Stream Me Up, Scotty! (part 1)

By Vikram Vaswani

Table of Contents

File Transfer 101

Out here at Melonfire, we're big fans of PHP. We use it for a variety of reasons − Web site development, image generation, database connectivity − and we find it friendly, powerful and easy to use...all factors which come in handy when working against tight delivery deadlines.

You've already seen how PHP can be used to create GIF and JPEG images which dynamically respond to content from a database. But that's just the tip of the iceberg − the newest version of the language also comes with powerful functions that can dramatically ease the task of transferring files over the Web.

In this special two−part article, we're going to explore how PHP can be used to transfer files over HTTP and FTP connections, together with some simple applications that demonstrate just how powerful this really is. Keep reading!

Out With The Old...

The first thing you should know is that PHP allows file transfer across both HTTP and FTP connections. The ability to upload files over HTTP has been available since PHP3, while the newer FTP functions have made an appearance only in later versions of PHP3 and in PHP4.

Before we begin, you need to make sure that your version of PHP includes FTP support. You can check this by creating a PHP file containing the code

```
<? phpinfo(); ?>
```

and checking the resulting output through your Web browser. There's a section named "Additional Modules" which lists the modules supported by your build; in case you don't find the FTP module listed, you'll need to recompile your build with FTP support.

To start with, let's refresh your memory of what a typical FTP session looks like:

```
$ ftp ftp.server.com Connected to ftp.server.com 220
server.com FTP server ready. Name (server:john): john 331
Password required for john. Password: 230 User john logged in.
Remote system type is UNIX. Using binary mode to transfer
files. ftp> ls 200 PORT command successful. 150 Opening ASCII
mode data connection for /bin/ls. drwxr−xr−x 5 john users 3072
Nov 2 11:03 . drwxr−xr−x 88 root root 2048 Nov 1 23:26 ..
drwxr−−r−− 2 john users 1024 Oct 5 13:26 bin drwx−−x−−x 8 john
users 1024 Nov 2 10:59 public_html drwxr−−r−− 4 john users
1024 Nov 2 11:26 tmp −rw−r−−r−− 1 john users 2941465 Oct 9
17:21 data.zip 226 Transfer complete. ftp> bin 200 Type set to
I. ftp> get data.zip local: data.zip remote: data.zip 200 PORT
command successful. 150 Opening BINARY mode data connection
for data.zip(2941465 bytes). 226 Transfer complete. ftp> bye
221 Goodbye.
```

As you can see, the process can be broken down into clearly−identifiable segments. There's the connection phase (which opens a connection to the FTP server); the authentication phase (where the user identifies himself or herself and is permitted access to the FTP site); the transaction phase (which involves operations like directory navigation, file listing, and file GET or PUT); and the disconnection phase (which terminates the FTP connection cleanly). Nice and symmetrical, right?

...In With The New

Initiating an FTP connection in PHP follows the same basic principles: open an FTP connection, send authentication information, and then use built−in PHP functions to navigate through directories or transfer files. Let's take a look at the PHP version of the session you just saw.

```
<? // connect to FTP server $conn =
ftp_connect("ftp.server.com"); // log in with username and
password ftp_login($conn, "john", "doe"); // get remote system
type ftp_systype($conn); // obtain file listing $filelist =
ftp_nlist($conn, "."); // download file ftp_get($conn,
"data.zip", "data.zip", FTP_BINARY); // close connection
ftp_quit($conn); ?>
```

Let's go through this step by step:

In order to initiate an FTP connection, PHP offers the ftp_connect() function, which takes a host name and port number as parameters. In the example above, the host name is "ftp.server.com"; since a port is not specified, PHP will attempt a connection to the default FTP port, 21.

The ftp_connect() function returns a handle for the FTP connection if successful; this handle is used by all subsequent FTP functions.

```
<? // connect to FTP server $conn =
ftp_connect("ftp.server.com"); ?>
```

Once connected, the ftp_login() function is used to send the FTP server a user name and password. As you can see, ftp_login() also uses the handle returned by ftp_connect() to ensure that the authentication information is transmitted to the correct server.

```
<? // log in with username and password ftp_login($conn,
"john", "doe"); ?>
```

At this point, you can begin playing with some of the other functions available to you, specifically those that provide you with system information and directory listing (they're covered in detail on the next page).

