The Mantys Package

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

v0.1.2 2024-04-23 MIT

Helpers to build manuals for Typst packages.

Jonas Neugebauer

https://github.com/jneug/typst-mantys

Mantys is a Typst template to help package and template authors to write manuals. It provides functionality for consistent formatting of commands, variables, options 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 was 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 to write consistently formatted 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.

The design and functionality of Mantys was inspired by the fantastic LaTeX package CNLTX[3] by Clemens Niederberger[4].

This manual is supposed to be a complete reference of Mantys, but might be out of date for the most recent additions and changes. On the other hand, 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.

> Mantys is in active development and its functionality is subject to change. Until version 1.0.0 is reached, the command signatures and layout may change and break previous versions. Keep that in mind while using Mantys.
>
> Contributions to the package are very welcome!

[1] https://rust-lang.github.io/mdBook/
[2] https://asciidoc.org
[3] https://ctan.org/pkg/cnltx
[4] clemens@cnltx.de

Part II.
Usage

II.1. Using Mantys

Just import MANTYS inside your `typ` file:

```
#import "@preview/mantys:0.1.2": *
```

II.1.1. Initializing the template

After importing MANTYS the template is initialized by applying a show rule with the `#mantys()` command passing the necessary options using `with`:

```
#show: mantys.with(
...
)
```

`#mantys()` takes a bunch of arguments to describe the package. These can also be loaded directly from the `typst.toml` file in the packages' root directory:

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

```
#mantys(
  ⟨name⟩: none,
  ⟨title⟩: none,
  ⟨subtitle⟩: none,
  ⟨info⟩: none,
  ⟨authors⟩: (),
  ⟨url⟩: none,
  ⟨repository⟩: none,
  ⟨license⟩: none,
  ⟨version⟩: none,
  ⟨date⟩: none,
  ⟨abstract⟩: [],
  ⟨titlepage⟩: #titlepage,
  ⟨examples-scope⟩: (:),
  ..⟨args⟩
)[⟨body⟩]
```

Argument

```
⟨titlepage⟩: #titlepage
```

A function that renders a titlepage for the manual. Refer to `#name()` for details.

Argument

⟨examples-scope⟩: (:)

Default scope for code examples.

```
examples-scope: (
  cmd: mantys.cmd
)
```

For further details refer to `#name()`.

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

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

II.2. Available commands

`#arg()`	`#cmdref()`	`#meta()`
`#args()`	`#command()`	`#module-commands()`
`#argument()`	`#default()`	`#opt()`
`#barg()`	`#doc()`	`#relref()`
`#choices()`	`#dtype()`	`#sarg()`
`#cmd()`	`#dtypes()`	`#symbol()`
`#cmd-label()`	`#func()`	`#value()`
`#cmd-selector()`	`#lambda()`	`#var()`
`#var-label()`		
`#variable()`		

II.2.1. Describing arguments and values

`#meta(⟨name⟩)` ⟶

Highlight an argument name. `#meta[variable]` ⟶ ⟨variable⟩

Argument

⟨name⟩

Name of the argument.

`#value(⟨value⟩)` ⟶

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.

`#default(⟨value⟩)` ⟶

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

- `#default("default-value")` ⟶ `"default-value"`
- `#default(true)` ⟶ `true`
- `#choices(1, 2, 3, 4, default: 3)` ⟶ `1|2|3|4`

Argument
⟨value⟩ any

The value to highlight.

`#doc(⟨target⟩, ⟨name⟩: none, ⟨anchor⟩: none, ⟨fnote⟩: false)` ⟶

Create a link to the reference documentation at https://typst.app/docs/reference/.

```
See the #doc("meta/locate") function.
```

See the `locate` function.

Argument
⟨target⟩

Path to the subpage of `https://typst.app/docs/reference/`. The `lowercase` command for example is located in the category `text` and has `⟨target⟩: "text/lowercase"`.

Argument
⟨name⟩: none

Optional name for the link. With `auto`, the `⟨target⟩` is split on `/` and the last part is used.

Argument
⟨anchor⟩: none

An optional HTML page anchor to append to the link.

Argument
⟨fnote⟩: false bool

Show the reference link in a footnote.

`#dtype(⟨type⟩, ⟨fnote⟩: false, ⟨parse-type⟩: false)` ⟶

Shows a highlightd data type with a link to the reference page.

`⟨t⟩` may be any value to pass to `type` to get the type or a with the name of a datatype. To show the `string` type, use `#dtype("string")`. To force the parsing of the values type, set `⟨parse-type⟩: true`.

