The typopts Package

Typst **opt**ion**s** management

v0.0.3 2023-07-05

A **Typst** package to conveniently handle options and arguments.

Jonas Neugebauer

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

typopts is a **Typst** package with the intend to make handling options for packages and templates as easy as possible.

It provides functionality to load options from various sources, merge them together and make them accessible throughout the document.

Table of contents

Part I.
About

TYPOPTS was inspired by *LaTeX* packages like `pgfkeys`[1] and modules like `argparse`[2] in *Python*.

[1] https://ctan.org/pkg/pgfkeys
[2] https://docs.python.org/3/library/argparse.html

Part II.
Usage

II.1. Use as a module

To use TYPOPTS as a module for one project, get the file `options.typ` from the repository and save it in your project folder.

Import the module as usual:

```
#import "options.typ"
```

To use the `state` module do the same with the file `states.typ`:

```
#import "states.typ"
```

II.2. Use as a package

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

Either download the current release from GitHub[3] and unpack the archive into yout system dependent local repository folder or clone it directly:

```cmd
git clone https://github.com/jneug/typst-typopts.git typopts-0.0.3
```

In either case make sure the files are placed in a folder with the correct version number: `typopts-0.0.3`

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

```
#import "@local/typopts:0.0.3": options
```

II.3. Available functions

TYPOPTS provides several commands in three categories: Options access, argument parsing and configuration loading.

II.3.1. Accessing options

Options are simply key/value-pairs that are stored in a global state variable. This allows them to be access anywhere, even outside a main template function.

[3] https://github.com/jneug/typst-typopts/releases/latest

Namespaces are a way to create logical groups of options. All commands handling options accept an `ns` argument to specify the namespace. Alternatively the namespace may be defined in dot-notation with thr option name.

`#options.get("colors.red")` and `#options.get("red", ns:"colors")` will both retrieve the option `red` from the namespace `colors`. The argument takes precedence though and will prevent any namespaces before a dot to take effect. This means `#options.get("colors.red", ns:"colors")` will look for an option `colors.red` in the namespace `colors`.

```
#get(name, func, default: none, ns: none, final: false, loc: none)
```

`name` Name of the option. `string`

`func` Function to pass the value to. `any => none`

`default: none` Default value, if an option `name` does not exist. `any`

`final: false` If set to `true`, the options final value is retrieved, otherwise the local value. `boolean`

`loc: none` A `location` to use for retrieving the value. `location`

`ns: none` The namespace to look for the value in. `string`

Retrieves the value for the option by the given `name` and passes it to `func`, which is a function of one argument.

If no option `name` exists, the given `default` is passed on.

If `final: true`, the final value for the option is retrieved, otherwise the current value. If `loc` is given, the call is not wrapped inside a `locate` call and the given `location` is used.

```
#update(name, value, ns: none)
```

Sets the option `name` to `value`.

```
#update-all(values, ns: none)
```

Updates all key/value-pairs in the dictionary `values`. Each key is used as the option name.

```
#remove(name, ns: none)
```

Remove the option `name`.

```
#display(name, format: (v) => v, default: none, final: false, ns: none)
```

Show the value of option `name` formated with the function `format`.

If no such option exists, `default` is used instead.

`format: (v) => v` A function of one argument, that receives the optons value and transforms it into the content to be set. `(any) => content`

II.4. Parsing arguments

```
#add-argument(name, type: ("string", "content"), required: false, default: none,
choices: none, store: true, pipe: none, code: none)
```

```
#parseconfig(_unknown: none, _opts: none, ..args)

#extract(var, _prefix: "", _positional: false, ..keys)

#getconfig(name, final: false)
```

II.5. Loading configuration files

```
#load(filename)
```

`filename` The file to load options from or a `dictionary` options. `string` | `dict` Supported are YAML, TOML and JSON files.

Loads options from a json, toml or yaml file.

Any key on the girst level that has a `dict` as a value will be considered a namespace and the dictionary will be unpacked as options within this namespace.

config.toml

```
[colors]
red = 255,0,0
green = 0,255,0
blue = 0,0,255
```

```
#options.load("config.toml")
#text(
fill:#options.get(
"colors.red",
v => rgb(..v.split(","))
),
[Hello World!]
)
```

`filename` may be a prepopulated `dict` to load in the same way described above.

If you want to load a file without namespaces, just do something like this:

```
#options.update-all(toml("config.toml"))
```

Part III.
Index