

## Exception Handling - PHP

Error handling is the process of catching errors raised by your program and then taking appropriate action. If you would handle errors properly then it may lead to many unforeseen consequences.

Its very simple in PHP to handle an errors.

### Using die() function:

While wirting your PHP program you should check all possible error condition before going ahead and take appropriate action when required.

Try following example without having “test.txt” file and with this file. The output is given in figure 1

Example 1: exception\_eg1.php

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

//output

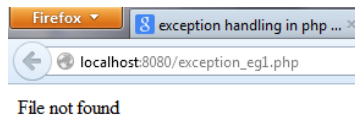


Figure 1

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

## 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.

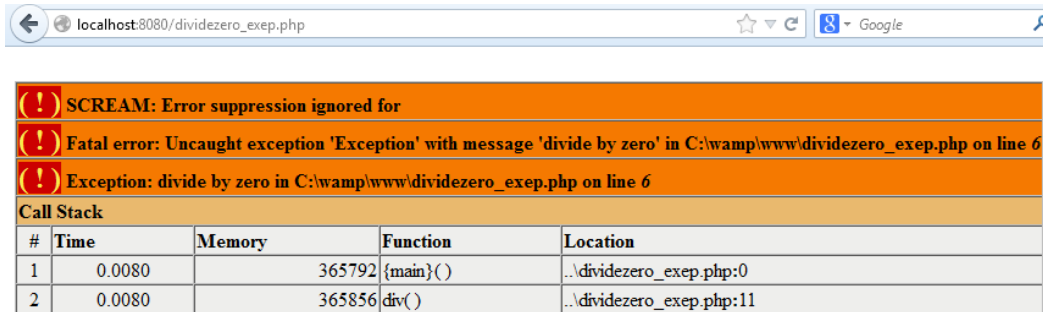
If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Lets try to throw an exception without catching it: The output can be seen in figure 2.

Example 2 :dividezero\_exep.php

```
<?php
function div($a,$b)
{
if($b==0)
{
throw new Exception("divide by zero");
}
return($a/$b);
}
echo div(5,0);
?>
```

//output



localhost:8080/dividezero\_exep.php

SCREAM: Error suppression ignored for

Fatal error: Uncaught exception 'Exception' with message 'divide by zero' in C:\wamp\www\dividezero\_exep.php on line 6

Exception: divide by zero in C:\wamp\www\dividezero\_exep.php on line 6

Call Stack

#	Time	Memory	Function	Location
1	0.0080	365792	{main}()	..\dividezero_exep.php:0
2	0.0080	365856	div()	..\dividezero_exep.php:11

Figure 2

This way you can write an efficient code. Using above technique you can stop your program whenever it errors out and display more meaningful and user friendly message.

## 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 3: dividezero\_exep2.php

```
<?php
function div($a,$b)
{
if($b==0)
{
throw new Exception("divide by zero");
}
return($a/$b);
}
try
{
echo div(5,0);
}
catch(Exception $e)
{
echo $e->getMessage();
}
?>
```

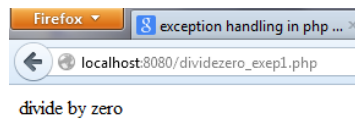


Figure 3

#### Explanation

- 1) The div() function is created. It checks if the divisor is 0. If it is, an exception is thrown
- 2) The div() function is called in a “try” block.
- 3) The excetion within the div() 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.

