Accessing Files

Introduction

- The files access is abstracted within the stream layer.
- There are two types of streams. The first includes streams that provide access to specific stream resource. The second includes streams (AKA 'filter streams') that can be “installed” on top of streams that belong to the first group.

Specific Resources Streams

- The most important streams of the first group of streams (streams connected with specific resources) are the following:

php.*	standard PHP input/output stream
file	standard files access
http	accessing resources via HTTP
ftp	accessing resources via FTP
compress.zlib	accessing compressed data using zlib compression library.

Filter Streams

- The most important streams of the second group of streams (streams connected with specific resources streams) are the following:

string.rot13	encodes the data stream using ROT-13 algorithm
string.troupper	converts strings to uppercase
string.tolower	converts strings to lowercase
string.strip_tags	removes XML tags from a stream
convert.*	family of filters that convert to/from base64 encoding
mcrypt.*	family of filters that encrypt/decrypt
zlib.*	family of filters that compress/decompress data using zlib

Traditional C Like Files Access`

- PHP includes c-style functions that enable to access files.

  These methods include (among others) the following ones:

```
fopen
filesize
fgets
ftruncate
fseek
fwrite
...
```

Traditional C Like Files Access`

- When calling the `fopen` function we should pass two arguments.

  The first is the filename. The second is a string that indicates whether we are reading, writing or doing both. In addition, it indicates about the position at which the next byte will be read or written. That position can be either in the beginning or at the end of the file.

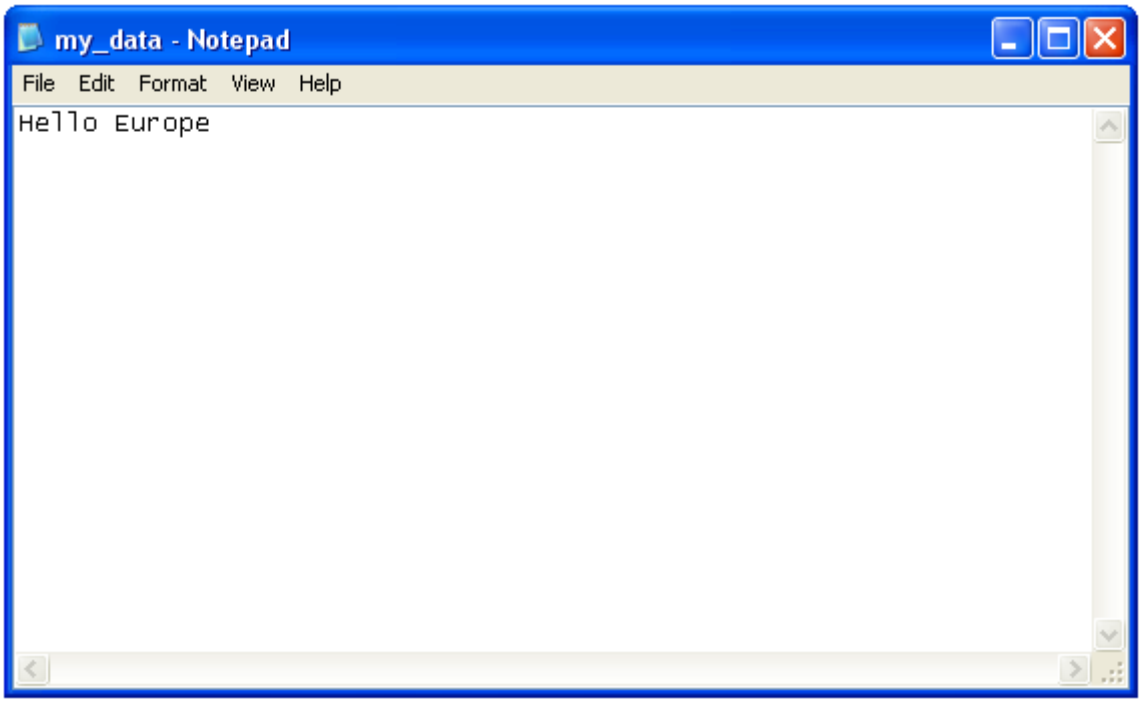
Traditional C Like Files Access`

- The second parameter can be any of the possible strings listed at http://il2.php.net/manual/en/function.fopen.php.

