type="hidden" name="otherid" value="'.$row["id"].'"/>
67                          <input type="submit" name="submit" id="submit" value="remove"/>
68                          </form></td>';
69                  echo '</td></tr>';
70          }
71          mysql_close($con);
72  ?>
73          </table>
74          </div>
```

The next several lines throughout this block of HTML are **<form>** tags that will display input for *online* or *offline* status, setting the profile picture, and a search bar for the user.

```
75          <p><?php echo $_SESSION["screenname"];?></p>
76      <form action="processStatusUpdate.php">
77                  <p>Status</p>
78                  <input type="dropdown" id="status" name="status"
79                         value="<?php echo $_SESSION["status"];?>"/>
80                  <input type="submit" name="submit" id="submit" value="Update"/>
81      </form>
82       <img width="200" height="100" src="get_pic.php"/><br/>
83      <form enctype="multipart/form-data" action="upload.php" method="post">
84      <p>Set Picture</p>
85       <input name="userfile" type="file" />
86       <input type="submit" value="Update" />
87      </form>
88      <form action="index.php">
89                  <p>Search</p>
90                  <input type="text" id="search" name="search"/>
91                  <input type="submit" name="submit" id="submit" value="Search"/>
92      </form>
```

You've seen this before; I start by creating a table, and then create a conditional to check a connection to the MySQL database for accessing the department servers.

```
93      <table>
94      <?php
95  if (isset($_REQUEST["search"])){
96          $id=$_SESSION["id"];
97          $con = mysql_connect("mysql","cjordan","phapaigo");
98          if (!$con)
99          {
100                 die('Could not connect: ' . mysql_error());
101         }
102         mysql_select_db("cos264s09", $con);
```

Starting on **line 103** I run a MySQL search query for record data, followed by a `SELECT` statement searches for Pictures, ScreenName, and id within the Accounts element; and then checking the result of the query. Just like in my previous block of PHP, starting on **line 106** I run my conditional checks against the query, followed by the while loop on **line 115** that returns the result of *ScreenName* in the associative array. The rest of the while loop echoes *(displays)* the output of the specified parameters for adding a friend. Then, just as before, we close the MySQL connection, followed by the closing php tag, and then the closing table tag.

```
103         $Search=$_REQUEST["search"];
104         $query="SELECT Picture, ScreenName,id FROM Accounts WHERE ScreenName LIKE '%$Search%';";
105         $result=mysql_query($query);
106         if (!$result) {
107                 print "Error - the query could not be executed ";
108                 $error = mysql_error();
109                 print "<php>" . $error . "</p>";
110                 exit;
111         }
112         if (mysql_num_rows($result)==0){
113                 print "<tr><td> No Results Found </td></tr>";
114         }
115         while($row=mysql_fetch_assoc($result)){
116                 echo '<tr><td>';
117                 echo $row['ScreenName'];
118                 echo  '</td>';
119                 echo '<td> <form action="addfriend.php">
120                         <input type="hidden" name="otherid" value="'.$row["id"].'"/>
121                         <input type="submit" name="submit" id="submit" value="add"/>
122                         </form></td>';
123                 echo '</td></tr>';
124         }
125         mysql_close($con);
126 }
127 ?>
128         </table>
```

The remaining code in the index file repeats the same process with previous elements, except this time it's being done to allow the users to message each other in the window.

