Developer's Toolbox

Manual

---

typst toolbox 0.1.0 MIT

Maycon F. Melo*

Contents

*https://github.com/mayconfmelo

Description

Easily implement frequently-used code through multiple projects. This package was created as a development companion for my other Typst projects: it contains basic code and structures that often appear in multiple projects and must be manually rewritten each time — and therefore are hardly equal and homogeneous among each other, even if they do the exactly same tasks. It do not intends to be a full heavy-loaded general development toolbox, but a compartimentalization of my other projects' shared features.

This manual contains a technical reference for the package, for practical examples of usage refer to the `/tests/` folder content; the `docs/assets/example.typ` file can also be helpful on basic usage.

Main Namespace

```
#import "@preview/toolbox:0.1.0"
```

Date Command

```
#date(..date)
```

Create a `datetime` date from named and/or positional arguments, combined or not. Panics if two values are set for the same thing, like `data.pos().at(0)` and also `data.named().year`.

`..date` arguments *(required)*

`(year, month, day)`

Date data, like year, month, and day. Fallback to the current year, month 1, and day 1 when not set.

Default Command

```
#default(
  when: false,
  value: (:),
  otherwise: (:),
  original,
)
```

Allows to substitute the original defaults for custom ones, allowing to set new defaults that can be easily changed using `set` rules — for example, change default font from *Libertinus Serif* to *Comic Sans*.[1]

`when:` boolean

Test whether the original default is currently being used.

`value:` dictionary any

Custom default, set when the original default is being used.

`otherwise:` dictionary any

Alternative value, set when the original default is not being used.

`original` boolean

Use original defaults instead of the custom ones.

This command is commonly used inside `set` rules, and might require `#context` to access some defaults data for `#default(when)` test.

Content to String Command

```
#content2str(data)
```

[1] Sorry, designers.

Convert simple text content into string.

`data` content

Content data.

Storage Namespace

```
#import "preview/toolbox:0.1.0": storage
```

Set Namespace Command

```
#storage.namespace(name)
```

Set the namespace used as storage. Namespaces allows multiple packages/templates to use `#storage` at the same time, each accessing its own proper space.

`name` string *(required)*

Namespace name.

Add Command

```
#storage.add(
  key,
  value,
  append: false,
  namespace: auto,
)
```

Insert a new entry in the storage.

`key` string

Storage entry name.

`value` ant

Value to be stored.

`append:` boolean

Append new value if entry already exists, otherwise replaces it.

`namespace:` string

Add to the given namespace.

Remove Command

```
#storage.remove(
  key,
  namespace: auto,
)
```

Removes an existing entry from storage.

`key` string

Storage entry name.

`namespace:` string

Remove from the given namespace.

Get Command

```
#storage.get(
  ..args,
  namespace: auto,
)
```

Retrieves a value from storage, or the entire database itself.

`..args` arguments

The first argument is the storage entry and the second one a default value, returned if the storage entry is not found. If no argument is set, the whole storage database is returned.

`namespace:` string

Get from the given namespace.

Final Command

```
#storage.final(
    ..args,
  namespace: auto,
)
```

The final storage state. Returns the whole storage database.

`..args` arguments

The first argument is the storage entry and the second one a default value, returned if the storage entry is not found. If no argument is set, the whole storage database is returned.

`namespace:` string

Final state from the given namespace.

Reset Command

```
#storage.reset(
  data,
  namespace: auto,
)
```

Set a new value for the entire storage database. While it can be of any type, the other storage commands can only be used if it is a `dictionary` value.

`data` dictionary any

New storage database value.

`namespace:` string

Reset the given namespace.

Components Namespace

```
#import "@preview/toolbox: 0.1.0": comp
```

URL Command

```
#comp.url(url, ..data)
```

Creates a paper-friendly link, attached to a footnote containing the URL itself to ensure readability when printed.

`url` string label *(required)*

URL set as link and shown in footnote.

`data.pos()` arguments

The last argument is used as the link name (fallback to the URL itself) and the second-last one is a label set for future referencing.

Package URL Command

```
#comp.pkg(url)
```

Generates paper-friendly links to packages from its URL. The package name is inferred from the last path from the URL slug (like `/path`) or set using curly brackets (like `/{path}/path`).

`url` string

Package URL.

Callout Command

```
#comp.callout(
  icon: "information-circle",
  title: none,
  fill: gray.lighten(85%),
  fill-text: auto,
  body,
)
```

Creates a customizable callout box.

`icon:` string

Icon name, as set by *Heroicons*[2].

`title:` string content none

Set title, if any.

`fill:` color

Set background color.

`fill-text:` color

Set text color.

`body` content string

The callout content.

Get Namespace

```
#import "@preview/toolbox: 0.1.0": get
```

[2] https://heroicons.com/

Null Value

```
#get.null
```

A null value that matches only itself. Useful as substitute for `none` default values, as it is truly unique and not naturally returned by anything in Typst.

Auto Value Changer Command

```
#get.auto-val(
  origin,
  replace,
)
```

Resolve an `auto` value by replacing it by a custom one only when it is `auto`; otherwise the value is not changed.

`origin` auto any

Value to check if it is `auto`.

`replace` any

Value to replace the origin. Ignored when it is not `auto`.

Has Namespace

```
#import "@preview/toolbox: 0.1.0": has
```

Field Command

```
#has.field(data, values)
```

Check if content has a given field.

`data` string

The content itself.

`values` string array of strings

One or more field names.

Key Command

```
#has.key(data, values)
```

Check if dictionary or module has a given key.

`data` string
The dictionary or module itself.

`values` string array of strings
One or more key names.

Value Command

```
#has.key(data, values)
```

Check if dictionary or module has a given value.

`data` string
The dictionary ir module itself.

`values` string array
One or more values.

Item Command

```
#has.key(data, values)
```

Check if array has a given item.

`data` string
The array itself.

`values` string array of strings
One or more field names.

It's Namespace

```
#import "@preview/toolbox: 0.1.0": its
```

None Command

```
#its.none-val(data)
```

Check whether a value is `none`.

data none any
Value to be checked.

Null Command

```
#its.null(data)
```

Check whether a value is `#get.null`.

data any
Value to be checked.

Empty Command

```
#its.empty(data)
```

Check whether a value is empty: `""` or `[]` or `()` or `(:)`.

data any
Value to be checked.

Context Command

```
#its.empty(data)
```

Check whether a value is a `context()`.

data any
Value to be checked.

Sequence Content Command

```
#its.empty(data)
```

Check whether a value is a sequence of contents.

data any
Value to be checked.

Space Content Command

```
#its.empty(data)
```

Check whether a value is a content witu just a space.

data any
  Value to be checked.

Space Content Command

```
#its.empty(data)
```

Check whether a value function is one of the given ones.

data any
  Value to be checked.

values function array of functions
  One or more functions.

Type Command

```
#its.empty(data)
```

Check whether a value type is one of the given ones.

data any
  Value to be checked.

values type array of types
  One or more types.

Copyright

The logo was obtained from Flaticon website.

The Fluent support is a fork of a *linguify*[3] feature, and all the overall project concept is heavily inspired in this great package.

[3] https://typst.app/universe/package/linguify