Docs

# Utils Category

Utils

Anonymous · 09/09/2026

# Table of Contents

# Utils Category

Utils

Anonymous · 09/09/2026

# Utils

Last updated on 09/09/2026

# ESCPOS Project

Derafu ESCPOS

Anonymous · 09/09/2026 · #php

# Derafu ESCPOS

Last updated on 09/09/2026

# Introduction

PHP Library for ESC/POS Printers

Anonymous · 09/09/2026

# PHP Library for ESC/POS Printers


A PHP library for generating ESC/POS commands to communicate with thermal receipt printers.

Underneath, it uses mike42/escpos-php to talk to the printer, but that library is an implementation detail: nothing in this package's public API exposes a `Mike42\Escpos\*` type or constant. See Architecture for why that matters and how it is enforced.

## Features

- Fluent, chainable API for common receipt printing tasks.
- Own vocabulary (enums and constants) for every ESC/POS value, instead of the underlying library's constants — see Vocabulary.
- 1D and 2D barcode generation, including PDF417 (native command or as a printed bitmap when the printer has no native support) — see Printing Images.
- Sending the generated bytes to a real network printer, without a raw socket or the underlying library's connectors — see Network Delivery.
- Special character handling and replacement.
- Receipt cutting and cash drawer opening.

## Installation

```
composer require derafu/escpos
```

## Quick Start

```
use Derafu\Escpos\Enum\Justification;
use Derafu\Escpos\EscposPrinter;

// Initialize the printer. With no options, it buffers ESC/POS bytes in
// memory instead of talking to a real printer — useful for building the
// receipt and deciding what to do with the bytes afterwards.
```

```
$printer = new EscposPrinter();

// Print some text.
$printer
    ->println('Hello World!')
    ->setJustification(Justification::Center)
    ->println('Centered Text')
    ->println('Multiple lines')
    ->feed(2); // Add some empty lines.

// Finalize the receipt (adds final line breaks, cuts paper, opens the
// drawer if configured) and get the ESC/POS bytes.
$escposData = $printer->end();

// Do whatever you need with the bytes: return them from an API, save them
// to a file, or send them to a real printer (see Network Delivery).
echo $escposData;
```

## Configuration Options

The `EscposPrinter` constructor accepts an options array as its first argument:

| Option | Type | Default | Description |
|---|---|---|---|
| `connector` | array | `['type' => 'memory']` | Printer connector configuration. `memory` is the only type supported today — it buffers bytes so `dump()`/`end()` can return them. |
| `profile` | string | `'default'` | Printer capability profile (from Mike42's `capabilities.json`). |
| `cut` | bool | `true` | Automatically cut the paper when `end()` is called. |
| `specialchars` | bool | `true` | When `false`, accented characters (á, ñ, etc.) are replaced by their unaccented equivalent instead of being sent as-is. |
| `pulse` | array | `['pin' => 0, 'on_ms' => 120, 'off_ms' => 240]` | Cash drawer pulse, sent together with the cut when `cut` is `true`. |

A second, optional constructor argument lets you replace the default backend — see Architecture.

## Full Example

The repository's `examples/full-example.php` exercises every feature (text formatting, image, barcode, PDF417, special characters). Run it with:

```
php examples/full-example.php | nc 172.16.1.5 9100
```

**Note**: you need `netcat` installed, and `172.16.1.5/9100` should be the IP and port of your thermal printer. See Network Delivery for a pure-PHP alternative to `nc`.

Last updated on 09/09/2026

# Vocabulary

Vocabulary

Anonymous · 09/09/2026

# Vocabulary

Every value `EscposPrinter` accepts (justification, print modes, barcode types, etc.) has its own type under `Derafu\Escpos\Enum\*` or `Derafu\Escpos\Constant\*`. You never need to import anything from `Mike42\Escpos\*` to call any method.

These values are **not arbitrary numbers chosen by this library**. They are the ESC/POS protocol itself — the bytes a thermal printer actually expects, defined by the standard and shared by every ESC/POS library regardless of language. That is why they are safe to hardcode here instead of referencing the underlying library's constants: they would not change even if the underlying library were replaced.

## Enum vs. constant

- **Enums** (`Derafu\Escpos\Enum\*`) are used when the values are a mutually exclusive choice — you pick exactly one (e.g. left, center or right justification).
- **Constants** (`Derafu\Escpos\Constant\*`) are used when the values are independent bits meant to be combined with `|` (e.g. font + emphasized + double width at the same time). An enum would force unwrapping `->value` on every term just to combine them, and the combined result would no longer be the enum type anyway — a plain `int` constant says what it is more honestly.

