UNIT-4

Advanced PHP Programming

I. PHP and Web Forms

There are two common methods for passing data from one script to another: GET and POST.
Although GET is the default, users want to use POST because it's capable of handling considerably more data, an important characteristic when using forms to insert and modify large blocks of text.

A Simple Example

The following script renders a form that prompts the user for his name and e-mail address. Once completed and submitted, the script (named subscribe.php) displays this information back to the browser window.

```
<?php
// If the name field is filled in
if (isset($_POST['name']))
{
$name = $_POST['name'];
$email = $_POST['email'];
printf("Hi %s! <br />", $name);
printf("The email address  is %s  <br />",
$email);
}
?>
<form action="subscribe.php"
method="post">
<p>
Name:<br />
<input type="text" id="name"
name="name" size="20" maxlength="40" />
</p>
<p>
Email Address:<br />
<input type="text" id="email"
name="email" size="20" maxlength="40" />
</p>
<input type="submit" id="submit" name =
"submit" value="Go!" />
</form>
```

Assuming that the user completes both fields and clicks the Go! button, output similar to the following will be displayed:

Hi Arifa!
The email address is arifa@jbiet.edu.in

II. Working with Files

1. The Concept of a Resource

The term *resource* is commonly used to refer to any entity from which an input or output stream can be initiated.
Standard input or output, files, and network sockets are all examples of resources.

2. Recognizing Newline Characters
The newline character, represented by the \n character sequence (\r\n on Windows), denotes the end of a line within a file.

3. Recognizing the End-of-File Character
The feof() function determines whether a resource's EOF has been reached. It is used quite commonly in file I/O operations. Its prototype follows:

int feof(string *resource*)

An example follows:

```
<?php
// Open a text file for reading purposes
$fh = fopen('/www/data/users.txt', 'r');
```

```
// While the end-of-file hasn't been
reached, retrieve the next line
while (!feof($fh)) echo fgets($fh);

// Close the file
fclose($fh);
?>
```

For opening and Closing a File users need to create a *handle* before anything can be done with file's contents.

4. Opening and closing a File
a. Opening a File

The fopen() function binds a file to a handle.

Its prototype follows:

resource fopen(string *resource*, string *mode* [, int *use_include_path* [, *resource context*]])

While fopen() is most commonly used to open files for reading and manipulation, it's also capable of opening resources via a number of protocols, including HTTP, HTTPS, and FTP.

The *mode*, assigned at the time a resource is opened, determines the level of access available to that resource.

Mode Description

R	Read-only. The file pointer is placed at the beginning of the file.
r+	Read and write. The file pointer is placed at the beginning of the file.
W	Write only. Before writing, delete the file contents and return the file pointer to the beginning of the file. If the file does not exist, attempt to create it.
w+	Read and write. Before reading or writing, delete the file contents and return the file pointer to the beginning of the file. If the file does not exist, attempt to create it.
A	Write only. The file pointer is placed at the end of the file. If the file does not exist, attempt to create it. This mode is better known as Append.
a+	Read and write. The file pointer is placed at the end of the file. If the file does not exist, attempt to create it. This process is known as *appending to the file*.
x	Create and open the file for writing only. If the file exists, fopen() will fail and an error of level E_WARNING will be generated.
x+	Create and open the file for writing and writing. If the file exists, fopen() will fail and an error of level E_WARNING will be generated.

If the resource is found on the local file system, PHP expects it to be available by the path prefacing it.

Alternatively, you can assign fopen()'s *use_include_path* parameter the value of 1, which will cause PHP to look for the resource within the paths specified by the include_path configuration directive.

The final parameter, *context*, is used for setting configuration parameters specific to the file or stream and for sharing file- or stream-specific information across multiple fopen() requests.

b. Closing a File

Good programming practice dictates that pointers should be destroyed to any resources when finished working with them.

The fclose() function closes the previously opened file pointer specified by a file handle, returning TRUE on success and FALSE otherwise. Its prototype follows:

boolean fclose(resource *filehandle*)

The filehandle must be an existing file pointer opened using fopen() or fsockopen().

5. Reading from a File

a. Reading a File into an Array

The file() function is capable of reading a file into an array, separating each element by the newline character, with the newline still attached to the end of each element. Its prototype follows:

array file(string *filename* [int *use_include_path* [, resource *context*]])

users.txt

```
Alice alice@example.com
Nicole nicole@example.com
Laura laura@example.com
```

The following script reads in users.txt and parses and converts the data into a convenient Web based format.

```
<?php
// Read the file into an array
$users = file('users.txt');

// Cycle through the array
foreach ($users as $user) {

// Parse the line, retrieving the name and e-mail address list($name, $email) = explode(' ', $user);

// Remove newline from $email
$email = trim($email);

// Output the formatted name and e-mail address echo "<a href = \ " mailto : $email \">$name</a> <br /> ";
}
?>
```

This script produces the following HTML output:

```
<a href="alice@example.com">Alice
</a><br />
<a href="nicole@example.com">
Nicole</a><br />
<a href="laura@example.com">
Laura</a><br />
```

b. Reading File Contents into a String Variable

The file_get_contents() function reads the contents of a file into a string. Its prototype follows:

string file_get_contents(string *filename* [, int *use_include_path* [, resource *context* [, int *offset* [, int *maxlen*]]]])

```
<?php
// Read the file into a string variable
$userfile= file_get_contents('users.txt');
// Place each line of $userfile into array
$users = explode("\n", $userfile);
// Cycle through the array
```

```
foreach ($users as $user) {
```

c. Reading a CSV File into an Array

The convenient fgetcsv() function parses each line of a file marked up in CSV format. Its prototype follows:

array fgetcsv(resource *handle* [, int *length* [, string *delimiter* [, string *enclosure*]]])