```
129   <table>
130 <form action="message.php" method="POST">
131       Screen Name: <input type="text" name="author"><br>
132       Thread Title: <input type="text" name="title"><br>
133       Message<br><textarea cols="60" rows="5" name="message"></textarea><br>
134       <input type="submit" value="Post Message">
135 </form>
136   <?php
137        $sql = mysql_query("SELECT * FROM messages ORDER BY posted DESC");
138
139 while($r = mysql_fetch_array($sql)) {
140
141 $posted = date("jS M Y h:i",$r[posted]);
142 echo "<h3><a href='message.php?id=$r[id]'>$r[title]</a>
143               ($r[replies])</h3><h4>Posted by $r[author] on $posted</h4>";
144 }
145 ?>
146   </table>
147   </div>
148 </body>
149 </html>
```

When it comes to embedded PHP, we place the code within the **<table>** tags because everything outside of a pair of opening and closing tags is ignored by the PHP parser which allows PHP files to have mixed content. The index file may be done, but there's still a little more to do with the remaining files. Earlier on *page 4* I provided a visual diagram demonstration of the file structure for my Facebook Lite program. Those files are still being referenced, and needing to be written. The last php file that I referenced in my index file was the message file and since that file has a little more going on inside of it, I'll jump into that file next. We already have a general message *(text area)* space on the index page, but we know that we need more SQL queries thrown inside of PHP to handle the functionality of those HTML elements.

Starting from the beginning I connect to the supplied database. On **line 5** I get the current time as a UNIX timestamp variable. Next I insert the information sent from the form into the database. **Line 9** sends a string that the thread is posted, and returns the value to the index page. Next, on **line 10** this query selects the replies from the database where the thread ID matches the thread **$_GET** value. We have to store these results, so beginning on **line 11** the while loop gets the results and stores them into an array. Everything within the curly brackets

```
1  <?php
2  mysql_connect("localhost", "ScreenName", "password");
3  mysql_select_db("cos264s09");
4
5  $time = time();
6  mysql_query("INSERT INTO threads VALUES(NULL,'$_POST[title]',
7              '$_POST[message]','$_POST[author]','0','$time')");
8
9  echo "Thread Posted.<br><a href='index.php'>Return</a>";
10 $sql = mysql_query("SELECT * FROM threads WHERE id = '$_GET[id]'");
11 while($r = mysql_fetch_array($sql)) {
12
13     echo "<h2>$r[title]</h2>";
14     $posted = date("jS M Y h:i",$r[posted]);
15
16     echo "$r[message]<h4>Posted by $r[author] on $posted</h4><hr>";
17     }
18     mysql_connect("localhost", "ScreenName", "password");
19     mysql_select_db("cos264s09");
20     $time = time();
21     mysql_query("INSERT INTO replies VALUES(NULL,'$_POST[thread]',
22                     '$_POST[message]','$_POST[author]','$time')");
23     mysql_query("UPDATE threads SET replies = replies + 1 WHERE id = '$_POST[thread]'");
24     echo "Reply Posted.<br><a href='index.php?id=$_POST[thread]'>Return</a>";
25 ?>
```

can read from the database using **$r[]** and we need to convert the UNIX timestamp entered into the database for when a thread or message is posted into a readable date, using date(). This is done on **line 14**. **Line 21** inserts the information sent from the form into the database, then the next line allows us to update the reply count in the threads database, and finally, making a *"reply posted"* message with a link back to the index page. The facebook.php file will be used to store information from login details and messaging.

```php
<?php
$con = mysql_connect("mysql","cjordan","phapaigo");
if (!$con)
{
        die('Could not connect: ' . mysql_error());
}
mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM Persons");

while($row = mysql_fetch_array($result))
{
        echo $row['FirstName'] . " " . $row['LastName'];
        echo "<br />";
        echo "$r[message]<h4>Posted by $r[author] on $posted</h4>";
}
$sql = mysql_query("INSERT * FROM replies WHERE thread = '$_GET[id]'");
mysql_close($con);
?>
```

In order to have a proper session timeout, we need to write a timer to do so. The *set_time_limit* attribute performs a function call to 300 seconds which will give the user 5 minutes of idle time before disconnecting with the database.

```php
<?php
set_time_limit(300);
$id=$_SESSION["id"];
$con = mysql_connect("mysql","cjordan","phapaigo");
while ($i<150)
{
        echo "i=$i ";
        sleep(50);
        $i++;
}
mysql_select_db("cos264s09", $die);
$time = time();
$query="UPDATE Accounts SET Status='$status' WHERE id=$id, '$time'";
mysql_query($query);
header("Location:index.php");

mysql_close($con);
?>
```

As any changes are made to the user account and profile page, this data needs to be processed and updated in the system. This is where I created a file called ProcessStatusUpdate.php.

