File Handling

Fajar Pradana S.ST., M.Eng

Opening and Closing File Streams

- A **stream** is a channel used for accessing a resource that you can read from and write to
- The **input stream** reads data from a resource (such as a file)
- The **output stream** writes data to a resource
  1. Open the file stream with the `fopen()` function
  2. Write data to or read data from the file stream
  3. Close the file stream with the `fclose()` function

Opening a File Stream

- A **handle** is a special type of variable that PHP uses to represent a resource such as a file
- The `fopen()` function opens a handle to a file stream
- The syntax for the `fopen()` function is:

```
open_file =  fopen("text file", "mode");
```

- A **file pointer** is a special type of variable that refers to the currently selected line or character in a file

How ?

1. Opening File
2. Manipulating File
3. Closing File

File Open Mode

Argument	Description
a	Opens the specified file for writing only and places the file pointer at the end of the file; attempts to create the file if it doesn't exist
a+	Opens the specified file for reading and writing and places the file pointer at the end of the file; attempts to create the file if it doesn't exist
r	Opens the specified file for reading only and places the file pointer at the beginning of the file
r+	Opens the specified file for reading and writing and places the file pointer at the beginning of the file
w	Opens the specified file for writing only and deletes any existing content in the file; attempts to create the file if it doesn't exist
w+	Opens the specified file for reading and writing and deletes any existing content in the file; attempts to create the file if it doesn't exist
x	Creates and opens the specified file for writing only; returns false if the file already exists
x+	Creates and opens the specified file for reading and writing; returns false if the file already exists

Opening a File Stream (continued)

```
$BowlersFile = fopen("bowlers.txt", "r+");
```

Figure 6-1 Location of the file pointer when the `fopen()` function uses a *mode* argument of "r+"

Opening a File Stream (continued)

```
$BowlersFile = fopen("bowlers.txt", "a+");
```

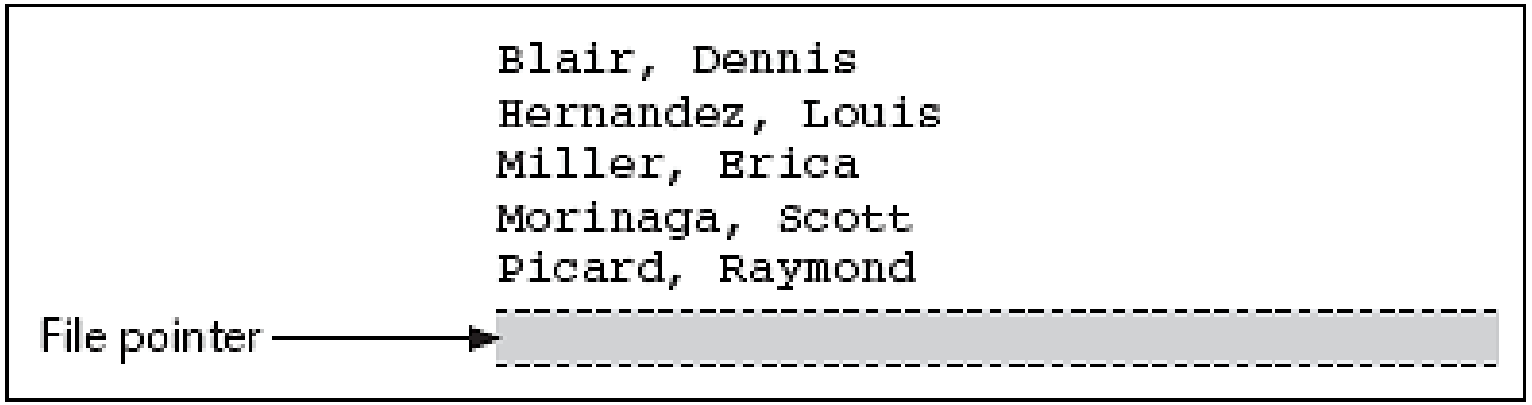


Figure 6-2 Location of the file pointer when the `fopen()` function uses a *mode* argument of "a+"

