The Complete Guide to Modern PHP

The Gemini Technical Writing Team

December 5, 2025

Preface

This book serves as an exhaustive reference and tutorial for PHP, focusing primarily on versions 8.0 and newer. The goal is to move beyond procedural scripts and into the realm of enterprise-level software engineering, covering everything from fundamental syntax and robust Object-Oriented Programming (OOP) to advanced features like the JIT compiler, security protocols, and modern dependency management. Whether you are migrating a legacy application or starting a new project, this guide provides the necessary theoretical depth and practical application to master the language that powers over 75% of the web.

Contents

Part I

PHP Foundations and Fundamentals

Chapter 1

Setting the Stage: History and Environment

1.1 A Brief History of PHP

PHP, originally standing for Personal Home Page, was created by Rasmus Lerdorf in 1994. It was designed as a set of Common Gateway Interface (CGI) binaries written in C.

1.1.1 Key Milestones

- **1995: PHP/FI** - Initial release, combining the parser with Form Interpreter capabilities.
- **1998: PHP 3** - Introduced by Andi Gutmans and Zeev Suraski, featuring improved architecture and extensibility. This version marked the beginning of PHP's mass adoption.
- **2000: PHP 4** - Powered by the first-generation Zend Engine, significantly improving performance and structure.
- **2004: PHP 5** - A massive leap forward with the Zend Engine II, introducing a robust, complete Object-Oriented Programming model that made PHP viable for large-scale enterprise applications.
- **2015: PHP 7** - A major performance revolution. The core engine was rewritten, leading to up to $2\times$ performance gains over PHP 5.6 and introducing scalar type hints and the Spaceship operator.
- **2020: PHP 8** - The current generation, introducing the Just-In-Time (JIT) compiler, Attributes, Union Types, and the `match` expression, confirming PHP's status as a modern, high-performance language.

1.2 Installation and Environment

PHP operates primarily as an interpreter running on a web server. The most common setup is the LAMP stack (Linux, Apache, MySQL, ~~PHP~~) or its variants (LEMP/Nginx, WAMP, MAMP).

1.2.1 The Server Request Lifecycle

1. A web request (HTTP) hits the web server (Apache/Nginx).
2. The server identifies the file extension (`.php`) and passes the request to the PHP interpreter (typically via FPM - FastCGI Process Manager).
3. The PHP engine executes the script, reading and executing the code line by line.
4. The engine may interact with databases (like MySQL) or external services.
5. The PHP script generates an output stream (usually HTML, JSON, or XML).
6. PHP FPM passes the output back to the web server.
7. The web server sends the final HTTP response back to the client.

Chapter 2

Syntax, Data Types, and Control Flow

2.1 Basic Syntax and Variables

PHP code blocks are embedded within HTML using the opening tag `<?php` and the closing tag `?>`.

2.1.1 Variables and Scoping

All variables in PHP are denoted by a leading dollar sign (`$`). PHP is dynamically typed, meaning a variable's type is determined by the data it holds.

```
<?php
$username = "Alice"; // string
$age = 30; // int
$is_active = true; // bool

// Function scope
function greet() {
    // $username here is undefined unless passed in or declared global
    global $username;
    echo "Hello, $username";
}
greet(); // Outputs: Hello, Alice
?>
```

2.2 Data Types

PHP supports eight primitive data types, categorized as Scalar, Compound, and Special.

2.2.1 Scalar Types

These hold a single value.

- `bool`: `true` or `false`.
- `int`: Whole numbers.

- `float`: Floating-point numbers (decimals).
- `string`: Sequence of characters.

2.2.2 Compound Types: Arrays

The PHP `array` is a versatile, ordered map that can be used as a list, a hash map (associative array), or a stack.

Table 2.1: Array Types in PHP

Type	Example	Description
Indexed	`[1, 5, 9]`	Simple, zero-indexed list.
Associative	`[ 'user' => 'Bob', 'id' => 101 ]`	Key-value pairs (like a dictionary).
Multidimensional	`[ [1, 2], [3, 4] ]`	Arrays containing other arrays.

2.3 Control Structures

Control structures dictate the flow of execution within a script.

2.3.1 Conditional Statements

Standard `if`, `else`, and `elseif` blocks are used for conditional execution.

The modern `match` expression (PHP 8.0+) is often preferred over `switch` due to its strict type comparison, lack of fall-through, and ability to return a value.

```
$result = match ($statusCode) {
    200, 201 => "Success",
    404 => "Not Found",
    default => "Unknown Error",
};
```

2.3.2 Loops

- `for`: Classic iterative loop.
- `while` / `do...while`: Condition-based loops.
- `foreach`: The most common loop for iterating over arrays and Traversable objects.

Part II

Object-Oriented Mastery

Chapter 3

Core Object-Oriented Principles

3.1 Classes, Properties, and Methods

Object-Oriented Programming (OOP) organizes code around objects—instances of classes.

3.1.1 Encapsulation and Visibility

Visibility keywords control access to properties and methods:

- `public`: Accessible from everywhere.
- `protected`: Accessible within the defining class and by inherited (child) classes.
- `private`: Accessible only within the defining class.