Traditional C Like Files Access`

```
<?php
    $fp = fopen('my_data.txt', 'w+');
    fwrite($fp, 'Hello');
    fwrite($fp, ' Europe');
    fclose($fp);
?>
```

Traditional C Like Files Access`



CSV Files

- Working with CSV files, PHP provides us with two basic functions that ease our work.
- Calling the '`fgetcsv()`' function we can iterate the file and get the next row each iteration. Calling the '`fputcsv()`' we cam write new rows into a given CSV file.

CSV Files Sample

Create CSV File

```
<?php
$file_handle = fopen('data.csv','w+');
$vec = array(
        array(42342,"Blue Table",2434.5),
        array(54234,"Red Chair",343.2),
        array(23342,"Tall Chair",532.2)
            );
foreach($vec as $row)
{
    fputcsv($file_handle,$row);
}
fclose($file_handle);
echo('The data.csv file was created');
?>
```

CSV Files Sample

Read CSV File

```
<?php
$file_handle = fopen('data.csv','r');
echo('<table>');
while($values = fgetcsv($file_handle))
{
    echo('<tr>');
    foreach($values as $value)
    {
        echo('<td>'.$value.'</td>');
    }
    echo('</tr>');
}
fclose($file_handle);
echo('</table>');
?>
```

CSV Files Sample


```
42342,"Blue Table",2434.5
54234,"Red Chair",343.2
23342,"Tall Chair",532.2
```

