# Publishing a Typst Template Package on Typst Universe

Tim Peko (TimerErTim)

As part of my current bachelor thesis writing and efforts to establish Typst at my university, I recently published the ready to use easy-hgb-thesis package for usage at Campus Hagenberg. The package can be found on the official Typst Universe, and I am going to show how easy that process was. This can be thought of as a tutorial with my personal tone and choice of tools: mise, jujutsu and Typst CLI.

---

# Table of Contents

1 Goal . . . 3
2 Writing the package . . . 3
2.1 The package itself . . . 4
2.1.1 Other resources . . . 5
2.2 The template . . . 5
3 Publishing to Typst Universe . . . 7
3.1 Technical introduction . . . 7
3.2 Creating the Pull Request . . . 7
4 Automation . . . 7
4.1 Formatting . . . 7
4.2 Local compilation . . . 8
4.3 Publishing . . . 9
References . . . 10

# 1 Goal

Being the **Typst** enthusiast that I am, my bachelor thesis shall be written in **Typst** as well. There is only one problem: There is no ready to use template for the university of applied sciences Upper Austria… at least not outside of the *LaTeX* world. That has to change before I can start writing! The easy-hgb-thesis package was born.

With that being settled, I have the following goals in mind:
1. The template should be easy to use and understand
2. It should get you going very quickly
3. Flexibility for every unique constellation
4. Support of german and english language
5. Covers advanced customization needs
6. Good documentation and examples.

All in all, the package should be a well-rounded solution for the typical bachelor thesis writing process. Easy for newcomers but powerful for experienced typesetting users, accessable on the official package registry, Typst Universe.

# 2 Writing the package

There are two main parts: The package itself and the ready to get going template. Conceptionally, the package is the real content with styles and convetions, whereas the template can be thought of a new project starter.

File structure of the package

```
├── template
│   ├── bib.yaml
│   ├── chapters
│   │   ├── abstract.typ
│   │   ├── acknowledgement.typ
│   │   ├── appendix.typ
│   │   ├── conclusion.typ
│   │   ├── introduction.typ
│   │   ├── kurzfassung.typ
│   │   ├── methodology.typ
│   │   └── preamble.typ
│   ├── deps.typ
│   ├── main.typ
│   └── mise.toml
├── components
│   ├── constants.typ
│   ├── declaration_de.typ
│   ├── declaration_en.typ
│   ├── i8n.typ
│   ├── sections.typ
│   ├── styles_classic.typ
│   ├── styles_modern.typ
│   ├── template.typ
│   ├── titlepage.typ
│   └── utils.typ
├── lib.typ
├── mise.toml
└── typst.toml
```

## 2.1 The package itself

A **Typst** package requires only a `typst.toml` file and a configurable entrypoint (per convention `lib.typ`). The `typst.toml` file contains metadata relevant for **Typst Universe**, with the full specification available here.

```toml
[package]
name = "easy-hgb-thesis"
description = "Opinionated bachelor's, master's thesis or protocols for the FH Upper Austria, especially campus Hagenberg."
version = "0.2.0"
compiler = "0.14.0"
entrypoint = "lib.typ"
authors = ["Tim Peko <timerertim@gmail.com> @TimerErTim"]
repository = "https://github.com/TimerErTim/hagenberg-thesis-typst"
license = "MIT-0"
⋮                                                                    ⋯
```

`lib.typ` works as a proxy for the public API surface of the package. With it we manually expose functions and types, while the rest stays hidden as internals. This is done by importing them: `#import "components/template.typ": full-thesis, .."`. With that out of the way, let's fill it with life! Let me quickly walk you through the design decisions serving our established goals:

**Easy to use and understand**
The `full-thesis` function is kept simple and straightforward. It pairs well with **Typst**-native concepts such as `#set document(..)` or `#set text(lang: ..)` to define the document's metadata. There are only a few parameter categories, all of which are optional and have sensible defaults:

- **Coverpage**
- **Feature toggles** (such as list of tables, print size control, etc.)
- **Section contents** such as preface, bibliography, main content; skips section per default
- **Style hooks** (`show`-Rules spanning the according sections) for customization

**Gets you going very quickly**
By providing a ready to use template with chapters, typical configurations and compilation scripts already set up, users can get going simple by invoking `typst init @preview/easy-hgb-thesis`. This will create a new directory with the template and all necessary files. Section 2.2 further down below expands on this.

**Flexibility for every unique constellation**
We expose all sections as individual functions so users can opt-out of the opinionated default order. Inserting an appendix is as simple as calling `#appendix-section[..]`. These functions provide the same customization options as the complete `full-thesis` function (including style hooks).

We also provide different premade thesis styles, controlling the default style for the individual sections, allowing users to quickly switch between completely different style variants they can build upon later on. This is simply added as another parameter to the `full-thesis` and sections functions.

```typst
#let THESIS_STYLE = (
```

```
2   "modern": 1,
3   "classic": 2,
4 )
```

**Multiple languages**
By using a `#set text(lang: ..)` rule before the `full-thesis` invocation, users can switch between german and english language. Every text content passes through a `#i8n(key)` function, returning the appropriate variant from a central translation dictionary based on the current `text.lang` value.