d. Reading a Specific Number of Characters

The **fgets()** function returns a certain number of characters read in through the opened resource handle, or everything it has read up to the point when a newline or an EOF character is encountered. Its prototype follows:

string fgets(resource *handle* [, int *length*])

```
<?php
// Open a handle to users.txt
$fh = fopen('/home/www/data/users.txt',
'r');
// While the EOF isn't reached, read in
another line and output it
while (!feof($fh)) echo fgets($fh);
// Close the handle
fclose($fh);
?>
```

e. Stripping Tags from Input

The fgetss() function operates similarly to fgets(), except that it also strips any HTML and PHP tags from the input. Its prototype follows:

string fgetss(resource *handle*, int *length* [, string *allowable_tags*])

f. Reading a File One Character at a Time

The fgetc() function reads a single character from the open resource stream specified by handle. If the EOF is encountered, a value of FALSE is returned. Its prototype follows:

string fgetc(resource *handle*)

g. Ignoring Newline Characters

The **fread()** function reads length characters from the resource specified by handle. Reading stops when the EOF is reached or when length characters have been read. Its prototype follows:

string fread(resource *handle*, int *length*)

Note that unlike other read functions, newline characters are irrelevant when using fread(), making it useful for reading binary files. Therefore, it's often convenient to read the entire file in at once using filesize() to determine the number of characters that should be read in:

h. Reading in an Entire File

The readfile() function reads an entire file specified by filename and immediately outputs it to the output buffer, returning the number of bytes read. Its prototype follows:

int readfile(string *filename* [, int *use_include_path*])

i.Reading a File According to a Predefined Format

The fscanf() function offers a convenient means for parsing a resource in accordance with a predefined format. Its prototype follows:

mixed fscanf(resource *handle*, string *format* [, string *var1*])

6.Writing a String to a File
The fwrite() function outputs the contents of a string variable to the specified resource. Its prototype follows:

int fwrite(resource *handle*, string *string* [, int *length*])

If the optional length parameter is provided, fwrite() will stop writing when length characters have been written. Otherwise, writing will stop when the end of the string is found.

7. Moving the File Pointer

a. Moving the File Pointer to a Specific Offset
The fseek() function moves the pointer to the location specified by a provided offset value. Its prototype follows:
int fseek(resource *handle*, int *offset* [, int *whence*])

If the optional parameter *whence* is omitted, the position is set offset bytes from the beginning of the file. Otherwise, *whence* can be set to one of three possible values, which affect the pointer's position:
SEEK_CUR: Sets the pointer position to the current position plus offset bytes.
SEEK_END: Sets the pointer position to the EOF plus offset bytes. In this case, offset must be set to a negative value.
SEEK_SET: Sets the pointer position to offset bytes. This has the same effect as omitting whence.

b.Retrieving the Current Pointer Offset
The ftell() function retrieves the current position of the file pointer's offset within the resource. Its prototype follows:

int ftell(resource *handle*)

c. Moving the File Pointer Back to the Beginning of the File
The rewind() function moves the file pointer back to the beginning of the resource. Its prototype follows:

int rewind(resource *handle*)

8. Reading Directory Contents
The process required for reading a directory's contents is quite similar to that involved in reading a file.

a.Opening a Directory Handle
Just as fopen() opens a file pointer to a given file, opendir() opens a directory stream specified by a path.Its prototype follows:

resource opendir(string *path* [, resource *context*])

b. Closing a Directory Handle
The closedir() function closes the directory stream. Its prototype follows:

void closedir(resource *directory_handle*)

c.Parsing Directory Contents
The readdir() function returns each element in the directory. Its prototype follows:

string readdir([resource *directory_handle*])

d.Reading a Directory into an Array
The scandir() function, introduced in PHP 5, returns an array consisting of files and directories found in directory or returns FALSE on error. Its prototype follows:

array scandir(string *directory* [,int *sorting_order* [, resource *context*]])

Setting the optional sorting_order parameter to 1 sorts the contents in descending order, overriding the default of ascending order.

III. PHP Authentication Methodologies

There are several ways you can implement authentication via a PHP script. In doing so, you should always consider the scope and complexity of your authentication needs. This section discusses four implementation methodologies:

- hard-coding a login pair directly into the script,
- using file-based authentication,
- using database-based authentication,
- using PEAR's HTTP authentication

Hard-Coded Authentication

The simplest way to restrict resource access is by hard-coding the username and password directly into the script

```
if (($_SERVER['PHP_AUTH_USER'] != 'client')
||
($_SERVER['PHP_AUTH_PW'] != 'secret')) {
header('WWW-Authenticate: Basic
Realm="Secret Stash"');
header('HTTP/1.0 401 Unauthorized');
print('You must provide the proper
credentials!');
exit;
}
```

In this example, if $_SERVER['PHP_AUTH_USER'] and $_SERVER['PHP_AUTH_PW'] are equal to client and secret, respectively, the code block will not execute, and anything ensuing that block will execute. Otherwise, the user is prompted for the username and password until either the proper information is provided or a 401 Unauthorized message is displayed due to multiple authentication failures.

Advantage:
Authentication against hard-coded values is very quick and easy to configure.

Disadvantages:

1. All users requiring access to that resource must use the same authentication pair. In most real-world situations, each user must be uniquely identified so that user-specific preferences or resources can be provided.
2. Changing the username or password can be done only by entering the code and making the manual adjustment.

File-Based Authentication

Often you need to provide each user with a unique login pair in order to track user-specific login times, movements, and actions. This is easily accomplished with a text file, much like the one commonly used to store information about Unix users (/etc/passwd).

Each line contains a username and an encrypted password pair, with the two elements separated by a colon.

