# Logger Documentation

***Release 1.0.0***

**Nickolas Whiting**

March 28, 2016

Contents

A stupid simple logger based on Python's logger.

CHAPTER 1

# Installation

## 1.1 XPSPL

Logger is already installed on your system.

## 1.2 Standalone

Download the latest.

Include in configuration.

```
require_once 'logger/__init__.php';
```

CHAPTER 2

# Logging

Logging is performed using the `logger` function.

This returns a `Logger` object used to log using the methods `debug`, `info`, `warning`, `error`, `critical` in that order using a 0 - 5 level.

Logger also provides the ability to globally share logs using a single api and with good practice control our systems entire log with a single constant definition.

CHAPTER 3

# Log to STDOUT

Create a new logger and send output to STDOUT.

```php
<?php

$log = logger('foo-log');
$formatter = new Formatter(
    '[{date}] [{str_code}] {message}'.PHP_EOL
);
$log->add_handler(new Handler(
    $formatter, STDOUT
));

$log->info('About to perform foo action');
```

CHAPTER 4

# Logging to multiple files at once

Creates a new logger and sends output to STDOUT and log file.

```
<?php

$log = logger('foo-log');
$formatter = new Formatter(
    '[{date}] [{str_code}] {message}'.PHP_EOL
);
$log->add_handler(new Handler(
    $formatter, STDOUT
));
$log->add_handler(new Handler(
    $formatter, '/tmp/foo.log'
));
```

CHAPTER 5

# Constants

LOGGER_LOG_LEVEL

The default log level code to use when logging.

This constant must only be set once!

CHAPTER 6

# Function - logger

**logger** ( [ *$name = false* ] )

Returns a logger identified by the given name.

If the logger does not exist it is created.

**Parameters** **string** – Name of the logger

**Return type** object Logger

CHAPTER 7

# Class - Logger

Stupid simple logging utility for XPSPL based on Pythons logger.

CHAPTER 8

# Methods

**instance**()
Returns an instance of the singleton.

Passes args to constructor

**__clone**()
Disallow cloning

**add_handler**(*$handler*)
Adds a handler.

**Parameters** **object** – HJandler

**Return type** void

**log**(*$code*, *$message*)
Logs a message.

**Parameters**

- **integer** – Log level code.
- **string** – Message to log.

**Return type** void

**debug**(*$message*)
Logs a debug message.

**Parameters** **string** – Message to log.

**Return type** void

**info**(*$message*)
Logs a info message.

**Parameters** **string** – Message to log.

**Return type** void

**warning**(*$message*)
Logs a warning message.

**Parameters** **string** – Message to log.

**Return type** void

**error**(*$message*)
Logs a error message.

**Parameters** **string** – Message to log.

**Return type** void

**critical**(*$message*)
Logs a critical message.

**Parameters** **string** – Message to log.

**Return type** void

**get_logger**(*$logger*)
Returns a new logger.

**Parameters** **string** – Name of the logger.

**Return type** object Logger

CHAPTER 9

# Constants

DEBUG

Detailed information, typically of interest only when diagnosing problems.

INFO

Confirmation that things are working as expected.

WARNING

An indication that something unexpected happened, or indicative of some problem in the near future (e.g. ‘disk space low’).

The software is still working as expected.

ERROR

Due to a more serious problem, the software has not been able to perform some function.

CRITICAL

A serious error, indicating that the program itself may be unable to continue running.

CHAPTER 10

# Class - Handler

Handler

Handles a log message

CHAPTER 11

# Methods

**__construct** (*$formatter*, *$output*[, *$level = 2* ])
Sets the formatter.

**Parameters**

- **object** –
- **resource** – Output resource or file
- **integer** – Code level to log, anything greater than the given code will be logged.

**Return type** void

**handle** (*$code*, *$message*)
Handles a message.

**Parameters**

- **integer** – Log level code.
- **string** – Message to handle.

**Return type** void

**_make_writeable** ()
Makes the output writeable.

This will create a non-blocking stream to the given file.

**Return type** boolean

CHAPTER 12

# Class - Formatter

Formatter

Formats a log message.

The formatter allows for the following parameters.

%date - Date of the log %message - Log message %code - Error Code Level %str_code - String representation of the error code

CHAPTER 13

# Methods

**__construct** (*$format*)
Create a formatter.

**Parameters** **object** – String format to log a message

**Return type** void

**format** (*$code*, *$message*)
Handles a message.

**Parameters**

- **integer** – Log level code.
- **string** – Message to handle.

**Return type** void

# psprintf

**psprintf**()
: Returns a formatted string. Accepts named arguments.

Last updated on 02/04/13 10:38pm

## Symbols



## A



## C



## D



## E



## F



## G



## H



## I



## L



## P



## W