The Mantys Package

MANuals for **TY**p**S**t packages and templates

v0.0.1 2023-07-18

A Typst template to create consistens and readable manuals for pakcages and templates.

Jonas Neugebauer

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

Weit hinten, hinter den Wortbergen, fern der Länder Vokalien und Konsonantien leben die Blindtexte. Abgeschieden wohnen sie in Buchstabhausen an der Küste des Semantik, eines großen Sprachozeans. Ein kleines Bächlein namens Duden fließt durch ihren Ort und versorgt sie mit den nötigen Regelialien. Es ist ein paradiesmatisches Land, in dem einem gebratene Satzteile in den Mund fliegen. Nicht einmal von der allmächtigen Interpunktion werden die Blindtexte beherrscht – ein geradezu unorthographisches Leben.

Table of contents

Part I.
About

MANTYS is a **TYPST** package.

MANTYS is in active development and its functionality is subject to change.

Contributions are welcome.

Part II. Usage

II.1. Using Mantys

II.1.1. Loading as a package

Currently the package needs to be installed into the local package repository.

Either download the current release from GitHub[1] and unpack the archive into your system dependent local repository folder[2] or clone it directly:

```
git clone https://github.com/jneug/typst-mantys.git mantys-0.0.1
```

After installing the package just import it inside your `typ` file:

```
#import "@local/mantys:0.0.1": *
```

II.1.2. Loading as a module

To load Mantys into a single project as a module download the necessary files and place them inside the project directory. The required files are `mantys.typ` and `mty.typ`.

Import the module into your manual file:

```
#import "mantys.typ": *
```

II.1.3. Initialising 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(
    ...
)
```

[1] https://github.com/jneug/typst-typopts/releases/latest
[2] https://github.com/typst/packages#local-packages

#mantys(name: none, title: none, subtitle: none, info: none, authors: (), url: none, version: none, date: none, abstract: sequence(children: ()), titlepage: titlepage, example-imports: (:), ..args)[body]

`titlepage: titlepage`	A function of nine arguments to render a titlepage for the manual. Refer to command `#titlepage()` on page 11 for details.	`function`
`example-imports: (:)`	Default imports for code examples. Each entry should have the full package identifier as a key and the imports as a value. If the package should be imported es a whole, the value should be `""`.	`dictionary`

```
example-imports: (
  "@local/mantys:0.0.1": "*",
  "@preview/tablex:0.0.1": "",
  "@preview/cetz:0.0.1": "canvas"
)
```

For further details refer to command `#example()` on page 8.

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

All uppercase occurences of `name` will be highlighted as a packagename. For example MANTYS will appear as MANTYS.

II.2. Available commands

II.2.1. Describing arguments and values

#meta(name) -> content

Used to highlight argument names. `#meta[variable]` ⟶ `variable`

#value(variable) -> content

Used to display the value of a variable. The command will highlight the value depending on the type.

- `#value[name]` ⟶ `[name]`
- `#value("name")` ⟶ `"name"`
- `#value((name: "value"))` ⟶ `(name: "value")`
- `#value(range(4))` ⟶ `(0, 1, 2, 3)`

#arg(name) -> content

Renders 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"`

#sarg(name) -> array

Renders an argument sink. `#sarg[args]` ⟶ `..args`

`#barg(name) -> content`

Renders a body argument. `#barg[body]` ⟶ `[body]`

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

`#args(..args) -> content`

Creates an array of all its arguments rendered either by `#arg()` or `#barg()`. All values of type `content` will be passed to `#barg()` and everything else to `#arg()`.

This command is intendend to be unpacked as the arguments to one of `#cmd()` or `#command()`.