How to Close

- Use the `fclose` function when finished working with a file stream to save space in memory

```
$BowlersFile = fopen("bowlers.txt", "a");
fclose($BowlersFile);
```

Try This !

```
<?php
$namafile = "data.txt";
$handle = fopen ($namafile, "r");
if (!$handle) {
echo "<b>File tidak dapat dibuka atau belum
  ada</b>";
} else {
echo "<b>File berhasil dibuka</b>";
}
fclose($handle);
?>
```

Reading Data

- There are two main functions to read data:

`fgets($handle,$bytes)`

- Reads up to `$bytes` of data, stops at newline or end of file (EOF)

`fread($handle,$bytes)`

- Reads up to `$bytes` of data, stops at EOF.

Reading Data

- We need to be aware of the End Of File (EOF) point...

`feof($handle)`

- Whether the file has reached the EOF point. Returns true if have reached EOF.

Writing Data to Files

- PHP supports two basic functions for writing data to text files:
  - `fwrite()` function incrementally writes data to a text file
  - `file_put_contents()` function writes or appends a text string to a file

Writing Data Incrementally

- Use the `fwrite()` function to incrementally write data to a text file
- The syntax for the `fwrite()` function is:

```
fwrite($handle, data[, length]);
```

- The `fwrite()` function returns the number of bytes that were written to the file
- If no data was written to the file, the function returns a value of 0

Try This

```
<?php
$namafile = "data.txt";
$handle = fopen ($namafile, "w");
if (!$handle) {
echo "<b>File tidak dapat dibuka atau belum
  ada</b>";
} else {
fwrite ($handle, "Fakultas Teknologi Informasi\n");
fputs ($handle, "Universitas Budi Luhur\n");
//file_put_contents ($namafile, "Jakarta");
echo "<b>File berhasil ditulis</b>";
}
fclose($handle);
?>
```

Example

```
<?php
$namafile = "data.txt";
$handle = @fopen($namafile, "r");
if ($handle) {
while (!feof($handle)) {
$buffer = fgets($handle, 4096);
echo $buffer."<br>";
}
fclose($handle);
}
?>
```

Implementation (User Counter)

```
<?php
$counter_file="counter.txt";
if (!file_exists ($counter_file)) {
fopen ($counter_file, "w");
}
$file = fopen($counter_file,"r");
$counter = fread($file,10);
fclose($file);
$counter++;
echo "<h2>Anda adalah pengunjung ke - $counter</h2>";
$file = fopen($counter_file, "w");
fwrite($file,$counter);
fclose($file);
?>
```

Writing an Entire File

- The `file_put_contents()` function writes or appends a text string to a file
- The syntax for the `file_put_contents()` function is:

```
file_put_contents (filename, string[, options])
```

file_put_contents() Function

```
$TournamentBowlers = "Blair, Dennis\n";
$TournamentBowlers .= "Hernandez, Louis\n";
$TournamentBowlers .= "Miller, Erica\n";
$TournamentBowlers .= "Morinaga, Scott\n";
$TournamentBowlers .= "Picard, Raymond\n";
$BowlersFile = "bowlers.txt";
file_put_contents($BowlersFile, $TournamentBowlers);
```

file_put_contents() Function (continued)

- If no data was written to the file, the function returns a value of 0
- Use the return value to determine whether data was successfully written to the file

```
if (file_put_contents($BowlersFile, $TournamentBowlers) > 0)
        echo "<p>Data was successfully written to the
              $BowlersFile file.</p>";
else
        echo "<p>No data was written to the $BowlersFile
  file.</p>";
```

Writing an Entire File (continued)

- The `FILE_USE_INCLUDE_PATH` constant searches for the specified filename in the path that is assigned to the `include_path` directive in your php.ini configuration file
- The `FILE_APPEND` constant appends data to any existing contents in the specified filename instead of overwriting it

