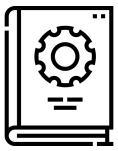

Minimal Manuals

Manual

Modern yet sober manuals using Typst, Markdown, and documentation comments

---

typst | min-manual[*] | 0.3.0 | MIT-0

Maycon F. Melo[†]

Contents

[*] https://github.com/mayconfmelo/min-manual
[†] https://github.com/mayconfmelo

Quick Start

```
#import "@preview/min-manual:0.3.0": manual
#show: manual.with(
  title: "Package Name",
  manifest: toml("typst.toml"),
)
```

Description

Create modern and elegant manuals with a clean visual style and a focus on maintaining attention on the document's content. This package seeks a balance between new visual trends and the traditional simplicity of older manuals: there are no abstract designs, colorful sections, diverse themes, or anything that steals the focus; however, it adopts modern fonts, pleasant spacing, text layout inspired by web pages, as well as automation tools and practical features.

This was created with Typst in mind, but also aiming for the potential to universally document code from other languages: all the features of *min-book* are supported when documenting any type of program or code.

Options

```
#show: manual.with(
  title: none,
  description: none,
  by: none,
  package: none,
  authors: none,
  license: none,
  url: none,
  logo: none,
  manifest: none,
  typst-defaults: false,
  from-comments: none,
  from-markdown: none,
```

```
    comment-delim: auto,
  )
```

`title:` string content *(required)*

Descriptive name of the package (what is being documented).

`description:` string content

Short package description, generally two lines long or less.

`by:` string content

Manual author (fallback to `authors.at(0)` if not set).

`package:` string *(required)*

`"@namespace/name:version"` `"pkg:type/namespace/name@version"`

Package identification (see Package Identification section).

`authors:` string array of strings *(required)*

`"name <url>"`

Package authors, each followed by an optional `<url>`.

`license:` string content *(required)*

Package license.

`url:` string content

Package URL.

`logo:` image content

Manual logo.

`manifest:` toml dictionary

Retrieve essential data from *typst.toml* package manifest.

`typst-defaults:` boolean

Use Typst defaults instead of min-manual defaults.

`from-comments:` string read

Retrieve documentation from comments in source code files.

`from-markdown:` string read

Retrieve documentation from markdown files (experimental).

`comment-delim:` array of strings

`("///", "/**", "**/")`

Set documentation comment delimiters.

Package Identification

```
@namespace/name:version
```

```
pkg:type/namespace/name@version
```

The package identification can be done through an Typst import or a*package URL*[1] both are ways to reliably identify and locate packages, but the latter works in a mostly universal and uniform way across dofferent programming languages, package managers, packaging conventions, tools, APIs and databases.

The Typst import consists of the following components:

`@namespace/`
Typst namespace; can be omitted if it's `@preview/`.

`name:` *(required)*
Typst package name.

`version`
Typst package version.

The package URL used is a custom implementation that consists of the following components:

`pkg:` *(required)*
Package URL scheme

`type/`
Package protocol, platform, manager, etc.

`namespace/`
Additional prefix to the name, generally used for desambiguity.

`name@` *(required)*
Package name.

`version`
Package version.

[1] https://github.com/package-url/purl-spec/blob/main/README.rst#purl

Syntax

Terminal Simulation

```
```terminal
prompt$ command
output
```
```

Generates a generic terminal simulation, with syntax highlight for prompt, command, and output. It is implemented as `#raw(lang: "terminal")`, and also available under the shorter `"term"` name. The syntax detects key characters (yellow) and sets everything at its left as prompt (green) and at its right as command (red); lines without key characters are considered output (white).

```
usr@host ~ % command (zsh)
usr@host:~$ command (bash)
usr@host:~# command (root)
C:\> command (windows)
output
```

Commands

Arguments

```
#arg(
  title,
  body
)
```

Defines and explains possible arguments/parameters (see `/tests/commands/arg/`).

`title` string *(required)*

`"name <- type | type -> type <required>"`

Title data: A mandatory name identifier, followed by optional ASCII arrows indicating input/output types, and a final `<required>` to define required arguments.

Extract

```
#extract(
  name,
  from: auto,
  display: auto,
  lang: "typ",
  model: auto,
)
```

Extract code from another file or location (see `/tests/commands/extract/`).

