The "Mantys" template

MANuals for **TY**p**S**t

v1.0.0 2024-12-31 MIT

Helpers to build manuals for Typst packages and templates.

Jonas Neugebauer
@jneug

Mantys is a Typst template to help package and template authors write beautiful and useful manuals. It provides functionality for consistent formatting of commands, variables and source code examples. The template automatically creates a table of contents and a command index for easy reference and navigation.

For even easier manual creation, Mantys works well with Tidy, the Typst docstring parser.

The main idea and design were inspired by the LaTeX package CNLTX by Clemens Niederberger.

Table of Contents

Part I
About

Mantys is a Typst package to help package and template authors write manuals. The idea is that, as many Typst users are switching over from TeX, they are used to the way packages provide a PDF manual for reference. Though in a modern ecosystem there are other ways to write documentation (like mdBook[1] or AsciiDoc[2]), having a manual in PDF format might still be beneficial since many users of Typst will generate PDFs as their main output.

This manual is a complete reference of all of Mantys features. The source file of this document is a great example of the things Mantys can do. Other than that, refer to the README file in the GitHub repository and the source code for Mantys.

I.1 Acknowledgements

Mantys was inspired by the fantastic LaTeX package CNLTX[3] by Clemens Niederberger[4].

Thanks to @tingerrr[5] and others for contributing to this package and giving feedback.

Thanks to @Mc-Zen[6] for developing Mc-Zen/tidy[7].

I.2 Dependencies

Mantys is build using some of the great packages provided by the Typst community:

- valkyrie[8] (0.2.1)
- tidy[9] (0.4.0)
- typearea[10] (0.2.0)
- hydra[11] (0.5.1)
- marginalia[12] (0.1.1)
- showybox[13] (2.0.3)
- codly[14] (1.1.1)
- octique[15] (0.1.0)

[1] https://rust-lang.github.io/mdBook/
[2] https://asciidoc.org
[3] https://ctan.org/pkg/cnltx
[4] clemens@cnltx.de
[5] https://github.com/tingerrr
[6] https://github.com/Mc-Zen
[7] https://github.com/Mc-Zen/tidy
[8] https://typst.app/universe/package/valkyrie/0.2.1
[9] https://typst.app/universe/package/tidy/0.4.0
[10] https://typst.app/universe/package/typearea/0.2.0
[11] https://typst.app/universe/package/hydra/0.5.1
[12] https://typst.app/universe/package/marginalia/0.1.1
[13] https://typst.app/universe/package/showybox/2.0.3

I.3 Some Terminology

Since Mantys was first developed as a port of CNLTX, some terms used are derived from the original LaTeX package.

Functions are called “commands” and paramteres “arguments”. This has the benefit of avoiding collisions with the native `function` type.

To display formatted commands, arguments and types inline use the abbreviated command versions like `#cmd` or `#arg`.

To fully document a command or argument use the block commands like `#command` and `#argument`.

Some commands add an entry to the Index. Those commands usually have a Minus-variant that skips this step (like `#cmd` and `#cmd-`).

A “custom type” is a type defined by the package usually in the form of a dictionary schema. Read Section IV.2.0.a for more information.

[14] https://typst.app/universe/package/codly/1.1.1
[15] https://typst.app/universe/package/octique/0.1.0

Part II
Quickstart

In your project root run `typst init`:

```
typst init "@preview/mantys" docs
```

Your project folder should look something like this:

```
.
├── docs
│   └── manual.typ
└── typst.toml
```

Open `docs/manual.typ` in your editor, delete the arguments in the `#mantys` call at the top from ⟨`name`⟩ to ⟨`respository`⟩. Then uncomment the line `..toml("../typst.toml"),`.

The top of your manual should look like this:

```
#show: mantys(
  ..toml("../typst.toml"),
)
```

Fill in the rest of the information like ⟨`subtitle`⟩ or ⟨`abstract`⟩ to your liking. Select a Theme you like.

All uppercase occurrences of ⟨`name`⟩ will be highlighted as a package name. For example MANTYS will appear as Mantys.

Start writing your manual.

If you alread use Tidy to document your functions, use `#tidy-module` to parse and display a module directly in Mantys:

```
#tidy-module("utils", read("../src/lib/utils.typ"))
```

Read Section IV.1 for more details about using Tidy with Mantys.

Part III
Usage

Initialize your manual using `typst init`:

```
typst init "@preview/mantys" docs
```

> We suggest to initialize the template inside a `docs` subdirectory to keep your manual separated from your packages source files.

If you prefer to manually setup your manual, create a `.typ` file and import Mantys at the top:

```
#import "@preview/mantys:1.0.0": *
```

III.1 Project structure

You can setup your project in any way you like, but a common project structure for Typst packages looks like this:

```
.
├── LICENSE
├── README.md
├── docs
│   ├── assets
│   │   └── example.typ
│   └── manual.typ
├── src
│   └── lib.typ
├── tests
└── typst.toml
```

Mantys' defaults are configured with this structure in mind and will let you easily setup your manual.

III.2 Initializing the template

After importing Mantys the template is initialized by applying a show rule with the `#mantys` command.

`#mantys` requires some information to setup the template with an initial title page. Most of the information can be read directly from the `typst.toml` of your package:

```
#show: mantys(
  ..toml("../typst.toml"),
  ... // other options
)
```

Change the path to the `typst.toml` file according to your project structure.

Note that since 1.0.0 `#mantys` no longer requires the use of `#with`.

`#mantys(..⟨doc⟩)`

Argument
`..⟨doc⟩`

MANTYS initializes the `document` from the provided arguments. Refer to the scheme in Section III.3 for all possible options and how to use the `document`.

Argument
`⟨theme⟩` `theme`

The `theme` to use for the manual.

All other arguments will be passed to `#titlepage`.

III.3 The MANTYS document

The arguments passed to `#mantys` are used to initialize the `document`, a dictionary holding information required for the manual.

The following keys can be passed to `#mantys`:

```
(
  ⟨title⟩ content
  ⟨subtitle⟩: none content
  ⟨urls⟩: none array of url
  ⟨date⟩: none date
  ⟨abstract⟩: none content
  ⟨package⟩: (:) package
  ⟨template⟩: none template
  ⟨theme-options⟩: (:) dictionary
  ⟨show-index⟩: true boolean
  ⟨show-outline⟩: true boolean
  ⟨show-urls-in-footnotes⟩: true boolean
  ⟨index-references⟩: true boolean
  ⟨examples-scope⟩: none examples-scope
  ⟨assets⟩: () array of (
      ⟨id⟩ string
      ⟨src⟩ string
      ⟨dest⟩ string

    )
  ⟨git⟩: none (
      ⟨branch⟩: "main" string
      ⟨hash⟩ string

    )
)
```

Argument

`⟨title⟩: none` `content`

If no title is provided, the title is taken from the `package`. If no package information is provided, an error is thrown.

Will be populated from the information in `⟨package⟩` if omitted.

Argument

`⟨subtitle⟩: none` `content`

A subtitle for the manual.

Argument

`⟨urls⟩: none` `array` | `string`

An array of URLs associated with this package.

Argument

`⟨date⟩: none` `datetime` | `string`

A date for the manual or package.

Argument
⟨abstract⟩: none `content`

An abstract to appear on the #titlepage.

Argument
⟨package⟩: (:) `package`

The `package` information (usually read from `typst.toml`).

Argument
⟨template⟩: none `template`

The `template` information (usually read from `typst.toml`).

Argument
⟨show-index⟩: true `boolean`

By default, an index of commands, variabes and other keywords is generated at the end of the document. Setting this to `false` will disable the index. You can manually generate an index by using #make-index.

Argument
⟨show-outline⟩: true `boolean`

By default, a table of contents is generated on the title page. Setting this to `false` will disable the outline.

> The title page is generated by the theme and might ignore this setting.

Argument
⟨show-urls-in-footnotes⟩: true `boolean`

By default, the URLs of links generated by #link will be shown in a footnote. `false` disables this behaviour.

Argument
⟨index-references⟩: true `boolean`

By default, referencing a command, argument or type will create an index entry. This can be disabled on a per reference basis.

Setting ⟨index-references⟩ to `false` will reverse this and disable index entries but allows you to enable them per reference.

See Section IV.3 for more information about references.

Argument
⟨examples-scope⟩: (:) `dictionary`

Default scope for code examples. The examples scope is a `dictionary` with two keys: `scope` and `imports`. The `scope` is passed to `#eval` for evaluation. `imports` maps module names to a set of imports that should be prepended to example code as a preamble.

Schema:

```
(
  ⟨scope⟩ dictionary
  ⟨imports⟩ dictionary
)
```

For example, if your package is named `my-pkg` and you want to import everything from your package into every examples scope, you can add the following `⟨examples-scope⟩`:

```
examples-scope: (
  scope: (
    pkg: my-pkg
  ),
  imports: (
    pkg: "*"
  )
)
```

The `⟨scope⟩` and `⟨imports⟩` are passed to Tidy for evaluating docstring examples.

For further details refer to Section IV.4 and `#example`.

Argument

`⟨theme-options⟩: (:)` `dictionary`

Options to be used by themes (see Section V.1).

Argument

`⟨assets⟩: (:)` `array` | `asset`

Mantys can add `#metadata` to the manual to be queried by external tools. See Section III.3.0.c for more information.

The repository at jneug/typst-mantys[16] contains an `assets` script to query Typst assets from a Mantys manual and compile them before compiling the manual.

[16] https://github.com/jneug/typst-mantys

Argument

⟨git⟩ dictionary

Mantys can show information about the current commit in the manuals footer. This is useful if you compile your manual with a CI workflow like GitHub Actions.