Once you're finished, it's important that you remember to close your FTP session with ftp_quit().

```
<? // close connection ftp_quit($conn); ?>
```

Where Am I?

Once logged in to an FTP server, PHP offers a number of functions that can provide system−specific information, together with file and directory information.

First, the ftp_pwd() function comes in very useful if you need to find out where you're currently located in the directory tree.

```
<? // get current location $here = ftp_pwd($conn); ?>
```

In case you need to know the operating system that the FTP server is currently running, ftp_systype() will be able to give you this information.

```
<? // get system type $server_os = ftp_systype($conn); ?>
```

In case you need to switch passive (PASV) mode on or off, PHP has the ftp_pasv() function, which acts as a toggle switch, turning PASV mode on and off. In case you don't know what passive mode...don't worry about it!

```
<? // turn PASV on ftp_pasv($conn, 1); ?>
```

How about a file listing? Well, PHP offers two types of listings − a compact listing containing only file and directory names, and a long listing containing detailed information on file sizes, permissions and timestamps.

The first listing is provided by the ftp_nlist() function, while the second comes via ftp_rawlist(). Both functions require a directory name to be passed as a parameter, and both return the listing as an array. Each element of this array corresponds to one line of text from the listing.

```
<? // obtain file listing $filelist = ftp_nlist($conn, ".");
?>
```

And in case you're curious about file sizes, there's a very handy ftp_size() function which returns the size of the specified file in bytes. An important point to be noted here is that this function returns a value of −1 on error − a fact which comes in handy if you're trying to distinguish between files and directories, since this function, when invoked on a directory, typically returns a value of −1. You'll see this technique being used extensively in the example coming up later.

```
<? // obtain file size of file "data.zip" $filelist =
ftp_size($conn, "data.zip"); ?>
```

GETting It Right

Now that you know where you are and who's around you, it's time to start moving around in the directory tree − and the function that lets you do that is ftp_chdir(), which accepts a directory name as parameter.

```
<? // change directory to "public_html" ftp_chdir($conn,
"public_html"); ?>
```

If all you want to do is go up to the parent directory, an alternative way of accomplishing this is the ftp_cdup() command, which transports you one level up from wherever you are currently.

```
<? // go up one level in the directory tree ftp_cdup($conn);
?>
```

You can also create and remove directories with ftp_mkdir() and ftp_rmdir(); note that ftp_mkdir() returns the name of the new directory if successful.

```
<? // make the directory "test" ftp_mkdir($conn, "test"); //
remove the directory "test" ftp_rmdir($conn, "test"); ?>
```

The purpose of opening an FTP session is usually to transfer files − so let's get to that!

Uploading a file is accomplished by means of the ftp_put() command, which requires you to specify a name for the file on the remote system, the name of the file to be uploaded, and the type of transfer. For example, if you'd like to upload the file "abc.txt" and rename it to "xyz.txt" on the remote system, the command would look something like this:

```
<? // upload ftp_put($conn, "xyz.txt", "abc.txt", FTP_ASCII);
?>
```

Alternatively, if you'd like to download a file, the built−in function is ftp_get(), which also needs both remote and local file names, although the order is reversed. If the file to be downloaded is named "his.zip" on the remote system and you plan to save it as "hers.zip" on the local system, the syntax would change to

```
<? // download ftp_get($conn, "hers.zip", "his.zip",
FTP_BINARY); ?>
```

PHP defines two modes for file transfer − FTP_BINARY and FTP_ASCII − which need to be specified as they are displayed here, without quotes. Note that both ftp_get() and ftp_put() return result codes, indicating whether or not the transfer was successful.

Start Me Up!