## Enums

| Enum | Cases | Used by |
|---|---|---|
| `Justification` | `Left`, `Center`, `Right` | `EscposPrinter::setJustification()` |
| `BarcodeType` | `Upca`, `Upce`, `Jan13`, `Jan8`, `Code39`, `Itf`, `Codabar`, `Code93`, `Code128` | `EscposPrinter::barcode()` |
| `Pdf417Option` | `Standard`, `Truncated` | `EscposPrinter::pdf417()` |
| `BarcodeTextPosition` | `None`, `Above`, `Below` | *(vocabulary only — no method exposes it yet)* |
| `Font` | `A`, `B`, `C` | *(vocabulary only)* |
| `Color` | `Color1`, `Color2` | *(vocabulary only)* |

| Enum | Cases | Used by |
|---|---|---|
| `CutMode` | `Full`, `Partial` | *(vocabulary only)* |
| `Underline` | `None`, `Single`, `Double` | *(vocabulary only)* |
| `QrErrorCorrectionLevel` | `L`, `M`, `Q`, `H` | *(vocabulary only)* |
| `QrModel` | `Model1`, `Model2`, `Micro` | *(vocabulary only)* |

The “vocabulary only” enums exist because they are a small, closed set defined by the protocol — completing them costs nothing and keeps this package generic instead of shaped only around what its first consumer needed. They are ready for the day a `setFont()`, `setColor()`, `setUnderline()`, `setBarcodeTextPosition()` or `qrCode()` method is added to `EscposPrinter`, without needing new vocabulary at that point.

```
use Derafu\Escpos\Enum\Justification;

$printer->setJustification(Justification::Center);
```

## Constants (bitmask flags)

| Class | Constants | Used by |
|---|---|---|
| `PrintMode` | `FONT_A`, `FONT_B`, `EMPHASIZED`, `DOUBLE_HEIGHT`, `DOUBLE_WIDTH`, `UNDERLINE` | `EscposPrinter::selectPrintMode()` |
| `ImageSize` | `STANDARD`, `DOUBLE_WIDTH`, `DOUBLE_HEIGHT` | `EscposPrinter::image()` |

```
use Derafu\Escpos\Constant\PrintMode;

// Combine flags with a plain bitwise OR.
$printer->selectPrintMode(PrintMode::FONT_B | PrintMode::EMPHASIZED);
```

## What is deliberately not vocabulary

A few things found in `mike42/escpos-php` are **not** mapped, on purpose:

- **Status request codes and raw control bytes** (`STATUS_*`, `ESC`, `GS`, `LF`, …): these are not parameters of any command a caller chooses — they are either internal bytes the underlying library uses to build each command, or (for `STATUS_*`) not used by any method in the vendored version at all. Mapping them would be vocabulary for a feature that does not exist.
- **Character tables / code pages**: unlike everything above, the correct table number is not a fixed protocol value — it depends on the specific printer’s capability profile. This is a known open item, not yet designed.

Last updated on 09/09/2026

## Printing Images

Printing Images

Anonymous · 09/09/2026

# Printing Images

`EscposPrinter::image()` never asks for a type from the underlying ESC/POS library — it takes `Derafu\Escpos\ValueObject\RasterImage`, a small wrapper around a GD image (`\GdImage`). You decide how the pixels were produced; this package only knows how to print them.

## From an existing GD resource

```
use Derafu\Escpos\ValueObject\RasterImage;

$gd = imagecreatefrompng('/path/to/logo.png');

$printer->image(new RasterImage($gd));
```

## From raw image bytes

If you already have PNG bytes (downloaded, generated, read from a database, etc.) instead of a GD resource, decode them with the named constructor:

```
use Derafu\Escpos\ValueObject\RasterImage;

$logo = RasterImage::fromPngData(file_get_contents('/path/to/logo.png'));

$printer->image($logo);
```

`fromPngData()` throws `Derafu\Escpos\Exception\InvalidImageDataException` if the given bytes cannot be decoded as an image — it never returns a broken `RasterImage`.

## PDF417 as a bitmap fallback

`EscposPrinter::pdf417()` sends the printer's native PDF417 command. Some cheaper thermal printers do not support it. For those, `Derafu\Escpos\Renderer\Pdf417Renderer` renders a PDF417 code as a raster image (using `tecnickcom/tc-lib-barcode`, a required dependency) that you then print like any other image:

```
use Derafu\Escpos\Renderer\Pdf417Renderer;

$renderer = new Pdf417Renderer();
$image = $renderer->render('some-content-to-encode');

$printer->image($image);
```

This is a rendering concern, not a printing one, so it lives outside `EscposPrinter` on purpose: `EscposPrinter` has no idea `Pdf417Renderer` exists, and never will need to. You only reach for it when the native command does not work on your printer.

Last updated on 09/09/2026

# Network Delivery

Network Delivery

Anonymous · 09/09/2026

# Network Delivery

`EscposPrinter::end()`/`dump()` return a plain string of ESC/POS bytes. What to do with that string — save it to a file, return it from an API, or send it to a real printer — is up to you, using whatever your application already uses for that (file writes, HTTP responses, etc.). This package does not wrap trivial I/O like `file_put_contents()`.