The git information is read with the `#git-info` command. To allow Mantys to read local files from your project you need to provide a reader function to `#git-info`.

```
#mantys(
  ..toml("../typst.toml"),

  git: git-info((file) => read(file))
)
```

The function assumes the project structure seen in Section III.1. For other layouts provide the location of the `.git` folder via the ⟨git⟩ argument.

Schema for package information

```
(
  ⟨name⟩ string
  ⟨version⟩ version
  ⟨entrypoint⟩ string
  ⟨authors⟩: () author
  ⟨license⟩ string
  ⟨description⟩ string
  ⟨homepage⟩: none url
  ⟨repository⟩: none url
  ⟨keywords⟩: none array of string
  ⟨categories⟩: none array of one of ("components", "visualization" … )
  ⟨disciplines⟩: none array of one of ("agriculture", "anthropology" … )
  ⟨compiler⟩: none version
  ⟨exclude⟩: none array of string
)
```

The `package` is exactly the same schema used for the `package` key in the `toml.typst` file. See the official doumentation[17] for a full description of all keys.

Providing a ⟨name⟩ for the package is mandatory.

[17] https://github.com/typst/packages?tab=readme-ov-file#package-format

> Usually the ⟨package⟩ is loaded directly from the `typst.toml` file and passed to `#mantys`.

Schema for template information

```
(
  ⟨path⟩ string
  ⟨entrypoint⟩ string
  ⟨thumbnail⟩: none string
)
```

The `template` is exactly the same schema used for the `template` key in the `toml.typst` file. See the official doumentation[18] for a full description of all keys.

> The ⟨template⟩ is optional and may be `none`.

Schema for asset information

```
(
  ⟨id⟩ string
  ⟨src⟩ string
  ⟨dest⟩ string
)
```

MANTYS can add `#metadata` about required assets to the document. External tooling may query the document for these assets at the `<mantys:asset>` label and compile these before compiling the manual itself.

> You can find a simple script in the MANTYS GitHub repository (scripts/assets[19]) to automatically compile Typst assets.

External tools should query the document with the input `mode=assets`. This will stop rendering of the document after setting the required metadata and thus speed up the query.

```
typst query --root . --input mode=assets --field 'value' docs/manual.typ
'<mantys:asset>'
```

Each queried asset has an `id`, a source file `src` and a description `dest`. An external tool should compile `src` to `dest`.

[18] https://github.com/typst/packages?tab=readme-ov-file#templates
[19] https://github.com/jneug/typst-mantys/tree/main/scripts/assets

Usually the order of assets is important since later assets might depend on earlier ones. For example the first two assets for this manual look like this:

```
{
  "id": "theme-cnltx-pages",
  "src": "assets/examples/theme-cnltx-pages.typ",
  "dest": "assets/examples/theme-cnltx-pages/{n}.png"
},
{
  "id": "assets/examples/theme-cnltx.png",
  "src": "assets/examples/theme-cnltx.typ",
  "dest": "assets/examples/theme-cnltx.png"
}
```

The first entry compiles a multipage example for the CNLTX theme into multiple `png` images and the second combines them into one. The result can be seen in Section V.1.2.c.

If your manual requires a lot of assets it might be a good idea to collect them into a separate file like docs/assets.typ[20] and import it in your manual.

Schema for author information

```
(
  ⟨name⟩ string
  ⟨email⟩: none string
  ⟨github⟩: none string
  ⟨urls⟩: none array of url
  ⟨affiliation⟩: none string
)
```

Information about the package authors can be provided in different formats. In the document they will be accessible as dictionaries with a `name` key. The other information is optional.

If the author is provided as a `string`, Mantys will try to find additional information like an email address.

For example:

```
"J. Neugebauer @jneug <github@neugebauer.cc>"
```

[20] https://github.com/jneug/typst-mantys/tree/main/docs/assets.typ

will be parsed into

```
{
  name: "J. Neugebauer",
  email: "github@neugebauer.cc",
  github: "jneug",
}
```

Loading git information

#git-info(⟨reader⟩, ⟨git-root⟩: "../.git")

Loads information about the current commit from the git repository at ⟨git-root⟩.

Argument

⟨reader⟩ `function`

A function that reads a file and returns its content: (`string`)→ `string`

Usually this will look like this:

```
(filename) => read(filename)
```

```
#git-info((filename) => read(filename))
```

```
(
  branch: "v1.0.0",
  hash: "d2764dc3e0cd9b62e8870f8de71b0521154432ef",
)
```

III.3.1 Accessing document data

There are two methods to access information from the MANTYS `document`:

1. Using commands from the `document` module or
2. using `#mantys-init` instead of `#mantys`.

Using the `document` module

The usual way to access the `document` is by calling one of the `document` functions.

#document.create	#document.get-value	#document.update-value
#document.final	#document.save	#document.use
#document.get	#document.update	#document.use-value

#document.create(..⟨args⟩) → `document`

Creates a document by parsing the supplied arguments agains the document schema using VALKYRIE[21].

Argument
..⟨args⟩ any

Arguments accepted by document.

context **#document.final**

Retrieves the final document from the internally saved state.

context **#document.get**

Retrieves the document at the current location from the internally saved state.

context **#document.get-value(⟨key⟩, ⟨default⟩: none)**

Gets a value from the internally saved document.

#document.save(⟨doc⟩) → content

Saves the document in an internal state.

Argument
⟨doc⟩ document

The document created by #document.create.

#document.update(⟨func⟩) → content

Updates the document in the internal state.

Argument
⟨func⟩ function

An update function to be passed to #state: (document)→ document

#document.update-value(⟨key⟩, ⟨func⟩)

Updates the value at ⟨key⟩ with the update function ⟨func⟩: (any , any)→ none ⟨key⟩ may be in dot-notation to update values in nested dictionaries.

> see #utils.dict-update

context **#document.use(⟨func⟩)**

Retrieves the document from the internal state and passes is to ⟨func⟩.

Argument
⟨func⟩ function

A function to receive the document.

context **#document.use-value(⟨key⟩, ⟨func⟩, ⟨default⟩: none)**

Gets a value from the internally saved document.

Argument
⟨key⟩ string

[21] https://typst.app/universe/package/valkyrie

Key to retrieve. May be in dot-notation.

Argument
`⟨func⟩` `function`

Function to receive the value.

Argument
`⟨default⟩: none` `any`

default value to use, if `⟨key⟩` is not found.

Custom initialization

Instead of using `#mantys` in a `#show` rule, you can initialize Mantys using `#mantys-init` directly (`#mantys` essentially is a shortcut for using `#mantys-init`).

`#mantys-init` → `array`

Calling this function will return a tuple with two elements:

[0] The Mantys `document`.
[1] The Mantys template function to be used in a `#show` rule.

Calling `#mantys-init` directly will give you direct access to the `document` in your manual:

```
#let (doc, mantys) = mantys-init(..toml("../typst.toml"))

#show: mantys

This is the manual for #doc.package.name version
#str(doc.package.version).
```

Part IV
Documenting commands

⚠ This section need to be written. Refer to Section VI for the documentation of all available commands.

IV.1 Using Tidy

Mantys was build with Tidy in mind and replaces the default template used by Tidy. If you already use docstrings to document your code, you can easily show your function documentation in your Mantys manual.

`#tidy-module` is the main entrypoint for using Tidy in Mantys. The command will call `#tidy.parse-module` and `#tidy.show-module` for you and setup Mantys as the template.

Since Mantys can't read your packages files, you need to call `#read` and pass the result to the function (same as you would do for `#tidy.parse-module`).

```
#tidy-module(
  ⟨name⟩,
  ⟨data⟩,
  ⟨scope⟩: (:),
  ⟨module⟩: none,
  ⟨filter⟩: func => true,
  ⟨legacy-parser⟩: false,
  ..⟨tidy-args⟩
) → content
```

Parses and displays a library file with Tidy.

```
#tidy-module("utils", read("../src/lib/utils.typ"))
```

Argument
⟨name⟩ `string`

Name of the module.

Argument
⟨data⟩ `string`

Data of the module, usually read with `#read`.

Argument
⟨scope⟩: (:) `dictionary`

Additional scope for evaluating the modules docstrings.

Argument
`⟨module⟩: none` `string`

Optional module name for functions in this module. By default, all functions will be displayed without a module prefix. This will add a module to the functions by passing `⟨module⟩` to `#command`.

Without module: `#some-command`

With module: `#util.another-command`

Note that setting this will also change function labels to include the module.

Argument
`⟨filter⟩: func => true` `function`

A filter function to apply after parsing the module data. For each function in the module the parsed information is passed to `⟨filter⟩`. It should return `true` if the function should be displayed and `false` otherwise.

Argument
`⟨legacy-parser⟩: false` `boolean`

Set to `true` to enable Tidys legacy parser (pre version 0.4.0).

Argument
`..⟨tidy-args⟩` `any`

Additional arguments to be passed to `#tidy.show-module`.

For easier usage it is recommended to define a custom function in the header of your manual like this:

```
#let show-module(name, ..tidy-args) = tidy-module(
  name,
  read("../src/" + name + ".typ"),
  // Some defaults you want to set
  ..tidy-args.named(),
)
```

See Section VI for an example of the result of `#tidy-module`.

When using Tidy, most Mantys concepts also apply to docstrings. For example, cross-referencing commands is done with the `cmd:` prefix. All Mantys commands like `#arg` or `#property` are available in docstring.

IV.2 Documenting custom types and validation schemas

MANTYS provides support for documentation of custom data types and validation schemas as provided by VALKYRIE[22].

In general a custom type is an anchor in the document that defines a structured schema for some kind of data, that is used in your package. A `dictionary` with some mandatory keys for example. See `document` and other schmeas in this manual for examples.

