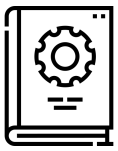

Minimal Manuals

Manual

Modern but sober manuals inspired by the manpages of old.

---

typst	min-manual*	0.2.0	MIT

Maycon F. Melo†

Contents

*typst.app/universe/package/min-manual/
†https://github.com/mayconfmelo

Quick Start

```
#import "@preview/min-manual:0.2.0": manual
#show: manual.with(
  title: "Package Name",
  description: "Short description, no longer than two lines.",
  package: "pkg-name:0.4.2",
  authors: "Author <mailto:author@email.com>",
  license: "MIT",
  logo: image("assets/logo.png")
)
```

Description

Generate modern manuals without losing the simplicity of old manpages. This package draws inspiration from old manuals while adopting the facilities of modern tools, like Markdown and documentation embedded in comments. The design aims to be sober: a minimal informative header, technical text in comfortable fonts and well-formatted code examples.

The package was created with Typst in mind, but also targeting the potential to universally document code from other languages: all *min-book* features 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,
  use-defaults: false,
  from-comments: none,
  from-markdown: none,
  comment-delim: auto,
  body,
)
```

`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*)
`"pkg:type/namespace/name@version"`
Package identification,[1] where `pkg:type/namespace/` is optional (fallback to `pkg:typst/`) and `name@version` can also be written `name:version`.

`authors:` string array of strings (*required*)
`"name <url>"`
Package author or authors, each followed by an optional `<url>`.

`license:` string content (*required*)
Package license.

`url:` string content
Package URL.

`logo:` image content
Manual logo.

`use-defaults:` boolean
Use Typst defaults instead of min-manual defaults.

`from-comments:` string read
Retrieve documentation from comments in source files.

`from-markdown:` string read
Retrieve documentation from markdown files (experimental).

`comment-delim:` array of strings
`("///", "/**", "**/")`
Set documentation comment delimiters.

[1] Inspired by *https://github.com/package-url/purl-spec/*

Command 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.

Command Extract

```
#extract(
  name,
  from: auto,
  rule: none,
  lang: "typ",
  model: auto,
  display: none,
)
```

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

`name` string

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

`from:` string read (*required*)

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

`rule:` "show.with" "show" "call" "set" "let" "arg" "str" none

Render Typst code in different ways.

`lang:` string

Programming language of the code.

`model:` string

Custom regex pattern to retrieve code — spaces captured before the code are used to normalize indentation.

`display:` string

Custom way to render retrieved code. Replaces `<name>` and `<capt>` markers by the name and retrieved code, respectively.

When extracting Typst code, the `#extract(rule)` supply almost all use cases of a Typst code; otherwise, the `#extract(model, display)` options can be used to achieve any other desired result.

Command Code-Result Example

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

Generates a code-result example, consisting in a `#raw` code block and an annex block representing the code result. If only a Typst code block is passed, the result is automatically evaluated. Can also be used as `#raw(lang: "example")` language to evaluate Typst codes; a shorter `"eg"` name can also be used as an alias.

`scope:` dictionary yaml toml

Define the `#eval` scope of automatic Typst results.

`output-align:` top bottom left right false

Set position of result block — `false` disables it.

`..data` raw string

The code and result blocks — the optional latter can be a content block.

Command URL

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

Creates a paper-friendly link, attached to a footnote containing the URL itself for readability when printed.

`url` string label (*required*)

URL set to link and shown in footnote.

`id` label

Label set to the footnote for future reference.

`text` string content

Text to be shown in-place as the link itself.

Commands for Package URLs

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

Generates paper-friendly links to packages from different sources/platforms using only essential data like its name (see `/tests/commands/links/`).

`url` string

Package URL (used by `#pkg`). The package name is extracted if enclosed in `{}` or fallback to the last `/slug` of the URL.

`name` string

Package name as it is in the source repository/platform (used by `#pip`, `#univ`, and `#crate`).

`slug` string

A `user/name` path, as it appears in GitHub repository paths (used by `#gh`).

Terminal Emulation

````
```terminal
~$ command
output
```
````

```terminal
~$ command
output
```

This `#raw(lang: "terminal")` language emulates a terminal window, with prompt highlight; a shorter `"term"` name can also be used as an alias. Prompts are any line inside the cod block that obeys a certain basic syntax (green paths are optional):

```terminal
~/path$ user command
~/path# root command
C:path> windows command
command output
```

Comment Documentation

```
/// = Feature
/// :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 contains Typst code retrieved by *min-mannual* to generate a complete manual, while at the same time they are useful as in-code documentation.

By default, documentation comments are a extension of Typst comments, both one-line and block comments:

Normal	Documentation
`//`	`///`
`/* */`	`/** **/`

Custom comment delimiters can be set by `#manual(comment-delim)` option. In addition to Typst code, documentation comments also support all *min-book* features both as commands and through special syntax:

```
// #extract (from documentation files itself)
:rule name: lang "model" => display

// #arg (optional arrows)
name <- type | type -> type <required>
  body |
```

The *min-manual* itself is documented through comments in source code, check it out to see how it looks like in practice.

Markdown Documentation

````
# Feature
```typst
#feature(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*[2] package, with some little tweaks — therefore some *cmarker* features are available.

Some Markdown-only and Typst-only code can be written using the *cmarker*-inherited features:

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

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

Refer to `tests/markdown/` for the Markdown (HTML5) structure used to get `#arg` commands.

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.

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