### CREATING A CUSTOM EXCEPTION CLASS

Creating a custom exception handler is quite simple. We simply 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 dateException class:

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

Through `custom_excep_eg1.html` we get date input which we sent to `custom_excep_eg.php` which process the date value. The outputs can be seen in figures 4,5,6 and figure 7

```
//custom_excep_eg1.html
```

```
<html>
<title>date exception</title>
<body>
<form action="custom_excep_eg.php" method="get">
Enter date :<input type="text" name="fdate"/>
<input type="submit"/>
</form>
</body>
</html>
```

```
//custom_excep_eg.php
```

```
<?php
class dateException extends Exception
{
function err_date_Msg()
{
echo $this->getMessage(). " Proper format dd-mm-yyyy";
}
} //dateException

try
{
$date=$_GET['fdate'];
getdatefun($date);
echo "$date in correct format";
}
catch(dateException $e)
{
echo $e->err_date_Msg();
}

function getdatefun($dt)
{
if(!date_create_from_format("d-m-Y",$dt))
{
throw new dateException("Date not in proper format");
}
}
}
```

?>

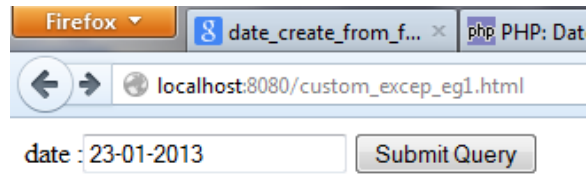


Figure 4

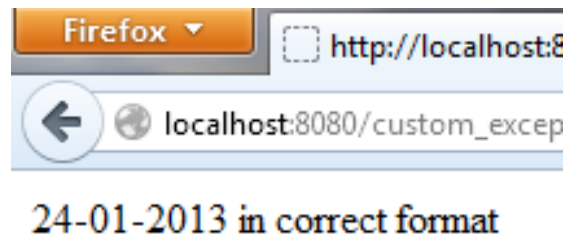


Figure 5

When you change the input format like below the exception is raised

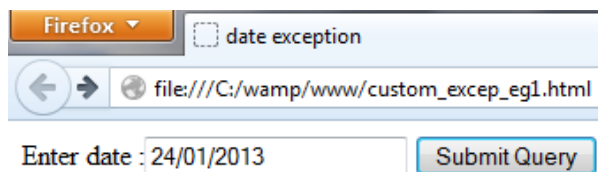


Figure 6

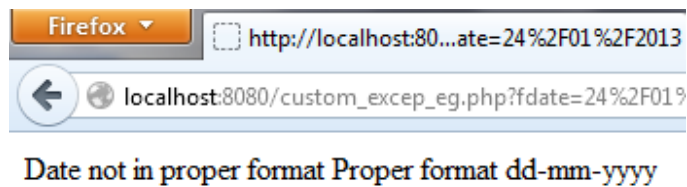


Figure 7

## Example explained:

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

1. The dateException() 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 err\_date\_Msg() function is created. This function returns an error message if the date is not entered in the prescribed format.
3. The \$date variable is received through \$\_GET function and passed as argument to getdatefun()
4. The getdatefun() checks whether the date is in given format and if not throw exception. This error is caught in catch block and displays error message.
5. If everything is ok the line next to getdatefun() will be executed and the program terminates.

## Multiple catch – custom exception

The following example shows how to write more than one custom exception in a single program for checking multiple inputs validation.

//custom\_excep\_eg2.php

```
<?php
class dateException extends Exception
{
function err_date_Msg()
{
echo $this->getMessage(). " Proper format dd-mm-yyyy <br>";
}
} //dateException

class nameException extends Exception
{
function err_name_msg()
{
echo $this->getMessage(). " Name should not be empty <br>";
}
} //nameException

try
{
$name=$_GET['fname'];
$date=$_GET['fdate'];
getdatefun($date);
echo "<br> $date in correct format";
getnamefun($name);
echo "<br> $name is correct";
}
catch(dateException $e)
{
echo $e->err_date_Msg();
}
catch(nameException $e)
{
```

```

echo $e->err_name_Msg();
}

function getdatefun($dt)
{
if(!date_create_from_format("d-m-Y",$dt))
{
throw new dateException(" Date not in proper format");
}
}
function getnamefun($n)
{
if($n=="")
{
throw new nameException("must enter some value");
}
}
?>