The authenticationFile.txt File Containing Encrypted Passwords

```
jason:60d99e58d66a5e0f4f89ec3ddd1d9a8
donald:d5fc4b0e45c8f9a333c0056492c191c
mickey:bc180dbc583491c00f8a1cd134f751
```

A crucial security consideration regarding authenticationFile.txt is that this file should be stored outside the server document root. If it's not, an attacker could discover the file through brute-force guessing, revealing half of the login combination.

The PHP script required to parse this file and authenticate a user against a given login pair is only a tad more complicated than the script used to authenticate against a hard-coded authentication pair. The difference lies in the script's additional duty of reading the text file into an array, and then cycling through that array searching for a match. This involves the use of several functions, including the following:

- **file(string *filename*):** The file() function reads a file into an array, with each element of the array consisting of a line in the file.
- **explode(string *separator*, string *string* [, int *limit*]):** The explode() function splits a string into a series of substrings, with each string boundary determined by a specific separator.
- **md5(string *str*):** The md5() function calculates an MD5 hash of a string, using RSA Security Inc.'s MD5 Message-Digest algorithm (www.rsa.com). Because the passwords are stored using the same encrypted format, you first use the md5() function to encrypt the provided password, comparing the result with what is stored locally.

```
<?php
// Preset authentication status to false
$authorized = FALSE;
if (isset($_SERVER['PHP_AUTH_USER']) &&
isset($_SERVER['PHP_AUTH_PW'])) {
// Read the authentication file into an array
$authFile =
file("/usr/local/lib/php/site/authenticate.txt
");
// Search array for authentication match
// If using Windows, use \r\n
if (in_array($_SERVER['PHP_AUTH_USER'].
":"
.md5($_SERVER['PHP_AUTH_PW'])."\n",
$authFile))
$authorized = TRUE;
}
// If not authorized, display authentication
prompt or 401 error
if (! $authorized) {
header('WWW-Authenticate: Basic
Realm="Secret Stash"');
header('HTTP/1.0 401 Unauthorized');
print('You must provide the proper
credentials!');
exit;
}
// restricted material goes here...
?>
```

Disadvantage:
This strategy can quickly become inconvenient when you're handling a large number of users; when users are regularly being added, deleted, and modified; or when you need to incorporate an authentication scheme into a larger information infrastructure such as a preexisting user table. Such requirements are better satisfied by implementing a database-based solution.

Database-Based Authentication

Implementing a database driven solution is the most powerful because it not only enhances administrative convenience and scalability, but also can be integrated into a larger database infrastructure

```
CREATE TABLE logins (
id INTEGER UNSIGNED NOT NULL
AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(255) NOT NULL,
pswd CHAR(32) NOT NULL
);
```

A few lines of sample data follow:

```
id username password
1 wjgilmore
098f6bcd4621d373cade4e832627b4f6
2 mwade
0e4ab1a5a6d8390f09e9a0f2d45aeb7f
3 jgennick
3c05ce06d51e9498ea472691cd811fb6
```

```
<?php
/* Because the authentication prompt
needs to be invoked twice,
embed it within a function.
*/
function authenticate_user() {
header('WWW-Authenticate: Basic
realm="Secret Stash"');
header("HTTP/1.0 401 Unauthorized");
exit;
}
/* If $_SERVER['PHP_AUTH_USER'] is blank,
the user has not yet been
prompted for the authentication
information.
*/
if (! isset($_SERVER['PHP_AUTH_USER'])) {
authenticate_user();
} else {
$db = new mysqli("localhost", "webuser",
"secret", "chapter14");
$stmt = $db->prepare("SELECT username,
pswd FROM logins WHERE username=?
AND pswd=MD5(?)");
$stmt->bind_param('ss',
$_SERVER['PHP_AUTH_USER'],
$_SERVER['PHP_AUTH_PW']);
$stmt->execute();
$stmt->store_result();
if ($stmt->num_rows == 0)
authenticate_user();
}
?>
```

Although database authentication is more powerful than the previous two methodologies described, it is really quite trivial to implement. Simply execute a selection query against the logins table, using the entered username and password as criteria for the query. Of course, such a solution is not dependent upon specific use of a MySQL database; any relational database could be used in its place.

Taking Advantage of PEAR: Auth_HTTP

While the approaches to authentication discussed thus far work just fine, it's always nice to hide some of the implementation details within a class. The PEAR class Auth_HTTP satisfies this desire quite nicely, taking advantage of Apache's authentication mechanism and prompt to produce an identical prompt but using PHP to manage the authentication information. Auth_HTTP encapsulates many of the messy aspects of user authentication, exposing the information and features you're looking for by way of a convenient interface. Furthermore, because it inherits from the Auth class, Auth_HTTP also offers a broad range of authentication storage mechanisms, some of which include the DB database abstraction package, LDAP, POP3, IMAP, RADIUS, and SAMBA.

Installing Auth_HTTP

To take advantage of Auth_HTTP's features, you need to install it. Therefore, invoke PEAR and pass it the following arguments:
%>pear install -o auth_http

Authenticating Against a MySQL Database
Because Auth_HTTP subclasses the Auth package, it inherits all of Auth's capabilities. Because Auth subclasses the DB package, Auth_HTTP can take advantage of this popular database abstraction layer to store authentication information in a database table. To store the information, this example uses a table identical to one used earlier in this chapter:

```
CREATE TABLE logins (
id INTEGER UNSIGNED NOT NULL
AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(255) NOT NULL,
pswd CHAR(32) NOT NULL
);
```