- `#dtype("int")` ⟶ `int`
- `#dtype(1)` ⟶ `int`
- `#dtype(1deg)` ⟶ `angle`
- `#dtype(true)` ⟶ `bool`
- `#dtype(())` ⟶ `array`
- `#dtype(red)` ⟶ `color`

Argument

⟨type⟩ `any`

Either a value to take the type from or a string with the datatype name.

Argument

⟨fnote⟩: `false` `bool`

If `true`, the reference lin kis shown in a footnote.

Argument

⟨parse-type⟩: `false` `bool`

If ⟨t⟩ should always be passed to `type`.

`#dtypes(..⟨types⟩, ⟨sep⟩: box(inset: (left: 1pt, right: 1pt), body: [|]))` ⟶

Shows a list of datatypes.

- `#dtypes(false, "integer", (:))` ⟶ `bool` | `int` | `dictionary`

Argument

..⟨types⟩ `any`

List of values to get the type for or strings with datatype names.

`#arg(..⟨args⟩)` ⟶

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

`#barg(⟨name⟩)` ⟶

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⟩

Name of the argument.

#sarg(⟨name⟩) ⟶

Shows an argument sink.

- `#sarg[args]` ⟶ `..⟨args⟩`

Argument

⟨name⟩

Name of the argument.

#args(..⟨args⟩) ⟶ `array`

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

s and named arguments are passed to `#arg()`, while is passed to `#barg()`. The result is to 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 () or a (name: value) pair either as a named argument or as exactly two positional arguments.

#choices(⟨default⟩: "__none__", ..⟨values⟩) ⟶

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 given in ⟨values⟩, the value is highlighted at its current position. Otherwise ⟨default⟩ is added as the first choice in the list.

- `#choices(left, right, center)` ⟶ `left|right|center`
- `#choices(left, right, center, default:center)` ⟶ `left|right|center`
- `#choices(left, right, default:center)` ⟶ `center|left|right`
- `#arg(align: choices(left, right, default:center))` ⟶ `⟨align⟩: center|left|right`

#symbol(⟨name⟩, ⟨module⟩: none) ⟶

Shows a Typst reserved symbol argument.

- `#symbol("dot")` ⟶ `dot`
- `#symbol("angle.l", module:"sym")` ⟶ `sym.angle.l`
- `#arg(format: symbol("angle.l", module:"sym"))` ⟶ `⟨format⟩: sym.angle.l`

#func(⟨name⟩, ⟨module⟩: none)

Create a function argument. Function arguments may be used as an argument value with `#arg()`.

- `#func("func")` ⟶ *#func*
- `#func("clamp", module:"calc")` ⟶ *#calc.clamp*
- `#arg(format: func("upper"))` ⟶ ⟨format⟩: *#upper*

#lambda(..⟨args⟩, ⟨ret⟩: none) ⟶

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("integer", "boolean", ret:"string")` ⟶ (`int`, `bool`) =>
- `#lambda("..any", ret:"boolean")` ⟶ (.. `any`) => `bool`
- `#arg(format: lambda("string", ret:"content"))` ⟶ ⟨format⟩: (_) =>

II.2.2. Describing commands

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

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

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

All positional arguments will be rendered first, then named arguments and all body arguments will be added after the closing paranthesis.

- `#cmd("cmd", arg[name], sarg[args], barg[body])` ⟶ `#cmd(⟨name⟩, ..⟨args⟩)[⟨body⟩]`
- `#cmd("cmd", ..args("name", [body]), sarg[args])` ⟶ `#cmd(⟨name⟩, ..⟨args⟩)[⟨body⟩]`

Argument

⟨name⟩

Name of the command.

Argument

⟨module⟩: none

Name of a module, the command belongs to.

Argument

⟨ret⟩: none — any

Returned type.

Argument

⟨index⟩: true — bool

Whether to add an index entry.

Argument

⟨unpack⟩: false bool

If true, the arguments are shown in separate lines.

Argument

..⟨args⟩ any

Arguments for the command, created with the argument commands above or #args().

#opt(⟨name⟩, ⟨index⟩: true, ⟨clr⟩: rgb("#0074d9")) ⟶

Shows the option ⟨name⟩ and adds an entry with ⟨kind⟩: "option" to the index.

- #opt[examples-scope] ⟶ examples-scope

Argument

⟨name⟩

Name of the option.