```php
<?php
session_start();
$id=$_SESSION["id"];
$status=$_REQUEST["status"];
$con = mysql_connect("mysql","cjordan","phapaigo");
if (!$con)
{
        die('Could not connect: ' . mysql_error());
}
mysql_select_db("cos264s09", $con);
$query="UPDATE Accounts SET Status='$status' WHERE id=$id;";
mysql_query($query);
header("Location:index.php, timeout.php, friendsList.php");

if (set_time_limit > 300)
{
        die('Timed out session: ' . mysql_error($con));
}
mysql_close($con);
?>
```

In this previous block, we can see from **line 15** that we need to run a conditional check if we were idle too long and the timer ran out, resulting in a disconnect from the database. This file is used in multiple instances throughout the Facebook program and executed numerous times as any changes are updated to the user profile. This next block is the smallest external file with code in the whole program. Logout.php doesn't really require much at all. We simply start a session for the action to be executed, and then destroy the session when the user clicks the logout button. The header must be set to login.php if the code is executed.

```
1 <?php
2 session_start();
3 session_destroy();
4 header("Location:login.php");
5 ?>
```

This next block *(upload.php)* gives the ability to upload a file to the user profile page. The if statement on **line 3** checks the instance of the userfile attribute in an associative array. Essentially, **$_FILES** declares a variable that performs an HTTP request to upload variables. **Line 9** uploads and stores the image and temporary name in the database. The next section is just like previous blocks where we check to gain access to the database, and the query on **line 17** updates the *Accounts* parameter and applies the image data to the id of the user.

```
1 <?php
2 session_start();
3 if(!isset($_FILES['userfile'])) {
4         echo '<p>Please select a file</p>';
5 }
6 else
7 {
8         try {
9                 $imgData =addslashes(file_get_contents($_FILES['userfile']["tmp_name"]));
10                $id=$_SESSION["id"];
11                $con = mysql_connect("mysql","cjordan","phapaigo");
12                if (!$con)
13                {
14                        die('Could not connect: ' . mysql_error());
15                }
16                mysql_select_db("cos264s09", $con);
17                $query="UPDATE Accounts SET Picture='$imgData' WHERE id=$id;";
18                mysql_query($query);
19
20                mysql_close($con);
21        }
22        catch(Exception $e) {
23                echo $e->getMessage();
24                echo 'Sorry, could not upload file';
25        }
26 }
27 header("Location:index.php");
28 ?>
```

The primary method of handling exceptions in PHP is the **try-catch**. In a nutshell, the try-catch is a code block that can be used to deal with thrown exceptions without interrupting program execution. In other words, you can *"try"* to execute a block of code, and *"catch"* any PHP

exceptions that are thrown. This is helpful when testing multiple exceptions and/or the properties of multiple exceptions like class name, message, and code because even once an exception is thrown the PHP cannot return to the line of code that comes after the line that had thrown the exception. This block of code *(getPic.php)* is how we can get the picture that we want to upload. At this stage of the guide, the vast majority of this code is really just repeated again from earlier files that I've already explained. The only difference in this file is from lines **19** to **21**. For the header link, we make a call to *Content-type* rather than a specific file. **Line 20** *(if the conditionals are successful)* takes the result from the *imagecreatefromstring* datatype and sets the value to the variable *im* for the image.

```
 1 <?php
 2 session_start();
 3 $id=$_SESSION["id"];
 4 $con = mysql_connect("mysql","cjordan","phapaigo");
 5 if (!$con)
 6 {
 7         die('Could not connect: ' . mysql_error());
 8 }
 9 mysql_select_db("cos264s09", $con);
10 $query="SELECT Picture FROM Accounts WHERE id=$id;";
11 $result=mysql_query($query);
12 if (!$result) {
13         print "Error - the query could not be executed ";
14         $error = mysql_error();
15         print "<php>" . $error . "</p>";
16         exit;
17 }
18 mysql_close($con);
19 header("Content-type: image/jpeg");
20 $im=imagecreatefromstring(mysql_result($result, 0));
21 imagejpeg($im);
22 ?>
```

On **line 21**, the *imagejpeg* function is an inbuilt function in PHP which is used to display image to browser or file. The main use of this function is to view an image in the browser, convert any other image type to JPEG and altering the quality of the image.

Facebook wouldn't be Facebook without the ability to add or remove friends, so we need to implement this functionality into the program. This is starting point for my *addFriend.php* file, and the first 10 lines of code are the same process that I've demonstrated in previous files.

