# File Handling

H S Rana

CIT,UPES

March 24, 2014

# Checking Whether a File Exists

To determine whether a file already exists, you can use the file_exists function, which returns either TRUE or FALSE,

*if (file_exists("testfile.txt")) echo "File exists";*

# Creating a file

If a file does not exist create it by using fopen function

```
<?php // testfile.php
$fh = fopen("testfile.txt", 'w') or die("Failed to create file'
$text = <<<_END
Line 1
Line 2
Line 3
_END;
fwrite($fh, $text) or die("Could not write to file");
fclose($fh);
echo "File 'testfile.txt' written successfully";
?>
```

# Steps in file handling

1. Create/Open a file using fopen function.
2. Perform read/write operation on file.
3. Finish by closing the file (fclose).

# Reading a file

The easiest way to read the contents of a disk file in PHP is with the file_get_ contents() function. This function accepts the name and path to a disk file, and reads the entire file into a string variable

```
<?php
// read file into string
$str = file_get_contents('example.txt') or die('ERROR: Cannot f
echo $str;
?>
```

# Reading a file

An alternative method of reading data from a file is PHP's file() function, which accepts the name and path to a file and reads the entire file into an array, with each element of the array representing one line of the file. To process the file, all you need do is iterate over the array using a foreach loop.

```
<?php
// read file into array
$arr = file('example.txt') or die('ERROR: Cannot find file');
foreach ($arr as $line) {
  echo $line;
}
?>
```

# Reading Remote Files

Both file_get_contents() and file() also support reading data from URLs, using either the HTTP or FTP protocol.

```
<?php
// read file into array
$arr = file('http://www.google.com') or die('ERROR: Cannot find
foreach ($arr as $line) {
  echo $line;
}
?>
```

# Writing a file

file_put_contents() function accepts a filename and path, together with the data to be written to the file, and then writes the latter to the former

```
<?php
// write string to file
$data = "A fish \n out of \n  water\n";
file_put_contents('output.txt', $data)
 or die('ERROR: Cannot write file');
echo 'Data written to file';
?>
```

# Writing a file

If the file specified in the call to file_put_contents() already exists on disk, file_put_contents() will overwrite it by default. If, instead, you'd prefer to preserve the file's contents and simply append new data to it, add the special FILE_APPEND flag to your file_put_contents() function call as a third argument.

```
<?php
// write string to file
$data = "A fish \n out of \n  water\n";
file_put_contents('output.txt', $data, FILE_APPEND)
or die('ERROR: Cannot write file');
echo 'Data written to file';
?>
```

# Another function to write a file

fwrite(file_Pointer,data) write the data in file file_name

fputs(file_Pointer,Data) write data in a line by line

fputc(file_ponter,character) write file a character by character.

# Copying Files

PHP copy function to create a clone of a file.

```
<?php // copyfile.php
copy('testfile.txt', 'testfile2.txt') or die("Could not copy fi
echo "File successfully copied to 'testfile2.txt'";
?>
```

# Moving a File

To move a file, rename it with the rename function,

```
<?php // movefile.php
if (!rename('testfile2.txt', 'testfile2.new'))
    echo "Could not rename file";
else echo "File successfully renamed to 'testfile2.new'";
?>
```

# Deleting a file

Deleting a file is just a matter of using the unlink function to remove it from the file system,

```
<?php // deletefile.php
if (!unlink('testfile2.new')) echo "Could not delete file";
else echo "File 'testfile2.new' successfully deleted";
?>
```

# Read/write a file randomly

The file pointer is the position within a file at which the next file access will take place, whether it's a read or a write.

to read and write a file randomly use fseek function

**fseek()** This function moves the file pointer from its current position to a new position, forward or backward, specified by the number of bytes.This function returns 0 on success, or -1 on failure.

this function takes three argument

**file** Specifies the open file to seek in

**offset** Specifies the new position (measured in bytes from the beginning of the file)

**whence** possible values

- SEEK_SET - Set position equal to offset. Default
- SEEK_CUR - Set position to current location plus offset
- SEEK_END - Set position to EOF plus offset (to move to a position before EOF, the offset must be a negative value)

# Examle

see example for random access

```
<?php // update.php
$fh = fopen("testfile.txt", 'r+') or die("Failed to open file")
$text = fgets($fh);
fseek($fh, 0, SEEK_END);
fwrite($fh, "$text") or die("Could not write to file");
fseek($fh, 8, SEEK_CUR);
fwrite($fh, "$text") or die("Could not write to file");
fseek($fh, 15, SEEK_SET);
fwrite($fh, "$text") or die("Could not write to file");
fclose($fh);
echo "File 'testfile.txt' successfully updated";
?>
```

# Locking Files for Multiple Accesses

Web programs are often called by many users at the same time. If more than one person tries to write to a file simultaneously, it can become corrupted. And if one person writes to it while another is reading from it, the file is all right but the person reading it can get odd results. To handle simultaneous users, it's necessary to use the file locking flock function.This function queues up all other requests to access a file until your program releases the lock. So, whenever your programs use write access on files that may be accessed concurrently by multiple users, you should also add file locking to them.

# Locking Files for Multiple Accesses

example

```
<?php
$fh = fopen("testfile.txt", 'r+') or die("Failed to open file")
$text = fgets($fh);
fseek($fh, 0, SEEK_END);
if (flock($fh, LOCK_EX))
{
    fwrite($fh, "$text") or die("Could not write to file");
    flock($fh, LOCK_UN);
}
fclose($fh);
echo "File 'testfile.txt' successfully updated";
?>
```