Writing an Entire File (continued)

Writing an Entire File (continued)

```
<h1>Coast City Bowling Tournament</h1>
<?php
if (isset($_GET['first_name']) && isset($_GET['last_name'])) {
      $BowlerFirst = $_GET['first_name'];
      $BowlerLast = $_GET['last_name'];
      $NewBowler = $BowlerLast . ", " . "$BowlerFirst" . "\n";
      $BowlersFile = "bowlers.txt";
      if (file_put_contents($BowlersFile, $NewBowler, FILE_APPEND) > 0)
          echo "<p>{$_GET['first_name']} {$_GET['last_name']} has
               been registered for the bowling tournament!</p>";
      else
          echo "<p>Registration error!</p>";
}
else
      echo "<p>To sign up for the bowling tournament, enter your first
          and last name and click the Register button.</p>";
?>
<form action="BowlingTournament.php" method="get"
enctype="application/x-www-form-urlencoded">
<p>First Name: <input type="text" name="first_name" size="30" /></p>
<p>Last Name: <input type="text" name="last_name" size="30" /></p>
<p><input type="submit" value="Register" /></p>
</form>
```

Other File Operations

- Delete file

```
unlink('filename');
```

- Rename (file or directory)

```
rename('old name', 'new name');
```

- Copy file

```
copy('source', 'destination');
```

- And many, many more!
  - www.php.net/manual/en/ref.filesystem.php

Directories

Reading Directories

Table 6-6 PHP directory functions

Function	Description
`chdir(directory)`	Changes to the specified directory
`chroot(directory)`	Changes to the root directory
`closedir($handle)`	Closes a directory handle
`getcwd()`	Gets the current working directory
`opendir(directory)`	Opens a handle to the specified directory
`readdir($handle)`	Reads a file or directory name from the specified directory handle
`rewinddir($handle)`	Resets the directory pointer to the beginning of the directory
`scandir(directory[, sort])`	Returns an indexed array containing the names of files and directories in the specified directory

Reading Directories (continued)

- To iterate through the entries in a directory, open a handle to the directory with the `opendir()` function
- Use the `readdir()` function to return the file and directory names from the open directory
- Use the `closedir()` function to close a directory handle

Reading Directories (continued)

```
$Dir = "C:\\PHP";
$DirOpen = opendir($Dir);
while ($CurFile = readdir($DirOpen)) {
        echo $CurFile . "<br />";
}
closedir($DirOpen);
```

scandir() Function

- Returns an indexed array containing the names of files and directories in the specified directory

```
$Dir = "C:\\PHP";
$DirEntries = scandir($Dir);
foreach ($DirEntries as $Entry) {
        echo $Entry . "<br />";
}
```

Creating Directories

- The `mkdir()` function creates a new directory
- To create a new directory within the current directory:
  - Pass just the name of the directory you want to create to the `mkdir()` function