**Advanced customization**
Style hooks are very powerful and allow users to override stylings as well as the whole content of individual sections. By adhering to the principle of only using fully reversible `#set`- or impactless `#show`-rules inside the package, users retain the ability to fully override all styling.

**Documentation**
Every publicly facing function is well documented, the template is blastered with helpful comments and a manual is provided with detailed explanations and examples for common customizations.

> **The repository is available on GitHub**
>
> If you want to inspect the source code or contribute yourself, the repository is available at: TimerErTim/hagenberg-thesis-typst

### 2.1.1 Other resources

There are other resources you might want to include when uploading to **Typst Universe** without being required for the package to work in **Typst**, for example a thumbnail image referenced in the `README.md` (so **Typst Universe** can display a nice preview image).

There is an option specifically exclude published files from being fetched by **Typst** when installing the package:

```toml
1 [package]
⋮                                                    ⋯
8 exclude = ["thumbnail.png"]
```

You can read more about this here.

## 2.2 The template

**Typst Universe** allows packages to define a project template with optional thumbnail, which makes setup for new users a breeze. All we need is an additional `[template]` section in `typst.toml` to point to the template directory and the entrypoint:

```toml
10 [template]
11 path = "template"
12 entrypoint = "main.typ"
13 thumbnail = "thumbnail.png"
```

Everything inside the `template` directory is copied to the new project directory when invoking `typst init @preview/easy-hgb-thesis`. I also decided to include a `mise.toml` file to make it easier to get started compiling and formatting the thesis project.

The `main.typ` is the entrypoint for the template and should always use the **Typst Universe** hosted package `@preview/easy-hgb-thesis` to ensure setup agnosticity.

```typst
#import "@preview/easy-hgb-thesis:0.2.0": full-thesis, titlepage, WORK_TYPES

#set document(
  title: "Thesis Title",
  author: ("Author Name", "Name Two", "Name Three"),
  description: "Thesis Description",
  keywords: ("Keyword 1 ", "Keyword 2"),
)
// If German, set to "de" instead of "en"
#set text(lang: "en")

#show: full-thesis.with(
  titlepage: titlepage(
    "Computer Science",
    "Dr. Max Mentorman",
    work-type: WORK_TYPES.bachelor-thesis,
  ),
  kurzfassung: include "chapters/kurzfassung.typ",
  abstract: include "chapters/abstract.typ",
  appendix: include "chapters/appendix.typ", // Can be deleted if not required
  bibl: bibliography("bib.yaml"), // Can be replaced with a BibLaTex file,

  // Style hooks can be used to apply custom styles to sections
  content-style: it => {
    show table.cell.where(y: 0): strong
    it
  },
)

// Include your chapters here, content can also be written here directly but
// may become confusing and hard to maintain with very long contents
#include "chapters/introduction.typ"
#include "chapters/methodology.typ"
#include "chapters/conclusion.typ"
```

An additional user serving artifact is the manual, which is available in the repository. It uses the template with `THESIS_STYLE.modern`. The source functions as example for how to use the package template.

# 3 Publishing to Typst Universe

## 3.1 Technical introduction

**Typst** went for an easy and straightforward approach for its early-stage package registry: Hosting everything in a public GitHub repository typst/packages. Because this is a (albeit indentionally long-term) bandaid solution, [1] the only available package namespace for now is `@preview/`.

**Typst Universe**, the package browser on the web, is effectively just a mirror of this GitHub repo. Submitting a new package is therefore as simple as opening a Pull Request.

## 3.2 Creating the Pull Request

Using the GitHub Pull Request Flow
1. Create a new branch on the locally cloned fork
2. Copy the package files into `packages/preview/easy-hgb-thesis/<version>/`
3. Create and push a new commit
4. Open a Pull Request on the upstream packages repo

That's it. After filling out the PR description, your job is to wait for approval. There is good documentation for rules and best practices, so the package does not stay in review hell.

# 4 Automation

Let's finally mention mise-en-place for automating some of the publishing process. Some of these are genuinely great tips.

We are going to start by adding basic tools and environment variables to the `mise.toml` file that ensure proper compilation.

```toml
[tools]
typst = "0.15.0"
⋮                                                    …
[env]
TYPST_ROOT = "{{config_root}}"
TYPST_FONT_PATHS = "{{config_root}}/template/fonts"
⋮                                                    …
```

## 4.1 Formatting

typstyle is a great **Typst** formatter that we can easily integrate into our workflow with just one tool and according task defintion:

```toml
[tools]
⋮                                                    …
typstyle = "0.15.0"
⋮                                                    …
[tasks."fmt:typst"]
description = "Format the typst code"
run = [
```

```
13      "typstyle -i ./*.typ ./components/*.typ ./template/**/*.typ --timing"
14  ]
⋮                                                                    …
```

Listing the individual subdirectories manually is tedius but required for the next chapter.

