Logbook Project

PHP Logging Library

Anonymous · 21/08/2026 · #php

PHP Logging Library

A flexible, PSR-3 compliant PHP logging library that leverages Monolog and add additional features while maintaining a clean and simple API.

Features

- **PSR-3 Compatible**: Implements the PSR-3 logging interface for standardized usage.
- **In-Memory Log Storage**: Store logs in memory for retrieval during runtime.
- **Caller Information**: Automatically track which class, method, file, and line generated each log message.
- **Custom Formatting**: Enhanced line formatter that includes caller information.
- **Multiple Log Levels**: Support for all standard log levels (debug, info, notice, warning, error, critical, alert, emergency).
- **Context Support**: Add structured data to your log messages.
- **Monolog Integration**: Built on top of Monolog for robust logging capabilities.

Installation

```
composer require derafu/log
```

Basic Usage

```
use Derafu\Log\Level;
use Derafu\Log\Service\Logger;
use Derafu\Log\Service\LogService;
use Derafu\Log\Service\LineFormatter;

// Create a logger with a line formatter.
$logger = new Logger(
    formatter: new LineFormatter()
```

```
);

// Create the log service.
$logService = new LogService($logger);

// Log messages with different levels.
$logService->logger()->debug('Debug message');
$logService->logger()->info('Info message');
$logService->logger()->warning('Warning message', ['context' => 'value']);
$logService->logger()->error('Error message', ['error_code' => 500]);

// Retrieve all logs.
$allLogs = $logService->logs();

// Retrieve only error logs.
$errorLogs = $logService->logs(Level::ERROR);

// Retrieve logs, newest first (default).
$newestFirst = $logService->logs(null, true);

// Retrieve logs, oldest first.
$oldestFirst = $logService->logs(null, false);

// Clear logs.
$logService->clear(); // Clear all logs.
$logService->clear(Level::ERROR); // Clear only error logs.

// Retrieve logs and clear them in one operation.
$logs = $logService->flush();
```

Caller Information

One of the key features of this library is automatic tracking of the caller information. Every log includes details about which class, method, file, and line generated the log message:

```
$logService->logger()->error('Something went wrong');

// The log will include caller information automatically.
$logs = $logService->logs();
$caller = $logs[0]->getCaller();

echo $caller; // Outputs: in /path/to/file.php on line 42, called by
Namespace\Class::method()
```

Working with Log Records

Each log record provides methods to access its data:

```
$logs = $logService->logs();
$log = $logs[0];

// Access log data.
$level = $log->getCode();
$message = $log->getMessage();
$context = $log->getContext();
$caller = $log->getCaller();
```

Advanced Configuration

You can customize the logger with additional handlers:

```
use Monolog\Handler\StreamHandler;
use Monolog\Level;

// Create a logger with additional handlers.
$logger = new Logger(
    configuration: ['channel' => 'app'],
    handlers: [
        new StreamHandler('path/to/your.log', Level::Debug),
        // Add more handlers as needed.
    ],
    formatter: new LineFormatter()
);

$logService = new LogService($logger);
```

Contributing

Contributions are welcome! Please feel free to submit a Pull Request. For major changes, please open an issue first to discuss what you would like to change.

License