Now, that covers the various FTP functions that PHP provides − but, just by itself, it isn't enough. The true power of these functions becomes visible only when they're combined into a single application which provides file transfer and file management capabilities. Since we're talking about FTP functions, it seems obvious that the best application for such a demonstration would be a Web−based FTP client − which is exactly what's coming up next!

Before we get into a code listing, I should make it clear that the example below is meant only to illustrate the various functions you've just learned. While it is functional and can be used as a "real" FTP client, I've omitted many basic error checking routines to make it easier to explain, and you should consider this use−at−your−own−risk software if you decide to implement it in a live environment. And cigarette smoking causes cancer. You have been warned.

The application code is separated across three different files:

index.html − the log in screen

actions.php4 − the code for all FTP transactions

include.php4 − the main interface, which displays file listings and control buttons

A complete code listing (with comments) is available at the end of the article − I'll simply be highlighting the important areas here.

Let's begin at the top, with "index.html":

```
<table border=0 align=center> <form action="actions.php4"
method=post> <input type=hidden name=action value=CWD> <tr>
<td> Server </td> <td> <input type=text name=server> </td>
</tr> <tr> <td> User </td> <td> <input type=text
name=username> </td> </tr> <tr> <td> Password </td> <td>
<input type=password name=password> </td> </tr> <tr> <td
colspan=2 align=center> <input type="submit" value="Beam Me
Up, Scotty!"> </td> </tr> </form> </table>
```

This is the log−in form, with fields for the FTP server name, the FTP user name and the password; these are stored in the variables $server, $username and $password respectively. Once the form is submitted, it calls the PHP script "actions.php4", which then takes over to initiate the FTP connection.

Note the hidden field, which sends the directive "action=CWD" to "actions.php4" − you'll see what role this plays on the next page.

Lights! Camera! Action!

The primary workhorse of the application is the file "actions.php4", which is reproduced below:

```
<html> <head> <basefont face=Arial> </head> <body> <!-- the
include.php4 interface will be inserted into this page --> <?
// check for valid form entries else print error if (!$server
|| !$username || !$password) { echo "Form data incomplete!"; }
else { // keep reading } ?> </body> </html>
```

Right up front, "actions.php4" checks if the various form variables exist − if not, it simply prints an error message. If the variables exist, the script will proceed to the "else" clause of the conditional statement.

Next come the various "actions". This script has been built to allow the following types of actions:

"action=CWD"

change working directory

"action=Delete"

delete selected file(s)

"action=Download"

download selected file(s)

"action=Upload"

upload selected file

If you take a look at the file "include.php4", which contains the HTML interface, you'll see that it consists of a number of forms, each one linked to a specific function. Each of these forms contains a field (usually hidden) which specifies the action to be taken when that specific form is submitted.

For example, the button marked "Delete", when clicked, sends the directive "action=Delete" to the PHP script "actions.php4", while the button "Go" sends the directive "action=CWD" to the script.

In order to handle these four actions, the meat of "actions.php4" is a set of branching conditional statements, like this:

```
<? // action: change directory if ($action == "CWD") { //
script code } // action: delete file(s) else if ($action ==
"Delete") { // script code } // action: download files else if
($action == "Download") { // script code } // action: upload
file else if ($action == "Upload") { // script code } ?>
```

Each of these branches includes code to connect to the FTP server, perform the selected action and exit.

Typically, this code follows a simple progression:

```
connect to the FTP server and log in through the user-defined
function connect(); change to the appropriate directory;
perform the selected action; refresh the list of available
files to reflect changes; display the file list and control
buttons through the include()d file "include.php4"; close the
FTP connection.
```

These sections are commented liberally in the code listing which follows.

Some points to be noted here:

The actions that deal with multiple files − namely, "action=Delete" and "action=Download" − use "for" loops to iterate through the array of selected files and delete or download each of them.

The variables $cdir and $here are refreshed at each stage to indicate the current directory.

Well−Formed Ideas

Let's now take a quick look at the third file, "include.php4", which sets up the user interface for the application.