Argument

⟨index⟩: true bool

Whether to create an index entry.

Argument

⟨clr⟩: rgb("#0074d9") color

A color

#var(⟨name⟩) ⟶

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

- #var[colors] ⟶ #colors

#cmd-label(⟨name⟩, ⟨module⟩: "") ⟶ label

Creates a label for the command with name ⟨name⟩.

Argument

⟨name⟩

Name of the command.

Argument

⟨module⟩: ""

Optional module name.

#var-label(⟨name⟩, ⟨module⟩: "") ⟶ label

Creates a label for the variable with name ⟨name⟩.

Argument

⟨name⟩

Name of the variable.

Argument

⟨module⟩: ""

Optional module name.

#command(⟨name⟩, ⟨label⟩: auto, ..⟨args⟩)[⟨body⟩] ⟶

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

Argument

⟨name⟩

Name of the command.

Argument

⟨label⟩: auto

Custom label for the command. Defaults to auto.

Argument

..⟨args⟩

List of arguments created with the argument functions like #arg().

Argument

⟨body⟩

Description for the command.

#variable(⟨name⟩, ⟨types⟩: none, ⟨value⟩: none, ⟨label⟩: auto)[⟨body⟩] ⟶

Displays information for a variable defintion.

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

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

Primary color.

Argument

⟨name⟩

Name of the variable.

Argument

⟨types⟩: none array

Array of types to be passed to #dtypes().

Argument

⟨value⟩: none any

Default value.

Argument

⟨body⟩

Description of the variable.

```
#argument(
  ⟨name⟩,
  ⟨is-sink⟩: false,
  ⟨types⟩: none,
  ⟨choices⟩: none,
  ⟨default⟩: "__none__"
)[⟨body⟩] ⟶
```

Displays information for a command argument. See the argument list below for an example.

Argument

⟨name⟩

Name of the argument.

Argument

⟨is-sink⟩: false `bool`

If this is a sink argument.

Argument

⟨types⟩: none `array`

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

Argument

⟨choices⟩: none `array`

Optional array of valid values for this argument.

Argument

⟨default⟩ `any`

Optional default value for this argument.

Argument

⟨body⟩

Description of the argument.

```
#module-commands(⟨module⟩)[⟨body⟩] ⟶
```

A wrapper around `#command()` calls that belong to an internal module. The module name is displayed as a prefix to the command name.

```
#module-commands("mty")[
  #command("rawi")[
   Shows #arg[code] as inline #doc("text/raw") text (with #arg(block: false)).
  ]
]
```

#mty.rawi()

Shows ⟨code⟩ as inline `raw` text (with ⟨block⟩: `false`).

Argument

⟨module⟩

Name of the module.

Argument

⟨body⟩

Content with `#command()` calls.

#cmd-selector(⟨name⟩) ⟶

Creates a selector for a command label.

Argument

⟨name⟩

Name of the command.

#cmdref(⟨name⟩, ⟨module⟩: none, ⟨format⟩: (..) => ..) ⟶ _

Creates a reference to a command label.

Argument

⟨name⟩

Name of the command.

Argument

⟨module⟩: none

Optional module name.

Argument

⟨format⟩: (..) => .. function

Function of (_, `location`) => _ to format the reference. The first argument is the name of the command (the same as ⟨name⟩) and the second is the location of the referenced label.

#relref(⟨label⟩) ⟶ _

Creats a relative reference showing the text “above” or “below”.

- `#relref(cmd-label("meta"))` ⟶ above
- `#relref(cmd-label("shortex"))` ⟶ below

Argument

⟨label⟩ label

The label to reference.

#cmd-

Same as #cmd(), but does not create an index entry.

#opt-

Same as #opt(), but does not create an index entry.

#var-

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

II.2.3. Source code and examples

Mantys provides several commands to handle source code snippets and show examples of functionality. The usual raw command still works, but theses commands allow you to highlight code in different ways or add line numbers.

Typst code examples can be set with the #example() command. Simply give it a fenced code block with the example code and Mantys will render the code as highlighted Typst code and show the result underneath.

````
#example[```
This will render as *content*.

Use any #emph[Typst] code here.
```]
````

```
This will render as *content*.