Next, you need to create a script that invokes Auth_HTTP, telling it to refer to a MySQL database.

```
<?php
require_once("Auth/HTTP.php");
// Designate authentication credentials,
table name,
// username and password columns,
password encryption type,
// and query parameters for retrieving
other fields
$dblogin = array (
'dsn' =>
"mysqli://webuser:secret@localhost/chapt
er14",
'table' => "logins",
'usernamecol' => "username",
'passwordcol' => "pswd",
'cryptType' => "md5",
'db_fields' => "*"
);
// Instantiate Auth_HTTP
$auth = new Auth_HTTP("MDB2", $dblogin)
or die("Can't connect!");
// Message to provide in case of
authentication failure
$auth->setCancelText('Authentication
credentials not accepted!');
// Begin the authentication process
$auth->start();
// Check for credentials. If not available,
prompt for them
if($auth->getAuth())
echo "Welcome, {$auth-
>getAuthData('username')}<br />";
?>
```

Executing the script and passing along information matching that found in the logins table allows the user to pass into the restricted area. Otherwise, the error message supplied in setCancelText() is displayed.

IV. User Login Administration

When you incorporate user logins into your application, providing a sound authentication mechanism is only part of the total picture. How do you ensure that the user chooses a sound password of sufficient difficulty that attackers cannot use it as a possible attack route? Furthermore, how do you deal with the inevitable event of the user forgetting his password?

Testing Password Guessability with the CrackLib Library

In an ill-conceived effort to prevent forgetting their passwords, users tend to choose something easy to remember, such as the name of their dog, their mother's

maiden name, or even their own name or age.

Ironically, this practice often doesn't help users to remember the password and, even worse, offers attackers a rather simple route into an otherwise restricted system, either by researching the user's background and attempting various passwords until the correct one is found, or by using brute force to discern the password through numerous repeated attempts. In either case, the password typically is broken because the user has chosen a password that is easily guessable, resulting in the possible compromise of not only the user's personal data, but also the system itself.

Reducing the possibility that easily guessable passwords could be introduced into your system is quite simple; you turn the procedure of unchallenged password creation into one of automated password approval. PHP offers a wonderful means for doing so via the CrackLib library, created by Alec Muffett (www.crypticide.com). CrackLib is intended to test the strength of a password by setting certain benchmarks that determine its guessability, including:

- **Length:** Passwords must be longer than four characters.
- **Case:** Passwords cannot be all lowercase.
- **Distinction:** Passwords must contain adequate different characters. In addition, the password cannot be blank.
- **Familiarity:** Passwords cannot be based on a word found in a dictionary. In addition, passwords cannot be based on the reverse spelling of a word found in the dictionary.
- **Standard numbering:** Because CrackLib's author is British, he thought it a good idea to check against patterns similar to what is known as a National Insurance (NI) number. The NI number is used in Britain for taxation, much like the Social Security number (SSN) is used in the United States. Coincidentally, both numbers are nine characters long, allowing this mechanism to efficiently prevent the use of either, if a user is naive enough to use such a sensitive identifier for this purpose.

Using the CrackLib Extension

Using PHP's CrackLib extension is quite easy.

```
<?php
$pswd = "567hejk39";
/* Open the dictionary. Note that the
dictionary
filename does NOT include the extension.
*/
$dictionary =
crack_opendict('/usr/lib/cracklib_dict');

// Check password for guessability
$check = crack_check($dictionary, $pswd);
// Retrieve outcome
echo crack_getlastmessage();
// Close dictionary
crack_closedict($dictionary);
?>
```

In this particular example, crack_getlastmessage() returns the string "strong password" because the password denoted by $pswd is sufficiently difficult to guess. However, if the password is weak, one of a number of different messages could be returned.

Password Response

Mary It is too short.
12 It's WAY too short.

1234567 It is too simplistic/systematic. Street It does not contain enough DIFFERENT characters.

One-Time URLs and Password Recovery

A one-time URL is commonly given to a user to ensure uniqueness when no other authentication mechanisms are available, or when the user would find authentication perhaps too tedious for the task at hand.

Ex:
http://www.example.com/newsletter/0503.php?id=9b758e7f08a2165d664c2684fddbcde2

Suppose the users forget his password and thus click the Forgot password? link, commonly found near a login prompt. The user arrives at a page in which he is asked to enter his e-mail address. Upon entering the address and submitting the form, a script similar to that shown in

```
<?php
$db = new mysqli("localhost", "webuser", "secret", "chapter14");
// Create unique identifier
$id = md5(uniqid(rand(),1));
// User's email address
$address = filter_var($_POST[email], FILTER_SANITIZE_EMAIL);
// Set user's hash field to a unique id
$stmt = $db->prepare("UPDATE logins SET hash=? WHERE email=?");
$stmt->bind_param('ss', $id, $address);
$stmt->execute();
$email = <<< email
Dear user,
Click on the following link to reset your password:
http://www.example.com/users/lostpassword.php?id=$id
email;
// Email user password reset options
mail($address,"Password recovery","$email","FROM:services@example.com");
echo "<p>Instructions regarding resetting your password have been sent to $address</p>";
?>
```

When the user receives this e-mail and clicks the link, the script lostpassword.php,

Resetting a User's Password

```
<?php
$db = new mysqli("localhost", "webuser", "secret", "chapter14");
// Create a pseudorandom password five characters in length
$pswd = substr(md5(uniqid(rand())),5);
// User's hash value
$id = filter_var($_GET[id], FILTER_SANITIZE_STRING);
// Update the user table with the new password
$stmt = $db->prepare("UPDATE logins SET pswd=? WHERE hash=?");
$stmt->execute();

// Display the new password
echo "<p>Your password has been reset to {$pswd}.</p>";
?>
```

V. Uploading Files with PHP

Successfully managing file uploads via PHP is the result of cooperation between various configuration directives, the $_FILES superglobal, and a properly coded web form.

