Error & Exception Handling

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.

It’s very simple in PHP to handle an error.

## **Using die() function**

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

Try following example without having **/tmp/test.xt** file and with this file.

|  |  |
| --- | --- |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9** | **<**?php **if(!**file\_exists**(**"/tmp/test.txt"**)) {** **die(**"File not found"**);** **}else** **{** $file **=** fopen**(**"/tmp/test.txt"**,**"r"**);** **print** "Opend file sucessfully"**;** **}** // Test of the code here.?**>** |

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.

## **Defining Custom Error Handling Function**

You can write your own function to handling any error. PHP provides you a framework to define error handling function.

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**);**

|  |  |
| --- | --- |
| **No.** | **Parameter & Description** |
| 1 | **error\_level**Required - Specifies the error report level for the user-defined error. Must be a value number. |
| 2 | **error\_message**Required - Specifies the error message for the user-defined error |
| 3 | **error\_file**Optional - Specifies the file name in which the error occurred |
| 4 | **error\_line**Optional - Specifies the line number in which the error occurred |
| 5 | **error\_context**Optional - Specifies an array containing every variable and their values in use when the error occurred |

### **Possible Error levels**

These error report levels are the different types of error the user-defined error handler can be used for. These values cab used in combination using **|** operator

|  |  |  |
| --- | --- | --- |
| **No.** | **Constant & Description** | **Value** |
| 1 | **.E\_ERROR**Fatal run-time errors. Execution of the script is halted | 1 |
| 2 | **E\_WARNING**Non-fatal run-time errors. Execution of the script is not halted | 2 |
| 3 | **E\_PARSE**Compile-time parse errors. Parse errors should only be generated by the parser. | 4 |
| 4 | **E\_NOTICE**Run-time notices. The script found something that might be an error, but could also happen when running a script normally | 8 |
| 5 | **E\_CORE\_ERROR**Fatal errors that occur during PHP's initial start-up. | 16 |
| 6 | **E\_CORE\_WARNING**Non-fatal run-time errors. This occurs during PHP's initial start-up. | 32 |
| 7 | **E\_USER\_ERROR**Fatal user-generated error. This is like an E\_ERROR set by the programmer using the PHP function trigger\_error() | 256 |
| 8 | **E\_USER\_WARNING**Non-fatal user-generated warning. This is like an E\_WARNING set by the programmer using the PHP function trigger\_error() | 512 |
| 9 | **E\_USER\_NOTICE**User-generated notice. This is like an E\_NOTICE set by the programmer using the PHP function trigger\_error() | 1024 |
| 10 | **E\_STRICT**Run-time notices. Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code. | 2048 |
| 11 | **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()) | 4096 |
| 12 | **E\_ALL**All errors and warnings, except level E\_STRICT (E\_STRICT will be part of E\_ALL as of PHP 6.0) | 8191 |

All the above error level can be set using following PHP built-in library function where level cab be any of the value defined in above table.

**int** error\_reporting **( [int** $level**] )**

Following is the way you can create one error handling function:

|  |  |
| --- | --- |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9** | **<**?php **function** handleError**(**$errno**,** $errstr**,**$error\_file**,**$error\_line**) {** **echo** "<b>Error:</b> [$errno] $errstr - $error\_file:$error\_line"**;** **echo** "<br />"**;** **echo** "Terminating PHP Script"**;** **die();** **}****?>** |

Once you define your custom error handler you need to set it using PHP built-in library **set\_error\_handler** function. Now lets examine our example by calling a function which does not exist.

|  |  |
| --- | --- |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10****11****12****13****14****15****16****17** | **<**?php error\_reporting**(** E\_ERROR **);**  **function** handleError**(**$errno**,** $errstr**,**$error\_file**,**$error\_line**) {** **echo** "<b>Error:</b> [$errno] $errstr - $error\_file:$error\_line"**;** **echo** "<br />"**;** **echo** "Terminating PHP Script"**;**  **die();** **}**  //set error handler set\_error\_handler**(**"handleError"**);**  //trigger error myFunction**();**?**>** |

## **Exceptions Handling**

PHP has an exception model similar to that of other programming languages. Exceptions are important and provides a better control over error handling.

Let’s explain their new keyword related to exceptions.

* **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".
* **Throw** − This is how you trigger an exception. Each "throw" must have at least one "catch".
* **Catch** − A "catch" block retrieves an exception and creates an object containing the exception information.