```
#cmd("my-command", ..args("arg1", arg2: false, [body]))
```

`#my-command(arg1, arg2: false)[body]`

`#dtype(t, fnote: false, parse-type: false) -> string`

Shows the (data-)type of `t` and a link to the Typst documentation of that type.

`fnote: true` will show the reference link in a footnote (useful for print versions of the manual).

The type is determined by passing `t` to `type`. If `t` is a string however, it is assumed to already be a type name. For example `"fraction"` will give the type `fraction`. Setting `parse-type: true` will prevent this and always call `type` on `t`.

- `#dtype(false)` ⟶ `boolean`
- `#dtype(1%)` ⟶ `ratio`
- `#dtype(left)` ⟶ `alignment`
- `#dtype([some content], fnote:true)` ⟶ `content`[3]
- `#dtype("dictionary")` ⟶ `dictionary`
- `#dtype("dictionary", parse-type:true)` ⟶ `string`

`#dtypes(..types, sep: box(inset: (left: 1pt, right: 1pt), body: [|])) -> content`

Will produce a list of types from the provided arguments. Each value is passed to `#dtype()` and the results joined by `sep`.

- `#dtypes(false, 1cm, "array", [world])` ⟶ `boolean` | `length` | `array` | `content`
- `#dtypes(false, 1cm, "array", [world], sep: " or ")` ⟶ `boolean` or `length` or `array` or `content`

`#choices(default: "__none__", ..values)`

Creates a list of possible values for an argument.

[3]https://typst.app/docs/reference/types/content

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(..range(5))` ⟶ `0|1|2|3|4`
- `#choices(..range(5), default:3)` ⟶ `0|1|2|3|4`
- `#choices(..range(5), default:5)` ⟶ `5|0|1|2|3|4`

`#opt(name, ..args)[body]`

Renders the option `name` and adds an entry to the index.

- `#opt[example-imports]` ⟶ `example-imports`

`#opt-(name, ..args)[body]`

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

II.2.2. Describing commands

`#cmd(name, ..args)[body]`

Renders the command `name` with arguments and creates an entry in the command index.

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

All positional arguments will be rendered first, then named arguemnts 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]`

`#cmd-(name, ..args)[body]`

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

`#var(name, default: none)`

`#var-(name, default: none)`

`#command(name, ..args)[body]`

Shows

`#argument(name, type)`

`#variable(name, ..args)[body]`

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.

2.2.3 Available commands

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 is subject to its limitations. Each `eval` call is run in a local scope and does not have access to previously imported commands. To use your packages commands, you have to import it as a package:

~~~
#example[```
#import "@local/mantys:0.0.1": dtype

#dtype(false)
```]
~~~

```
#import "@local/mantys:0.0.1": dtype

#dtype(false)
```

`boolean`

You can only import packages and not local files.

2.2.3 Available commands

To automatically add imports to every example code, you can set the option `example-imports` at the initial call to `#mantys()`. For example this manual was compiled with `example-imports: ("@local/mantys:0.0.1": "*")`. This imports the MANTYS commands into all example code, without explicitly importing it in the code.

~~~
#example[```
#mty.value(false)
```]
~~~

```
#mty.value(false)
```

```
false
```

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

To use fenced code blocks in your example, pass the code as a string to `raw` like this:

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

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

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

`#example(side-by-side: false, imports: (:))[example-code][result]`

`example-code`	A block of raw code representing the example TYPST code.	`content`
`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.	`boolean`

`imports: (:)` A dictionary of package imports that should be added to the evaluated code. `dictionary`

`result` The result of the example code. Usually the same code as `example-code` but without the raw markup. `content`

> `result` is optional and will be omitted in most cases!

Sets `[example-code]` as a raw block with `lang: "typc"` 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.

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, `#example()` accepts an optional second positional argument `[result]`. If provided, `[example-code]` is not evaluated and `[result]` ist used instead.

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

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

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

`#side-by-side()[example-code][result]`

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

`#sourcecode(code)`

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

`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.	`content`
`title: none`	A title to show above the code in a titlebar.	`content`
`file: none`	A filename to show above the code in a titlebar.	`content`

`#sourcecode()` will render a raw block with linenumbers and proper tab indentions using `#mty.sourcecode()` 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*

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

The sourcecode set with this command is set line by line in a `grid` and will not be selectable as a whole without including the line numbers. If you want the code to be selectable (to allow copy&paste) you should set `linenos: false`.

II.2.4. Other commands

`#pkg()`

Shows a package name:

