Forms and PHP

Objectives

Upon completion of this session you should be able to:

- extract data from forms using PHP
- create formatted output pages based on the data
- store the extracted data into a text file

Overview

Examine the code for a sample form which is used to collect data. You are required to download the form and deploy it on the Deakin webserver. The form should be accessible from http://www.deakin.edu.au/~username/week10/form.html

When the submit button on the form is clicked, the destination is "SaveData.php". You are required to write the PHP file called "SaveData.php" which will extract the data from the html form and save the data into a file (on the server).

Tasks

1. Deploying the form on the Deakin webserver
2. Write a PHP file which will:
    i. Extract the values of the various form elements
    ii. Write those values into a file
    iii. Send an HTML document back to the user to indicate that the form has been received and stored successfully

Deploying the form on the Deakin webserver

Locate the "public_html" folder on the network drive. Create a subfolder "week10" in "public_html" and place the html file (form.html) there. If the network drive is not available then it can mapped manually:

> Select the Start menu -> All Programs -> Accessories -> Windows Explorer.
> In Windows Explorer, select Tools -> Map Network Drive. The folder to be mapped is obtained from *https://www.deakin.edu.au/its/homedirectory/*

The file can be accessed from the web browser from the URL: http://www.deakin.edu.au/~username/week10/form.html.

Extracting form values

Create a new file **SaveData.php** in notepad and save it to \public_html\week10\\ on the network folder (same location as form.html). The process of extracting form data in PHP involves the pre-defined **$_REQUEST** function, which returns a value into a variable. Insert this code block into the file:

```
<?php
$dayofbirth = $_REQUEST['dayofbirth'] ;
$monthofbirth = $_REQUEST['monthofbirth'] ;
$yearofbirth = $_REQUEST['yearofbirth'] ;
```

```
$postcode = $_REQUEST['postcode'] ;
?>
```

The field names must match the form field names *exactly*, but you can call the variables - such as *$dateofbirth* - whatever you like (provided of course that they are legal PHP variable names). Add the remaining fields to the file by adding extra $_REQUEST statements; if you don't want or need a particular field, you don't need to extract it (although one might wonder why it was included if you are not going to process it).

Creating a response page

If we now load up our form (*from the Deakin server*) and click on the Submit button we should see the empty SaveData.php. The code to extract the field data will have been executed, but there is no visible indication of this. Let's check that the form data is being extracted properly by adding a number of echo statements; put this code **below** the first block:

```
<?php
echo ("<h2>Thank you for completing our Online Survey</h2>") ;
echo ("<p>You were born on day ".$dayofbirth." in month ".$monthofbirth." in the year ". $yearofbirth.".</p>");
echo ("<p>Congratulations!</p>");
echo ("<p></p>");
?>
```

You will notice that we are displaying the values sent from the form, so we are getting 1-12 for the month, not the true 'name' of the month. We *could* change the values in the form from 1 to January, 2 to February, and so on - or we could leave them as they are and change the code to improve the readability of the message. Replace this second block of code with the following, more complex code:

```
<?php

// Display a heading

echo ("<h2>Thank you for completing our Online Survey</h2>") ;

// Create a short string that includes the appropriate choice of th, st, nd or rd using the $dayofbirth variable

if ($dayofbirth == 1 or $dayofbirth == 21 or $dayofbirth == 31)
   $datesub = "<span style=vertical-align:sup>st</span>";
elseif ($dayofbirth == 2 or $dayofbirth == 22)
   $datesub = "<span style=vertical-align:super>nd</span>";
elseif ($dayofbirth == 3 or $dayofbirth == 23)
   $datesub = "<span style=vertical-align:super>rd</span>";
else
   $datesub = "<span style=vertical-align:super>th</span>";

// Pick the name of the month using the $monthofbirth variable

switch ($monthofbirth) {
  case 1:
    $monthname = "January";
    break;
  case 2:
    $monthname = "Febraury";
    break;
  case 3:
    $monthname = "March";
    break;
  case 4:
    $monthname = "April";
    break;
  case 5:
    $monthname = "May";
    break;
  case 6:
    $monthname = "June";
    break;
  case 7:
    $monthname = "July";
    break;
  case 8:
    $monthname = "August";
    break;
  case 9:
    $monthname = "September";
    break;
  case 10:
    $monthname = "October";
    break;
  case 11:
    $monthname = "November";
    break;
  case 12:
    $monthname = "December";
    break;
}

// Print out the core response statements

echo ("<p>You claim to have been born on the ".$dayofbirth.$datesub." of ".$monthname." in ".$yearofbirth.".</p>");
echo ("<p>Come on, who are you kidding - you've got to be older than <em>that</em>.</p>");
echo ("<p></p>");
?>
```