Delivering bytes to a network thermal printer over TCP is a different matter: it needs connection timeout handling and clear error reporting, which is easy to get subtly wrong by hand (and it is the standard delivery mechanism for ESC/POS printers, not a generic file/API concern). `Derafu\Escpos\Transport\NetworkTransport` covers exactly that, with plain PHP streams — it has no dependency on `mike42/escpos-php` or any other ESC/POS library.

```
use Derafu\Escpos\Transport\NetworkTransport;

$escposData = $printer->end();

(new NetworkTransport())->send($escposData, '172.16.1.5');
// Port defaults to NetworkTransport::DEFAULT_PORT (9100, the de facto
// standard raw ESC/POS printing port). Pass a third argument to override
// it, and a fourth to change the connection timeout (default 5 seconds).
```

If the connection fails, or not all bytes could be delivered, it throws `Derafu\Escpos\Exception\NetworkDeliveryException` with the real connection error — never a silently swallowed failure.

```
use Derafu\Escpos\Exception\NetworkDeliveryException;
use Derafu\Escpos\Transport\NetworkTransport;

try {
    (new NetworkTransport())->send($escposData, '172.16.1.5');
} catch (NetworkDeliveryException $e) {
    // $e->getMessage() includes the host, port, and the underlying error.
}
```

### `bin/print.php`

The repository also has `bin/print.php`, a thin CLI wrapper around `NetworkTransport` used to quickly validate ESC/POS output against a real printer during development:

```
bin/print.php 172.16.1.5:9100 ticket.escpos
cat ticket.escpos | bin/print.php 172.16.1.5
php examples/full-example.php | bin/print.php 172.16.1.5
```

This is an internal development tool, not part of the package's public API: it is not registered as a Composer binary, so it will not appear in `vendor/bin/` when you require this package. It still physically exists inside the installed package, at `vendor/derafu/escpos/bin/print.php`, so it can be run from there if you know the path — but it is not documented or supported as something to build on. Use `NetworkTransport` directly in your own code instead.

---

Last updated on 09/09/2026

# Architecture

Architecture

Anonymous · 09/09/2026

# Architecture

`derafu/escpos` wraps mike42/escpos-php instead of implementing the ESC/POS protocol itself — but that library is treated as a replaceable implementation detail, not part of this package's contract. Nothing in the public API accepts or returns a `Mike42\Escpos\*` type or constant; see Vocabulary and Printing Images for the two places that leak would normally happen (command values, and image types) and how they are closed.

## Why this matters

If a consumer's code imports `Mike42\Escpos\*` directly to call this package, upgrading `mike42/escpos-php` to a new major version can break that consumer's code too — even though they never touched the underlying library's version themselves. Isolating it behind this package's own API means a breaking change in the underlying library only needs to be absorbed in one place: here.

## The seam: `PrinterInterface`

`Derafu\Escpos\Contract\PrinterInterface` defines every operation `EscposPrinter` needs from a backend: `text()`, `feed()`, `selectPrintMode()`, `setJustification()`, `barcode()`, `pdf417()`, `image()`, `cut()`, `pulse()`, `getOutput()`, `close()`.

`Derafu\Escpos\Adapter\Mike42Printer` is the only class in this package allowed to `use Mike42\Escpos\*`, and it is the default implementation of `PrinterInterface`. `EscposPrinter` is a thin facade: it owns option resolution, the `sprintf()`/line-break convenience of `print()`/`println()`, special character handling, and the `end()` workflow (finish, optionally cut and pulse) — and delegates everything else to whichever `PrinterInterface` it was given.

```
use Derafu\Escpos\Contract\PrinterInterface;
use Derafu\Escpos\EscposPrinter;

// Normally you never need this — omitting the second argument builds the
// default Mike42Printer from your options array.
$printer = new EscposPrinter($options, $customPrinterImplementation);
```

That second constructor argument is the extension point: a different backend (or a test double) can

be swapped in without changing `EscposPrinter`'s public API, or any code that already depends on `PrinterInterface` instead of the concrete class.

## Exceptions

Every exception this package throws implements `Derafu\Escpos\Exception\EscposExceptionInterface`, so you can catch anything it raises without needing to know the specific class:

| Exception | Thrown when |
|---|---|
| `ConnectorNotSupportedException` | The configured connector type has no known implementation (only `memory` exists today). |
| `ProfileNotFoundException` | The configured capability profile does not exist. |
| `UnbufferedConnectorException` | `dump()`/`end()` is called on a connector that does not buffer data in memory (`memory` always does; this exists so the failure mode is a clear exception if that ever changes, not a corrupted read). |
| `InvalidImageDataException` | `RasterImage::fromPngData()` is given data that cannot be decoded as an image. |
| `NetworkDeliveryException` | `NetworkTransport::send()` cannot connect, or cannot deliver all bytes. |

None of these extend a `Mike42\Escpos\*` exception, for the same reason described above.

Last updated on 09/09/2026