Tidy Types

v0.0.1

https://github.com/SillyFreak/typst-packages/tree/main/tidy-types

Clemens Koza

Abstract

Helpers for writing complex types in tidy documentation and rendering types like tidy outside of tidy-generated signatures.

Contents

I Introduction

This package contains helpers for documenting the types of values with tidy, just as tidy itself shows them in function signatures. For example, this lets you write documentation such as this:

> The result of `range(5).enumerate()` is a `((integer, integer), ...)`.

To do so, it produces raw blocks with language `"tidy-type"`, which can be then styled using a show rule as follows:

```typ
#import "@preview/tidy:0.2.0"
#import "@preview/tidy-types:0.1.0" as tt

// using the default style with default colors
#let style = tidy.styles.default
#show raw.where(lang: tt.lang): it => {
  style.show-type(it.text, style-args: (colors: style.colors))
}

// using the minimal style
#let style = tidy.styles.minimal
#show raw.where(lang: tt.lang): it => style.show-type(it.text)
```

For example, the raw block ```` ```tidy-type content``` ```` would be displayed as `content` in the default style, or `content` in the minimal style. This can be more easily written using the basic function of this package, `tt.type()`: `#tt.type("content")`. In practice, you will not need to use this function directly but instead use the utility functions and variables built on top.

II Built-in Typst types

There are constants for all the built-in types that Typst provides. Note how two of them are prefixed with `"t-"` as their names are keywords – `none` and `auto` – and another because its name is taken by a tidy types function – `type`:

`t-none`	`none`	`bool`	`boolean`	`int`	`integer`
`float`	`float`	`str`	`string`	`bytes`	`bytes`
`array`	`array`	`dictionary`	`dictionary`	`t-type`	`type`
`function`	`function`	`t-auto`	`auto`	`datetime`	`datetime`
`duration`	`duration`	`regex`	`regex`	`version`	`version`
`content`	`content`	`symbol`	`symbol`	`length`	`length`
`ratio`	`ratio`	`relative`	`relative length`	`fraction`	`fraction`
`angle`	`angle`	`color`	`color`	`stroke`	`stroke`
`alignment`	`alignment`	`location`	`location`	`styles`	`styles`
`label`	`label`	`selector`	`selector`	`module`	`module`
`plugin`	`plugin`	`arguments`	`arguments`		

III Module reference

III.a `tidy-types`

type(text: `string`) -> `content`

Wraps the given string, a type name, into a `raw` element with the language `"tidy-type"`. By itself, that doesn't do anything, but it allows styling that text using a `show` rule; see the introduction.

```
1  tt.type("foo")                    typc
```

`foo`

Parameters:

`text` (`string`) – the type name

arr(element: `content`) -> `content`

A function for rendering an array type including element type information:

```
1  tt.arr(tt.int)                    typc
```

(`integer`, ...)

This representation uses the array spread syntax to convey that there may be any number of `integer` elements in the array.

The name of this function is `arr` because `tt.array` (`array`) exists already.

Parameters:

`element` (`content`) – the element type of the array

dict(value: `content`) -> `content`

A function for rendering a dictionary type including element type information:

```
1  tt.dict(tt.int)                   typc
```

(`string`: `integer`, ...)

This representation uses the implicit `string` key type to convey that there may be any number of mappings in the dictionary.

The name of this function is `dict` because `tt.dictionary` (`dictionary`) exists already.

Parameters:

`value` (`content`) – the value type of the dictionary

tuple(..elements: `content`) -> `content`

A function for rendering an array type containing exactly the given elements:

```
1 tt.tuple(tt.str, tt.int)          typc
```

(`string`, `integer`)

Parameters:

`..elements` (`content`) – the tuple element types given as positional parameters

object(..pairs: `content`) -> `content`

A function for rendering a dictionary type containing exactly the given pairs:

```
1 tt.object(a: tt.str, b: tt.int)   typc
```

(`a:` `string`, `b:` `integer`)

Parameters:

`..pairs` (`content`) – the object attribute name/type pairs given as named parameters

func(..args: `content`) -> `content`

A function for rendering a function type taking the given parameters and having the given return type:

```
1 tt.func(                          typc
2   tt.str, opt: tt.bool,
3   tt.int)
```

(`string`, `opt:` `boolean`) ⟶ `integer`

Note that the relative order of positional and named parameters is not preserved; all named parameters come after all positional parameters. It makes sense to, as a convention, put the result type after any named parameters. There is one exception to this rule though: if the last positional parameter is `sink`, it will be put after any named arguments:

```
1 tt.func(                          typc
2   tt.str, opt: tt.bool, tt.sink,
3   tt.int)
```

(`string`, `opt:` `boolean`, ...) ⟶ `integer`

The name of this function is `func` because `tt.function` (`function`) exists already.

Parameters:

`..args` (`content`) – the function parameter types and return type (last positional argument) of the function

either(..options: `content`) -> `content`

A function for rendering a choice between the given types:

```
1 tt.either(tt.str, tt.int)                    typc
```

`string` | `integer`

Parameters:

`..options` (`content`) – the possible types given as positional parameters

optional(type: `content`) -> `content`

A function for rendering a parameter/element that may be omitted

```
1 tt.optional(tt.str)                          typc
2 parbreak()
3 tt.func(tt.optional(tt.str), tt.str)
```

`string` ?

(`string` ?) ⟶ `string`

Parameters:

`type` (`content`) – the tuple element types given as positional parameters

group(type: `content`) -> `content`

Surrounds a type with parentheses for grouping

```
1 tt.group(tt.str)                             typc
```

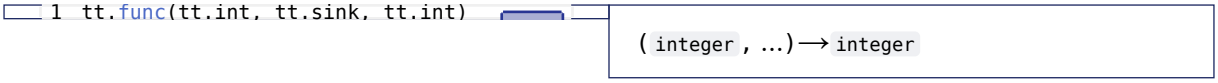
(`string`)

Parameters:

`type` (`content`) – the type to delimit using parentheses

sink `content`

This is not a real type, but it can be used as the last parameter in `func()` to indicate that the function uses an argument sink to take additional positional or named parameters:


1 tt.func(tt.int, tt.sink, tt.int)
(integer, ...)⟶integer