A custom type can appear anyplace in the manual where a data type can appear, like in argument descriptions:

```
#argument("theme", types:("theme","module"))[
  The theme for this manual.
]
```

Argument

⟨theme⟩ `theme` | `module`

The theme for this manual.

Defining custom types

Place a custom type anchor with the `#custom-type` command.

`#custom-type(⟨name⟩, ⟨color⟩: auto)`

Places a custom type anchor in the document. Any occurrences of the data type `⟨name⟩` will link to this location in the manual. THe anchor itself is invisible.

Defining a custom type schema

If your custom type is defined by a dictionary schema, you cann simply pass an example to `#schema` to show a summary of the required keys and types.

`#schema` also accepts a VALKYRIE[23] validation schema.

`#schema(⟨name⟩, ⟨definition⟩, ⟨color⟩: auto)`

> Support for VALKYRIE schemas is still in development. Some aspects (like optional keys) are not yet supported.

See `document` and other custom types in this manual for examples.

[22] https://typst.app/universe/package/valkyrie
[23] https://typst.app/universe/package/valkyrie

IV.3 Referencing commands and types

You can use the builtin `@` short-syntax for referencing commands, arguments and custom-types in your document.

Use the `cmd` prefix to reference custom types in the manual or use `#cmdref`.

```
@cmd:mantys
```

`#mantys`

Add an argument name after a dot to reference arguments of a command or use `#argref`.

```
@cmd:mantys.theme
```

`#mantys.theme`

Use the `type` prefix to reference custom types in the manual or use `#typeref`.

```
@type:custom-type
```

`custom-type`

Referencing a command will create an index entry. To prevent this, add `[-]` as a supplement. If `⟨index-references⟩` was set to `false`, no index entries are created by default but by adding `[+]` to a reference it will be.

```
@cmd:utils:dict-get[-]

@cmd:mantys.index-references[+]
```

`#utils.dict-get`

`#mantys.index-references`

Referencing the builtin commands and types can be done via the `#builtin` and `#dtype` commands. For these cases Mantys also provides shortcuts in the `#typ` dictionary.

- `#typ.raw` → `#raw`
- `#typ.t.dict` → `dictionary`
- `#typ.v.false` → `false`

See Section VI.3 for a full list of available shortcuts.

IV.4 Showing examples

Showing examples is easy by using the example commands. Wrapping any typst code in `#example` or `#side-by-side` will show the raw code and the evaluated result in a `#frame`.

> `#side-by-side` is an alias for `#example` with `⟨side-by-side⟩: true` set.

By default, any `#raw` blocks with the language set to `example` or `side-by-side` will automatically wrapped inside the corresponding command.

~~~
```example
Some *bold* text.
```

```side-by-side
Some *bold* text.
```
~~~

```
Some *bold* text.
```

Some **bold** text.

```
Some *bold* text.
```

Some **bold** text.

IV.4.1 Setting the evaluation scope

Examples are evaluated with the scope set by `#mantys.examples-scope`.

`#example` takes two arguments to modify the scope of examples: `#example.scope` and `#example.imports`.

The ⟨`scope`⟩ is passed to `#eval` as the scope argument while the ⟨`imports`⟩ are prepended to the raw code as `#import` statements.

The ⟨`scope`⟩ passed to `#example` is merged with the `scope` from ⟨`examples-scope`⟩ passed to `#mantys`. By passing ⟨`use-examples-scope`⟩`: false`, the ⟨`examples-scope`⟩ is ignored.

The ⟨`imports`⟩ are parsed into a preamble by `#utils.build-preamble`. The value is a `dictionary` with `(module: import)` pairs that are prepended to the raw code of the example:

```
#utils.add-preamble(
  "#rawi[Some] #cmd[command].",
  (
    mantys: "cmd",
    utils: "rawi",
  ),
)
```

#import mantys: cmd; #import utils: rawi;
#rawi[Some] #cmd[command].

The `#mantys.examples-scope` is passed to Tidy for evaluating examples in docstrings.

Part V

Customizing the template

V.1 Themes

MANTYS provides support for color themes and can be styled within certain boundries. The template comes with a few bundled themes but you can easily create a custom theme.

> ⚠ Theme support is considered **experimental** and might be removed in future versions if it proves to be not stable enough. Compilation times can get somewhat slow and my guess is that themes are a major factor.

V.1.1 Using themes

To set the theme for your manual, simply provide a `⟨theme⟩` argument to `#mantys` and set it to one of the bundled themes (see Section V.1.2), a `dictionary` or a `#module` with the required color, font and style information.

Some themes can be further customized by options that get passed to `#mantys` in the `⟨theme-options⟩` key.

```
#show: mantys(
  ..toml-info(read),

  theme: themes.orly,
  theme-options: (
    pic: image("assets/logo.png", width: 100%, )
  )
)
```

V.1.2 Bundled themes

Typst theme

The default theme for MANTYS. Based on the Typst documentation and website.

```
⟨theme⟩: #themes.default
```

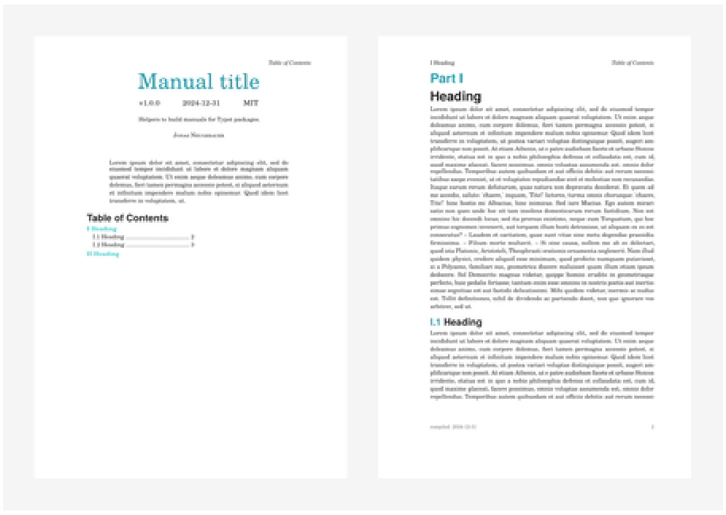


Table of Contents

Manual title

v1.0.0 2024-12-31 MIT

Helpers to build manuals for Typst packages

Jonas Neugebauer

[illegible]

Table of Contents

I Heading
I.1 Heading 2
I.2 Heading 3
II Heading

I Heading Table of Contents

Part I

Heading

[illegible]

I.1 Heading

[illegible]

Modern theme

A slightly more modern theme for the digital age. Based on the Creative Commons Style Guide[24].

```
⟨theme⟩: #themes.modern
```

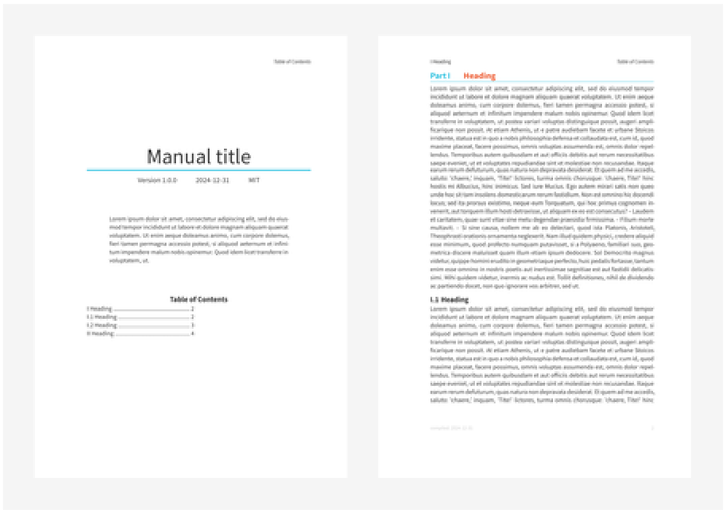


Table of Contents

Manual title

Version 1.0.0 2024-12-31 MIT

[illegible]

Table of Contents

I Heading 2
I.1 Heading 2
I.2 Heading 3
II Heading 4

I Heading Table of Contents

Part I Heading

[illegible]

I.1 Heading

[illegible]

CNLTX theme

This theme is based on the original CNLTX template.

```
⟨theme⟩: #themes.cnltx
```

[24] https://creativecommons.org/2019/10/30/cc-style-guide/

Table of Contents

MANUAL TITLE

v1.0.0 2024-12-31 MIT

Helpers to build manuals for Typst packages

JONAS NEUGEBAUER

[illegible]

Table of Contents

I Heading

I.1 Heading 2

I.2 Heading 3

II Heading

I Heading Table of Contents

Part I

Heading

[illegible]

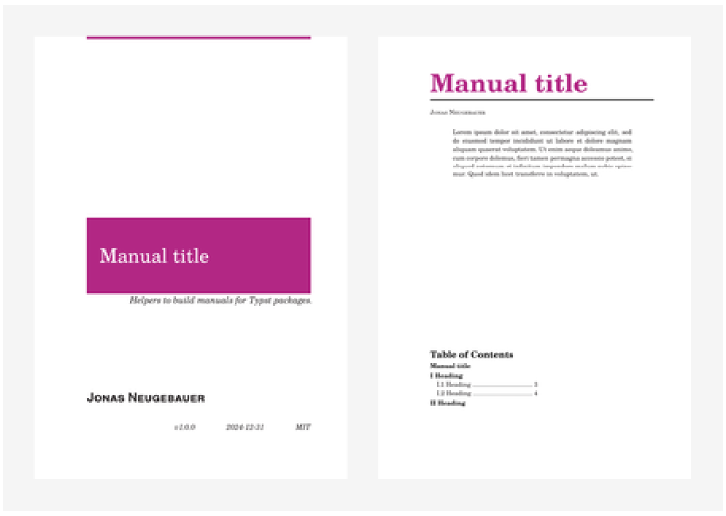
I.1 Heading

[illegible]

O’Rly? theme

This theme uses the FAUXREILLY[25] package to create a style similar to an O’Reilly book.

```
⟨theme⟩: #themes.orly
```


Manual title

Helpers to build manuals for Typst packages.