Encapsulation is the practice of bundling data (properties) and the methods that operate on that data, restricting direct access.

3.2 Constructors and Destructors

- **Constructor (`__construct`)**: A special method called automatically when an object is instantiated (`new ClassName()`). Used for setup and initialization.
- **Destructor (`__destruct`)**: Called when the object is destroyed or the script ends. Used for cleanup tasks (e.g., closing file handles).

3.2.1 PHP 8.0: Constructor Property Promotion

A feature simplifying class definitions by allowing properties to be declared and initialized directly in the constructor signature, reducing boilerplate code.

```
class User {
    public function __construct(
        private int $id,
        private string $name
    ) {}
    // Getters/setters omitted for brevity
}
```

Chapter 4

Inheritance, Contracts, and Traits

4.1 Inheritance and Polymorphism

Inheritance is a mechanism where a new class (*child* or *subclass*) is derived from an existing class (*parent* or *superclass*) using the `extends` keyword. It promotes code reuse.

Polymorphism means ”many forms.” In OOP, it allows an object to take many forms through method overriding (child classes implementing a parent’s method) or interface implementation.

4.1.1 Abstract Classes and Methods

An `abstract` class cannot be instantiated directly. It is designed to be a base class for others to inherit from. It can contain `abstract` methods, which must be implemented by any non-abstract child class.

4.2 Interfaces: Defining Contracts

An `interface` defines a contract: a set of methods that a class must implement. A class uses the `implements` keyword to adopt an interface. Interfaces ensure that different classes can be used interchangeably based on their capabilities, promoting loose coupling.

4.3 Traits: Horizontal Reuse

PHP supports single inheritance, meaning a class can only inherit from one parent. `Traits` were introduced to solve the problem of code reuse across unrelated classes. A trait is a group of methods that can be injected into multiple classes using the `use` keyword.

Part III

Modern PHP and Enterprise-Grade Features

Chapter 5

PHP 8 and Beyond: The New Landscape

5.1 The Just-In-Time (JIT) Compiler

Introduced in PHP 8.0, the JIT compiler is the most significant performance innovation since PHP 7.

5.1.1 How JIT Works

Unlike a standard interpreter, which executes Zend Opcode one instruction at a time, the JIT compiler identifies ”hot” (frequently executed) pieces of code and compiles them into native machine code. This is particularly beneficial for:

- CPU-intensive tasks (e.g., complex math, cryptography, data processing).
- Long-running processes and command-line scripts.

While web requests often remain I/O bound, the JIT still offers substantial gains in micro-benchmarks and high-load scenarios.

5.2 Attributes (Annotations)

Attributes (PHP 8.0+) provide a standardized way to add structured metadata to declarations of classes, properties, methods, functions, and parameters. They replace the reliance on parsing doc-block comments (`PHPDoc`) for configuration.

```
use App\Routing\Route;

class PostController
{
    #[Route("/posts/{id}", methods: ["GET"])]
    public function show(int $id)
    {
        // ...
    }
}
```

5.3 Enhanced Type System

Modern PHP focuses heavily on static analysis and type safety.

5.3.1 Union Types

(PHP 8.0+) Allows a variable, argument, or return value to accept multiple different types.

$$\text{function process}(\text{int}|\text{float } \$value) : \text{void}$$

5.3.2 Intersection Types

(PHP 8.1+) Requires a variable to simultaneously implement multiple interfaces.

$$\text{function loggable}(\text{Logger\&Contextable } \$object) : \text{void}$$

This forces `$object` to be an instance of a class that implements *both* the `Logger` and `Contextable` interfaces.

5.3.3 Readonly Properties

(PHP 8.1+) Properties can be declared with the `readonly` keyword, meaning they can only be initialized once (either in the declaration or the constructor) and cannot be modified thereafter. This is crucial for creating immutable value objects.

Chapter 6

Error Handling and Logging

6.1 Exceptions vs. Errors

- **Exceptions**: Recoverable errors that should be anticipated and handled gracefully using `try`/`catch` blocks.
- **Errors**: Non-recoverable problems (e.g., syntax errors, memory exhaustion) handled by PHP's internal error handler. Since PHP 7, many fatal errors are now thrown as `Error` exceptions, which can still be caught.

6.2 The `try...catch...finally` Block

This structure is fundamental for robust code:

```
try {
    // Code that might throw an exception
    $result = riskyFunction();
} catch (SpecificException $e) {
    // Handle the specific, expected failure
    logError($e->getMessage());
} catch (\Throwable $e) {
    // Catch-all for unexpected Exceptions or Errors
    // Always catch the Throwable interface (PHP 7+)
    handleFatal($e);
} finally {
    // Code that runs regardless of success or failure
    cleanupResources();
}
```

6.3 Logging Standards (PSR-3)

The PHP Standards Recommendations (PSRs) define common interfaces. PSR-3 defines the Logger Interface, ensuring that any logging library (like Monolog) can be used consistently across frameworks. Standardized logging is vital for application monitoring and debugging in production environments.

