Error Handling

- Throwable errors
  - Global exception handler
  - Extending exceptions
  - Throwable hierarchy
- Diagnostic errors
  - User-defined diagnostic handler

Errors are a normal part of the development and life cycle of an application. As such, PHP provides multiple ways to report and deal with errors.

PHP has two main mechanisms for reporting errors:

- Throwable errors, which are further split into *programming* errors via the Error class hierarchy and *recoverable* errors via the Exception class hierarchy. These errors can be thrown with the ENTITY keyword and caught with the ENTITY keyword.
- Diagnostic errors, which are classified into different severities.

Throwable errors

A Throwable error is an execution event which disrupts the normal flow of instructions. When a Throwable error is ENTITYn, it will unwind the call stack until it is caught by a ENTITY block or the global scope is reached. All ENTITY blocks it encounters along the way will be executed. An uncaught Throwable error is passed to the exception handler set with `set_exception_handler`, if one is set; otherwise, the program is terminated with a fatal error.

To throw a Throwable error from within PHP code, use the ENTITY keyword.

> The Standard PHP Library (SPL) provides various built-in exceptions. It is also possible to create custom exceptions by extending the Exception class.

It is possible to handle a Throwable error within PHP by surrounding the code that may throw an error in a ENTITY block.

> Throwable objects cannot be cloned. Attempting to clone such an object throws an Error with the following message: `Trying to clone an uncloneable object of class ClassName`. This happens even if a subclass declares its own __clone() method. See Exception::__clone and Error::__clone.

Global exception handler

It's possible to add a handler for any uncaught Throwable with `set_exception_handler`. This is similar to wrapping the entire code in a ENTITY-ENTITY block, except that execution does not resume afterwards: the script terminates once the handler returns.

Extending exceptions

A user-defined exception can be defined by extending the built-in Exception class.

If a class extends the built-in Exception class and redefines the constructor, it is highly recommended that it also call `parent::__construct()` to ensure all available data has been properly assigned. The __toString() method can be overridden to provide custom output when the object is used as a string.

Extending Exception

```
<?php
class CustomException extends Exception
{
    // Redefine the constructor so message isn't optional
    public function __construct($message, $code = 0, ?Throwable $previous = null) {
        // some code

        // Ensure everything is assigned properly
        parent::__construct($message, $code, $previous);
    }

    // Custom string representation of object
```

```
    public function __toString() {
        return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
    }

    public function customFunction() {
        echo "A custom function for this type of exception\n";
    }
}

try {
    throw new CustomException("Something happened", 5);
} catch (CustomException $e) {
    echo "Caught custom exception\n", $e;
    $e->customFunction();
}
```

ENTITY

```
Caught custom exception
CustomException: [5]: Something happened
A custom function for this type of exception
```

Throwable hierarchy

- Throwable
  - Error
    - ArithmeticError
      - DivisionByZeroError
    - AssertionError
    - CompileError
      - ParseError
    - TypeError
      - ArgumentCountError
    - ValueError
    - UnhandledMatchError
    - FiberError
  - Exception

The Exception class has many subclasses, among them the SPL exceptions and those defined by extensions.

Diagnostic errors

Diagnostic errors, also known as traditional errors, are used to signal a number of different conditions, and can be displayed and/or logged as required. By default, PHP will report all diagnostic errors. (Prior to PHP 8.0.0, `E_NOTICE` and `E_DEPRECATED` diagnostics were not reported by default.)

In contrast to Throwable errors, most diagnostic errors do *not* disrupt the normal flow of execution. Fatal diagnostics, such as `E_ERROR`, terminate the script.

Which diagnostics are reported and which are ignored is controlled by the `error_reporting` ENTITY directive, or at runtime by calling `error_reporting`. It is strongly recommended that the configuration directive be set, as some errors can occur before execution of your script begins.

> In a development environment, `error_reporting` should always be set to `E_ALL` so that issues raised by PHP are noticed and fixed. In production, the level of verbosity can be reduced to `E_ALL ENTITY ~E_NOTICE ENTITY ~E_DEPRECATED`, although in many cases `E_ALL` is still appropriate, as it may provide early warning of potential issues.

What happens to the diagnostics depends on two further ENTITY directives. `display_errors` controls whether the diagnostic is shown as part of the script's output. This should always be disabled in a production environment, as the output can include confidential information such as database passwords, but is often useful to enable in development, as it ensures immediate reporting of issues.

In addition to displaying errors, PHP can log diagnostics when the `log_errors` directive is enabled. This will log any diagnostic to the file or syslog defined by `error_log`. This can be extremely useful in a production environment, as diagnostics are logged when they occur, and reports can then be generated from those logs.

Individual diagnostics can be suppressed using the `@` operator.

> Prior to PHP 8.0.0, it was possible to suppress critical diagnostics that would terminate the execution of the PHP script.

> Between PHP versions, diagnostics may be added or removed, have their severity changed, or be elevated to Throwable errors.

User-defined diagnostic handler

If PHP's default diagnostic handling is inadequate, it is possible to override the default diagnostic handler with a custom one which is set using `set_error_handler`. Some diagnostics, such as `E_ERROR` and `E_PARSE`, cannot be handled this way (see `set_error_handler` for the full list). The others can be handled in various ways. For example, a custom handler can show an error page to the user and report the error more directly than a log would, such as by sending an email.

Another typical usage is to elevate diagnostic errors to exceptions with ErrorException.

Elevating diagnostic errors to exceptions

```
<?php
function exceptions_error_handler($severity, $message, $filename, $lineno) {
    if (error_reporting() & $severity) {
        throw new ErrorException($message, 0, $severity, $filename, $lineno);
    }
}

set_error_handler('exceptions_error_handler');
```

> It is important to verify that the severity is included in the current list of reportable diagnostics; otherwise, the `@` operator will be broken, as an exception will be thrown for a suppressed diagnostic.

ENTITY

ENTITY	ENTITY
8.5.0	Fatal errors (such as an exceeded maximum execution time) now include a stack trace in their error message.

□□□ #2
□ □□□ □□□ 2026-10-31 17:39:26 UTC
□ □□□ □□□ 2026-10-31 17:39:28 UTC