Use any #emph[Typst] code here.
```

This will render as **content**.

Use any *Typst* code here.

The result will be generated using eval and thus run in a local scope without access to imported functions. To pass your functions or modules to #example() either set the examples-scope option in the intial #mantys() call or pass a ⟨scope⟩ argument to #example() directly.

See below for how to use the #example() command.
```

To use fenced code blocks in your example, add an extra backtick to the example code:

````
#example[````
  ```rust
  fn main() {
    println!(\"Hello World!\");
  }
  ```
````]
````

---

````
  ```rust
  fn main() {
    println!(\"Hello World!\");
  }
  ```
````

---

```rust
fn main() {
  println!(\"Hello World!\");
}
```

**`#example`**`(`⟨**`side-by-side`**⟩`: false, `⟨**`imports`**⟩`: (:), `⟨**`mode`**⟩`: "code")[`⟨**`example-code`**⟩`][`⟨**`result`**⟩`]`

Sets `[`⟨`example-code`⟩`]` as a `raw` block with ⟨`lang`⟩`: "typ"` and the result of the code beneath. `[`⟨`example-code`⟩`]` need to be `raw` code itself.

````
#example[```
  Some lorem ipsum:\
  #lorem(40)
```]
````

```
*Some lorem ipsum:*\
#lorem(40)
```

Some lorem ipsum:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere.

Argument

⟨`example-code`⟩

A block of `raw` code representing the example Typst code.

Argument

⟨`side-by-side`⟩`: false`

Usually, the ⟨example-code⟩ is set above the ⟨result⟩ separated by a line. Setting this to `true` will set the code on the left side and the result on the right.

Argument

⟨scope⟩: (:)

The scope to pass to `eval`.

Examples will always import the `examples-scope` set in the initial `#mantys()` call. Passing this argument to an `#example()` call *additionally* make those imports available in thsi example. If an example should explicitly run without imports, pass ⟨scope⟩: `none`:

```
#example[`I use #opt[examples-scope].`]

#example(scope:none)[```
// This will fail: #opt[examples-scope]
I can't use `#opt()`, because i don't use `examples-scope`.
```]
```

Argument

⟨mode⟩: "code"

The mode to evaluate the example in. See `eval/mode` for more information.

Argument

⟨result⟩

The result of the example code. Usually the same code as ⟨example-code⟩ but without the `raw` markup. See below for an example of using [⟨result⟩].

⟨result⟩ is optional and will be omitted in most cases!

Setting ⟨side-by-side⟩: `true` will set the example on the left side and the result on the right and is useful for short code examples. The command `#side-by-side()` exists as a shortcut.

```
#example(side-by-side: true)[```
  Some lorem ipsum:\
  #lorem(20)
```]
```

```
*Some lorem ipsum:*\
#lorem(20)
```

Some lorem ipsum:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

[⟨example-code⟩] is passed to `#mty.sourcecode()` for processing.

If the example-code needs to be different than the code generating the result (for example, because automatic imports do not work or access to the global scope is required), `#example()` accepts an optional second positional argument `[⟨result⟩]`. If provided, `[⟨example-code⟩]` is not evaluated and `[⟨result⟩]` is used instead.

````
#example[```
  #value(range(4))
```][
  The value is: #value(range(4))
]
````

```
#value(range(4))
```

The value is: `(0, 1, 2, 3)`

`#side-by-side(⟨scope⟩: (:), ⟨mode⟩: "code")[⟨example-code⟩][⟨result⟩]`

Shortcut for `#example(⟨side-by-side⟩: true)`.

`#sourcecode(⟨title⟩: none, ⟨file⟩: none)[⟨code⟩]`

If provided, the `⟨title⟩` and `⟨file⟩` argument are set as a titlebar above the content.

Argument

`⟨code⟩`

A `#raw()` block, that will be set inside a bordered block. The `raw` content is not modified and keeps its `⟨lang⟩` attribute, if set.

Argument

`⟨title⟩: none`

A title to show above the code in a titlebar.

Argument

`⟨file⟩: none`

A filename to show above the code in a titlebar.

`#sourcecode()` will render a `raw` block with linenumbers and proper tab indentions using CODELST and put it inside a `#mty.frame()`.

If provided, the `⟨title⟩` and `⟨file⟩` argument are set as a titlebar above the content.
```

```
#sourcecode(title:"Some Rust code", file:"world.r")[```rust
fn main() {
println!("Hello World!");
}
```]
```

---

Some Rust code — *world.r*

```rust
fn main() {
  println!("Hello World!");
}
```

#codesnippet()[⟨code⟩]

A short code snippet, that is shown without line numbers or title.

