Catppuccin for Typst

🪶 Soothing pastel theme for Typst

v1.0.0 November 25, 2024

https://github.com/catppuccin/typst

TimeTravelPenguin

Abstract

The **catppuccin** package provides colourful Catppuccin aesthetics for Typst documents. It provides four soothing pastel themes that is easy on the eyes. This manual provides a detailed documentation of the package.

Contents

1. Overview

1.1. About

This document provides a detailed documentation of the **catppuccin** package for Typst. Inspired by the LaTeX Catppuccin package, this package hopes to make writing in Typst more pleasurable and easy to use.

As someone who has done a lot of LaTeX, I found myself spending a lot of time writing in dark themes (usually by inverting the document colors). Eventually I found the Catppuccin package for LaTeX, and I incorporated it into my custom preable to allow me to enable, disable, or configure the enabled theme. When I finished, I would submit my work with the theme disabled, without explicitly removing code!

I have plans for the future of this package, such as added styling and perhaps integration with other packages (if that ever becomes easier to do without making a new package).

1.2. Basic Usage

Using this package is simple. See Listing 1 for an example of how to use the package.

```
#import "catppuccin.typ": catppuccin, flavors

#show: catppuccin.with(flavor: flavors.mocha)

// The rest of your document
```

Listing 1: Example usage of the Catppuccin package

You can disable the theme by commenting out or deleting the show block.

2. Modules

2.1. Catppuccin

- catppuccin()

2.1.1. catppuccin

Configure your document to use a Catppuccin flavor.

Example:

```
#import "@preview/catppuccin": catppuccin, flavors

#show: catppuccin.with(flavors.mocha)
```

This should be used at the top of your document.

- flavor (string | flavor): The flavor to set.

Parameters

```
catppuccin(
  flavor,
  body: content
) -> content
```

body `content`

The content to apply the flavor to.

3. Flavors

The Catppuccin package comes with four flavors: **Latte**, **Frappe**, **Macchiato**, and **Mocha**. Each flavor has its own unique color palette that is easy on the eyes. You can choose a flavor by setting the `flavor` parameter in the `catppuccin.with` function.

In this package, we refer to the dictionary related to each flavor with the type alias `flavor`.

3.1. Flavor Schema

Here we describe the schema for the `flavor` dictionary. Use *`get-flavor()`* function to

- **name** `string` — The name of the flavor (e.g. Frappé)
- **emoji** `string` — The emoji associated with the flavor.
- **order** `integer` — The order of the flavor in the Catppuccin lineup.
- **dark** `boolean` — Whether the flavor is a dark theme.
- **light** `boolean` — Whether the flavor is a light theme.
- **colors** `dictionary` — A dictionary of colors used in the flavor. Keys are the color names as a `string` and values are dictionaries with the following keys:
  - **name** `string` — The name of the color.
  - **order** `integer` — The order of the color in the palette.
  - **hex** `string` — The hex value of the color.
  - **rgb** `string` — The RGB value of the color.
  - **accent** `boolean` — Whether the color is an accent color.

- get-flavor()
- parse-flavor()

Variables:

- flavors

3.1.1. get-flavor

Get the palette for the given flavor.

Example

```
#let items = flavors.values().map(flavor => [
  #let rainbow = (
    "red", "yellow", "green",
    "blue", "mauve",
  ).map(c => flavor.colors.at(c).rgb)

  #let fills = (
    gradient.linear(..rainbow),
    gradient.radial(..rainbow),
    gradient.conic(..rainbow),
  )

  #stack(
    dir: ttb,
    spacing: 4pt,
    text(flavor.name + ":"),
    stack(
      dir: ltr,
      spacing: 3mm,
      ..fills.map(fill => square(fill: fill))
    )
  )
])

#grid(columns: 1, gutter: 1em, ..items)
```

Parameters

`get-flavor(flavor: string) -> dictionary`

flavor `string`

The flavor name as a string to get the flavor for. This function is provided as a helper for anyone requiring dynamic resolution of a flavor.

3.1.2. parse-flavor

Parse a flavor. If the flavor is a string, get the flavor from the dictionary. Otherwise, assert that the flavor is a valid flavor.

Parameters

`parse-flavor(flavor: string flavor) -> dictionary`

flavor `string` or `flavor`

The flavor to parse.

3.1.3. flavors `dictionary`

The available flavors for Catppuccin. Given simply by the dictionary

```
#let flavors = (
  latte: { ... },
  frappe: { ... },
  macchiato: { ... },
  mocha: { ... },
)
```

4. Styling

Please note that this module is still in development and may be subject to change.

Until Typst supports relative paths in libraries, there may not be much change here. The current implementation and style is not perfect, but if you don't want to style things manually, this is the best you can get. If you want to style things manually, you can use the library codly to style code blocks. In the future, we may eventually use this approach.

4.1. Code Blocks

- config-code-blocks()

4.1.1. config-code-blocks

Configures the appearance of code blocks and code boxes.

Note that code-syntax requires manually installing the theme corresponding to the flavor into the directory `/tmThemes`, relative to the project root. This is due to a limitation in the current implementation of Typst and how it handles relative paths, preventing libraries from accessing files outside of the project root directory.

Example

```
#import "@preview/catppuccin": flavors, config-code-blocks
#show: config-code-blocks.with(flavors.mocha)
```

Parameters

```
config-code-blocks(
  flavor: string dictionary,
  code-block: boolean,
  code-syntax: boolean,
  block-config: dictionary,
  inline-config: dictionary,
  body: content
) -> content
```

flavor `string` or `dictionary`

The flavor to set.

code-block `boolean`

Whether to apply the configuration to code blocks.

Default: `true`

code-syntax `boolean`

Whether to apply the configuration to code syntax highlighting.

Default: `true`

block-config `dictionary`

Additional configuration for code blocks.

Default: `(:)`

inline-config `dictionary`

Additional configuration for code boxes.

Default: `(:)`

body `content`

The content to apply the configuration to.

4.2. Tidy Styles

- get-tidy-colors()

4.2.1. get-tidy-colors

Parameters

`get-tidy-colors(flavor: string) -> dictionary`

flavor `string`

The name of the flavor to use.

Default: `flavors.mocha`

5. Miscellaneous

5.1. Version

Variables:

- version

5.1.1. version `version`

The package version of Catppuccin.

Example:

```
This package's version is #version.
```

This package's version is 1.0.0.