```
 1 <?php
 2 session_start();
 3 $myid=$_SESSION["id"];
 4 $otherid=$_REQUEST["otherid"];
 5 $con = mysql_connect("mysql","cjordan","phapaigo");
 6 if (!$con)
 7 {
 8         die('Could not connect: ' . mysql_error());
 9 }
10 mysql_select_db("cos264s09", $con);
```

Starting on **Line 11** we use the MySQL `INSERT` operator for values passed to the Friends parameter, and then check the condition of the current user id, and other variable id's in the query.

```
11 $query="INSERT INTO Friends(First, Second) VALUES($myid, $otherid);";
12 if($myid < $otherid){
13         $query="INSERT INTO Friends(First, Second) VALUES($myid, $otherid);";
14 }
15 else
16         $query="INSERT INTO Friends(First, Second) VALUES($otherid, $myid);";
17
18 mysql_query($query);
19
20 print(mysql_error());
21 mysql_close($con);
22 header('Location:index.php')
23 ?>
```

The next block is the process of removing a friend with my *removeFriend.php* file. This script is really straight forward even more so than adding a friend. All you're really doing is simply running a MySQL `DELETE` operator from the query and specifying the variable id of the user you are removing from index.

```
 1 <?php
 2     session_start();
 3         $id=$_SESSION["id"];
 4         $otherid=$_REQUEST["otherid"];
 5         $con = mysql_connect("mysql","cjordan","phapaigo");
 6         if (!$con)
 7         {
 8                 die('Could not connect: ' . mysql_error());
 9         }
10         mysql_select_db("cos264s09", $con);
11         $query="DELETE FROM Friends WHERE
12                         First=$id AND Second=$otherid OR
13                         Second=$id AND First=$otherid;";
14     mysql_query($query);
15         mysql_close($con);
16         header("Location: index.php");
17 ?>
```

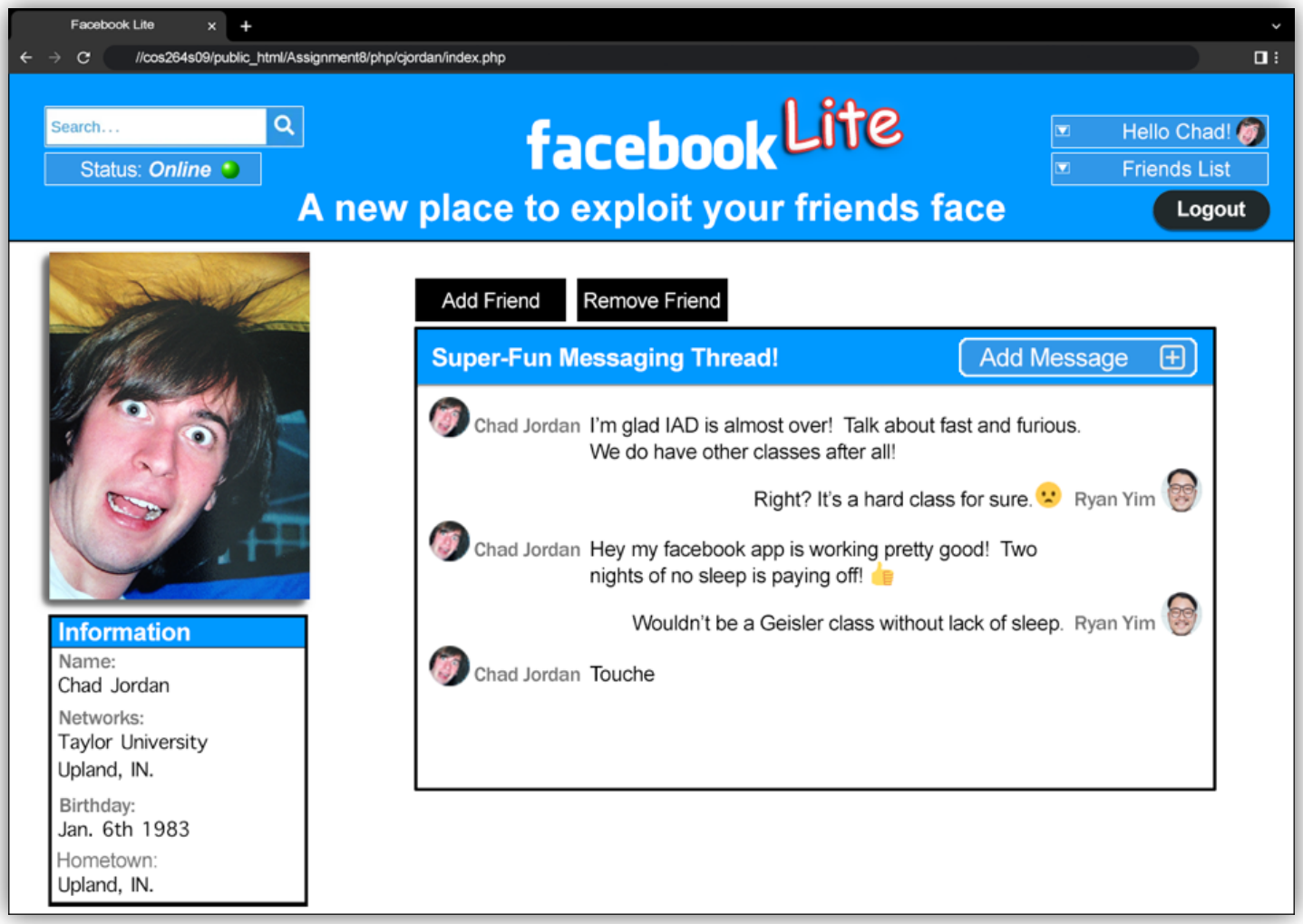
The final page for this program is the ability to view your friends list on the index page. This is basically done in one large query and then printing the results to the screen. We grab the data stored in userid, the profile, along with their image and will display the current user as you scroll through the list.