```
#codesnippet[```shell-unix-generic
git clone https://github.com/jneug/typst-mantys.git mantys-0.0.3
```]
```

```shell-unix-generic
git clone https://github.com/jneug/typst-mantys.git mantys-0.0.3
```

#shortex(⟨sep⟩: sym.arrow.r)[⟨code⟩]

Display a very short example to highlight the result of a single command. ⟨sep⟩ changes the separator between code and result.

```
- #shortex(`#emph[emphasis]`)
- #shortex(`#strong[strong emphasis]`, sep:"::")
- #shortex(`#smallcaps[Small Capitals]`, sep:sym.arrow.r.double.long)
```

- `#emph[emphasis]` ⟶ *emphasis*
- `#strong[strong emphasis]` :: **strong emphasis**
- `#smallcaps[Small Capitals]` ⟹ Small Capitals

II.2.4. Other commands

#package()

Shows a package name:

- `#package[tablex]` ⟶ TABLEX
- `#mty.package[tablex]` ⟶ TABLEX

#module()

Shows a module name:

- `#module[mty]` ⟶ mty

- `#mty.module[mty]` ⟶ `mty`

`#doc(⟨target⟩, ⟨name⟩: none, ⟨fnote⟩: false)`

Displays a link to the Typst reference documentation at https://typst.app/docs. The `⟨target⟩` need to be a relative path to the reference url, like `"text/raw"`. `#doc()` will create an appropriate link URL and cut everything before the last `/` from the link text.

The text can be explicitly set with `⟨name⟩`. For `(fnote: true)` the documentation URL is displayed in an additional footnote.

```
Remember that #doc("meta/query") requires a #doc("meta/locate",
name:"location") obtained by #doc("meta/locate", fnote:true) to work.
```

Remember that `query` requires a `location` obtained by `locate`[5] to work.

> Footnote links are not yet reused if multiple links to the same reference URL are placed on the same page.

`#command-selector(⟨name⟩)`

Creates a `selector` for the specified command.

```
// Find the page of a command.
#let cmd-page( name ) = locate(loc => {
  let res = query(cmd-selector(name), loc)
  if res == () {
    panic("No command " + name + " found.")
  } else {
    return res.last().location().page()
  }
})

The #cmd-[mantys] command is documented on page #cmd-page("mantys").
```

The `#mantys()` command is documented on page 3.

II.2.5. Using Tidy

MANTYS can be used with the docstring parser TIDY, to create a manual from the comments above each function. See the TIDY manual for more information on this.

MANTYS ships with a TIDY template and a helper function to use it.

```
#tidy-module(
  ⟨data⟩,
  ⟨include-examples-scope⟩: false,
  ⟨extract-headings⟩: 2,
  ⟨tidy⟩: none,
```

[5] https://typst.app/docs/reference/meta/locate

```
  ..⟨args⟩
)
```

`#tidy-module()` calls `#tidy.parse-module()` and `#tidy.show-module()` on the provided ⟨tidy⟩ instance. If no instance is provided, the current Tidy version from the preview repository is used.

Setting ⟨include-examples-scope⟩: `true` will add the `examples-scope` passed to `#mantys()` to the evaluation of the module.

To extract headings up to a certain level from function docstrings and showing them between function documentations, set ⟨extract-headings⟩ to the highest heading level that should be extracted. ⟨extract-headings⟩: `none` disables this.

This manual was compiled with ⟨extract-headings⟩: `3` and thus the Section II.2.1 heading was shown before the description of `#api.meta()`.

II.2.6. Templating and styling

```
#titlepage(
  ⟨name⟩,
  ⟨title⟩,
  ⟨subtitle⟩,
  ⟨info⟩,
  ⟨authors⟩,
  ⟨urls⟩,
  ⟨version⟩,
  ⟨date⟩,
  ⟨abstract⟩,
  ⟨license⟩
)
```

The `#titlepage()` command sets the default titlepage of a Mantys document.

To implement a custom title page, create a function that takes the arguments shown above and pass it to `#mantys()` as ⟨titlepage⟩:

```
#let my-custom-titlepage( ..args ) = [*My empty title*]

#show: mantys.with(
  ..toml("typst.toml"),
  titlepage: my-custom-titlepage
)
```

A ⟨titlepage⟩ function gets passed the package information supplied to `#mantys()` with minimal preprocessing. The function has to check for `none` values for itself. The only argument with a guaranteed value is ⟨name⟩.

II.2.7. Utilities

Most of MANTYS functionality is located in a module named `mty`. Only the main commands are exposed at a top level to keep the namespace pollution as minimal as possible to prevent name collisions with commands belonging to the package / module to be documented.

