PHP Error Handling

In this tutorial you will learn how to use the PHP's error handling functions to deal with the error conditions gracefully.

Handling Errors

Sometimes your application will not run as it supposed to do, resulting in an error. There are a number of reasons that may cause errors, for example:

- The Web server might run out of disk space
- A user might have entered an invalid value in a form field
- The file or database record that you were trying to access may not exist
- The application might not have permission to write to a file on the disk
- A service that the application needs to access might be temporarily unavailable

These types of errors are known as runtime errors, because they occur at the time the script runs. They are distinct from syntax errors that need to be fixed before the script will run.

A professional application must have the capabilities to handle such runtime error gracefully. Usually this means informing the user about the problem more clearly and precisely.

Understanding Error Levels

Usually, when there's a problem that prevents a script from running properly, the PHP engine triggers an error. Each error is represented by an integer value and an associated constant. The following table list some of the common error levels:

Error Level	Value	Description
`E_ERROR`	1	A fatal run-time error, that can't be recovered from. The execution of the script is stopped immediately.
`E_WARNING`	2	A run-time warning. It is non-fatal and most errors tend to fall into this category. The execution of the script is not stopped.
`E_NOTICE`	8	A run-time notice. Indicate that the script encountered something that could possibly an error, although the situation could also occur when running a script normally.
`E_USER_ERROR`	256	A fatal user-generated error message. This is like an `E_ERROR`, except it is generated by the PHP script using the function `trigger_error()` rather than the PHP engine.
`E_USER_WARNING`	512	A non-fatal user-generated warning message. This is like an `E_WARNING`, except it is generated by the PHP script using the function `trigger_error()` rather than the PHP. engine

E_USER_NOTICE	1024	A user-generated notice message. This is like an `E_NOTICE`, except it is generated by the PHP script using the function `trigger_error()` rather than the PHP engine.
E_STRICT	2048	Not strictly an error, but triggered whenever PHP encounters code that could lead to problems or forward incompatibilities
E_ALL	8191	All errors and warnings, except of `E_STRICT` prior to PHP 5.4.0.

For more error levels, please check out the reference on PHP Error Levels.

The PHP engine triggers an error whenever it encounters a problem with your script, but you can also trigger errors yourself to generate more user friendly error messages. This way you can make your application more sofisticated. The following section describes some of common methods used for handling errors in PHP:

Basic Error Handling Using the die() Function

Consider the following example that simply tries to open a text file for reading only.

Example **Download**

```php
<?php
// Try to open a non-existent file
$file = fopen("sample.txt", "r");
?>
```

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

```
Warning: fopen(sample.txt) [function.fopen]: failed to open stream: No such file or directory in C:\wamp\www\project\test.php on line 2
```

If we follow some simple steps we can prevent the users from getting such error message.

Example **Download**

```php
<?php
if(file_exists("sample.txt")){
    $file = fopen("sample.txt", "r");
} else{
    die("Error: The file you are trying to access doesn't exist.");
}
?>
```

Now if you run the above script you will get the error message like this:

```
Error: The file you are trying to access doesn't exist.
```

As you can see by implementing a simple check whether the file exist or not before trying to access it, we can generate an error message that is more meaningful to the user.

The `die()` function used above simply display the custom error message and terminate the current script if 'sample.txt' file is not found.

Creating a Custom Error Handler

You can create your own error handler function to deal with the run-time error generated by PHP engine. The custom error handler provides you greater flexibility and better control over the errors, it can inspect the error and decide what to do with the error, it might display a message to the user, log the error in a file or database or send by e-mail, attempt to fix the problem and carry on, exit the execution of the script or ignore the error altogether.