When an exception is thrown, code following the statement will not be executed, and PHP will attempt to find the first matching catch block. If an exception is not caught, a PHP Fatal Error will be issued with an "Uncaught Exception ...

* An exception can be thrown, and caught ("catched") within PHP. Code may be surrounded in a try block.
* Each try 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) within a catch block.

### **Example**

Following is the piece of code, copy and paste this code into a file and verify the result.

|  |  |
| --- | --- |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10****11****12****13****14** | **<**?php **try** **{** $error **=** 'Always throw this error'**;** **throw new** Exception**(**$error**);**  // Code following an exception is not executed. **echo** 'Never executed'**;** **}catch** **(**Exception $e**) {** **echo** 'Caught exception: '**,** $e->getMessage**(),** "\n"**;** **}**  // Continue execution **echo** 'Hello World'**;**?**>** |

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

### **Creating Custom Exception Handler**

You can define your own custom exception handler. Use following function to set a user-defined exception handler function.

**string** set\_exception\_handler **(** callback $exception\_handler **)**

Here **exception\_handler** is the name of the function to be called when an uncaught exception occurs. This function must be defined before calling set\_exception\_handler().

### **Example**

|  |  |
| --- | --- |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10** | **<**?php **function** exception\_handler**(**$exception**) {** **echo** "Uncaught exception: " **,** $exception->getMessage**(),** "\n"**;** **}**  set\_exception\_handler**(**'exception\_handler'**);** **throw new** Exception**(**'Uncaught Exception'**);**  **echo** "Not Executed\n"**;**?**>** |

## **Error & Logging Functions**

These are functions dealing with error handling and logging. They allow you to define your own error handling rules, as well as modify the way the errors can be logged. This allows you to change and enhance error reporting to suit your needs.

Using these logging functions, you can send messages directly to other machines, to an email, to system logs, etc., so you can selectively log and monitor the most important parts of your applications and websites.

## **Runtime Configuration**

The behavior of these functions is affected by settings in php.ini. These settings are defined below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Default** | **Changeable** | **Changelog** |
| error\_reporting | NULL | PHP\_INI\_ALL |  |
| display\_errors | "1" | PHP\_INI\_ALL |  |
| display\_startup\_errors | "0" | PHP\_INI\_ALL | Available since PHP 4.0.3. |
| log\_errors | "0" | PHP\_INI\_ALL |  |
| log\_errors\_max\_len | "1024" | PHP\_INI\_ALL | Available since PHP 4.3.0. |
| ignore\_repeated\_errors | "0" | PHP\_INI\_ALL | Available since PHP 4.3.0. |
| ignore\_repeated\_source | "0" | PHP\_INI\_ALL | Available since PHP 4.3.0. |
| report\_memleaks | "1" | PHP\_INI\_ALL | Available since PHP 4.3.0. |
| track\_errors | "0" | PHP\_INI\_ALL |  |
| html\_errors | "1" | PHP\_INI\_ALL | PHP\_INI\_SYSTEM in PHP <= 4.2.3. Available since PHP 4.0.2. |
| docref\_root | "" | PHP\_INI\_ALL | Available since PHP 4.3.0. |
| docref\_ext | "" | PHP\_INI\_ALL | Available since PHP 4.3.2. |
| error\_prepend\_string | NULL | PHP\_INI\_ALL |  |
| error\_append\_string | NULL | PHP\_INI\_ALL |  |
| error\_log | NULL | PHP\_INI\_ALL |  |
| warn\_plus\_overloading | NULL |  | This option is no longer available as of PHP 4.0.0 |

## **PHP Error and Logging Constants**

**PHP** − indicates the earliest version of PHP that supports the constant.

You can use any of the constant while configuring your php.ini file.