PHP's File Upload/Resource Directives

1. **file_uploads = *On | Off***

Scope: PHP_INI_SYSTEM; Default value: On
The file_uploads directive determines whether PHP scripts on the server can accept file uploads.

2. **max_input_time = *integer***

Scope: PHP_INI_ALL; Default value: 60
The max_input_time directive determines the maximum amount of time, in seconds, that a PHP script will spend attempting to parse input before registering a fatal error. This is relevant because particularly large files can take some time to upload, eclipsing the time limit set by this directive.

3. **max_file_uploads = *integer***

Scope: PHP_INI_SYSTEM; Default value: 20
Available since PHP 5.2.12, the max_file_uploads directive sets an upper limit on the number of files which can be simultaneously uploaded.

4. **memory_limit = *integerM***

Scope: PHP_INI_ALL; Default value: 16M
The memory_limit directive sets a maximum allowable amount of memory in megabytes that a script can allocate (note that the integer value must be followed by M for this setting to work properly). It prevents runaway scripts from monopolizing server memory and even crashing the server in certain situations.

5. **post_max_size = *integer*M**

Scope: PHP_INI_PERDIR; Default value: 8M
The post_max_size places an upper limit on the size of data submitted via the POST method. Because files are uploaded using POST, you may need to adjust this setting upwards along with upload_max_filesize when working with larger files.

6. **upload_max_filesize = *integerM***

Scope: PHP_INI_PERDIR; Default value: 2M
The upload_max_filesize directive determines the maximum size in megabytes of an uploaded file. This directive should be smaller than post_max_size because it applies only to information passed via the file input type and not to all information passed via the POST instance.

upload_tmp_dir = *string*

Scope: PHP_INI_SYSTEM; Default value: NULL
Because an uploaded file must be successfully transferred to the server before subsequent processing on that file can begin, a staging area of sorts must be designated for such files where they can be temporarily placed until they are moved to their final location. This staging location is specified using the upload_tmp_dir directive.

The $_FILES Array

The $_FILES superglobal stores a variety of information pertinent to a file uploaded to the server via a PHP script. In total, five items are available in this array, each of which is introduced here:

- **$_FILES['userfile']['error']:** This array value offers important information pertinent to the outcome of the upload attempt. In total, five return values are possible: one signifying a successful outcome and four others denoting specific errors that arise from the attempt. The name and meaning of each return value is introduced in the "Upload Error Messages".
  - **$_FILES['userfile']['name']:** This variable specifies the original name of the file, including the extension, as declared on the client machine. Therefore, if you browse to a file named vacation.png and upload it via the form, this variable will be assigned the value vacation.png.
  - **$_FILES['userfile']['size']:** This variable specifies the size, in bytes, of the file uploaded from the client machine. For example, in the case of the vacation.png file, this variable could plausibly be assigned a value such as 5253, or roughly 5KB.
  - **$_FILES['userfile']['tmp_name']:** This variable specifies the temporary name assigned to the file once it has

been uploaded to the server. This is the name of the file assigned to it while stored in the temporary directory (specified by the PHP directive upload_tmp_dir).

- **$_FILES['userfile']['type']:** This variable specifies the MIME type of the file uploaded from the client machine. Therefore, in the case of the vacation.png image file, this variable would be assigned the value image/png. If a PDF was uploaded, the value application/pdf would be assigned. Because this variable sometimes produces unexpected results, you should explicitly verify it yourself from within the script.

PHP's File-Upload Functions

In addition to the number of file-handling functions made available via PHP's file system library, PHP offers two functions specifically intended to aid in the fileupload process, is_uploaded_file() and move_uploaded_file().

Determining Whether a File Was Uploaded

The is_uploaded_file() function determines whether a file specified by the input parameter filename is uploaded using the POST method. Its prototype follows:
boolean is_uploaded_file(string *filename*)

This function is intended to prevent a potential attacker from manipulating files not intended for interaction via the script in question. For example, consider a scenario in which uploaded files are made immediately available for viewing via a public site repository. Say an attacker wants to make a file somewhat juicier than the boring old class notes available for his perusal, say /etc/passwd. Rather than navigate to a class notes file as would be expected, the attacker instead types /etc/passwd directly into the form's file-upload field.

Moving an Uploaded File

The move_uploaded_file() function provides a convenient means for moving an uploaded file from the temporary directory to a final location. Its prototype follows:
boolean move_uploaded_file(string *filename*, string *destination*)
Although copy() works equally well, move_uploaded_file() offers one additional feature: it will check to ensure that the file denoted by the filename input parameter was in fact uploaded via PHP's HTTP POST upload mechanism. If the file has not been uploaded, the move will fail and a FALSE value will be returned. Because of this, you can forgo using is_uploaded_file() as a precursor condition to using move_uploaded_file().

Upload Error Messages

- **UPLOAD_ERR_OK**: A value of 0 is returned if the upload is successful.
- **UPLOAD_ERR_INI_SIZE:** A value of 1 is returned if there is an attempt to upload a file whose size exceeds the value specified by the upload_max_filesize directive.
- **UPLOAD_ERR_FORM_SIZE:** A value of 2 is returned if there is an attempt to upload a file whose size exceeds the value of the max_file_size directive, which can be embedded into the HTML form
- **UPLOAD_ERR_PARTIAL:** A value of 3 is returned if a file is not completely uploaded.This might happen if a network error causes a disruption of the upload process.

- **UPLOAD_ERR_NO_FILE**: A value of 4 is returned if the user submits the form without specifying a file for upload.
- **UPLOAD_ERR_NO_TMP_DIR:** A value of 6 is returned if the temporary directory does not exist.
- **UPLOAD_ERR_CANT_WRITE:** Introduced in PHP 5.1.0, a value of 7 is returned if the file can't be written to the disk.
- **UPLOAD_ERR_EXTENSION:** Introduced in PHP 5.2.0, a value of 8 is returned if an issue with PHP's configuration caused the upload to fail.