"include.php4" contains three forms, together with some PHP code that takes the current list of files and separates it into three arrays:
$files (whichcontains a list of only the files in the current directory,
$file_sizes (which contains a list of the corresponding file sizes),
and $dirs (which contains a list of sub−directories)

The first of these uses the $dirs array to generate a drop−down list of available directories, and is linked to "action=CWD".

The second uses the $files and $file_sizes to create a list of available files, with a checkbox for each. This form is linked to "action=Delete" and "action=Download", and will send an array of selected files to "actions.php4" for processing.

You should note that downloaded files will appear in the directory where these scripts are running from. Since this directory is sure to be under your Web server root, it opens up a security hole, and it's therefore essential that you specify another location for the downloaded files to reside.

The third form is used specifically to upload a file to the FTP site, and therefore must be of the form

```
<form enctype="multipart/form−data" action=actions.php4
method=post> ... <input type=file name=upfile> ... </form>
```

When PHP receives a file encoded in this manner, a number of variables come into existence. These variables specify the size of the file, a temporary filename and the file type, and will be covered in detail in the second part of this article. For the moment, all you need to know is that the original filename is stored in the variable $upfile_name, while the temporary name of the file, once uploaded, is stored in the variable $upfile (this name is assigned by PHP).

Using this information, it's possible to construct the following ftp_put() statement:

```
ftp_put($result, $upfile_name, $upfile, FTP_BINARY);
```

Depending on whether or not the upload was successful, a message is generated and displayed the next time the page loads.

Note that it's not very difficult to add additional functions to this application − for example, you could easily add functions to create and delete directories, or to filter out files of a specific type. And in case you're still stuck with PHP3, or prefer the HTTP method of file transfer, tune in for the second part of this article, where I'll be showing you how to perform equivalent file management tasks with a combination of PHP's file manipulation functions and HTTP file transfer.

Appendix: Code Listing

```
<!-- code for index.html begins here --> <html> <head>
<basefont face=arial> </head> <body> <table border=0
align=center> <form action="actions.php4" method=post> <input
type=hidden name=action value=CWD> <tr> <td> Server </td> <td>
<input type=text name=server> </td> </tr> <tr> <td> User </td>
<td> <input type=text name=username> </td> </tr> <tr> <td>
Password </td> <td> <input type=password name=password> </td>
</tr> <tr> <td colspan=2 align=center> <input type="submit"
value="Beam Me Up, Scotty!"> </td> </tr> </form> </table>
</body> </html> <!-- code for index.html ends here -->
```

```
<!-- code for actions.php4 begins here --> <html> <head>
<basefont face=Arial> </head> <body> <? /*
```

DISCLAIMER:

This is use-at-your-own-risk code.

It is meant only for illustrative purposes and is not meant for production environments. No warranties of any kind are provided to the user.