JONAS NEUGEBAUER

v1.0.0 2024-12-31 MIT

Manual title

Jonas Neugebauer

[illegible]

Table of Contents

Manual title

I Heading

I.1 Heading 3

I.2 Heading 4

II Heading

Theme Options

Argument

⟨pic⟩ `content`

`#content` to be passed to the ⟨pic⟩ argument of `#fauxreilly.orly`.

V.1.3 Creating a custom theme

A theme is a `dictionary` of `#module` with a set of predefined keys for color and font information. See the default theme for a full list of keys and their meaning.

[25] https://typst.app/universe/package/fauxreilly

```
(
  ⟨primary⟩ color
  ⟨secondary⟩ color
  ⟨fonts⟩ (
      ⟨serif⟩ array
      ⟨sans⟩ array
      ⟨mono⟩ array
  )
  ⟨muted⟩ (
      ⟨fill⟩ color
      ⟨bg⟩ color
  )
  ⟨text⟩ (
      ⟨size⟩ length
      ⟨font⟩ array
      ⟨fill⟩ color
  )
  ⟨heading⟩ (
      ⟨font⟩ array
      ⟨fill⟩ color
  )
  ⟨header⟩ (
      ⟨size⟩ length
      ⟨fill⟩ color
  )
  ⟨footer⟩ (
      ⟨size⟩ length
      ⟨fill⟩ color
  )
  ⟨code⟩ (
      ⟨size⟩ length
      ⟨font⟩ array
      ⟨fill⟩ color
  )
  ⟨alert⟩ function
  ⟨tag⟩ function
  ⟨emph⟩ (
      ⟨link⟩ color
      ⟨package⟩ color
      ⟨module⟩ color
      ⟨since⟩ color
      ⟨until⟩ color
      ⟨changed⟩ color
      ⟨deprecated⟩ color
      ⟨compiler⟩ color
      ⟨context⟩ color
  )
  ⟨commands⟩ (
      ⟨argument⟩ color
      ⟨command⟩ color
      ⟨variable⟩ color
      ⟨builtin⟩ color
      ⟨comment⟩ color
      ⟨symbol⟩ color
  )
  ⟨values⟩ (
      ⟨default⟩ color
  )
  ⟨page-init⟩ function
  ⟨title-page⟩ function
  ⟨last-page⟩ function
)
```

`⟨primary⟩` and `⟨secondary⟩` are the main color scheme of the theme. `⟨fonts⟩` is a `dictionary` of the main fontsets used.

`⟨page-init⟩` is a `function` called during template initialization to add custom `#set` rules and other global settings to the document. `⟨title-page⟩` and `⟨last-page⟩` are called once at the beginning and end of the document to add a title and final page to the manual respectively. All three are functions of `(document, theme)→content`.

`⟨alert⟩` is a function `(string, content)→content` that receives an `⟨alert-type⟩`

> When writing a custom theme, remember to add `#pagebreak` at the end of `⟨title-page⟩`, if your title page is supposed to be on its own page. Same goes for `⟨last-page⟩`.

V.1.4 Theme helpers

If you don't want to create a complete `theme` on your own, but want to modify the color scheme of an existing theme, you can quickly do that with one of these helper functions.

`#create-theme(..⟨theme-spec⟩, ⟨base-theme⟩: #themes.default)`

Creates a theme from the passed in arguments. . . `⟨theme-spec⟩` should be key-value pairs from the `theme` specification. Any missing keys are copied from the theme passed in as `⟨base-theme⟩`.

#color-theme(⟨primary⟩, ⟨secondary⟩, ..⟨theme-spec⟩, ⟨base-theme⟩: #themes.default)

Creates a new theme from a ⟨primary⟩ and a ⟨secondary⟩ color. Further arguments are passed to #create-theme along with ⟨base-theme⟩.

```
#show: mantys(
  ..toml-info(read),

  theme: color-theme(blue, red, muted: (fill: yellow), base:
  themes.cnltx),
)
```

V.2 The index

Mantys adds an index of all commands and custom types to the end of the manual. You can modify this index in several ways.

V.2.1 Adding entries to the index

Using #idx you can add new entries to the index. Entries may be categorized by ⟨kind⟩. Commands have ⟨kind⟩: "cmd" set and custom types ⟨kind⟩: "type". You may add arbitrary new types. If your package handles colors, you may want to add a “color” category like this:

```
idx("red", kind: "color")
```

V.2.2 Showing index entries by category

The default index can be disabled by passing ⟨show-index⟩: false to #mantys.

To manually show an index in the manual, use #make-index.

#idx(⟨term⟩, ⟨kind⟩: "term", ⟨main⟩: false, ⟨display⟩: auto) → none | content

Adds ⟨term⟩ to the index.

Each entry can be categorized by setting ⟨kind⟩. #make-index can be used to generate the index for one kind only.

Argument
⟨term⟩ string | content

An optional term to use, if it differs from ⟨body⟩.

Argument
⟨kind⟩: "term" string

A category for this term.

Argument
⟨main⟩: false boolean

If this is the “main” entry for this ⟨term⟩.

Argument
⟨display⟩: auto content

An optional content element to show in the index instead of ⟨term⟩,

```
#idx-term(⟨term⟩) → string
```

Removes special characters from ⟨term⟩ to make it a valid format for the index.

Argument
⟨term⟩ string | content

The term to sanitize.

```
#make-index(
  ⟨kind⟩: auto,
  ⟨heading-format⟩: text => heading(depth: 2, numbering: none, outlined: false,
bookmarked: false, text),
  ⟨entry-format⟩: (term, pages) => [#term #box(width: 1fr, repeat[.])
#pages.join(", ")\ ],
  ⟨sort-key⟩: it => it.term,
  ⟨grouping⟩: it => upper(it.term.at(0))
) → content
```

Creates an index from previously set entries.

Argument
⟨kind⟩: auto string

An optional kind of entries to show.

Argument
⟨heading-format⟩: text => heading(depth: 2, numbering: none, outlined: false, bookmarked: false, text) function

Function to format headings in the index: (string)→ content

Argument
⟨entry-format⟩: (term, pages) => [#term #box(width: 1fr, repeat[.]) #pages.join(", ")\ ] content

Function to format index entries. Receives the index-entry and an array of page numbers.

(content , array)→ content

Argument

⟨sort-key⟩: `it => it.term` `function`

Sorting function to sort index entries.

(`string`)→ `string`

Argument

⟨grouping⟩: `it => upper(it.term.at(0))` `function`

Grouping function to group index entries by. Usually entries are grouped by the first letter of ⟨term⟩, but this can be changed to group by other keys. See below for an example.

(`string`)→ `string`

This example creates an index of hex-colors. Since they all start with `#`, the grouping function is changed to group by the red component of the color.

```
#for c in (red, green, yellow, blue) {
  idx(
    c.to-hex(),
    kind:"color",
    display:box(inset:2pt,baseline:3pt,fill:c, text(white, c.to-hex())))
}

#block(height:10em, columns(2)[
  #make-index(
    kind:"color",
       entry-format:  (term,  pages)  =>  [#term  #box(width:  1fr,  repeat[.])
(#pages.join(", "))\ ],
    grouping: it => it.term.slice(1, count:2)
  )
])
```

00

#0074d9 .... (28)

2e

#2ecc40 .... (28)

ff

#ff4136 .... (28)

#ffdc00 .... (28)

Index entries are defined by a ⟨term⟩ and a ⟨kind⟩ that groups terms.

(
  ⟨term⟩ `string`
  ⟨kind⟩ `string`

```
  ⟨main⟩ boolean
  ⟨display⟩ content
)
```

V.3 Examples

Part VI

Available commands

VI.1 API

VI.1.1 Commands

#arg	#cmd	#sarg
#argref	#cmd-	#typeref
#args	#cmdref	#var
#argument	#command	#var-
#barg	#lambda	#variable
#builtin	#meta	
#carg	#property	

`#meta(⟨name⟩, ⟨l⟩: sym.angle.l, ⟨r⟩: sym.angle.r) → content`

Highlight an argument name.

`#meta[variable]` → `⟨variable⟩`

Argument

`⟨name⟩` — `string` | `content`

Name of the argument.

Argument

`⟨l⟩: sym.angle.l` — `string` | `content` | `symbol`

Prefix to `⟨name⟩`.

Argument

`⟨r⟩: sym.angle.r` — `string` | `content` | `symbol`

Prefix to `⟨name⟩`.

`#arg(..⟨args⟩) → content`

Shows an argument, either positional or named. The argument name is highlighted with `#meta` and the value with `#value`.

- `#arg[name]` → `⟨name⟩`
- `#arg("name")` → `⟨name⟩`
- `#arg(name: "value")` → `⟨name⟩: "value"`
- `#arg("name", 5.2)` → `⟨name⟩: 5.2`

Argument

`..⟨args⟩` — `any`

Either an argument name (`string`) or a (`name`: `value`) pair either as a named argument or as exactly two positional arguments.

`#barg(⟨name⟩) → content`

Shows a body argument.

> Body arguments are positional arguments that can be given as a separat content block at the end of a command.