A Simple Example

To formalize the scenario, suppose that a professor invites students to post class notes to his web site, the idea being that everyone might have something to gain from such a collaborative effort. Of course, credit should nonetheless be given where credit is due, so each file upload should be renamed to the last name of the student. In addition, only PDF files are accepted.

```
<form action="listing15-1.php"
enctype="multipart/form-data"
method="post">
<label form="email">Email:</label><br />
<input type="text" name="email" value=""
/><br />
<label form="classnotes">Class
notes:</label><br />
<input type="file" name="classnotes"
value="" /><br />
<input type="submit" name="submit"
value="Submit Notes" />
</form>
<?php
// Set a constant
define
("FILEREPOSITORY","/var/www/4e/15/class
notes");
// Make sure that the file was POSTed.
if
(is_uploaded_file($_FILES['classnotes']['tmp
_name'])) {
// Was the file a PDF?
if ($_FILES['classnotes']['type'] !=
"application/pdf") {
echo "<p>Class notes must be uploaded in
PDF format.</p>";
} else {
// Move uploaded file to final destination.
$name = $_POST['name'];
$result =
move_uploaded_file($_FILES['classnotes']['t
mp_name'],
FILEREPOSITORY.$_FILES['classnotes']['nam
e']);
if ($result == 1) echo "<p>File successfully
uploaded.</p>";
else echo "<p>There was a problem
uploading the file.</p>";
}
}
?>
```

VI. Sending E-mail Using a PHP Script

E-mail can be sent through a PHP script in amazingly easy fashion, using the mail() function. Its prototype follows:

boolean mail(string *to*, string *subject*, string *message* [, string *addl_headers* [, string *addl_params*]])

The mail() function can send an e-mail with a subject and a message to one or several recipients.You can tailor many of the e-mail properties using the addl_headers parameter; you can even modify your SMTP server's behavior by passing extra flags via the addl_params parameter.

On the Unix platform, PHP's mail() function is dependent upon the sendmail MTA. If you're using an alternative MTA (e.g., qmail), you need to use that MTA's sendmail wrappers. PHP's Windows implementation of the function depends upon establishing a socket connection to an MTA designated by the SMTP configuration directive.

Sending a Plain-Text E-mail

Sending the simplest of e-mails is trivial using the mail() function, done using just the three required parameters, in addition to the fourth parameter which allows you to identify a sender. Here's an example:

```
<?php
mail("test@example.com", "This is a subject", "This is the mail body",
"From:admin@example.com\r\n");
?>
```

Taking Advantage of PEAR: Mail and Mail_Mime

While it's possible to use the mail() function to perform more complex operations such as sending to multiple recipients, annoying users with HTML-formatted e-mail, or including attachments, doing so can be a tedious and error-prone process. However, the Mail (http://pear.php.net/package/Mail) and Mail_Mime (http://pear.php.net/package/Mail_Mime) PEAR packages make such tasks a breeze. These packages work in conjunction with one another: Mail_Mime creates the message, and Mail sends it. This section introduces both packages.

Installing Mail and Mail_Mime

To take advantage of Mail and Mail_Mime, you'll first need to install both packages. To do so, invoke PEAR and pass along the following arguments:

```
%>pear install Mail Mail_Mime
```

Sending an HTML-Formatted E-mail

```
<?php
// Include the Mail and Mime_Mail
Packages
include('Mail.php');
include('Mail/mime.php');
// Recipient Name and E-mail Address
$name = "Jason Gilmore";

$recipient = "jason@example.org";
// Sender Address
$from = "bram@example.com";
// Message Subject
$subject = "Thank you for your inquiry -
HTML Format";
// E-mail Body
$html = <<<html
<html><body>
<h3>Example.com Stamp Company</h3>
<p>
Dear $name,<br />
Thank you for your interest in
<b>Example.com's</b> fine selection of
collectible stamps. Please respond at your
convenience with your telephone
number and a suggested date and time to
chat.
</p>
<p>I look forward to hearing from you.</p>
<p>
Sincerely,<br />
Bram Brownstein<br />
President, Example.com Stamp Supply
html;
// Identify the Relevant Mail Headers
$headers['From'] = $from;
$headers['Subject'] = $subject;
// Instantiate Mail_mime Class
$mimemail = new Mail_mime();
// Set HTML Message
```

```
$mimemail->setHTMLBody($html);
// Build Message
$message = $mimemail->get();
// Prepare the Headers
$mailheaders = $mimemail-
>headers($headers);
// Create New Instance of Mail Class
$email =& Mail::factory('mail');
// Send the E-mail Already!
$email->send($recipient, $mailheaders,
$message) or die("Can't send message!");
?>
```

Sending an Attachment

Mail_Mime object's addAttachment() method is used for passing in the attachment name and extension, and dentifying its content type:

```
$mimemail-
>addAttachment('inventory.pdf',
'application/pdf');
```

PHP's Encryption Functions

Encryption over the Web is largely useless unless the scripts running the encryption schemes are operating on an SSL-enabled server. Why? PHP is a server-side scripting language, so information must be sent to the server in plain-text format *before* it can be encrypted. There are many ways that an unwanted third party can watch this information as it is transmitted from the user to the server if the user is not operating via a secured connection.

Encrypting Data with the md5() Hash Function

The md5() function uses MD5, a third-party hash algorithm often used for creating digital signatures . Digital signatures can, in turn, be used to uniquely identify the sending party.