There is nothing too complex going on here (believe it or not). In the first section we are using the value of the date (1-31) to set the formatting of the day (whether it is followed by *st*, *nd*, *rd* or *th*). To do this we use an **if ... elseif ... else** statement, which is just a 'multiple choice' version of the normal **if ...** statement (so it's a *conditional* statement too). In the second section we are using the **switch** statement to make a similar 'multiple choice', using the value (1-12) of the month from the form to set the *$monthname* variable. The **switch** statement and the *if ... elseif ... else* statement are both *conditional* statements and are to some extent interchangeable in PHP (as in other languages). We could easily have used the **if** statement in the second case; however, using the switch statement in the first section would have produced a much larger piece code. It is also important to note that the *condition* that determines what outcome we get from a **switch** statement - in this case $monthofbirth - must be an integer, whilst we can use numbers or text (or even boolean variables) in **if** comparisons.

Storing the data

The final part of the process is to store the data in some way (although we don't always *need* to store form data). In the next couple of sessions we will be doing this in a 'real' (relational) database system, but for now we are going to simply write the data into a text file on the Deakin server.

Place the following code in the page; it must be *after* the first block of $_REQUEST statements:

```
$filename = 'datafile.txt';
$fp = fopen($filename, "a");
```

```
$write = fwrite($fp, $dayofbirth);
$write = fwrite($fp, $monthofbirth);
$write = fwrite($fp, $yearofbirth);
$write = fwrite($fp, $postcode);
fclose($fp);
```

This code creates a text file called datafile.txt and opens it for writing data. Actually, the file is being opend to *append* data, because we have included the "a" *mode*; each completed form adds data to the end of the file. [The modes that we might use when creating files include "r" (read), "w" (write) and "a" (append).] You should add an extra *$write* statement for each of the remaining fields.

Finally, we close the file to make sure the data is not lost.

If you run this code you will probably get an error, and no file will be created. The problem is that the script may not have permission to actually create, open and write to a file. On a Unix system (such as the Deakin server) your permissions almost certainly do not include this 'world write' setting (they are not needed to deliver web documents). You will probably therefore need to set them - but **only for the directory that you are using for this practical**, not your entire public_html area! To set these permissions you will need to Telnet to Deakin (using *TeraTerm Pro*, for example), and set the permissions 'manually'. You should be able to telnet by SSH to bajor.its.deakin.edu.au, using port 22. If, after successfully accessing the system, you see a text menu you need to use the *Shell* option to get to the command prompt. From there enter the various commands.

Let's assume that you have put your files for this session into a new folder called *week10*. Go into your public_html area and change directories until you are in the week10 folder; enter **ls** to find out where you are in the directory structure, and **cd dirname** to change directories, in this case into the directory called *dirname*. Type in

```
chmod 777 datafile.txt
```

The **777** setting gives scripts the privilege to write to the file datafile.txt; if this does not get done - or is not done properly - you will get a series of permission-based error messages from the PHP interpreter as it tries (but is not allowed) to execute the various file access statements. Also, if you need to use the User Management System to re-set your permissions after changing the php file, you will need to re-do the *chmod 777* process.

When you get this working without permission errors, you can look at the file (in Notepad, for example) and see the data you entered through the form. If you have filled in the form more than once, you will see that the data are 'joined up'; each 'record' is not on a new line! This makes it hard to work with (and read) the file; the solution is to 'force' a new line (technically it's a *carriage return/line feed* combination) at the end of each record; just add the following line immediately before the *fclose* statement:

```
$write = fwrite($fp, "\r\n");
```

You should now get each data record on a new line - but the data are all run together! We should separate them in some way, and the simplest way is to put a *separator* between each field. If we used a comma, for example, the resulting file would a *comma-separated* (*CSV*) file that could, for example, be imported directly into Microsoft Excel. Another common separator that we might use is the tab (ASCII character 9), strictly called the *horizontal tab*. We can write a tab character in PHP by including the 'special character' "\t", so you just need to change every $write statement to append the tab character to each field value:

```
$write = fwrite($fp, $dayofbirth."\t");
```

Finally, we do not want a 'spare' tab at the end of the line, just the carriage-return/line-feed, so we don't add a tab to the last field and it remains:

```
$write = fwrite($fp, $other);
```