```

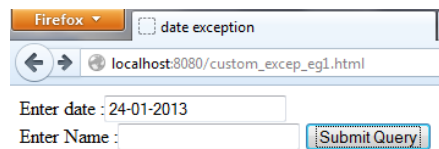


Figure 8

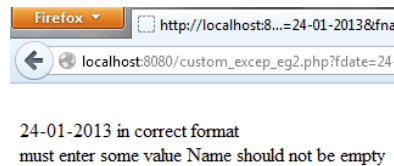


Figure 9

## Example explained:

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

1. The dateException() 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 err\_date\_Msg() function is created. This function returns an error message if the date is not entered in the prescribed format
3. The nameException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
4. The err\_name\_Msg() function is created. This function returns an error message if the name is left empty i.e without entering any value.
5. The \$date variable and \$name is received through \$\_GET function and passed as argument to getdatefun() and getnamefun() respectively.

6. The `getdatefun()` checks whether the date is in given format and if not throw exception. This error is caught in catch block (`catch(dateException($e))`) and displays error message.
7. The `getnamefun()` checks whether the user entered any value in the name text box and if not an error is thrown. This error is caught in catch block (`catch(dateException($e))`) and displays error message.
8. If everything is ok the line next to `getdatefun()` and `getnamefun()` will get executed and the program terminates.

In the above example `$e->getMessage` function is used to get error message. There are following functions which can be used from **Exception** class.

- **getMessage()**- message of exception
- **getCode()** - code of exception
- **getFile()** - source filename
- **getLine()** - source line
- **getTrace()** - n array of the backtrace()
- **getTraceAsString()** - formated string of trace

## Rules for Exception

- 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 in a catch block within a try block.

## PHP - Predefined Variables

PHP provides a large number of predefined variables to any script which it runs. PHP provides an additional set of predefined arrays containing variables from the web server the environment, and user input. These new arrays are called superglobals:

All the following variables are automatically available in every scope.

### PHP Superglobals:

Variable	Description
<code>\$GLOBALS</code>	Contains a reference to every variable which is currently available within the global scope of the script. The keys of this array are the names of the global variables.
<code>\$_SERVER</code>	This is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these. See next section for a complete list of all



	the SERVER variables.
<code>\$_GET</code>	An associative array of variables passed to the current script via the HTTP GET method.
<code>\$_POST</code>	An associative array of variables passed to the current script via the HTTP POST method.
<code>\$_FILES</code>	An associative array of items uploaded to the current script via the HTTP POST method.
<code>\$_REQUEST</code>	An associative array consisting of the contents of <code>\$_GET</code> , <code>\$_POST</code> , and <code>\$_COOKIE</code> .
<code>\$_COOKIE</code>	An associative array of variables passed to the current script via HTTP cookies.
<code>\$_SESSION</code>	An associative array containing session variables available to the current script.
<code>\$_PHP_SELF</code>	A string containing PHP script file name in which it is called.
<code>\$php_errormsg</code>	<code>\$php_errormsg</code> is a variable containing the text of the last error message generated by PHP.

### Server variables: `$_SERVER`

`$_SERVER` is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these.

Variable	Description
<code>\$_SERVER['PHP_SELF']</code>	The filename of the currently executing script, relative to the document root
<code>\$_SERVER['argv']</code>	Array of arguments passed to the script. When the script is run on the command line, this gives C-style access to the command line parameters. When called via the GET method, this will contain the query string.
<code>\$_SERVER['argc']</code>	Contains the number of command line parameters passed to the script if run on the command line.
<code>\$_SERVER['GATEWAY_INTERFACE']</code>	What revision of the CGI specification the server is using; i.e. 'CGI/1.1'.
<code>\$_SERVER['SERVER_ADDR']</code>	The IP address of the server under which the current script is executing.
<code>\$_SERVER['SERVER_NAME']</code>	The name of the server host under which the current script is executing. If the script is running on a virtual host, this will be the value defined for that virtual host.
<code>\$_SERVER['SERVER_SOFTWARE']</code>	Server identification string, given in the headers when responding to requests.
<code>\$_SERVER['SERVER_PROTOCOL']</code>	Name and revision of the information protocol via which the page was requested; i.e. 'HTTP/1.0';