MD5 is considered to be a *one-way* hashing algorithm, which means there is no practical way to dehash data that has been hashed using md5(). Its prototype looks like this:

string md5(string *str*)

The MD5 algorithm can also be used as a password verification system. Because it is (in theory)
extremely difficult to retrieve the original string that has been hashed using the MD5 algorithm, you could hash a given password using MD5 and then compare that encrypted password against those that a user enters to gain access to restricted information.

For example, assume that your secret password *toystore* has an MD5 hash of 745e2abd7c52ee1dd7c14ae0d71b9d76. You can store this hashed value on the server and compare it to the MD5 hash equivalent of the password the user attempts to enter. Even if an intruder gets hold of the encrypted password, it wouldn't make much difference because that intruder can't return the string to its original format through conventional means.

An example of hashing a string using md5() follows:

```
<?php
$val = "secret";
$hash_val = md5 ($val);
// $hash_val =
"5ebe2294ecd0e0f08eab7690d2a6ee69";
?>
```

The MCrypt Package

MCrypt is a popular data-encryption package available for use with PHP, providing support for two-way encryption (i.e., encryption and decryption). Before you can use it, you need to follow these installation instructions:

1. Go to http://mcrypt.sourceforge.net and download the package source.
2. Extract the contents of the compressed distribution and follow the installation instructions as specified in the INSTALL document.
3. Compile PHP with the --with-mcrypt option.

MCrypt supports the following encryption algorithms:

• ARCFOUR • ENIGMA • RC (2, 4)
• TEAN • ARCFOUR_IV • GOST
• RC6 (128, 192, 256) • THREEWAY
• BLOWFISH • IDEA
• RIJNDAEL (128, 192, 256) • 3DES
• CAST • LOKI97 • SAFER (64, 128, and PLUS) • TWOFISH (128, 192, and 256) • CRYPT • MARS
• SERPENT (128, 192, and 256)
• WAKE • DES • PANAMA
• SKIPJACK • XTEA

Encrypting Data with MCrypt

The mcrypt_encrypt() function encrypts the provided data, returning the encrypted result. The prototype follows:

string mcrypt_encrypt(string *cipher*, string *key*, string *data*, string *mode* [, string *iv*])

The provided cipher names the particular encryption algorithm, and the parameter *key* determines the key used to encrypt the data. The *mode* parameter specifies one of the six available encryption modes:

- electronic codebook,
- cipher block chaining,
- cipher feedback,
- 8-bit output feedback,
- N-bit output feedback,
- special stream mode.

Each is referenced by an abbreviation: ecb, cbc, cfb, ofb, nofb, and stream, respectively.

Finally, the *iv* parameter initializes cbc, cfb, ofb, and certain algorithms used instream mode. Consider an example:

```
<?php
$ivs = mcrypt_get_iv_size(MCRYPT_DES,
MCRYPT_MODE_CBC);
$iv = mcrypt_create_iv($ivs,
MCRYPT_RAND);
$key = "F925T";
$message = "This is the message I want to
encrypt.";
$enc = mcrypt_encrypt(MCRYPT_DES, $key,
$message, MCRYPT_MODE_CBC, $iv);
echo bin2hex($enc);
?>
```

This returns the following:

```
f5d8b337f27e251c25f6a17c74f93c5e9a8a2
1b91f2b1b0151e649232b486c93b36af4679
14bc7d8
```

You can then decrypt the text with the mcrypt_decrypt() function.

Decrypting Data with MCrypt

The mcrypt_decrypt() function decrypts a previously encrypted cipher, provided that

the cipher, key,and mode are the same as those used to encrypt the data. Its prototype follows:
string mcrypt_decrypt(string *cipher*, string *key*, string *data*, string *mode* [, string *iv*])

Insert the following line into the previous example, directly after the last statement:
echo mcrypt_decrypt(MCRYPT_DES, $key, $enc, MCRYPT_MODE_CBC, $iv);

This returns the following:

This is the message I want to encrypt.

Building Web Sites for the World

Translating Web Sites with Gettext
Gettext (www.gnu.org/software/gettext), one of the many great projects created and maintained by the Free Software Foundation, consists of a number of utilities useful for internationalizing and localizing software. Over the years it's become a de facto standard solution for maintaining translations for countless applications and web sites. PHP interacts with gettext through a namesake extension, meaning you need to download the gettext utility and install it on your system. If you're running Windows,download it from http://gnuwin32.sourceforge.net and make sure you update the PATH environment variable to point to the installation directory.
Because PHP's gettext extension isn't enabled by default, you probably need to reconfigure PHP. If you're on Linux, you can enable it by rebuilding PHP with the --with-gettext option. On Windows, just uncomment the php_gettext.dll line found in the php.ini file.

Step 1: Update the Web Site Scripts

Gettext must be able to recognize which strings you'd like to translate. This is done by passing all translatable output through the gettext() function. Each time gettext() is encountered, PHP will look to the language-specific localization repository (more about this in Step 2) and match the string encompassed within the function to the corresponding translation. The script knows which translation to retrieve due to earlier calls to setlocale(), which tells PHP and gettext which language and country you want to conform to, and then to bindtextdomain() and textdomain(), which tell PHP where to look for the translation files.