The commands provide some helpful low-level functionality, that might be useful in some cases.

> Some of the utilities of previous versions are now covered by TOOLS4TYPST.

```
#add-mark()          #gitlink()          #name()
#alert()             #has-mark()         #package()
#author()            #idx()              #pkglink()
#cblock()            #idx-term()         #place-marker()
#code-example()      #make-index()       #rawc()
#date()              #marginnote()       #rawi()
#footlink()          #marker()           #sourcecode()
#frame()             #module()           #ver()
```

`#mty.rawi(⟨lang⟩: none)[⟨code⟩]` ⟶

Shows ⟨code⟩ as inline `raw` text (with ⟨block⟩: `false`).

- `#mty.rawi("some inline code")` ⟶ some inline code

Argument

⟨code⟩

String content to be displayed as `raw`.

Argument

⟨lang⟩: none

Optional language for syntax highlighting.

`#mty.rawc(⟨color⟩)[⟨code⟩]` ⟶

Shows ⟨code⟩ as inline `raw` text (with ⟨block⟩: `false`) and with the given ⟨color⟩. This supports no language argument, since ⟨code⟩ will have a uniform color.

- `#mty.rawc(purple, "some inline code")` ⟶ some inline code

Argument

⟨color⟩ color

Color for the `raw` text.

Argument

⟨code⟩

String content to be displayed as `raw`.

`#mty.cblock(⟨width⟩: 90%, ..⟨args⟩)[⟨body⟩]` ⟶

A `block` that is centered in its parent container.

```
#mty.cblock(width:50%)[#lorem(40)]
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere.

Argument

⟨width⟩: 90% `length`

Width of the block.

Argument

..⟨args⟩ `any`

Argeuments for `block`.

Argument

⟨body⟩

Content of the block

#mty.frame(..⟨args⟩) ⟶

Create a frame around some content.

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

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

Some lorem text

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

..args (any): Arguments for SHOWYBOX.

```
#mty.alert(
  ⟨color⟩: rgb("#0074d9"),
  ⟨width⟩: 100%,
  ⟨size⟩: 0.88em,
  ..⟨style⟩,
  ⟨body⟩
)
```

An alert box to highlight some content.

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