You have been warned!


```
*/ // function to connect to FTP server function connect() {
global $server, $username, $password; $conn =
ftp_connect($server); ftp_login($conn, $username, $password);
return $conn; } // main program begins // check for valid form
entries else print error if (!$server || !$username ||
!$password) { echo "Form data incomplete!"; } else { //
connect $result = connect(); // action: change directory if
($action == "CWD") { // at initial stage $rdir does not exist
// so assume default directory if (!$rdir) { $path = "."; } //
get current location $cdir and add it to requested directory
$rdir else { $path = $cdir . "/" . $rdir; } // change to
requested directory ftp_chdir($result, $path); } // action:
delete file(s) else if ($action == "Delete") {
ftp_chdir($result, $cdir); // loop through selected files and
delete for ($x=0; $x<sizeof($dfile); $x++) {
ftp_delete($result, $cdir . "/" . $dfile[$x]); } } // action:
download files else if ($action == "Download") {
ftp_chdir($result, $cdir); // download selected files //
IMPORTANT: you should specify a different download location
```

```
here!! for ($x=0; $x<sizeof($dfile); $x++) { ftp_get($result,
$dfile[$x], $dfile[$x], FTP_BINARY); } } // action: upload
file else if ($action == "Upload") { ftp_chdir($result,
$cdir); // put file /* a better idea would be to use $res_code
= ftp_put($result, $HTTP_POST_FILES["upfile"]["name"],
$HTTP_POST_FILES["upfile"]["tmp_name"], FTP_BINARY); as it
offers greater security */ $res_code = ftp_put($result,
$upfile_name, $upfile, FTP_BINARY); // check status and
display if ($res_code == 1) { $status = "Upload successful!";
} else { $status = "Upload error!"; } } // create file list
$filelist = ftp_nlist($result, "."); // and display interface
include("include.php4"); // close connection
ftp_quit($result); } ?> </body> </html> <!-- code for
actions.php4 ends here -->
```

```
<!-- code for include.php4 begins here --> <? // get current
location $here = ftp_pwd($result); /* since ftp_size() is
quite slow, especially when working on an array containing all
the files in a directory, this section performs an ftp_size()
on all the files in the current directory and creates three
arrays. */ // array for files $files = Array(); // array for
directories $dirs = Array(); // array for file sizes
$file_sizes = Array(); // counters $file_list_counter = 0;
$dir_list_counter = 0; // check each element of $filelist for
($x=0; $x<sizeof($filelist); $x++) { if (ftp_size($result,
$filelist[$x]) != -1) { // create arrays
$files[$file_list_counter] = $filelist[$x];
$file_sizes[$file_list_counter] = ftp_size($result,
$filelist[$x]); $file_list_counter++; } else {
$dir_list[$dir_list_counter] = $filelist[$x];
$dir_list_counter++; } } ?> <!-- header - where am I? -->
<center> You are currently working in <b><? echo $here; ?></b>
<br> <!-- status message for upload function --> <? echo
$status; ?> </center> <hr> <p> <!-- directory listing in
drop-down list --> Available directories: <form
action=actions.php4 method=post> <!-- these values are passed
hidden every time --> <!-- a more optimal solution might be to
place these in session variables --> <input type=hidden
name=username value=<? echo $username; ?>> <input type=hidden
name=password value=<? echo $password; ?>> <input type=hidden
name=server value=<? echo $server; ?>> <input type=hidden
name=cdir value=<? echo $here; ?>> <!-- action to take when
THIS form is submitted --> <input type=hidden name=action
value=CWD> <!-- dir listing begins; first item is for parent
dir --> <select name=rdir> <option value=".."><parent
directory></option> <? for ($x=0; $x<sizeof($dir_list); $x++)
{ echo "<option value=" . $dir_list[$x] . ">" . $dir_list[$x]
. "</option>"; } ?> </select> <input type=submit value=Go>
```

```
</form> <hr> <!-- file listing begins --> Available files:
<form action=actions.php4 method=post> <!-- these values are
passed hidden every time --> <input type=hidden name=server
value=<? echo $server; ?>> <input type=hidden name=username
value=<? echo $username; ?>> <input type=hidden name=password
value=<? echo $password; ?>> <input type=hidden name=cdir
value=<? echo $here; ?>> <table border=0 width=100%> <? //
display file listing with checkboxes and sizes for ($y=0;
$y<sizeof($files); $y++) { echo "<tr><td><input type=checkbox
name=dfile[] value=" . $files[$y] . ">". $files[$y] . " <i>("
. $file_sizes[$y] . " bytes)</i><td>"; } ?> </table> <!--
actions for this form --> <center> <input type=submit
name=action value=Delete>   <input type=submit name=action
value=Download> </center> </form> <p> <hr> <!-- file upload
form --> File upload: <form enctype="multipart/form-data"
action=actions.php4 method=post> <!-- these values are passed
hidden every time --> <input type=hidden name=username
value=<? echo $username; ?>> <input type=hidden name=password
value=<? echo $password; ?>> <input type=hidden name=server
value=<? echo $server; ?>> <input type=hidden name=cdir
value=<? echo $here; ?>> <table> <tr> <td> <!-- file selection
box --> <input type=file name=upfile> </td> </tr> <tr> <td>
<!-- action for this form --> <input type=submit name=action
value=Upload> </td> </tr> </table> </form> <!-- code for
include.php4 ends here -->
```