Common Country and Language Code Combinations

Combination	Locale
pt_BR	Brazil
fr_FR	France
de_DE	Germany
en_GB	Great Britain
he_IL	Israel
it_IT	Italy
es_MX	Mexico
es_ES	Spain
en_US	United States

Presents a simple example that seeks to translate the string **Enter your email address:** to its Italian equivalent.

```
<?php
// Specify the target language
$language = 'it_IT';
// Assign the appropriate locale
setlocale(LC_ALL, $language);
// Identify the location of the translation
files
bindtextdomain('messages',
'/usr/local/apache/htdocs/locale');
```

```
// Tell the script which domain to search within when translating text
textdomain('messages');
?>
<form action="subscribe.php" method="post">
<?php echo gettext("Enter your e-mail address:"); ?><br />
<input type="text" id="email" name="email" size="20" maxlength="40" value="" />
<input type="submit" id="submit" value="Submit" />
</form>
```

Step 2: Create the Localization Repository

Next, you need to create the repository where the translated files will be stored. One directory should be created for each language/country code combination, and within that directory you need to create another directory named LC_MESSAGES. So if you plan on localizing the web site to support English (the default), German, Italian, and Spanish, the directory structure would look like this:

```
locale/
de_DE/
LC_MESSAGES/
it_IT/
LC_MESSAGES/
es_ES/
LC_MESSAGES/
```

Step 3: Create the Translation Files

Next, you need to extract the translatable strings from the PHP scripts. You do so with the **xgettext** command, which is a utility bundled with gettext. Executing the following command will cause xgettext to examine all of the files found in the current directory ending in .php, producing a file consisting of the desired strings to translate:

```
%>xgettext -n *.php
```

The -n option results in the file name and line number being included before each string entry in the output file. By default, the output file is named messages.po, although you can change this using the --default-domain=FILENAME option.

A sample output file follows:

```
# SOME DESCRIPTIVE TITLE.
# Copyright (C) YEAR THE PACKAGE'S COPYRIGHT HOLDER
# This file is distributed under the same license as the PACKAGE package.
# FIRST AUTHOR <EMAIL@ADDRESS>, YEAR.
#
#, fuzzy
msgid ""
msgstr ""
"Project-Id-Version: PACKAGE VERSION\n"
"Report-Msgid-Bugs-To: \n"
"POT-Creation-Date: 2010-05-16 13:13-0400\n"


"PO-Revision-Date: YEAR-MO-DA HO:MI+ZONE\n"
"Last-Translator: FULL NAME <EMAIL@ADDRESS>\n"
"Language-Team: LANGUAGE <LL@li.org>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=CHARSET\n"
"Content-Transfer-Encoding: 8bit\n"
#: homepage.php:12
msgid "Subscribe to the newsletter:"
msgstr ""
#: homepage.php:15
msgid "Enter your e-mail address:"
msgstr ""
#: contact.php:12
msgid "Contact us at info@example.com!"
```

msgstr ""

Step 4: Translate the Text
Open the messages.po file residing in the language directory you'd like to translate, and translate the strings by completing the empty msgstr entries that correspond to an extracted string. Then replace the placeholders represented in all capital letters with information pertinent to your application.

Step 5: Generate Binary Files
The final required preparatory step involves generating binary versions of the messages.po files, which will be used by gettext. This is done with the msgfmt command. Navigate to the appropriate language directory and execute the following command:
%>msgfmt messages.po

Executing this command produces a file named messages.mo, which is what gettext will ultimately use for the translations.

Step 6: Set the Desired Language Within Your Scripts
To begin taking advantage of your localized strings, all you need to do is set the locale using setlocale() and call the bindtextdomain() and textdomain() functions. The end result is the ability to use the same code source to present your web site in multiple languages.

Localizing Dates, Numbers, and Times

The setlocale() function can also affect how PHP renders dates, numbers, and times. This is important because of the variety of ways in which this often crucial data is represented among different countries. For example, suppose you are a United States–based organization providing an essential subscription based service to a variety of international corporations. When it is time to renew subscriptions, a special message is displayed at the top of the browser that looks like this:

Your subscription ends on 3-4-2011. Renew soon to avoid service cancellation.

For the United States–based users, this date means March 4, 2011. However, for European users, this date is interpreted as April 3, 2011. The result could be that the European users won't feel compelled to renew the service until the end of March, and therefore will be quite surprised when they attempt to log in on March 5. This is just one of the many issues that might arise due to confusion over data representation.

You can eliminate such inconsistencies by localizing the information so that it appears exactly as the user expects. PHP makes this a fairly easy task, done by setting the locale using setlocale(), and then using functions such as money_format(), number_format(), and strftime() per usual to output the data. For example, suppose you want to render the renewal deadline date according to the user's locale.
Just set the locale using setlocale(), and run the date through strftime() (also taking advantage of strtotime() to create the appropriate timestamp) like this:

```
<?php
setlocale(LC_ALL, 'it_IT');
printf("Your subscription ends on %s",
strftime('%x', strtotime('2011-03-04')));
?>
```

This produces the following:

Your subscription ends on 04/03/2011

The same process applies to formatting number and monetary values. For instance, the United States uses a comma as the thousands separator; Europe uses a period, a space, or nothing at all for the same purpose. Making matters more confusing, the United States uses a period for the decimal separator and Europe uses a comma for this purpose. As a result, the following numbers are ultimately

considered identical:

- 523,332.98
- 523 332.98
- 523332.98
- 523.332,98

Of course, it makes sense to render such information in a manner most familiar to the user in order to reduce any possibility of confusion. To do so, you can use setlocale() in conjunction with number_format() and another function named localeconv(), which returns numerical formatting information about a defined locale. Used together, these functions can produce properly formatted numbers, like so:

```
<?php
setlocale(LC_ALL, 'it_IT');
$locale = localeconv();
printf("(it_IT) Total hours spent commuting
%s <br />",
number_format(4532.23, 2,
$locale['decimal_point'],
$locale['thousands_sep']));
setlocale(LC_ALL, 'en_US');
$locale = localeconv();
printf("(en_US) Total hours spent
commuting %s",
number_format(4532.23, 2,
$locale['decimal_point'],
$locale['thousands_sep']));
?>
```

This produces the following result:

```
(it_IT) Total hours spent commuting
4532,23
(en_US) Total hours spent commuting
4,532.23
```