## 4.2 Local compilation

**Typst Universe** recommends compiling the exact template version locally before publishing to ensure compilability. So the unreleased, local package version has to become available to the template via `#import "@preview/easy-hgb-thesis:<unreleased-version>"` directive.

**Typst** supports local package registries via the `TYPST_PACKAGE_PATH` environment variable. Instead of probing into the `packages/` folder of the GitHub repo mentioned in Section 3, it probes into the folder this environment variable points to.

```
⋮                                                                    …
5    [env]                                                                T TOML
⋮                                                                    …
8    TYPST_PACKAGE_PATH = "{{config_root}}/local-packages/"
⋮                                                                    …
122  [tasks."setup:editable-package"]
123  description = "Setup the editable package to use @preview/... locally"
124  tools.yq = "4.48.1"
125  run = '''
126  #!/usr/bin/env bash
127  set -euo pipefail
128
129  name=$(yq e '.package.name' typst.toml)
130  version=$(yq e '.package.version' typst.toml)
131
132  dest="local-packages/preview/$name/$version"
133  echo "Setting up $name editable version $version to $dest..."
134  mkdir -p "$dest"
135  rm -rf "$dest"
136  ln -sf "../../../" "$dest"
137  '''
⋮                                                                    …
```

Let's break this down:

- `TYPST_PACKAGE_PATH = "{{config_root}}/local-packages/"`: Tells **Typst** to look for packages inside the `local-packages/` folder.
- `[tasks."setup:editable-package"]`: Symlinks the project root to the `local-packages/` folder so **Typst** can find the package.
  1. Extract package `name` and `version` from the `typst.toml` file using *yq* (the yaml equivalent of *jq*)
  2. **Typst** will look for the package at `<namespace>/<name>/<version>/` of the local package registry. Store that as symlink target.
  3. Create a relative symlink from the project root to the target directory.

Ta-da! Compilation will now work seamlessly.

## 4.3 Publishing

The whole git workspace management can be automated as well. The whole publishing process then consists of five steps:

1. **Preparation**: Compile artifacts and format code
2. **Workspace setup**: Clone the forked package repository under `packages`
3. **Extract version** from the package metadata
4. **Copy package files** to the correct version directory
5. **Commit**

```toml
⋮                                                  …
45  [tasks."publish:prepare"]
46  description = "Prepare the project for publication"
47  depends = ["fmt", "build:thumbnails", "build:manual", "check:template-compilable"]
⋮                                                  …
101 [tasks.publish]
102 description = "Publish the project to local packages mirror"
103 depends = ["setup:packages-workspace", "publish:prepare"]
104 tools.yq = "4.48.1"
105 tools.jujutsu = "0.40.0"
106 run = '''
107 #!/usr/bin/env bash
108 set -euo pipefail
109
110 # Use yq v4 syntax for TOML (note: yq v4 reads TOML with 'yq -e -p toml ...')
111 name=$(yq e '.package.name' typst.toml)
112 version=$(yq e '.package.version' typst.toml)
113
114 dest="packages/packages/preview/$name/$version"
115 echo "Pushing $name version $version to $dest..."
116 mise run publish:copy-to "$dest"
117 jj workspace update-stale --repository packages
118 jj commit -m "$name:$version" --repository packages
119 jj b s ght-publish -r @- --repository packages --allow-backwards
120 '''
⋮                                                  …
```

The magic happens in line 116, because the `publish:copy-to` task[1] has a certain gimmick: It respects a `.publishignore` file, that works like a `.gitignore` file but for the publish process. There are some files we do not want to publish but want to track in the project repo, such as the root `mise.toml` file.

Example for a `.publishignore` file

```gitignore
1 # Exclude additional resources
2 thumbnail/
```

[1] a simple python script

```
manual/

# Exclude development/IDE config
.vscode/
/mise.toml
```

That's it, the `publish` task will now take care of the rest. Thanks for reading this far!

# References


1. [Accessed 12 August 2026]. Available from: https://typst.app/blog/2023/package-manager/
2. Available from: https://typst.app/universe/package/easy-hgb-thesis
3. Available from: https://typst.app/universe
4. Available from: https://github.com/typst/packages/blob/main/docs/manifest.md
5. Available from: https://github.com/TimerErTim/hagenberg-thesis-typst
6. Available from: https://github.com/typst/packages/blob/main/docs/tips.md#what-to-commit-what-to-exclude
7. Available from: https://github.com/TimerErTim/hagenberg-thesis-typst/blob/main/easy-hgb-thesis-manual.pdf
8. Available from: https://github.com/typst/packages/
9. Available from: https://medium.com/@abhijit838/git-fork-development-workflow-and-best-practices-fb5b3573ab74
10. Available from: https://github.com/typst/packages/tree/main/docs
11. Available from: https://mise.jdx.dev/
12. Available from: https://github.com/typstyle-rs/typstyle