```
#mty.alert(color:purple, width:4cm)[#lorem(10)]
```

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

`#mty.marginnote(⟨pos⟩: left, ⟨gutter⟩: 0.5em, ⟨dy⟩: -1pt)[⟨body⟩]` ⟶

Places a note in the margin of the page.

Argument

`⟨pos⟩: left` `alignment`

Either `left` or `right`.

Argument

`⟨gutter⟩: 0.5em` `length`

Spacing between note and textarea.

Argument

`⟨dy⟩: -1pt` `length`

How much to shift the note up or down.

Argument

`⟨body⟩`

Content of the note.

`#mty.idx-term(⟨term⟩)` ⟶

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

Argument

`⟨term⟩`

The term to sanitize.

`#mty.idx(⟨term⟩: none, ⟨hide⟩: false, ⟨kind⟩: "term")[⟨body⟩]` ⟶

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⟩: none`

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

Argument

⟨hide⟩: false bool

If true, no content is shown on the page.

Argument

⟨kind⟩: "term"

A category for ths term.

Argument

⟨body⟩

The term or label for the entry.

#mty.make-index(⟨kind⟩: none, ⟨cols⟩: 3, ⟨headings⟩: (..) => .., ⟨entries⟩: (..) => ..) → _

Creates an index from previously set entries.

Argument

⟨kind⟩: none

An optional kind of entries to show.

Argument

⟨cols⟩: 3 int

Number of columns to show the entries in.

Argument

⟨headings⟩: (..) => .. function

Function to generate headings in the index. Gets the letter for the new section as an argument: (_) => _

Argument

⟨entries⟩: (..) => .. function

A function to format index entries. Gets the index term, the label and the location for the entry: (_, _, location) => _

#mty.ver(..⟨args⟩)

Generate a version number from a version string or array. The function takes a variable number of arguments and builds a version string in *semver* format:

- #mty.ver(0,1,1) ⟶ **0.1.1**
- #mty.ver(0,1,"beta-1") ⟶ **0.1.beta-1**
- #mty.ver("1.0.2") ⟶ **1.0.2**

#mty.name(⟨name⟩, ⟨last⟩: none) → _

Highlight human names (with first- and lastnames).

- #mty.name("Jonas Neugebauer") ⟶ Jonas Neugebauer
- #mty.name("J.", last:"Neugebauer") ⟶ J. Neugebauer

Argument

⟨name⟩

First or full name.

Argument

⟨last⟩: none

Optional last name.

#mty.author(⟨info⟩) ⟶

Show author information.

- #mty.author("Jonas Neugebauer") ⟶ Jonas Neugebauer
- #mty.author((name:"Jonas Neugebauer")) ⟶ Jonas Neugebauer
- #mty.author((name:"Jonas Neugebauer", email:"github@neugebauer.cc")) ⟶ Jonas Neugebauer <github@neugebauer.cc>

Argument

⟨info⟩ | dictionary

Either a string with an author name or a dictionary with the name and email keys.

#mty.date(⟨d⟩, ⟨format⟩: "[year]-[month]-[day]") ⟶

Show a date with a given format.

- #mty.date("2023-09-25") ⟶ 2023-09-25
- #mty.date(datetime.today()) ⟶ 2024-04-23

Argument

⟨d⟩ |

Either a date as a string or .

Argument

⟨format⟩: "[year]-[month]-[day]"

An optional format string.

#mty.package(⟨name⟩)

Show a package name.

- #mty.package("codelst") ⟶ codelst

Argument

⟨name⟩

Name of the package.

#mty.module(⟨name⟩)

Show a module name.

- #mty.module("util") ⟶ util

Argument

⟨name⟩

Name of the module.

`#mty.footlink(⟨url⟩, ⟨label⟩)` ⟶

Creates a `link` with an attached footnote showing the ⟨`url`⟩.

- `#mty.footlink("https://neugebauer.cc", "neugebauer.cc")` ⟶ neugebauer.cc[6]

Argument

⟨`url`⟩

The url for the link and the footnote.

Argument

⟨`label`⟩

The label for the link.

`#mty.gitlink(⟨repo⟩)` ⟶

Creates a `link` to a GitHub repository given in the format `user/repository` and shows the url in a footnote.

- `#mty.gitlink("jneug/typst-mantys")` ⟶ jneug/typst-mantys[7]

Argument

⟨`repo`⟩

Identifier of the repository.

`#mty.pkglink(⟨name⟩, ⟨version⟩, ⟨namespace⟩: "preview")` ⟶

Creates a `link` to a Typst package in the Typst package repository at typst/packages.

- `#mty.pkglink("codelst", (2,0,0))` ⟶ CODELST:2.0.0[8]

Argument

⟨`name`⟩

Name of the package.

Argument

⟨`version`⟩

Version string of the package as an array of ints (e.g. (0,0,1)).

Argument

⟨`namespace`⟩: "preview"

The namespace to use. Defaults to `preview`.

`#mty.add-mark(⟨mark⟩)[⟨elem⟩]`

Adds a label to a content element.

Argument

⟨`mark`⟩ | `label`

A label to attach to the content.

[6] https://neugebauer.cc
[7] https://github.com/jneug/typst-mantys
[8] https://github.com/typst/packages/tree/main/packages/preview/codelst/2.0.0

Argument

⟨elem⟩

Content to mark with the label.

#mty.has-mark(⟨mark⟩)[⟨elem⟩]

Tests if ⟨value⟩ has a certain label attached.

- #mty.has-mark(<x>, mty.add-mark(<x>, raw("some code"))) ⟶ true
- #mty.has-mark(<x>, [#raw("some code")<x>]) ⟶ true
- #mty.has-mark(<y>, [#raw("some code")<x>]) ⟶ false

Argument

⟨mark⟩ label

A label to check for.

Argument

⟨elem⟩

The content to test.

#mty.place-marker(⟨name⟩)

Places an invisible marker in the content that can be modified with a #show rule.

```
This marker not replaced: #mty.place-marker("foo1")

#show mty.marker("foo1"): "Hello, World!"
Here be a marker: #mty.place-marker("foo1")\
Here be a marker, too: #mty.place-marker("foo2")
```

This marker not replaced:

Here be a marker: Hello, World!
Here be a marker, too:

Argument

⟨name⟩

Name of the marker to be referenced later.

#mty.marker(⟨name⟩)

Creates a selector for a marker placed via #place-marker().