<code>\$_SERVER['REQUEST_METHOD']</code>	Which request method was used to access the page; i.e. 'GET', 'HEAD', 'POST', 'PUT'.
<code>\$_SERVER['REQUEST_TIME']</code>	The timestamp of the start of the request. Available since PHP 5.1.0.
<code>\$_SERVER['QUERY_STRING']</code>	The query string, if any, via which the page was accessed.
<code>\$_SERVER['DOCUMENT_ROOT']</code>	The document root directory under which the current script is executing, as defined in the server's configuration file.
<code>\$_SERVER['HTTP_ACCEPT']</code>	Contents of the Accept: header from the current request, if there is one.
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	Contents of the Accept-Charset: header from the current request, if there is one. Example: 'iso-8859-1,*;utf-8'.
<code>\$_SERVER['HTTP_ACCEPT_ENCODING']</code>	Contents of the Accept-Encoding: header from the current request, if there is one. Example: 'gzip'.
<code>\$_SERVER['HTTP_ACCEPT_LANGUAGE']</code>	Contents of the Accept-Language: header from the current request, if there is one. Example: 'en'.
<code>\$_SERVER['HTTP_CONNECTION']</code>	Contents of the Connection: header from the current request, if there is one. Example: 'Keep-Alive'.
<code>\$_SERVER['HTTP_HOST']</code>	Contents of the Host: header from the current request, if there is one.
<code>\$_SERVER['HTTP_REFERER']</code>	The address of the page (if any) which referred the user agent to the current page.
<code>\$_SERVER['HTTP_USER_AGENT']</code>	This is a string denoting the user agent being which is accessing the page. A typical example is: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586).
<code>\$_SERVER['HTTPS']</code>	Set to a non-empty value if the script was queried through the HTTPS protocol.
<code>\$_SERVER['REMOTE_ADDR']</code>	The IP address from which the user is viewing the current page.
<code>\$_SERVER['REMOTE_HOST']</code>	The Host name from which the user is viewing the current page. The reverse dns lookup is based off the REMOTE_ADDR of the user.
<code>\$_SERVER['REMOTE_PORT']</code>	The port being used on the user's machine to communicate with the web server.
<code>\$_SERVER['SCRIPT_FILENAME']</code>	The absolute pathname of the currently executing script.
<code>\$_SERVER['SERVER_ADMIN']</code>	The value given to the SERVER_ADMIN (for Apache) directive in the web server configuration file.

<code>\$_SERVER['SERVER_PORT']</code>	The port on the server machine being used by the web server for communication. For default setups, this will be '80'.
<code>\$_SERVER['SERVER_SIGNATURE']</code>	String containing the server version and virtual host name which are added to server-generated pages, if enabled.
<code>\$_SERVER['PATH_TRANSLATED']</code>	Filesystem based path to the current script.
<code>\$_SERVER['SCRIPT_NAME']</code>	Contains the current script's path. This is useful for pages which need to point to themselves.
<code>\$_SERVER['REQUEST_URI']</code>	The URI which was given in order to access this page; for instance, '/index.html'.
<code>\$_SERVER['PHP_AUTH_DIGEST']</code>	When running under Apache as module doing Digest HTTP authentication this variable is set to the 'Authorization' header sent by the client.
<code>\$_SERVER['PHP_AUTH_USER']</code>	When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the username provided by the user.
<code>\$_SERVER['PHP_AUTH_PW']</code>	When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the password provided by the user.
<code>\$_SERVER['AUTH_TYPE']</code>	When running under Apache as module doing HTTP authenticated this variable is set to the authentication type.