PHP - Error Handling

Error handling in PHP refers to the making a provision in PHP code to effectively identifying and recovering from runtime errors that the program might come across. In PHP, the errors are handled with the help of –

- The die() function
- The Error Handler Function

The die() Function

The die() function is an alias of exit() in PHP. Both result in termination of the current PHP script when encountered. An optional string if specified in the parenthesis, will be output before the program terminates.

```
die("message");
```

Example

The following code is a typical usage of die() in a PHP script. It displays the File not found message if PHP doesn't find a file, otherwise proceeds to open it for subsequent processing.

</> Open Compiler

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

      // Rest of the code here.
      fclose($file);
   }
?>
```

It will produce the following **output** –

```
File not found
```

Using above technique, you can stop your program whenever it errors out and display more meaningful and user friendly message, rather than letting PHP generate fatal error message.

The Error Handler Function

Using die() for error handling is considered an ungainly and poor program design, as it results in an ugly experience for site users. PHP offers a more elegant alternative with which you can define a custom function and nominate it for handling the errors.

The set_error_handler() function has the following parameters −

```
set_error_handler(?callable $callback, int $error_levels = E_ALL): ?callable
```

The first parameter is a user defined function which is called automatically whenever an error is encountered.

The custom error handler callback function should have the following parameters −

```
handler(
   int $errno,
   string $errstr,
   string $errfile = ?,
   int $errline = ?,
   array $errcontext = ?
): bool
```

Parameters

Parameter	Importance	Description
errno	Required	It specifies the error level for the user-defined error. It must be numerical value.
errstr	Required	It specifies the error message for the user-defined error.
errfile	Optional	It specifies the filename in which the error occurred.
errline	Optional	It specifies the line number at which the error occurred.
errcontext	Optional	It specifies an array containing variables and their values

		in use when the error occurred.

If the callback function returns false, the default error will be called.

The **$errno** is an integer corresponding to the predefined error levels.

Sr.No	Constant & Description	Value
1	**E_ERROR** (int) Fatal run-time errors that can not be recovered from. Execution of the script is halted.	1
2	**E_WARNING** (int) Run-time warnings (non-fatal errors). Execution of the script is not halted.	2
3	**E_PARSE** (int) Compile-time parse errors. Parse errors should only be generated by the parser.	4
4	**E_NOTICE** (int) Run-time notices. Something that could indicate an error, but could also happen in the normal course of running a script.	8
5	**E_CORE_ERROR** (int) Fatal errors that occur during PHP's initial startup. This is like an **E_ERROR**	16
6	**E_CORE_WARNING** (int) Warnings (non-fatal errors) that occur during PHP's initial startup. This is like an **E_WARNING**,	32
7	**E_COMPILE_ERROR** (int) Fatal compile-time errors. This is like an **E_ERROR**.	64
8	**E_COMPILE_WARNING** (int) Compile-time warnings (non-fatal errors). This is like an **E_WARNING**.	128
9	**E_USER_ERROR** (int) User-generated error message. This is like an **E_ERROR**, generated in PHP code by using the PHP function trigger_error().	256
10	**E_USER_WARNING** (int) User-generated warning message. This is like an **E_WARNING**, generated in PHP code by using the function trigger_error().	512
11	**E_USER_NOTICE** (int)	1024

	User-generated notice message. This is like an **E_NOTICE** generated in PHP code by using the function trigger_error().	
12	**E_STRICT** (int) Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code.	2048
13	**E_RECOVERABLE_ERROR** (int) Catchable fatal error. If the error is not caught by a user defined handler, the application aborts as it was an **E_ERROR**.	4096
14	**E_DEPRECATED** (int) Run-time notices. Enable this to receive warnings about code that will not work in future versions.	8192
15	**E_USER_DEPRECATED** (int) User-generated warning message. This is like an **E_DEPRECATED**, generated in PHP code by using the function trigger_error().	16384
16	**E_ALL** (int) All errors, warnings, and notices.	32767

Example

Take a look at the following example −

Open Compiler

```php
<?php
   error_reporting(E_ERROR);

   function myerrorhandler($errno, $errstr) {
      echo "error No: $errno Error message: $errstr" . PHP_EOL;
      echo "Terminating PHP script";
      die();
   }

   set_error_handler("myerrorhandler");