```
#show mty.marker("foo1"): "Hello, World!"
Here be a marker: #mty.place-marker("foo1")\
Here be a marker, too: #mty.place-marker("foo2")
```

Here be a marker: Hello, World!
Here be a marker, too:

Argument

⟨name⟩

Name of the marker to be referenced.

#mty.sourcecode(..⟨args⟩) ⟶

Shows sourcecode in a frame.

> Uses CODELST to render the code.

See Section II.2.3 for more information on sourcecode and examples.

Argument

..⟨args⟩ any

Argumente für #codelst.sourcecode()

```
#mty.code-example(
  ⟨side-by-side⟩: false,
  ⟨scope⟩: (:),
  ⟨mode⟩: "markup",
  ⟨breakable⟩: false,
  ..⟨args⟩
)[⟨example-code⟩]
```

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

See section II.2.3 for more information on sourcecode and examples.

Argument

⟨side-by-side⟩: false bool

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

⟨mode⟩: "markup"

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

Argument

⟨breakable⟩: false bool

If the frame may brake over multiple pages.

Argument

⟨example-code⟩

A raw block of Typst code.

Argument

..⟨args⟩

> 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.

`#primary`

Highlights some content with the primary color.

`#secondary`

Highlights some content with the secondary color.

`#mark-arg`

Mark content as an argument.

`#is-arg`

Test if ⟨`value`⟩ is an argument created with `#arg()`.

`#not-is-arg`

Test if ⟨`value`⟩ is no argument created with `#arg()`.

`#mark-body`

Mark content as a body argument.

`#is-body`

Test if ⟨`value`⟩ is a body argument created with `#barg()`.

`#not-is-body`

Test if ⟨`value`⟩ is no body argument created with `#barg()`.

`#mark-sink`

Mark content as an argument sink.

`#is-sink`

Test if ⟨`value`⟩ is an argument sink created with `#sarg()`.

`#not-is-sink`

Test if ⟨`value`⟩ is no argument sink created with `#sarg()`.

`#mark-choices`

Mark content as a choices argument.

`#is-choices`

Test if ⟨`value`⟩ is a choice argument created with `#choices()`.

`#not-is-choices`

Test if ⟨`value`⟩ is no choice argument created with `#choices()`.

`#mark-func`

Mark content as a function argument.

`#is-func`

Test if ⟨`value`⟩ is a function argument created with `#func()`.

`#not-is-func`

Test if ⟨`value`⟩ is no function argument created with `#func()`.

`#mark-lambda`

Mark content as a lambda argument.

`#is-lambda`

Test if ⟨value⟩ is a lambda argument created with `#lambda()`.

`#not-is-lambda`

Test if ⟨value⟩ is no lambda argument created with `#lambda()`.

`#lineref`

Show a reference to a labeled line in a sourcecode.

> Uses CODELST to show the reference.

II.2.8. Tidy template

```
#get-type-color()          #show-outline()          #show-parameter-list()
#show-function()           #show-parameter-block()  #show-type()
```

#mty-tidy.get-type-color(⟨type⟩) ⟶ color

Returns the color for a specific type.

Argument

⟨type⟩ any

Type to get the color for.

#mty-tidy.show-outline(⟨module-doc⟩, ⟨style-args⟩: none) ⟶

Shows the outline of a module (list pf functions).

Argument

⟨module-doc⟩ dictionary

Parsed module data.

Argument

⟨style-args⟩: none dictionary

Styling arguments.

#mty-tidy.show-type(⟨type⟩, ⟨style-args⟩: (:))

Create beautiful, colored type box

#mty-tidy.show-parameter-list(⟨fn⟩, ⟨display-type-function⟩) ⟶

Show a list of arguments for a given function.

Argument

⟨fn⟩ dictionary

Parsed function dictionary.

```
#mty-tidy.show-parameter-block(
  ⟨name⟩,
  ⟨types⟩,
  ⟨style-args⟩,
```

```
  ⟨show-default⟩: false,
  ⟨default⟩: none
)[⟨content⟩] ⟶
```

Create a parameter description block, containing name, type, description and optionally the default value. Passes the parameter information to `#argument()`.

Argument

⟨name⟩

Name of the function.

Argument

⟨types⟩ `array`

Array of possible types.

Argument

⟨content⟩

Description of the argument.

Argument

⟨style-args⟩ `dictionary`

TIDY configuration.

```
#mty-tidy.show-function(⟨fn⟩, ⟨style-args⟩, ⟨tidy⟩: none, ⟨extract-headings⟩: 2)
```

Show function

```
#fn-color
```

Color to highlight function names in

Part III.
Index