- `#barg[body] → [body]`

Argument
`⟨name⟩` `string`

Name of the argument.

`#carg(⟨name⟩) → content`

Shows a "code" argument. `alert`)["Code" are blocks og Typst code wrapped in braces: `{ ... }`. They are not an actual argument, but evaluate to some other type.]

- `#carg[code] → {code}`

Argument
`⟨name⟩` `string`

Name of the argument.

`#sarg(⟨name⟩) → content`

Shows an argument sink / variadic argument.

- `#sarg[args] → ..⟨args⟩`

Argument
`⟨name⟩` `string`

Name of the argument.

`#args(..⟨args⟩) → array`

Creates a list of arguments from a set of positional and/or named arguments.

`string`s and named arguments are passed to `#arg`, while `content` arguments are passed to `#barg`. The result should be unpacked as arguments to `#cmd`.

```
#cmd( "conditional-show", ..args(hide: false, [body]) )
```

```
#conditional-show(⟨hide⟩: false)[body]
```

Argument
`..⟨args⟩` `any`

Either an argument name (`string`) or a (`name`: `value`) pair either as a named argument or as exactly two positional arguments.

`#lambda(..⟨args⟩, ⟨ret⟩: none) → content`

Create a lambda function argument.

Lambda arguments may be used as an argument value with `#arg`.

> To show a lambda function with an argument sink, prefix the type with two dots.

- `#lambda(int, str)` → (`integer` , `string`)→ `none`
- `#lambda("ratio", "length")` → (`ratio` , `length`)→ `none`
- `#lambda("int", int, ret:bool)` → (`integer` , `integer`)→ `boolean`
- `#lambda("int", int, ret:(int,str))` → (`integer` , `integer`)→(`integer` , `string`)
- `#lambda("int", int, ret:(name: str))` → (`integer` , `integer`)→(name: `string`)
- `#lambda("int", int, ret:(str,))` → (`integer` , `integer`)→(`string` ,)

Argument
`..⟨args⟩` `string` | `type`

Argument types of the function parameters.

Argument
`⟨ret⟩: none` `string` | `type`

Type of the returned value.

```
#cmd(
  ⟨name⟩,
  ⟨module⟩: none,
  ⟨ret⟩: none,
  ⟨index⟩: true,
  ⟨unpack⟩: false,
  ..⟨args⟩
) → content
```

⇄ Changed in 1.0.0

Renders the command `⟨name⟩` with arguments and adds an entry with `⟨kind⟩: "cmd"` to the index.

`..⟨args⟩` is a collection of positional arguments created with `#arg`, `#barg` and `#sarg` (or `#args`).

All positional arguments will be rendered first, then named arguments and all body arguments will be added after the closing paranthesis. The relative order of each argument type is kept.

```
- #cmd("cmd", arg[name], sarg[args], barg[body])
- #cmd("cmd", ..args("name", [body]), sarg[args], module:"mod")
-  #cmd("clamp",  arg[value],  arg[min],  arg[max],  module:"math",  ret:int,
unpack:true)
```

- `#cmd(⟨name⟩, ..⟨args⟩)[body]`
- `#mod.cmd(⟨name⟩, ..⟨args⟩)[body]`
-
```
#math.clamp(
  ⟨value⟩,
  ⟨min⟩,
  ⟨max⟩,
) → integer
```

Argument

`⟨name⟩` `string`

Name of the command.

Argument

`⟨module⟩: none` `string`

Name of the commands module. Will be used as a prefix and appear in the index.

Argument

`⟨ret⟩: none` `string` | `type`

Return type.

Argument

`⟨index⟩: true` `boolean`

If `false`, this location is not added to the index.

Argument

`⟨unpack⟩: false` `boolean`

If `true`, the arguments are shown in separate lines.

Argument

`..⟨args⟩` `content`

Arguments for the command, created with individual argument commands (`#arg`, `#barg`, `#sarg`) or `#args`.

```
#cmd-(
  ⟨name⟩,
  ⟨module⟩: none,
  ⟨ret⟩: none,
  ⟨index⟩: false,
  ⟨unpack⟩: false,
  ..⟨args⟩
) → content
```

Same as `#cmd`, but does not create an index entry (`⟨index⟩: false`).

Argument
`⟨name⟩` `string`

Name of the command.

Argument
`⟨module⟩: none` `string`

Name of the commands module. Will be used as a prefix and appear in the index.

Argument
`⟨ret⟩: none` `string` | `type`

Return type.

Argument
`⟨index⟩: false` `boolean`

If `false`, this location is not added to the index.

Argument
`⟨unpack⟩: false` `boolean`

If `true`, the arguments are shown in separate lines.

Argument
`..⟨args⟩` `content`

Arguments for the command, created with individual argument commands (`#arg`, `#barg`, `#sarg`) or `#args`.

```
#var(⟨name⟩, ⟨module⟩: none, ⟨index⟩: true) → content
```

Shows the variable `⟨name⟩` and adds an entry to the index.

- `#var[colors]` → `#colors`

Argument
`⟨name⟩` `string`

Name of the variable.

Argument
`⟨module⟩: none` `string`

Name of the commands module. Will be used as a prefix and appear in the index.

Argument
`⟨index⟩: true` `boolean`

If `false`, this location is not added to the index.

`#var-(⟨name⟩, ⟨module⟩: none, ⟨index⟩: false) → content`

Same as var, but does not create an index entry.

Argument
`⟨name⟩` `string`

Name of the variable.

Argument
`⟨module⟩: none` `string`

Name of the commands module. Will be used as a prefix and appear in the index.

Argument
`⟨index⟩: false` `boolean`

If `false`, this location is not added to the index.

`#builtin(⟨name⟩, ⟨module⟩: none) → content`

Displays a built-in Typst function with a link to the documentation.

- `#builtin[context]` → `#context`
- `#builtin(module:"math")[clamp]` → `#math.clamp`

Argument
`⟨name⟩` `str, content`

Name of the function (eg. `raw`).

Argument
`⟨module⟩: none` `string`

Optional module name.

`#property(..⟨args⟩)`

Shows a command property (annotation). This should be used in the `[body]` of `#command` to annotate a function with some special meaning.

Properties are provided as named arguments to the `#property` function.

The following properties are currently known to Mantys:

↑ Introduced in 1.0.1

since `version` | `string` Marks this function as available since a given package version.

↓ Available until 0.1.4

until `version` | `string` Marks this function as available until a given package version.

⊘ deprecated

deprecated `boolean` | `version` | `string` Marks this function as deprecated. If set to a version, the function is supposed to stay availalbel until the given version.

⇄ Changed in 0.12.0

changed `version` | `string` Marks function that changed in a specific package version.

t 0.12.0

compiler `version` | `string` Marks this function as only available on a specific compiler version.

context

requires-context `boolean` Requires a function to be used inside `#context`.

see `#mantys`, https://github.vom/jneug/typst-mantys

see `array` **of** `string` | `label` Adds references to other commands or websites.

✓ **TODO**
- Add documentation.
- Add `⟨foo⟩` paramter.

todo `string` | `content` Adds a todo note to the function.

Other named properties will be shown as given:

```
module: utilities
```

Argument

`..⟨args⟩` `any`

Property name / value pairs.

```
#command(⟨name⟩, ⟨label⟩: auto, ⟨properties⟩: (:), ..⟨args⟩)[body] → content
```

Displays information of a command by formatting the name, description and arguments. See this commands description for an example.

The command is formated with `#cmd` and an index entry is added that is marked as the “main” index entry for this command.

Argument

`⟨name⟩` `string`

Name of the command.

Argument

`⟨label⟩: auto` `string` | `auto` | `none`

Custom label for the command.

Argument

`⟨properties⟩: (:)` `dictionary`

Dictionary of properties to be passed to `#property`.

Argument

`..⟨args⟩` `content`

List of arguments created with the argument functions (`#arg`, `#barg`, `#sarg`) or `#args`.

Argument

`⟨body⟩` `content`

Description of the command. Usually some text and a series of `#argument` descriptions.

```
#variable(
  ⟨name⟩,
  ⟨types⟩: none,
  ⟨value⟩: none,
  ⟨label⟩: auto,
  ⟨properties⟩: (:)
)[body] → content
```

Displays information for a variable definition.

```
#variable("primary", types:("color",), value:green)[
  Primary color.
]
```

`#primary: rgb("#2ecc40")` `color`

Primary color.

Argument

`⟨name⟩` `string`

Name of the variable.

Argument
```
⟨types⟩: none                                                 array
```
Array of types to be passed to #`dtypes`.

Argument
```
⟨value⟩: none                                                   any
```
Default value.

Argument
```
⟨label⟩: auto                                  string | auto | none
```
Custom label for the variable.

Argument
```
⟨properties⟩: (:)                                        dictionary
```
Dictionary of properties to be passed to #`property`.

Argument
```
⟨body⟩                                                      content
```
Description of the variable.

```
#argument(
  ⟨name⟩,
  ⟨is-sink⟩: false,
  ⟨types⟩: none,
  ⟨choices⟩: none,
  ⟨default⟩: "__none__",
  ⟨title⟩: "Argument",
  ⟨properties⟩: (:),
  ⟨command⟩: none
)[body] → content
```
Displays information for a command argument. See the argument list below for an example.

```
#argument("category", default:"utilities")[
  #lorem(10)
]

#argument("category", choices: ("a", "b", "c"), default:"d")[
  #lorem(10)
]

#argument("style-args", title:"Style Arguments",
    is-sink:true, types:(length, ratio))[
  #lorem(10)
]
```

Argument

`⟨category⟩: "utilities"` `string`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Argument

`⟨category⟩: "d"` `string`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Style Arguments

`..⟨style-args⟩` `length` | `ratio`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Argument

`⟨name⟩` `string`

Name of the argument.