Part IV

Interacting with the World

Chapter 7

Database Connectivity (PDO)

7.1 PHP Data Objects (PDO)

PDO is a thin, consistent layer for accessing databases in PHP. It provides a data-access abstraction layer, meaning that regardless of which database you use (MySQL, PostgreSQL, SQLite), you use the same methods to query and retrieve data.

7.1.1 Connection and Transactions

A typical PDO connection involves:

1. Creating a Data Source Name (DSN) string.
2. Instantiating the `PDO` class.
3. Handling potential connection exceptions.

Transactions are critical for data integrity, allowing a sequence of database operations to be treated as a single unit. Use `beginTransaction()`, `commit()`, and `rollBack()`.

7.2 Security: Prepared Statements

This is the single most important defense against SQL Injection (SQLi) attacks. Prepared statements separate the SQL command structure from the data input.

7.2.1 The Process of Parameterized Queries

1. **Prepare**: The SQL template is sent to the database server with placeholders (e.g., `?`).
2. **Bind**: The user-supplied data is securely bound to the placeholders. The database treats this data as pure data, not executable SQL code.
3. **Execute**: The query is run with the bound data.

Chapter 8

Web Security Protocols

8.1 Injection Prevention

8.1.1 Cross-Site Scripting (XSS)

XSS occurs when an attacker injects client-side scripts (HTML/JavaScript) into a web page viewed by others.

- **Rule**: Never trust user input.
- **Defense**: Always escape output. Use `htmlspecialchars()` or, preferably, templating engines like Twig or Blade that automatically escape variables by default.

8.1.2 SQL Injection (SQLi)

As covered in Chapter 7, the use of Prepared Statements is the primary defense. Never use string concatenation to build database queries.

8.2 Cross-Site Request Forgery (CSRF)

A CSRF attack forces an end-user to execute unwanted actions on a web application in which they are currently authenticated.

- **Defense**: Use anti-CSRF tokens. A unique, unpredictable token must be generated on the server for every state-changing form/request and validated upon submission. Modern frameworks handle this automatically.

8.3 Secure Authentication

8.3.1 Password Hashing

Passwords must **never** be stored in plaintext. PHP provides built-in, secure hashing functions:

- `password_hash($password, PASSWORD_ARGON2ID)`: Creates a secure hash. `ARGON2ID` is the recommended algorithm.
- `password_verify($password, $hash)`: Verifies a plaintext password against a stored hash.

Part V

Ecosystem and Frameworks

Chapter 9

The PHP Ecosystem and Standards

9.1 Composer: Dependency Management

Composer is the standard dependency manager for PHP. It manages project dependencies, autoloading classes, and adheres to strict version control.

9.1.1 The `composer.json` File

This configuration file declares a project's required dependencies (`require`), development-only dependencies (`require-dev`), and defines the autoloading strategy for custom code.

9.2 Packagist

Packagist is the main repository for Composer packages, hosting thousands of reusable PHP libraries. When you run `composer install`, Composer fetches package definitions from Packagist and downloads the code from GitHub (or other sources).

9.3 PHP Standards Recommendations (PSRs)

The PHP-FIG (Framework Interoperability Group) created PSRs to standardize component interfaces, coding styles, and common practices.

- **PSR-4**: Defines a standard for autoloading classes from file paths, making framework components interchangeable.
- **PSR-7/15/17**: Define HTTP message interfaces, server request handlers, and HTTP factories, crucial for modern, decoupled applications.

Chapter 10

Major Frameworks

10.1 Laravel

Laravel is the most popular modern PHP framework, known for its elegant syntax, comprehensive documentation, and developer-friendly features.

10.1.1 Key Features

- **Eloquent ORM**: An object-relational mapper providing an expressive, ActiveRecord implementation for database interaction.
- **Blade Templating Engine**: Simple yet powerful templating for views, offering automatic output escaping.
- **Artisan CLI**: A command-line interface providing dozens of helpful commands for database migration, testing, and code generation.

10.2 Symfony

Symfony is a highly stable, component-based framework often preferred for large, long-term enterprise projects due to its strict adherence to standards and architecture.

10.2.1 Key Features

- **Reusable Components**: Symfony consists of over fifty decoupled components (like Validator, DependencyInjection, Security) that can be used independently in any PHP project.
- **Dependency Injection**: Its core principle relies on robust dependency injection, making unit testing and maintenance easier.
- **Doctrine ORM**: A powerful Data Mapper ORM that focuses on decoupling business objects from the database.

Glossary

- **JIT**: Just-In-Time Compiler. Compiles frequent PHP bytecode into native machine code.
- **PDO**: PHP Data Objects. The standard extension for database access.
- **ORM**: Object-Relational Mapper. Maps database tables to PHP objects.
- **PSR**: PHP Standards Recommendations. Guidelines for PHP development created by the FIG.
- **FPM**: FastCGI Process Manager. The bridge between the web server and the PHP interpreter.
- **XSS**: Cross-Site Scripting. A type of injection vulnerability.