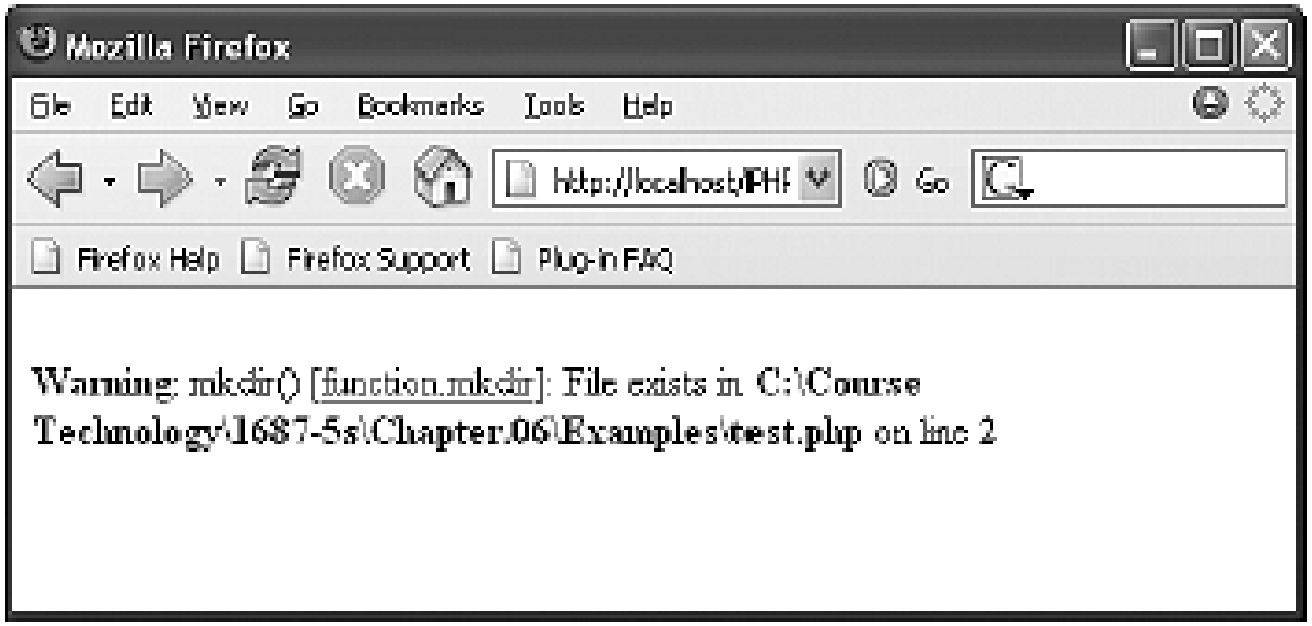
```
mkdir("bowlers");
```

Creating Directories (continued)

- To create a new directory in a location other than the current directory:
  - Use a relative or an absolute path

```
mkdir("..\\tournament");
mkdir("C:\\PHP\\utilities");
```

Creating Directories (continued)



F exists

Other Directory Operations

- Get current directory

```
getcwd()
```

- Change Directory

```
chdir('dirname');
```

- Create directory

```
mkdir('dirname');
```

- Delete directory (MUST be empty)

```
rmdir('dirname');
```

- And more!
  - www.php.net/manual/en/ref.dir.php

Upload

Upload

- The enctype attribute of the <form> tag specifies which content-type to use when submitting the form. "multipart/form-data" is used when a form requires binary data, like the contents of a file, to be uploaded
- The type="file" attribute of the <input> tag specifies that the input should be processed as a file. For example, when viewed in a browser, there will be a browse-button next to the input field
- By using the global PHP $_FILES array you can upload files from a client computer to the remote server.

$_FILES

- By using the global PHP $_FILES array you can upload files from a client computer to the remote server.
- The first parameter is the form's input name and the second index can be either "name", "type", "size", "tmp_name" or "error". Like this:
  - $_FILES["file"]["name"] - the name of the uploaded file
  - $_FILES["file"]["type"] - the type of the uploaded file
  - $_FILES["file"]["size"] - the size in bytes of the uploaded file
  - $_FILES["file"]["tmp_name"] - the name of the temporary copy of the file stored on the server
  - $_FILES["file"]["error"] - the error code resulting from the file upload

```
<html>
<head><title>Upload File</title></head>
<body>
<FORM ACTION="upload.php" METHOD="POST"
ENCTYPE="multipart/form-data">
Upload File : <input type="file"
  name="file"><br>
<input type="submit" name="Upload"
  value="Upload">
</FORM>
</body>
</html>
```

```php
<?php
if (isset($_POST['Upload'])) {
$dir_upload = "images/";
$nama_file = $_FILES['file']['name'];
//
if (is_uploaded_file($_FILES['file']['tmp_name'])) {
$cek = move_uploaded_file ($_FILES['file']['tmp_name'],
$dir_upload.$nama_file);
if ($cek) {
die ("File berhasil diupload");
} else {
die ("File gagal diupload");
}
}
}
?>
```