|  |  |  |
| --- | --- | --- |
| **Value** | **Constant & Description** | **PHP** |
| 1 | **E\_ERROR**Fatal run-time errors. Errors that cannot be recovered from. Execution of the script is halted |  |
| 2 | **E\_WARNING**Non-fatal run-time errors. Execution of the script is not halted |  |
| 4 | **E\_PARSE**Compile-time parse errors. Parse errors should only be generated by the parser |  |
| 8 | **E\_NOTICE**Run-time notices. The script found something that might be an error, but could also happen when running a script normally |  |
| 16 | **E\_CORE\_ERROR**Fatal errors at PHP startup. This is like an E\_ERROR in the PHP core | 4 |
| 32 | **E\_CORE\_WARNING**Non-fatal errors at PHP startup. This is like an E\_WARNING in the PHP core | 4 |
| 64 | **E\_COMPILE\_ERROR**Fatal compile-time errors. This is like an E\_ERROR generated by the Zend Scripting Engine | 4 |
| 128 | **E\_COMPILE\_WARNING**Non-fatal compile-time errors. This is like an E\_WARNING generated by the Zend Scripting Engine | 4 |
| 256 | **E\_USER\_ERROR**Fatal user-generated error. This is like an E\_ERROR set by the programmer using the PHP function trigger\_error() | 4 |
| 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() | 4 |
| 1024 | **E\_USER\_NOTICE**User-generated notice. This is like an E\_NOTICE set by the programmer using the PHP function trigger\_error() | 4 |
| 2048 | **E\_STRICT**Run-time notices. PHP suggest changes to your code to help interoperability and compatibility of the code | 5 |
| 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()) | 5 |
| 8191 | **E\_ALL**All errors and warnings, except of level E\_STRICT | 5 |

## **List of Functions**

**PHP** − indicates the earliest version of PHP that supports the function.

|  |  |  |
| --- | --- | --- |
| **No** | **Advanced & Description** | **PHP** |
| 1 | debug\_backtrace()Generates a backtrace | 4 |
| 2 | debug\_print\_backtrace()Prints a backtrace | 5 |
| 3 | error\_get\_last()Gets the last error occurred | 5 |
| 4 | error\_log()Sends an error to the server error-log, to a file or to a remote destination | 4 |
| 5 | error\_reporting()Specifies which errors are reported | 4 |
| 6 | restore\_error\_handler()Restores the previous error handler | 4 |
| 7 | restore\_exception\_handler()Restores the previous exception handler | 5 |
| 8 | set\_error\_handler()Sets a user-defined function to handle errors | 4 |
| 9 | set\_exception\_handler()Sets a user-defined function to handle exceptions | 5 |
| 10 | trigger\_error()Creates a user-defined error message | 4 |
| 11 | user\_error()Alias of trigger\_error() | 4 |

## **Bugs Debugging**

Programs rarely work correctly the first time. Many things can go wrong in your program that cause the PHP interpreter to generate an error message. You have a choice about where those error messages go. The messages can be sent along with other program output to the web browser. They can also be included in the web server error log.

To make error messages display in the browser, set the **display\_errors** configuration directive to **On**. To send errors to the web server error log, set **log\_errors** to On. You can set them both to On if you want error messages in both places.

PHP defines some constants you can use to set the value of **error\_reporting** such that only errors of certain types get reported: E\_ALL (for all errors except strict notices), E\_PARSE (parse errors), E\_ERROR (fatal errors), E\_WARNING (warnings), E\_NOTICE (notices), and E\_STRICT (strict notices).

While writing your PHP program, it is a good idea to use PHP-aware editors like **BBEdit** or **Emacs**. One of the special special features of these editors is syntax highlighting. It changes the color of different parts of your program based on what those parts are. For example, strings are pink, keywords such as if and while are blue, comments are grey, and variables are black.

Another feature is quote and bracket matching, which helps to make sure that your quotes and brackets are balanced. When you type a closing delimiter such as }, the editor highlights the opening { that it matches.

There are following points which need to be verified while debugging your program.

* **Missing Semicolons** − Every PHP statement ends with a semicolon (;). PHP doesn't stop reading a statement until it reaches a semicolon. If you leave out the semicolon at the end of a line, PHP continues reading the statement on the following line.
* **Not Enough Equal Signs** − When you ask whether two values are equal in a comparison statement, you need two equal signs (==). Using one equal sign is a common mistake.
* **Misspelled Variable Names** − If you misspelled a variable then PHP understands it as a new variable. Remember: To PHP, $test is not the same variable as $Test.
* **Missing Dollar Signs** − A missing dollar sign in a variable name is really hard to see, but at least it usually results in an error message so that you know where to look for the problem.
* **Troubling Quotes** − You can have too many, too few, or the wrong kind of quotes. So check for a balanced number of quotes.
* **Missing Parentheses and curly brackets** − They should always be in pairs.
* **Array Index** − All the arrays should start from zero instead of 1.

Moreover, handle all the errors properly and direct all trace messages into system log file so that if any problem happens then it will be logged into system log file and you will be able to debug that problem.