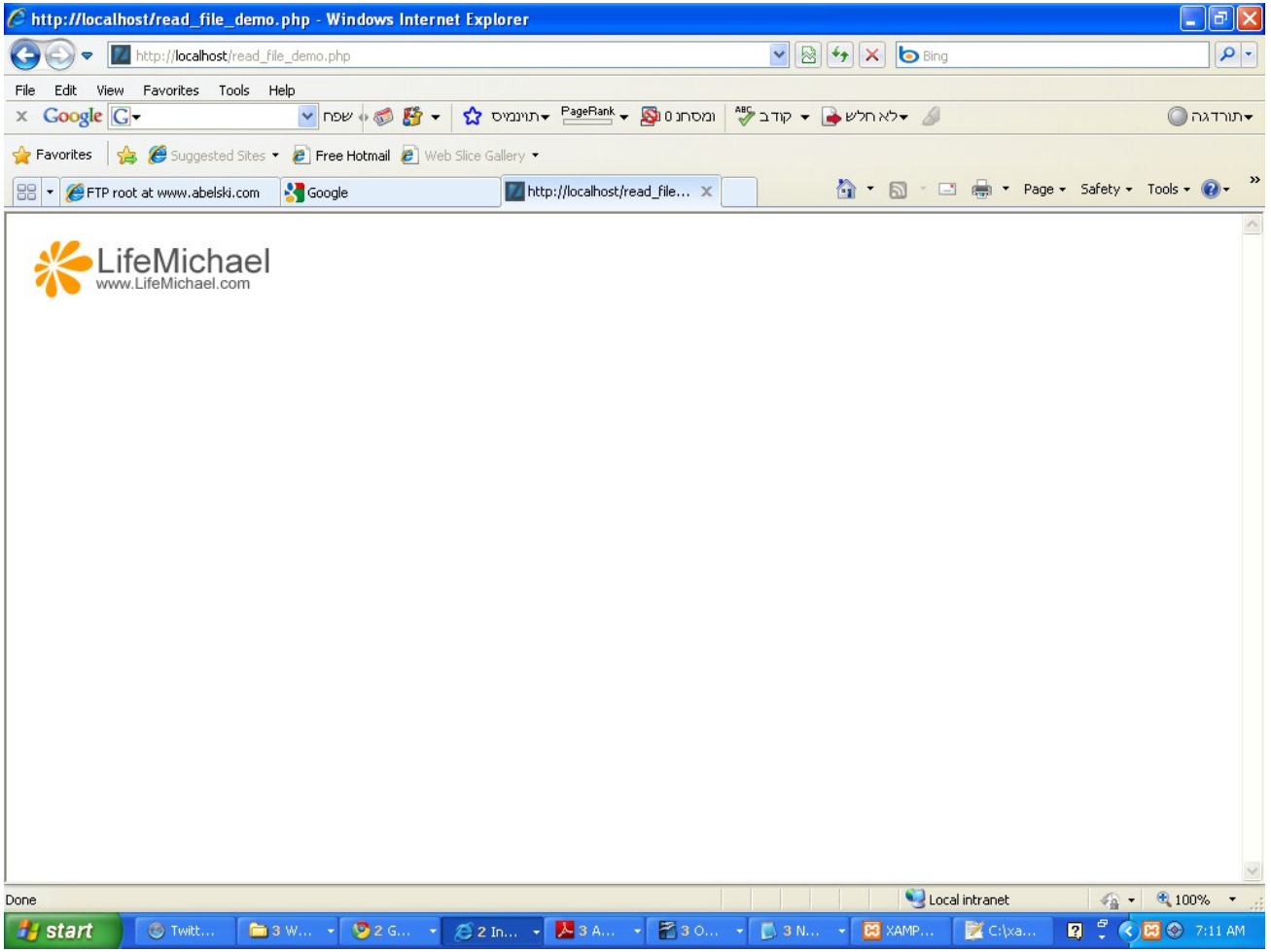
The '`readfile()`' Function

- Calling the '`readfile`()' function we can easily get the content of a specific file and have it immediately written to the script's standard output.
- This function is highly useful when we need to include within our script's output the content of a static file.

The 'readfile()' Function Sample

```
<?php
header("content-type: image/gif");
readfile("img.gif");
?>
```

The '`readfile()`' Function Sample



The '`file_get_contents()`' Function

- Calling the '`file_get_contents()`' function we can easily get the content of a specific file. This function returns the content as a string.

The '`file_put_contents()`' Function

- Calling the '`file_put_contents()`' function passing over a string together with a file name we can easily write the string to the specified file.

Sample

```
<?php
$str = file_get_contents('img.gif');
file_put_contents('imgcopy.gif',$str);
?>
```

Working with Directories

- PHP has a powerful set of directory manipulation functions.

  These functions include the following:

```
chdir('/temp');
```

  This function changes the current directory. If it fails it returns 'false'.

```
getcwd();
```

  This function returns the current directory.

```
mkdir('temp');
```

  This function creates a new directory.

Files Access

- PHP provides functions we can use in order to check whether a specific operation we want to perform related to specific file is permitted or not.

```
is_dir();
```

This function returns 'true' if the path you pass over is a path that leads to directory.

```
is_executable()
```

This function returns 'true' if the path you pass over is a path for an executable file.

Files Access

`is_file()`

This function returns 'true' if the path you pass it is of a regular file that already exists.

`is_readable()`

This function returns 'true' if the path you pass it is of a readable file.

`is_writeable()`

This function returns 'true' if the path you pass it is of a writeable file.

Files Access

`is_uploaded_file()`

This function returns 'true' if the path you pass it is of a file that was uploaded by the user (via HTTP).

- These functions use a cache memory to hold the result they return. Calling the same function twice, each time passing over another path might result in the same returned value.

  You can clear this cash calling the clearstatcache() function. Calling this function will ensure that each subsequent call to the same function shall return an accurate value (instead of the one that was stored in the cache).

Unix Permissions

- We can change the file Unix permissions calling the following functions:

```
chmod($filename,$mode)
chown($filename,$user)
chgrp($filename,$group)
```