PHP Error Handling

[❮ Previous](https://www.w3schools.com/php/php_filter_advanced.asp)[Next ❯](https://www.w3schools.com/php/php_exception.asp)

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.

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

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.

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 |

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.

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  
//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

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  
$test=2;  
if ($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  
//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.

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.

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  
//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.

PHP Exception Handling

[❮ Previous](https://www.w3schools.com/php/php_error.asp)[Next ❯](https://www.w3schools.com/php/php_mysql_intro.asp)

Exceptions are used to change the normal flow of a script if a specified error occurs.

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.

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  
//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**

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:

<?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();  
}  
?>

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:

1. The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
2. The checkNum() function is called in a "try" block
3. The exception within the checkNum() function is thrown
4. The "catch" block retrieves the exception and creates an object ($e) containing the exception information
5. 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.

Creating a Custom Exception Class

To create a custom exception handler you must create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()  
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example...com";  
  
try {  
  //check if  
  if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE) {  
    //throw exception if email is not valid  
    throw new customException($email);  
  }  
}  
  
catch (customException $e) {  
  //display custom message  
  echo $e->errorMessage();  
}  
?>

The new class is a copy of the old exception class with an addition of the errorMessage() function. Since it is a copy of the old class, and it inherits the properties and methods from the old class, we can use the exception class methods like getLine() and getFile() and getMessage().

Example explained:

The code above throws an exception and catches it with a custom exception class:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is not a valid e-mail address
4. The "try" block is executed and an exception is thrown since the e-mail address is invalid
5. The "catch" block catches the exception and displays the error message

Multiple Exceptions

It is possible for a script to use multiple exceptions to check for multiple conditions.

It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()  
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example.com";  
  
try {  
  //check if  
  if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE) {  
    //throw exception if email is not valid  
    throw new customException($email);  
  }  
  //check for "example" in mail address  
  if(strpos($email, "example") !== FALSE) {  
    throw new Exception("$email is an example e-mail");  
  }  
}  
  
catch (customException $e) {  
  echo $e->errorMessage();  
}  
  
catch(Exception $e) {  
  echo $e->getMessage();  
}  
?>

Example explained:

The code above tests two conditions and throws an exception if any of the conditions are not met:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block is executed and an exception is not thrown on the first condition
5. The second condition triggers an exception since the e-mail contains the string "example"
6. The "catch" block catches the exception and displays the correct error message

If the exception thrown were of the class customException and there were no customException catch, only the base exception catch, the exception would be handled there.

Re-throwing Exceptions

Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.

A script should hide system errors from users. System errors may be important for the coder, but are of no interest to the user. To make things easier for the user you can re-throw the exception with a user friendly message:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = $this->getMessage().' is not a valid E-Mail address.';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example.com";  
  
try {  
  try {  
    //check for "example" in mail address  
    if(strpos($email, "example") !== FALSE) {  
      //throw exception if email is not valid  
      throw new Exception($email);  
    }  
  }  
  catch(Exception $e) {  
    //re-throw exception  
    throw new customException($email);  
  }  
}  
  
catch (customException $e) {  
  //display custom message  
  echo $e->errorMessage();  
}  
?>

Example explained:

The code above tests if the email-address contains the string "example" in it, if it does, the exception is re-thrown:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block contains another "try" block to make it possible to re-throw the exception
5. The exception is triggered since the e-mail contains the string "example"
6. The "catch" block catches the exception and re-throws a "customException"
7. The "customException" is caught and displays an error message

If the exception is not caught in its current "try" block, it will search for a catch block on "higher levels".

Set a Top Level Exception Handler

The set\_exception\_handler() function sets a user-defined function to handle all uncaught exceptions:

<?php  
function myException($exception) {  
  echo "<b>Exception:</b> " . $exception->getMessage();  
}  
  
set\_exception\_handler('myException');  
  
throw new Exception('Uncaught Exception occurred');  
?>

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

**Exception:** Uncaught Exception occurred

In the code above there was no "catch" block. Instead, the top level exception handler triggered. This function should be used to catch uncaught exceptions.

Rules for exceptions

* Code may be surrounded in a try block, to help catch potential exceptions
* Each try block or "throw" must have at least one corresponding catch block
* Multiple catch blocks can be used to catch different classes of exceptions
* Exceptions can be thrown (or re-thrown) in a catch block within a try block

A simple rule: If you throw something, you have to catch it.