```
1 <?php
2 $sql = mysql_query("
3 SELECT profile_userid, profile_name, profile_image
4   FROM fsb_friendlist
5 LEFT JOIN fsb_profile ON (friendlist_friendid=profile_userid)
6 WHERE friendlist_memberid = THE_CURRENT_USER_ID");
```

The while loop takes the data stored in the associative array and prints it using the specified HTML tags and the variables for the table data. Retrieving this data will also assist in the messaging side of my program for displaying the user's photo. I can just piggyback off of this file for that data.

```
 8 while ($t = mysql_fetch_assoc($sql)) {
 9   echo "<div style='width:150px; float:left;'>"
10   echo "<strong>$t[profile_name]</strong><br />";
11   echo "<img src='$t[profile_image]' alt='' />";
12   echo "</div>"
13 }
14 ?>
```

That's essentially it as far as the backend. I still had to write a stylesheet to decorate the content up a little, but after finishing my CSS file, and making a few other small changes to my PHP code and my HTML, I managed to successfully complete the program and this was the final result of my Facebook Lite page running in Chrome:



Conclusion

This concludes the guide for my Facebook Lite in PHP & MySQL program. I hope it was a helpful learning experience for jumping into vanilla PHP v5.2.9 with MySQL. This program was the 8th assignment for my COS264 - *Interactive Application Development* course from my department at Taylor University. One of the greatest aspects of PHP is that it's open-source and free from cost unlike other overpriced server-side languages like Webfocus. It can be downloaded anywhere and is readily available to use for events or web applications. PHP is cross-platform so it can run on any OS like UNIX, Linux, or Windows. It's mainly used due to its faster rate of loading over slow internet speed than other programming languages. My entire website is also PHP-driven. I write my pages in HTML, but I wrap all of my code in PHP to handle content so I'm writing less code, and doing more with managing content. All diagrams and code presented in this guide were created and written by Chad Jordan for learning purposes only. The OS that this was displayed in was Ubuntu Linux 8.10 and written using the Vim code editor. For any possible inquiries such as general questions regarding this guide or other professional inquiries please feel free to email me at cjordan@wondercreationstudios.com

Resources Used:

- Sebesta, W. Robert - *Programming the World Wide Web – 4th Edition* – 2008
- Php.net