```
#pkg[tablex]

#mty.package[tablex]
```

TABLEX

TABLEX

II.2.5. Templating

`#titlepage(name, title, subtitle, info, authors, url, version, date, abstract)`

II.2.6. 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 polution as minimal as possible to prevent name collisons 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.

`#colors` `dictionary`

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.

`mty` » **`#type(variable) -> string`**

Alias for the buildin `type` command.

`mty` » **`#kv(key, value) -> dictionary`**

`key` Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do. `none`

Creates a `dictionary` containing the given `key`/`value`-pair. Useful for using `map` on the pairs of a dictionary:

```
#let dict = (a: 1, b: 2, c: 3)
dict.pairs().map(p => kv(..p)).map( ... )
```

mty » `#txt(variable) -> string`

Extracts the text content of `variable` as a `string`. The command attempts to extract as much text as possible by looking at possible children of a content element.

mty » `#rawi(lang: none)[code] -> content`

Inline `raw` content with an optional language fpr highlighting.

mty » `#rawc(color)[code] -> content`

Colored inline `raw` content. This supports no language argument, since `code` will have a uniform `color`.

mty » `#primary()`

mty » `#secondary()`

mty » `#cblock(width: 90%, ..block-args)[body] -> content`

Sets `body` inside a centered `block` with the given `width`. Any further arguments will be passed to the `block` command.

mty » `#box(header: none, footer: none, invert-headers: true, stroke-color: rgb("#239dac"), bg-color: rgb("#ffffff"), width: 100%, padding: 8pt, radius: 4pt)[body] -> content`

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 malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

mty » `#alert(color: rgb("#0074d9"), icon: none, title: none, width: 90%, size: 0.9em)[body] -> content`

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 malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

mty » `#marginnote(pos: left, margin: 0.5em, dy: 0pt)[body] -> content`

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.

mty » `#sourcecode(fill: rgb("#ffffff"), border: none, tab-indent: 4, gobble: auto, linenos: true, gutter: 10pt)[body] -> content`

2.2.6 Available commands

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.

`mty` » `#ver(major, minor, patch) -> content`

```
#mty.ver(0, 0, 1)
```

0.0.1

`mty` » `#name(name, last: none) -> content`

```
- #mty.name("Jonas Neugebauer")
- #mty.name("Jonas van Neugebauer")
- #mty.name("Jonas van", last:"Neugebauer")
- #mty.name("Jonas", last:"van Neugebauer")
```

`mty` » `#author(info) -> content`

```
- #mty.author("Jonas Neugebauer")
- #mty.author(
(name: "Jonas van Neugebauer")
)
- #mty.author((
name: "Jonas van Neugebauer",
email: "jonas@neugebauer.cc"
))
```

`mty` » `#date(d) -> content`

```
- #mty.date("2023-07-15")
- #mty.date(datetime(year:
2023, month:7, day:15))
- #mty.date(datetime.today())
- #mty.date(datetime.today(),
format:"[day].[month].[year]")
```

- 2023-07-15
- 2023-07-15
- 2023-07-18
- 18.07.2023

`mty` » `#package(name) -> content`

```
- #mty.package("Mantys")
- #mty.package("typopts")
```

`mty` » `#module(name) -> content`

```
- #mty.module("mty")
- #mty.module("emoji")
```

- `mty`
- `emoji`

`mty` » `#value(variable) -> content`

Returns the value of `variable` as content.

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

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

mty » `#default(value) -> content`

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

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

- `"default-value"`
- `true`
- `1|2|3|4`

II.2.6.1. Argument filters

mty » `#is-string(value)`

Checks if `value` is a `string`.

mty » `#is-content(value)`

Checks if `value` is `content`.

mty » `#is-choices(value)`

Checks if `value` is a choices value created with `#choices()`.

mty » `#is-body(value)`

Checks if `value` is a body argument created with `#barg()`.

mty » `#not-is-body(value)`

Negation of `#is-body()`.

mty » `#not-is-choices(value)`

Negation of `#is-choices()`.

Part III.
Index