`name` string

Name of the code structure to retrieve (uses last match).

`from:` string read *(required)*

File from where the code will be retrieved (required in the first use).

`display:` string

`"show.with"` `"show"` `"call"` `"set"` `"str"` `"src"` `"arg"` `"let"`

How to render the code retrieved: as one of the predefined Typst cases above, or an arbitrary template that interpolates <name>, <capt>, and <text> as the name, last capture, and last retrieved text, respectively.

`lang:` string

Programming language of the code.

`model:` string

`"func"` `"arg"` `"let"` `"var"` `"call"`

Regex pattern to retrieve code: one of the predefined names above, or a custom pattern (spaces before code normalizes indentation).

Code Example

```
#example(
  scope: auto,
  output-align: auto,
  ..data
)
```

Generates a code example that showcases its result. If only a Typst code is passed, the result is automatically generated. Can also be used as `#raw(lang: "example")` syntax for Typst codes; a shorter `"eg"` is also available.

`scope:` dictionary yaml toml

Additional scope available when generating Typst results.

`output-align:` top bottom left right false

Position of result block.

`..data` raw string

`(code, result)`

Positional arguments for code and optional result.

Paper-friendly URL

```
#url(target, id, text)
```

Creates a paper-friendly link, attached to a footnote containing the URL itself to ensure readability in paper.

`target` string label *(required)*

URL used as link and footnote, or or label referencing a previous `#url` command.

`id` label

Optional label for further reference.

`text` string content

Text shown as link (fallback to the URL).

Callout

```
#callout(
  icon: "information-circle",
  title: none,
  text: (:),
  background: (:),
  body,
)
```

Create a simple yet highly customizable callout box, used to highlight a text or showcase important content.

`icon:` string

Icon name, as set by*Heroicons*[2] .

`title:` string content none

Set title, if any.

`text:` color dictionary

Text (`#text`) options; the special `text.title` set title options.

`background:` color dictionary

Background style (`#block`) options.

Package URLs

```
#pkg(url)
#univ(name)
#pip(name)
#crate(name)
#npm(name)
#gh(slug)
```

Generates paper-friendly links to packages from different sources/platforms requiring only essential data. Supports generic packages, Typst Universe, Python Pypi, Rust crates, Node.js npm, GitHub repositories.

`url` string

`"https://repo.com/{name}/download"` `"https://repo.com/name"`

Package URL. Extracts the package name between `{}` or as the last `/slug`.

`name` string

Package name as it appears in the package source/platform.

`slug` string

The `user/repo` slug of the GitHub repository URL.

[2]https://heroicons.com/

Alternative Sources

Documentation from Comments

```
/// = Feature
/// :show.with feature:
/// The `#feature` command does something.
#let feature(title) = { }
```

The documentation can be embedded into the source code itself through special comments, sometimes called *doc-comments*. These comments contain Typst code that is retrieved by *min-manual* to generate the manual document — while also serving as documentation within the source code, like standard comments.

To use it with languages that have a different syntax for comments, check the `comment-delim` option.

Special syntax for argument command

```
name <- types -> types <required>
  body
```

```
#arg("name <- types -> types <required>", body)
```

Each argument must be between empty lines, or end with a `|` to separate them.

Special syntax for extract command

```
:display name: lang "model" => display-custom
```

```
#extract(
  name,
  display: display, lang: lang, model: model,
)
```

Documentation from Markdown

````
# Feature
```typst
#show: feature.with(title)
```
The `#feature` command does something.
````

The documentation can be written in Markdown and *min-manual* will manage to generate a manual from it. The conversion between Markdown and Typst code is done using the*cmarker*[3] package; therefore some *cmarker* features are supported, like the *raw-typst* and *typst-exclude* special comments:

```
<!--raw-typst
This code appears only in Typst
-->

<!--typst-begin-exclude-->
This code appears only in Markdown
<!--typst-end-exclude-->
```

There is also a special syntax structure for `#arg` using Markdown/HTML5 (see the `tests/markdown/assets/file.md` file).

> ⚠ **Experimental**
>
> As it is a recent implementation, the Markdown documentation is still experimental and may or may not present errors, bugs, or unexpected behaviors. Used it with caution for now.

Copyright

The logo was obtained from Flaticon website.

[3] https://typst.app/universe/package/cmarker