Catppuccin for Typst

🪶 Soothing pastel theme for Typst

v1.0.0 May 04, 2025

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. The following is an example of how to use the package.

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

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

// The rest of your document
```

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, code-block: true, code-syntax: true)
```

This should be used at the top of your document.

Parameters

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

flavor `string` or `flavor`

The flavor to set

code-block `boolean`

Whether to stylise code blocks

Default: `false`

code-syntax `boolean`

Whether to the Catppuccin flavor 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 flavor to

3. Flavor Schema

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`.

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

- **name** `string` — The name of the flavor (e.g. Frappé)
- **identifier** `string` — The identifier of the flavor (e.g. frappe)
- **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.

3.1. Flavors

- get-flavor()
- get-or-validate-flavor()
- validate-flavor()

Variables:

- color-names
- flavors

3.1.1. get-flavor

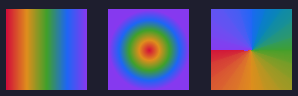
Get the palette for the given flavor.

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),
```

Latte:



Frappé:

Macchiato:

```
    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. get-or-validate-flavor

Get the flavor for the given flavor name or validate the given flavor. This function is provided as a helper for anyone requiring dynamic resolution of a flavor.

Parameters

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

flavor `string` or `dictionary`

The flavor name as a string to get the flavor for

3.1.3. validate-flavor

Validate that the given dictionary is a valid flavor.

Parameters

`validate-flavor(flavor: dictionary flavor) -> flavor`

flavor `dictionary` or `flavor`

The flavor to validate

3.1.4. color-names `dictionary`

The available color names for Catppuccin. Given simply by the dictionary

```
#let color-names = (
   rosewater: "Rosewater",
   flamingo: "Flamingo",
   pink: "Pink",
   // ...
 )
```

3.1.5. flavors `dictionary`

The available flavors for Catppuccin. Given simply by the dictionary

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

Variables:

- frappe
- latte
- macchiato
- mocha

3.1.6. frappe `flavor`

The Frappé flavor and palette.

Example

```
#let flavor = flavors.frappe
Selected flavor: #flavor.name #flavor.emoji
```

```
Selected flavor: Frappé 🪴
```

3.1.7. latte `flavor`

The Latte flavor and palette.

Example

```
#let flavor = flavors.latte
Selected flavor: #flavor.name #flavor.emoji
```

```
Selected flavor: Latte 🌻
```

3.1.8. macchiato `flavor`

The Macchiato flavor and palette.

Example

```
#let flavor = flavors.macchiato
Selected flavor: #flavor.name #flavor.emoji
```

```
Selected flavor: Macchiato 🌺
```

3.1.9. mocha `flavor`

The Mocha flavor and palette.

Example

```
#let flavor = flavors.mocha
Selected flavor: #flavor.name #flavor.emoji
```

```
Selected flavor: 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.

Parameters

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

flavor `string` or `flavor`

The flavor to set

code-block `boolean`

Whether to stylise code blocks

Default: `true`

code-syntax `boolean`

Whether to the Catppuccin flavor 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

4.2.1. ctp-tidy-style

Create a style dictionary to be used with Tidy.

The returned dictionary contains the following keys:

- `colors` : The color palette for the style.
- `show-outline` : A function to show the outline of a module.
- `show-type` : A function to show the type of a variable.
- `show-parameter-list` : A function to show the parameter list of a function.
- `show-parameter-block` : A function to show a parameter block.
- `show-function` : A function to show a function.
- `show-variable` : A function to show a variable.
- `show-reference` : A function to show a reference.
- `show-example` : A function to show an example.

Parameters

`ctp-tidy-style(flavor: flavor) -> dictionary`

flavor `flavor`

The Catppuccin flavor to use for the style

Default: `flavors.mocha`

4.2.2. default-layout-example

This function is temporarily used as the default layouter until the resolution of this issue.

Parameters

```
default-layout-example(
  code: raw,
  preview: content,
  dir: direction,
  ratio: int,
  scale-preview: auto ratio,
  code-block: function,
  preview-block: function,
  col-spacing: length
)
```

code `raw`

Code `raw` element to display.

preview `content`

Rendered preview.

dir `direction`

Direction for laying out the code and preview boxes.

Default: `ltr`

ratio `int`

Configures the ratio of the widths of the code and preview boxes.

Default: `1`

scale-preview `auto` or `ratio`

How much to rescale the preview. If set to auto, the the preview is scaled to fit the box.

Default: `auto`

code-block `function`

The code is passed to this function. Use this to customize how the code is shown.

Default: `block`

preview-block `function`

The preview is passed to this function. Use this to customize how the preview is shown.

Default: `block`

col-spacing `length`

Spacing between the code and preview boxes.

Default: `5pt`

4.2.3. get-tidy-colors

A style that can be used to generate documentation using Tidy for the Catppuccino theme. The returned dictionary is a tidy styles dictionary with some additional keys, such as `ctp-palette` whose value is the associated with the `colors` field of `flavor`.

Parameters

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

flavor `string` or `flavor`

The flavor to use

Default: `flavors.mocha`

4.2.4. show-module

A wrapper function around `tidy.show-module` .

Parameters

```
show-module(
  docs: dictionary,
  flavor: flavor,
  style-alt: dictionary,
  ..args
) -> content
```

docs `dictionary`

Module documentation information as returned by `tidy.parse-module` .

flavor `flavor`

The Catppuccin flavor to use for the style

Default: `flavors.mocha`

style-alt `dictionary`

Alternative style settings to use. See `ctp-tidy-style()`

Default: `(:)`

..args

Additional arguments to pass to `tidy.show-module`

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.