Argument

`⟨is-sink⟩: false` `boolean`

If this is a variadic argument.

Argument

`⟨types⟩: none` `array` | `none`

Array of types to be passed to `#dtypes`.

Argument

`⟨choices⟩: none` `array` | `none`

Optional array of valid values for this argument.

Argument
`⟨default⟩: "__none__"` `any`

Optional default value for this argument. Will be automatically included in `⟨choices⟩` if it is missing. To allow `none` as a default value, the default is `"__none___"`.

Argument
`⟨title⟩: "Argument"` `string` | `none`

Title in the border of the surrounding `#block`.

Argument
`⟨properties⟩: (:)` `dictionary`

Dictionary of properties to be passed to `#property`.

Argument
`⟨command⟩: none` `string` | `dictionary`

Optional information about the command this argument is attached to. Setting this to the name of a command will create a label for this argument in the form of `@cmd:cmd-name.arg-name`.

`@cmd:argument.title` ⟶ `#argument.title`

TIDY will automatically set this to the appropriate command.

Argument
`⟨body⟩` `content`

Description of the argument.

`#cmdref(⟨name⟩, ⟨module⟩: none) ⟶ content`

Creates a reference to the command `⟨name⟩`. This is equivalent to using `@cmd:name`.

- `#cmdref("cmdref")` ⟶ `#cmdref`
- `@cmd:cmdref` ⟶ `#cmdref`

Argument
`⟨name⟩` `string`

Name of the command.

Argument
`⟨module⟩: none` `string`

Optional module name.

`#argref(⟨command⟩, ⟨name⟩, ⟨module⟩: none) ⟶ content`

Creates a reference to the argument `⟨name⟩`. This is equivalent to using `@cmd:command.name`.

- `#argref("argref", "name")` → `#argref.name`
- `@cmd:argref.name` → `#argref.name`

Argument
`⟨command⟩` `string`

Name of the command.

Argument
`⟨name⟩` `string`

Name of the argument.

Argument
`⟨module⟩: none` `string`

Optional module name.

`#typeref[name]` → `content`

Creates a reference to the custom type `⟨name⟩`. This is equivalent to using `@type:name`.

Note that the custom type has to be declared first. See Section IV.2.0.a for more information about custom types.

Argument
`⟨name⟩` `content`

Name of the custom type.

VI.1.2 Types

`#dtypes`	`#is-custom-type`	`#type-box`

`#type-box(⟨name⟩, ⟨color⟩)`

Creates a colored box for a type, similar to those on the Typst website.

- `#type-box("color", red)` → `color`

Argument
`⟨name⟩` `string`

Name of the type.

Argument
`⟨color⟩` `color`

Color for the type box.

context

`#is-custom-type(⟨name⟩)`

Test if `⟨name⟩` was registered as a custom type.

`#dtypes(..⟨types⟩, ⟨link⟩: true, ⟨sep⟩: box(inset: (left: 1pt, right: 1pt), sym.bar.v))`

Creates a list of datatypes.

`#_type-map`

Dictionary of builtin types, mapping the types name to its actual type.

`#_type-aliases`

Dictionary of allowed type aliases, like `dict` for `dictionary`.

`#_type-colors`

Dictionary of colors to use for builtin types.

VI.1.3 Values

`#choices`	`#default`	`#value`

`#value(⟨value⟩, ⟨parse-str⟩: false) → content`

Shows `⟨value⟩` as content.

- `#value("string")` ⟶ `"string"`
- `#value([string])` ⟶ `[string]`
- `#value(true)` ⟶ `true`
- `#value(1.0)` ⟶ `1.0`
- `#value(3em)` ⟶ `3em`
- `#value(50%)` ⟶ `50%`
- `#value(left)` ⟶ `left`
- `#value((a: 1, b: 2))` ⟶ `(a: 1, b: 2)`

Argument
`⟨value⟩` `any`

- Value to show.

Argument
`⟨parse-str⟩: false` `boolean`

If `true`, parses strings as type names.

`#default(⟨value⟩, ⟨parse-str⟩: true) → content`

Highlights the default value of a set of `#choices`.

- `#default("default-value")` ⟶ `default-value`
- `#default(true)` ⟶ `true`

Argument
`⟨value⟩` `any`

The value to highlight.

Argument
`⟨parse-str⟩: true` `boolean`

If `true`, parses strings as type names.

`#choices(⟨default⟩: "__none__", ⟨sep⟩: sym.bar.v, ..⟨values⟩) → content`

Shows a list of choices possible for an argument.

If `⟨default⟩` is set to something else than `"__none__"`, the value is highlighted as the default choice. If `⟨default⟩` is already present in `⟨values⟩` the value is highlighted at its current position. Otherwise `⟨default⟩` is added as the first choice in the list.

Argument
`⟨default⟩: "__none__"` `any`

The default value to highlight.

Argument
`⟨sep⟩: sym.bar.v` `content`

Seperator between choices.

Argument
`..⟨values⟩` `any`

Values to choose from.

VI.1.4 Elements

#alert	#info-alert	#secondary
#changed	#module	#since
#colorize	#name	#success-alert
#compiler	#note	#until
#deprecated	#package	#ver
#error-alert	#primary	#warning-alert
#frame	#requires-context	

Styled by the theme

`#frame(..⟨args⟩) → content`

Create a frame around some content.

Uses SHOWYBOX and can take any arguments the `#showybox` command can take.

```
#frame(title:"Some lorem text")[#lorem(10)]
```

Some lorem text

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Argument

`..⟨args⟩` `content`

Arguments for Showybox.

🖌 Styled by the theme

#alert(⟨alert-type⟩)[body] → content

An alert box to highlight some content.

```
#alert("success")[#lorem(10)]
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Argument

`⟨alert-type⟩` `string`

The type of the alert. One of `"info"`, `"warning"`, `"error"` or `"success"`.

Argument

`⟨body⟩` `content`

Content of the alert.

🖌 Styled by the theme

#info-alert[body] → content

An info alert.

```
#info-alert[This is an #cmd-[info-alert].]
```

This is an `#info-alert`.

Argument

`⟨body⟩` `content`

Content of the alert.

🖌 Styled by the theme

#warning-alert[body] → content

A warning alert.

```
#warning-alert[This is an #cmd-[warning-alert].]
```

This is an `#warning-alert`.

Argument

`⟨body⟩` `content`

Content of the alert.

Styled by the theme

#error-alert[body] ⟶ content

An error alert.

```
#error-alert[This is an #cmd-[error-alert].]
```

This is an #error-alert.

Argument
⟨body⟩ content

Content of the alert.

Styled by the theme

#success-alert[body] ⟶ content

A success alert.

```
#success-alert[This is an #cmd-[success-alert].]
```

This is an #success-alert.

Argument
⟨body⟩ content

Content of the alert.

Styled by the theme

#package(⟨name⟩) ⟶ content

Show a package name.

- `#package("codelst")` ⟶ CODELST

Argument
⟨name⟩ string

Name of the package.

Styled by the theme

#module(⟨name⟩) ⟶ content

Show a module name.

- `#module("util")` ⟶ `util`

Argument
⟨name⟩ string

Name of the module.

`#name(⟨name⟩, ⟨last⟩: none) → content`

Highlight human names (with first- and lastnames).

- `#name("Jonas Neugebauer")` → Jonas Neugebauer
- `#name("J.", last:"Neugebauer")` → J. Neugebauer

Argument
`⟨name⟩` `string`

First or full name.

Argument
`⟨last⟩: none` `string` | `none`

Optional last name.

🖌 Styled by the theme

`#colorize(⟨color⟩: "primary")[body] → content`

Sets the text color of `⟨body⟩` to a color from the `theme`. `⟨color⟩` should be a key from the `theme`.

- `#colorize([Manual], color: "muted.fill")` → Manual

Argument
`⟨body⟩` `content`

Content to color.

Argument
`⟨color⟩: "primary"` `string`

Key of the color in the theme.

🖌 Styled by the theme

`#primary[body] → content`

Colors `[body]` in the themes primary color.

- `#primary[Manual]` → Manual

Argument
`⟨body⟩` `content`

Content to color.

🖌 Styled by the theme

`#secondary[body] → content`

Colors `[body]` in the themes secondary color.

- `#secondary[Manual]` → Manual

Argument
`⟨body⟩` `content`

Content to color.

`#ver(..⟨args⟩) → version`

Creates a `#version` from `..⟨args⟩`. If the first argument is a version, it is returned as given.

- `#ver(1, 4, 2)` → 1.4.2
- `#ver(version(1, 4, 3))` → 1.4.3

Argument
`..⟨args⟩` `version` | `integer`

Components of the version.

`#note(..⟨args⟩)[body]` → `content`

Show a margin note in the left margin. See `#since` and `#until` for examples.

Argument
`..⟨args⟩` `any`

Arguments to pass to `#drafting.margin-note`.

Argument
`⟨body⟩` `content`

Body of the note.

Styled by the theme

`#since(..⟨args⟩)` → `content`

↑ Introduced in 1.2.3

Show a margin-note with a minimal package version.

- `#since(1,2,3)` →

see `#note`, `#ver`

Argument
`..⟨args⟩` `integer` | `version`

Components of the version number.

Styled by the theme

`#until(..⟨args⟩)` → `content`

↓ Available until 1.2.3

Show a margin-note with a maximum package version.

- `#until(1,2,3)` →

see `#note`, `#ver`

Argument
`..⟨args⟩` `integer` | `version`

Components of the version number.

Styled by the theme

`#changed(..⟨args⟩)` → `content`

Show a margin-note with a version number.

⇄ Changed in 1.2.3

- `#changed(1,2,3)` →

> see #`note`, #`ver`

Argument

`..⟨args⟩` `integer` | `version`

Components of the version number.

Styled by the theme

`#deprecated` → `content`

Show a margin-note with a deprecated warning.

deprecated

- `#deprecated()` →

> see #`note`, #`ver`

Styled by the theme

`#compiler(..⟨args⟩)` → `content`

Show a margin-note with a minimal Typst compiler version.

1.2.3

- `#compiler(1,2,3)` →

> see #`note`, #`ver`

Argument

`..⟨args⟩` `integer` | `version`

Components of the version number.

Styled by the theme

`#requires-context` → `content`

Show a margin-note with a context warning.

context

- `#requires-context()` →

> see #`note`, #`ver`

VI.1.5 Examples

`#codesnippet`	`#show-git-clone`	`#sourcecode`
`#ex`	`#show-import`	
`#example`	`#side-by-side`	

`#sourcecode(⟨title⟩: none, ⟨file⟩: none, ..⟨args⟩)[code]` → `content`

Shows sourcecode in a #`frame`. See Section V.3 for more information on sourcecode and examples.