   $f = fopen("nosuchfile.txt", "r");
   echo "file opened successfully";
   // rest of the code
   fclose($f);
?>
```

It will produce the following **output** –

```
error No: 2 Error message: fopen(nosuchfile.txt): Failed to open stream: No
such file or directory
Terminating PHP script
```

PHP's error class hierarchy starts from throwable interface. All the predefined Error classes in PHP are inherited from Error class.

The ArithmeticError Class

The **ArithmeticError class** is inherited from the **Error class.** This type of error may occur while performing certain mathematical operations such as performing bitwise shift operation by negative amount.

Example

Take a look at the following example –

Open Compiler

```php
<?php
   try {
      $a = 10;
      $b = -3;
      $result = $a << $b;
   }
   catch (ArithmeticError $e) {
      echo $e->getMessage();
   }
?>
```

It will produce the following **output** –

```
Bit shift by negative number
```

This error is also thrown when call to intdiv() function results in value such that it is beyond the legitimate boundaries of integer.

Example

Take a look at the following example −

</> Open Compiler

```php
<?php
   try {
      $a = PHP_INT_MIN;
      $b = -1;
      $result = intdiv($a, $b);
      echo $result;
   }
   catch (ArithmeticError $e) {
      echo $e->getMessage();
   }
?>
```

It will produce the following **output** −

```
Division of PHP_INT_MIN by -1 is not an integer
```

DivisionByZeroError

DivisionByZeroError class is a subclass of ArithmeticError class. This type of error occurs when value of denominator is zero in the division operation.

Example: Modulo by Zero

Take a look at the following example:

</> Open Compiler

```php
<?php
   try {
      $a = 10;
      $b = 0;
      $result = $a%$b;
      echo $result;
   }
   catch (DivisionByZeroError $e) {
      echo $e->getMessage();
   }
?>
```

It will produce the following **output** –

```
Modulo by zero
```

This can also occur when a modulo operator (%) has 0 as second operator, and intdiv() function having second argument as 0.

Example: Division by Zero

Take a look at the following example –

Open Compiler

```
<?php
   try {
      $a = 10;
      $b = 0;
      $result = $a/$b;
      echo $result;
   }
   catch (DivisionByZeroError $e) {
      echo $e->getMessage();
   }
?>
```

It will produce the following **output** –

```
Division by zero
```

ArgumentCountError

PHP parser throws ArgumentCountError when arguments passed to a user defined function or method are less than those in its definition.

Example

Take a look at the following example −

Open Compiler

```
<?php
   function add($x, $y) {
      return $x+$y;
   }
   try {
      echo add(10);
   }
   catch (ArgumentCountError $e) {
      echo $e->getMessage();
   }
?>
```

It will produce the following **output** −

```
Too few arguments to function add(), 1 passed in C:\xampp\php\test.php on line 9 and e
```

TypeError

This error is raised when actual and formal argument types don't match, return type doesn't match the declared returned type.

Example

Take a look at the following example −

Open Compiler

```
<?php
   function add(int $first, int $second) {
      echo "addition: " . $first + second;
   }

   try {
      add('first', 'second');
   }
   catch (TypeError $e) {
      echo $e->getMessage(), "";
   }
?>
```

It will produce the following **output** –

```
add(): Argument #1 ($first) must be of type int, string given,
   called in /home/cg/root/63814/main.php on line 7
```

TypeError is also thrown when PHP's built-in function is passed incorrect number of arguments. However, the "strict_types=1" directive must be set in the beginning.

Example

Take a look at the following example –

</> Open Compiler

```
<?php
   declare(strict_types=1);
   try {
      echo pow(100,2,3);
   }
   catch (TypeError $e) {
      echo $e->getMessage(), "";
   }
?>
```

It will produce the following **output** –

```
pow() expects exactly 2 parameters, 3 given
```

Exceptions Handling in PHP

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

Lets explain there 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.

Open Compiler

```
<?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(), "";
   }

   // 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

Take a look at the following example −

```
<?php
   function exception_handler($exception) {
      echo "Uncaught exception: " , $exception->getMessage(), "\n";
   }

   set_exception_handler('exception_handler');
   throw new Exception('Uncaught Exception');

   echo "Not Executed";
?>
```

Check complete set of error handling functions at PHP Error Handling Functions