The custom error handler function must be able to handle at least two parameters (errno and errstr), however it can optionally accept an additional three parameters (errfile, errline, and errcontext), as described below:

Parameter	Description
Required — The following parameters are required	
errno	Specifies the level of the error, as an integer. This corresponds to the appropriate error level constant (`E_ERROR`, `E_WARNING`, and so on)
errstr	Specifies the error message as a string
Optional — The following parameters are optional	
errfile	Specifies the filename of the script file in which the error occurred, as a string
errline	Specifies the line number on which the error occurred, as a string
errcontext	Specifies an array containing all the variables and their values that existed at the time the error occurred. Useful for debugging

Here's an example of a simple custom error handling function. This handler, `customError()` is triggered whenever an error occurred, no matter how trivial. It then outputs the details of the error to the browser and stops the execution of the script.

Example **Download**

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

You need to tell the PHP to use your custom error handler function — just call the built-in `set_error_handler()` function, passing in the name of the function.

Example | Download

```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);
?>
```

Error Logging

Log Error Messages in a Text File

You can also logs details of the error to the log file, like this:

Example | Download

```php
<?php
function calcDivision($dividend, $divisor){
    if($divisor == 0){
        trigger_error("calcDivision(): The divisor cannot be zero",
E_USER_WARNING);
        return false;
    } else{
        return($dividend / $divisor);
    }
}
function customError($errno, $errstr, $errfile, $errline, $errcontext){
    $message = date("Y-m-d H:i:s - ");
    $message .= "Error: [" . $errno ."], " . "$errstr in $errfile on
line $errline, ";
    $message .= "Variables:" . print_r($errcontext, true) . "\r\n";

    error_log($message, 3, "logs/app_errors.log");
    die("There was a problem, please try again.");
}
set_error_handler("customError");
echo calcDivision(10, 0);
echo "This will never be printed.";
?>
```

Send Error Messages by E-Mail

You can also send e-mail with the error details using the same `error_log()` function.

Example

Download

```
<?php
function calcDivision($dividend, $divisor){
    if ($divisor == 0){
        trigger_error("calcDivision(): The divisor cannot be zero", 
E_USER_WARNING);
        return false;
    } else{
        return($dividend / $divisor);
    }
}
function customError($errno, $errstr, $errfile, $errline, $errcontext){
    $message = date("Y-m-d H:i:s - ");
    $message .= "Error: [" . $errno ."], " . "$errstr in $errfile on 
line $errline, ";
    $message .= "Variables:" . print_r($errcontext, true) . "\r\n";
    
    error_log($message, 1, "webmaster@example.com");
    die("There was a problem, please try again. Error report submitted 
to webmaster.");
}
set_error_handler("customError");
echo calcDivision(10, 0);
echo "This will never be printed.";
?>
```

Trigger an Error

Although the PHP engine triggers an error whenever it encounters a problem with your script, however you can also trigger errors yourself. This can help to make your application more robust, because it can flag potential problems before they turn into serious errors.

To trigger an error from within your script, call the `trigger_error()` function, passing in the error message that you want to generate:

```
trigger_error("There was a problem.");
```

Consider the following function that calculates division of the two numbers.

Example **Download**

```php
<?php
function calcDivision($dividend, $divisor){
    return($dividend / $divisor);
}

// Calling the function
echo calcDivision(10, 0);
?>
```

If a value of zero (0) is passed as the `$divisor` parameter, the error generated by the PHP engine will look something like this:

```
Warning: Division by zero in C:\wamp\www\project\test.php on line 3
```

This message doesn't look very informative. Consider the following example that uses the `trigger_error()` function to generate the error.

Example **Download**

```php
<?php
function calcDivision($dividend, $divisor){
    if($divisor == 0){
        trigger_error("The divisor cannot be zero", E_USER_WARNING);
        return false;
    } else{
        return($dividend / $divisor);
    }
}

// Calling the function
echo calcDivision(10, 0);
?>
```

Now the script generates this error message:

```
Warning: The divisor cannot be zero in C:\wamp\www\project\error.php on line 4
```

As you can see the error message generated by the second example explains the problem more clearly as compared to the previous one.