~~~
#sourcecode(
  title:"Example",
  file:"sourcecode-example.typ"
)[```typ
#let module-name = "sourcecode-example"
```]
~~~

```
Example                    sourcecode-example.typ
#let module-name = "sourcecode-example"
```

Argument
`⟨title⟩: none` `string`

A title to show on top of the frame.

Argument
`⟨file⟩: none` `string`

A filename to show in the title of the frame.

Argument
`..⟨args⟩` `any`

Arguments for `#codly.local`.

Argument
`⟨code⟩` `content`

A `#raw` block of Typst code.

`#codesnippet(⟨number-format⟩: none, ..⟨args⟩)[code] → content`

Shows some `#raw` code in a `#frame`, but without line numbers or other enhancements.

~~~
#codesnippet[```typc
let a = "some content"
[Content: #a]
```]
~~~

```
let a = "some content"
[Content: #a]
```

Argument
⟨number-format⟩: none boolean

If true, line numbers are shown.

Argument
..⟨args⟩ any

Arguments for #codly.local.

Argument
⟨code⟩ content

A #raw block of Typst code.

```
#example(
  ⟨side-by-side⟩: false,
  ⟨scope⟩: (:),
  ⟨imports⟩: (:),
  ⟨use-examples-scope⟩: true,
  ⟨mode⟩: "markup",
  ⟨breakable⟩: false,
  ..⟨args⟩
)[example-code] → content
```

Show an example by evaluating the given #raw code with Typst and showing the source and result in a #frame.

See Section V.3 for more information on sourcecode and examples.

Argument
⟨side-by-side⟩: false content

Shows the source and example in two columns instead of the result beneath the source.

Argument
⟨scope⟩: (:) dictionary

A scope to pass to #eval.

Argument
⟨imports⟩: (:) dictionary

Additional imports for evaluating this example. Imports will be added as a preamble to ⟨example-code⟩.

Argument
⟨use-examples-scope⟩: true boolean

Set to false to **not** use the global ⟨examples-scope⟩ passed to #mantys.

Argument
`⟨mode⟩: "markup"` `string`

The evaulation mode: `"markup"` | `"code"` | `"math"`

Argument
`⟨breakable⟩: false` `boolean`

If `true`, the frame may brake over multiple pages.

Argument
`⟨example-code⟩` `content`

A `#raw` block of Typst code.

Argument
`..⟨args⟩` `content`

An optional second positional argument that overwrites the evaluation result. This can be used to show the result of a sourcecode, that can not evaulated directly.

```
#side-by-side(
  ⟨side-by-side⟩: true,
  ⟨scope⟩: (:),
  ⟨imports⟩: (:),
  ⟨use-examples-scope⟩: true,
  ⟨mode⟩: "markup",
  ⟨breakable⟩: false,
  ..⟨args⟩
)[example-code] → content
```

Same as `#example`, but with `⟨side-by-side⟩: true`.

Argument
`⟨side-by-side⟩: true` `content`

Shows the source and example in two columns instead of the result beneath the source.

Argument
`⟨scope⟩: (:)` `dictionary`

A scope to pass to `#eval`.

Argument
`⟨imports⟩: (:)` `dictionary`

Additional imports for evaluating this example. Imports will be added as a preamble to `⟨example-code⟩`.

Argument
`⟨use-examples-scope⟩: true` `boolean`

Set to `false` to **not** use the global `⟨examples-scope⟩` passed to `#mantys`.

Argument
`⟨mode⟩: "markup"` `string`

The evaulation mode: `"markup"` | `"code"` | `"math"`

Argument
`⟨breakable⟩: false` `boolean`

If `true`, the frame may brake over multiple pages.

Argument
`⟨example-code⟩` `content`

A `#raw` block of Typst code.

Argument
`..⟨args⟩` `content`

An optional second positional argument that overwrites the evaluation result. This can be used to show the result of a sourcecode, that can not evaulated directly.

`#ex(⟨sep⟩: [ #sym.arrow.r ], ⟨mode⟩: "markup", ⟨scope⟩: (:))[code]` → `content`

Show a "short example" by showing `⟨code⟩` and the evaluation of `⟨code⟩` separated by `⟨sep⟩`. This can be used for quick one-line examples as seen in `#name` and other command docs in this manual.

```
- #ex(`#name("Jonas Neugebauer")`)
- #ex(`#meta("arg-name")`, sep: ": ")
```

- `#name("Jonas Neugebauer")` → Jonas Neugebauer
- `#meta("arg-name")`: `⟨arg-name⟩`

Argument
`⟨code⟩` `content`

The `#raw` code example to show.

Argument
`⟨sep⟩: [ #sym.arrow.r ]` `content`

The separator between `⟨code⟩` and its evaluated result.

Argument

`⟨mode⟩: "markup"` `string`

One of `"markup"` | `"code"` | `"math"`.

Argument

`⟨scope⟩: (:)` `dictionary`

A scope argument similar to `examples-scope`.

```
#show-import(
  ⟨repository⟩: "@preview",
  ⟨imports⟩: "*",
  ⟨name⟩: auto,
  ⟨version⟩: auto,
  ⟨mode⟩: "markup",
  ⟨code⟩: none
) → content
```

Shows an import statement for this package. The name and version from the document are used by default.

```
#show-import()
#show-import(repository: "@local", imports: "mantys", mode:"code")
```

```
#import "@preview/Mantys:1.0.0": *
```

```
import "@local/Mantys:1.0.0": mantys
```

Argument

`⟨repository⟩: "@preview"` `string`

Custom package repository to show.

Argument

`⟨imports⟩: "*"` `string` | `none`

What to import from the package. Use `none` to just import the package into the global scope.

Argument

`⟨name⟩: auto` `string` | `auto`

Package name for the import.

Argument

`⟨version⟩: auto` `version` | `auto`

Package version for the import.

Argument
`⟨mode⟩: "markup"` `string`

One of `"markup"` | `"code"`. Will show the import in markup or code mode.

Argument
`⟨code⟩: none` `string` | `auto`

Additional code to add after the import. Useful if your package requires some more steps for initialization.

```
#show-import(name: "codly", version: version(1,1,1), code:
"#show: codly-init")
```

```
#import "@preview/codly:1.1.1": *
#show: codly-init
```

`#show-git-clone(⟨repository⟩: auto, ⟨out⟩: auto, ⟨lang⟩: "bash")`

Shows a git clone command for this package. The name and version from the document are used by default.

```
#show-git-clone()
#show-git-clone(repository: "typst/packages", out:"preview/
mantys/1.0.0")
```

```
git clone https://github.com/jneug/typst-mantys Mantys/1.0.0
```

```
git clone https://github.com/typst/packages preview/mantys/1.0.0
```

Argument
`⟨repository⟩: auto` `string` | `auto`

Custom package repository to show.

Argument
`⟨out⟩: auto` `string` | `none` | `auto`

Output path to clone into.

Argument
`⟨lang⟩: "bash"` string

Syntax language to passs to `#raw`.

deprecated `#shortex`

Alias for `#ex`.

VI.1.6 Icons

`#icon`

`#icon(⟨name⟩, ⟨fill⟩: auto, ..⟨args⟩) → content`

Shows an icon from the 0x6b/typst-octique[26] package.

Argument
`⟨name⟩` string

- name: A name from the Octique icon set.

Argument
`⟨fill⟩: auto` color | auto

- fill: The fill color for the icon. `auto` will use the fill of the surrounding text.

Argument
`..⟨args⟩` any

- ..args: Further args for the `#octique` command.

`#info`

The default info icon: ⓘ

`#warning`

The default info icon: ⚠

`#typst`

Typst icon provided by CODLY[27]: t

[26] https://github.com/0x6b/typst-octique
[27] https://typst.app/universe/package/codly

VI.2 Utilities

`#utils.add-preamble(⟨code⟩, ⟨imports⟩)` → `string`

Adds a preamble for customs imports to `⟨code⟩`.

Argument
`⟨code⟩` `content` | `text`

A Typst code block as `#raw` or `#str`.

Argument
`⟨imports⟩` `dictionary` | `string`

The imports to add to the code. If it is a `dictionary` it will first be passed to `#utils.build-preamble`.

`#utils.build-preamble(⟨imports⟩)` → `string`

Creates a preamble to attach to code before evaluating. `⟨imports⟩` is a `dictionary` with `(module: imports)` pairs, like `(mantys: "*")`. This will create a preamble of the form `"#import mantys: *;"`

