Continue to Advance PHP

8.5 Modify a PHP Session Variable

To change a session variable, just overwrite it:

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>
</body>
</html>
```

Output: Array ([favcolor] => yellow [favanimal] => cat **)**

8.6 Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy()

Example:

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
echo "All session variables are now removed, and the session is destroyed."
?>
</body>
</html>
```

Output: All session variables are now removed, and the session is destroyed.

9.1 PHP Error Handling

The default error handling in PHP is very simple. An error message with filename, line number and a message describing the error is sent to the browser.

9.2 PHP Error Handling

When creating scripts and web applications, error handling is an important part. If your code lacks error checking code, your program may look very unprofessional and you may be open to security risks. This tutorial contains some of the most common error checking methods in PHP. We will show different error handling methods:

Simple "die()" statements

Custom errors and error triggers

Error reporting

9.3 Basic Error Handling: Using the die() function

The first example shows a simple script that opens a text file:

```
<?php
$file=fopen("welcome.txt","r");
?>
```

If the file does not exist you might get an error like this:

Warning: fopen(welcome.txt) [function.fopen]: failed to open stream:
No such file or directory in C:\webfolder\test.php on line 2

To prevent the user from getting an error message like the one above, we test whether the file exist before we try to access it:

```
<?php
if(!file_exists("welcome.txt")) {
  die("File not found");
} else {
  $file=fopen("welcome.txt","r");
}
?>
```

Now if the file does not exist you get an error like this:

File not found

The code above is more efficient than the earlier code, because it uses a simple error handling mechanism to stop the script after the error. However, simply stopping the script is not always the right way to go. Let's take a look at alternative PHP functions for handling errors.

9.4 Creating a Custom Error Handler

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP. This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

error_function(error_level,error_message,
error_file,error_line,error_context)

Parameter	Description
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

9.5 Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

Value	Constant	Description
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script

		normally
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

Now lets create a function to handle errors:

```
function customError($errno, $errstr) {
 echo "<b>Error:</b> [$errno] $errstr<br>";
 echo "Ending Script";
 die();
}
```

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.Now that we have created an error handling function we need to decide when it should be triggered.

9.6 Set Error Handler

The default error handler for PHP is the built in error handler. We are going to make the function above the default error handler for the duration of the script. It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler for all errors:

set_error_handler("customError");

Since we want our custom function to handle all errors, the set_error_handler() only needed one parameter, a second parameter could be added to specify an error level.

Example **Testing the error handler by trying to output variable that does not exist:**

```php
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr";
}
//set error handler
set_error_handler("customError");
//trigger error
echo($test);
?>
```

The output of the code above should be something like this:

Error: [8] Undefined variable: test

9.7 Trigger an Error

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the trigger_error() function.

Example

In this example an error occurs if the "test" variable is bigger than "1":

```php
<?php
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or below");
}
?>
```

The output of the code above should be something like this:

Notice: Value must be 1 or below
in C:\webfolder\test.php on line 6

An error can be triggered anywhere you wish in a script, and by adding a second parameter, you can specify what error level is triggered.

Possible error types:

E_USER_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted

E_USER_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted

E_USER_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally

Example

In this example an E_USER_WARNING occurs if the "test" variable is bigger than "1". If an E_USER_WARNING occurs we will use our custom error handler and end the script:

```php
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}
//set error handler
set_error_handler("customError",E_USER_WARNING);
//trigger error
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

The output of the code above should be something like this:

Error: [512] Value must be 1 or below
Ending Script

Now that we have learned to create our own errors and how to trigger them, lets take a look at error logging.

9.8 Error Logging

By default, PHP sends an error log to the server's logging system or a file, depending on how the error_log configuration is set in the php.ini file. By using the error_log() function you can send error logs to a specified file or a remote destination. Sending error messages to yourself by e-mail can be a good way of getting notified of specific errors.

9.9 Send an Error Message by E-Mail

In the example below we will send an e-mail with an error message and end the script, if a specific error occurs:

```php
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "someone@example.com","From: webmaster@example.com");
}
//set error handler
set_error_handler("customError",E_USER_WARNING);
//trigger error
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

The output of the code above should be something like this:

Error: [512] Value must be 1 or below
Webmaster has been notified

And the mail received from the code above looks like this:

Error: [512] Value must be 1 or below

This should not be used with all errors. Regular errors should be logged on the server using the default PHP logging system.

10.1 What is an Exception

With PHP 5 came a new object oriented way of dealing with errors.

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

This is what normally happens when an exception is triggered:

The current code state is saved

The code execution will switch to a predefined (custom) exception handler function

Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

We will show different error handling methods:

Basic use of Exceptions

Creating a custom exception handler

Multiple exceptions

Re-throwing an exception

Setting a top level exception handler

Note: Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

10.2 Basic Use of Exceptions

When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block. If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Lets try to throw an exception without catching it:

```php
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
//trigger exception
checkNum(2);
?>

The code above will get an error like this:
```

Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6

10.3 Try, throw and catch

To avoid the error from the example above, we need to create the proper code to handle an exception.

Proper exception code should include:

1. try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
2. throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
3. catch - A "catch" block retrieves an exception and creates an object containing the exception information

Lets try to trigger an exception with valid code example:

```php
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
} //trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
} //catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
?>
```

Output: The code above will get an error like this:
　　　　Message: Value must be 1 or below

Example explained:

The code above throws an exception and catches it:

The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown

The checkNum() function is called in a "try" block

The exception within the checkNum() function is thrown

The "catch" block retrieves the exception and creates an object ($e) containing the exception information

The error message from the exception is echoed by calling $e->getMessage() from the exception object

However, one way to get around the "every throw must have a catch" rule is to set a top level exception handler to handle errors that slip through.