typst-doc

A doctor for your documentation

June 7, 2023

Mc-Zen

Abstract

typst-doc is a package that generates documentation directly in Typst for your Typst modules. It parses docstring comments similar to javadoc and co. and can be used to easily build a reference section for each module.

```
true rgb("#0074d9")
```

Introduction

Feed **typst-doc** your in-code documented source files and get beautiful documentation of all your functions printed out. Enjoy features like type annotations, links to other documented functions and arbitrary formatting within function and parameter descriptions. Let's get started.

You can document your functions similar to javadoc by prepending a block of `///` comments.

Example

```
/// This function does something. It always returns true.
///
/// We can have *markdown* and
/// even $m^a t_h$ here. A list? No problem:
///  - Item one
///  - Item two
///
///
/// - param1 (string): This is param1.
/// - param2 (content, length): This is param2.
///           Yes, it really is.
/// -> boolean
#let something(param1, param2: 3pt) = { return true }
```

Each line needs to start with three slashes `///` (whitespace is allowed at the beginning of the line). Parameters of the function can be documented by listing them with `-` as showed above. The possible types for each parameter are given in parantheses and after a colon `:`, the parameter description follows. An optional return type can be annotated by ending with a line that contains `->` and the return type.

In front of the arguments, a function description can be put. Both function and parameter descriptions may span multiple lines and can contain any Typst code.

Calling `parse-code()` on the snippet above will read out the documentation of the given string. We can now invoke `show-module()` on the result to obtain the following result:

something

This function does something. It always returns true.

We can have **markdown** and even $m^a t_h$ here. A list? No problem:
- Item one
- Item two

Parameters

```
something(
  param1: string,
  param2: content length
) -> boolean
```

param1 `string`

This is param1.

param2 `content` or `length`

This is param2. Yes, it really is.

Default: `3pt`

Cool, he?

Usually, you'll want to parse a file, so you can instead just call `parse-module("myfile.typ")` and then `show-module()` the result.

There is another little nice feature: in the docstring, you can reference other functions by writing `@@other-function()`. This will automatically create a link that when clicked will lead you to the documentation of that function.

Of course, everything happens instantaneously, so you can see the live result while writing the docs for your package. Keep your code documented!

Let us now "self-document" this package:

typst-doc

parse-code

Parse the docstrings of Typst code. This function returns a dictionary with the keys
- `functions`: A list of function documentations as dictionaries.
- `label-prefix`: The prefix for internal labels and references.

The function documentation dictionaries contain the keys
- `name`: The function name.
- `description`: The functions docstring description.
- `args`: A dictionary of info objects for each fucntion argument.

These again are dictionaries with the keys
- `description` (optional): The description for the argument.
- `types` (optional): A list of accepted argument types.
- `default` (optional): Default value for this argument.

See `show-module()` for outputting the results of this function.

Parameters

```
parse-code(
  content: string,
  label-prefix: none string
)
```

content `string`

Typst code to parse for docs.

label-prefix `none` or `string`

Prefix for internally created labels and references. Use this to avoid name conflicts with labels.

Default: `none`

parse-module

Parse the docstrings of a typst module. This function returns a dictionary with the keys
- `name`: The module name as a string.
- `functions`: A list of function documentations as dictionaries.

The label prefix will automatically be the name of the module. See `parse-code()` for more details.

Parameters

```
parse-module(
  filename: string,
  name: string none
)
```

filename `string`

Filename for the `.typ` file to analyze for docstrings.

name `string` or `none`

The name for the module. If not given, the module name will be derived form the filename.

Default: `none`

show-module

Show given module in the style of the Typst online documentation. This displays all (documented) functions in the module sorted alphabetically.

Parameters

```
show-module(
  module-doc: dictionary,
  first-heading-level: integer,
  show-module-name: boolean,
  type-colors: dictionary,
  allow-breaking: boolean,
  omit-empty-param-descriptions: boolean
) -> content
```

module-doc `dictionary`

Module documentation information as returned by `parse-module()`.

first-heading-level `integer`

Level for the module heading. Function names are created as second-level headings and the "Parameters" heading is two levels below the first heading level.

Default: `2`

show-module-name `boolean`

Whether to output the name of the module.

Default: `true`

type-colors `dictionary`

Colors to use for each type. Colors for missing types default to gray (`"#eff0f3"`).

Default: `type-colors`

allow-breaking `boolean`

Whether to allow breaking of parameter description blocks

Default: `true`

omit-empty-param-descriptions `boolean`

Whether to omit description blocks for Parameters with empty description.

Default: `true`

parse-argument-list

Parse an argument list from source code at given position. This function returns `none`, if the argument list is not properly closed. Otherwise, a dictionary is returned with an entry for each parsed argument name. The values are dictionaries that may be empty or have an entry for `default` containing a string with the parsed default value for this argument.

Example

Let's take some source code:

```
#let func(p1, p2: 3pt, p3: (), p4: (entries: ())) = {...}
```

Here, we would call `parse-argument-list(source-code, 9)` and retrieve

```
(
  p0: (:),
  p1: (default: "3pt"),
  p2: (default: "()"),
  p4: (default: "(entries: ())"),
)
```

Parameters

```
parse-argument-list(
  module-content: string,
  index: integer
) -> none dictionary
```

module-content `string`

Source code.

index `integer`

Index where the argument list starts. This index should point to the character **next** to the function name, i.e. to the opening brace `(` of the argument list if there is one (note, that function aliases for example produced by `myfunc.where(arg1: 3)` do not have an argument list).