```
#utils.build-preamble((mantys: "*", tidy: "parse-module, show-module"))
```

```
#import mantys: *; #import tidy: parse-module, show-module;
```

Argument
`⟨imports⟩` `dictionary`

`(module: imports)` pairs.

`#utils.create-label(⟨command⟩, ⟨arg⟩: none, ⟨module⟩: none, ⟨prefix⟩: "cmd")` → `label`

Creates a `#label` to be placed in the document (usually by cmd:utils.place-reference). The created label is in the same format Tidy uses but will be prefixed with `cmd` to identify command references outside of docstrings.

- `#str(mantys.utils.create-label("create-label", arg:"module", module:"utils"))` → cmd:utils:create-label.module

Argument
⟨command⟩ `string`

Name of the command.

Argument
⟨arg⟩: none `string`

Argument name to add to the label

Argument
⟨module⟩: none `string`

Optional module of the command.

Argument
⟨prefix⟩: "cmd" `string`

Prefix for command labels. By default command labels are prefixed with `cmd`, eg. `cmd:utils.create-label`.

#utils.dict-get(⟨dict⟩, ⟨key⟩, ⟨default⟩: none) → `any`

Gets the value at ⟨key⟩ from the `dictionary` ⟨dict⟩. ⟨key⟩ can be in dot-notation to access values in nested dictionaries.

Argument
⟨dict⟩ `dictionary`

Dictionary to get Data from

Argument
⟨key⟩ `string`

String key of the value in dot-notation.

Argument
⟨default⟩: none `any`

Default value, if the key can't be found.

#utils.dict-merge(..⟨dicts⟩) → `dictionary`

Recursivley merges the passed in dictionaries.

```
#get.dict-merge(
    (a: 1, b: 2),
    (a: (one: 1, two:2)),
    (a: (two: 4, three:3))
)
// gives (a:(one:1, two:4, three:3), b: 2)
```

Argument

`..⟨dicts⟩` `dictionary`

Dictionaries to merge.

`#utils.dict-update(⟨dict⟩, ⟨key⟩, ⟨func⟩, ⟨default⟩: none) → dictionary`

Updates the value in `⟨dict⟩` at `⟨key⟩` by passing the value to `⟨func⟩` and storing the result. If `⟨key⟩` is not in `⟨dict⟩`, `⟨default⟩` is used instead.

`⟨key⟩` may be in dot-notation to update values in nested dictionaries.

Argument

`⟨dict⟩` `dictionary` | `any`

The `dictionary` to update.

Argument

`⟨key⟩` `string`

The key of the value. May be in dot-notation.

Argument

`⟨func⟩` `function`

Update function: `( any )→ any`

Argument

`⟨default⟩: none` `any`

Default value to use if `⟨key⟩` is not found in `⟨dict⟩`.

`#utils.get-text[it] → string`

Extracts text from `content`.

Argument

`⟨it⟩` `content`

A content element.

`#utils.get-text-color(⟨color⟩, ⟨light⟩: white, ⟨dark⟩: black) → color`

Returns a light or dark color, depending on the provided `⟨color⟩`.

```
- #utils.get-text-color(red)
- #utils.get-text-color(red.lighten(50%))
```

- `luma(100%)`
- `luma(0%)`

Argument
⟨color⟩ `color` | `gradient`

Paint to get the text color for.

Argument
⟨light⟩: `white` `color` | `gradient`

Color to use, if ⟨color⟩ is a dark color.

Argument
⟨dark⟩: `black` `color` | `gradient`

Color to use, if ⟨color⟩ is a light color.

#utils.parse-label(⟨label⟩) ⟶ `dictionary`

Parses a `#label` text into a `dictionary` with the command and module name (if present). A label in the format `"cmd:utils.split-cmd-name.arg-name"` will be split into
`(name: "split-cmd-name", arg:"arg-name", module: "utils", prefix:"cmd")`
TODO: removing "mantys" prefix should happen in tidy template

Argument
⟨label⟩ `label` | `string`

The label to parse.

#utils.place-reference(⟨label⟩, ⟨kind⟩, ⟨supplement⟩, ⟨numbering⟩: "1") ⟶ `content`

Places a hidden `#figure` in the document, that can be referenced via the usual `@label-name` syntax.

Argument
⟨label⟩ `label`

Label to reference.

Argument
⟨kind⟩ `string`

Kind for the reference to properly step counters.

Argument
⟨supplement⟩ `string`

Supplement to show when referencing.

Argument
⟨numbering⟩: `"1"` `string`

Numbering schema to use.

`#utils.rawc(⟨color⟩, ⟨code⟩, ⟨lang⟩: none) → content`

Shows `⟨code⟩` as inline `#raw` text (with `⟨block⟩: false`) and with the given `⟨color⟩`. The language argument will be passed to `#raw`, but will have no effect, since `⟨code⟩` will have an uniform color.

- `#utils.rawc(purple, "some inline code")` → `some inline code`

Argument
`⟨color⟩` `color`

Color for the `#raw` text.

Argument
`⟨code⟩` `string`

String content to be displayed as `#raw`.

Argument
`⟨lang⟩: none` `string`

Optional language name.

`#utils.rawi(⟨code⟩, ⟨lang⟩: none) → content`

Displays `⟨code⟩` as inline `#raw` code (with `⟨inline⟩: true`).

- `#utils.rawi("my-code")` → `my-code`

Argument
`⟨code⟩` `string` | `content`

The content to show as inline raw.

Argument
`⟨lang⟩: none` `string`

Optional language for highlighting.

`#utils.split-cmd-name(⟨name⟩) → dictionary`

Splits a string into a dictionary with the command name and module (if present).
A string of the form `"cmd:utils.split-cmd-name"` will be split into
`(name: "split-cmd-name", module: "utils")`
(Note that the prefix `cmd:` is removed.)

Argument
`⟨name⟩` `string`

The command optionally with module and `cmd:` prefix.

VI.3 Shortcut collection of builtin types

The `#typ` dictionary is a shortcut to the common Typst builtin functions, types (`#typ.t`) and values (`#typ.v`).

VI.3.1 Shortcuts for builtin commands

#typ.set
#typ.show
#typ.import
#typ.context
#typ.arguments
#typ.array
#typ.assert
#typ.auto
#typ.bool
#typ.bytes
#typ.with
#typ.calc
#typ.clamp
#typ.abs
#typ.pow
#typ.content
#typ.datetime
#typ.dictionary
#typ.duration
#typ.eval
#typ.float
#typ.function
#typ.int
#typ.label
#typ.module
#typ.none
#typ.panic
#typ.plugin
#typ.regex
#typ.repr
#typ.selector
#typ.str
#typ.style
#typ.sys
#typ.type
#typ.version
#typ.bibliography
#typ.cite
#typ.document
#typ.figure
#typ.emph
#typ.enum
#typ.list
#typ.numbering
#typ.outline
#typ.par
#typ.parbreak
#typ.quote
#typ.strong
#typ.ref
#typ.table
#typ.terms
#typ.link
#typ.raw
#typ.text
#typ.highlight
#typ.linebreak
#typ.lorem
#typ.lower
#typ.upper
#typ.overline
#typ.underline
#typ.smallcaps
#typ.smartquote
#typ.strike
#typ.sub
#typ.super
#typ.align
#typ.alignment
#typ.angle
#typ.block
#typ.box
#typ.colbreak
#typ.columns
#typ.direction
#typ.fraction
#typ.grid
#typ.h
#typ.hide
#typ.layout
#typ.length
#typ.measure
#typ.move
#typ.pad
#typ.page
#typ.pagebreak
#typ.place
#typ.ratio
#typ.relative
#typ.repeat
#typ.rotate
#typ.scale
#typ.stack
#typ.accent
#typ.attach
#typ.cancel
#typ.cases
#typ.class
#typ.equation
#typ.frac
#typ.lr
#typ.mat
#typ.op
#typ.primes
#typ.roots
#typ.sizes
#typ.styles
#typ.underover
#typ.variants
#typ.vec
#typ.circle
#typ.color
#typ.ellipse
#typ.gradient
#typ.image
#typ.line
#typ.path
#typ.pattern
#typ.polygon
#typ.rect
#typ.square
#typ.stroke
#typ.counter
#typ.here
#typ.locate
#typ.location
#typ.metadata
#typ.query
#typ.state
#typ.cbor
#typ.csv
#typ.json
#typ.read
#typ.toml
#typ.xml
#typ.yaml

VI.3.2 Shortcuts for builtin types

#typ.t.auto
#typ.t.none
#typ.t.arguments
#typ.t.array
#typ.t.boolean
#typ.t.bytes

- `#typ.t.content`
- `#typ.t.datetime`
- `#typ.t.dictionary`
- `#typ.t.float`
- `#typ.t.function`
- `#typ.t.integer`
- `#typ.t.location`
- `#typ.t.module`
- `#typ.t.plugin`
- `#typ.t.regex`
- `#typ.t.selector`
- `#typ.t.string`
- `#typ.t.type`
- `#typ.t.label`
- `#typ.t.version`
- `#typ.t.alignment`
- `#typ.t.angle`
- `#typ.t.direction`
- `#typ.t.fraction`
- `#typ.t.length`
- `#typ.t.ratio`
- `#typ.t.relative`
- `#typ.t.color`
- `#typ.t.gradient`
- `#typ.t.stroke`
- `#typ.t.bool`
- `#typ.t.str`
- `#typ.t.arr`
- `#typ.t.dict`
- `#typ.t.int`
- `#typ.t.func`

VI.3.3 Shortcuts for builtin values

- `#typ.v.false`
- `#typ.v.true`
- `#typ.v.none`
- `#typ.v.auto`
- `#typ.v.dict`
- `#typ.v.arr`

Part VII

Index