The LaTeX template generator: How micro-templates reduce template maintenance effort

Oliver Kopp

Abstract

Scientific findings are published by different publishers. These provide different templates. These differ in the documentation and packages provided. For example, `hyperref` or `microtype` are mostly not included or not configured properly. Furthermore, there is a demand for minimal examples in the body of the paper. For instance, how to typeset a listing with line numbers and hyperlink to that line number. These minimal examples should appear in any paper template. If the minimal example is updated, how can various paper templates be updated automatically? The “LaTeX Template Generator” is one answer to this question. It uses “micro-templates” to create full-fledged paper templates containing the same configurations for popular packages. Thus, it reduces the maintenance effort of LaTeX templates.

1 Introduction

In scientific research, starting a new paper often involves using a previous publication as a template. Researchers adapt this base structure to meet their current requirements. However, this practice introduces the potential for inconsistencies, particularly with respect to the usage and configuration of LaTeX packages. Although publishers offer templates for their publication venues, these templates are rather minimal and often omit configuration for `hyperref`, `microtype`, `listings`, etc.

One solution is to offer “extended” templates for each venue, including best practices for each LaTeX package. However, when the package is updated with new features or when new insights about the package emerge, all these templates must be manually updated — a process prone to errors. The “LaTeX Template Generator” (LTG) addresses these challenges. It introduces the concept of “micro templates”, each providing a preamble and an example for a specific LaTeX package. These micro templates are then automatically consolidated into templates for various outputs, such as journals, conferences, and student theses. This strategy reduces the risk of errors and simplifies the process of updating LaTeX templates.

Figure 1 presents the roles when working with the LTG:

- The role *package expert* is filled by an individual with extensive knowledge about a specific package. For instance, one expert might specialize in the `hyperref` package, while another might be well-versed in the `microtype` package. These experts contribute not only by configuring the packages, but also by providing minimal examples for each package. This guidance ensures proper usage and assists newcomers in understanding the package functionalities.
- The role *class expert* is held by an individual who understands the necessities of each class. They know which packages are required for each template based on the class and which ones might be unnecessary or counterproductive.
- The role *tooling expert* is held by someone proficient in the respective tooling, such as `latexmk` or `git`. They provide configurations for these tools for templates generated by the LTG.
- The role *content expert* provides guidance on how to write the scientific content of a paper. They may offer advice on structuring arguments, referencing sources, or other aspects of academic writing.
- Finally, the role *template user* describes the ultimate user of the template crafting scientific work.

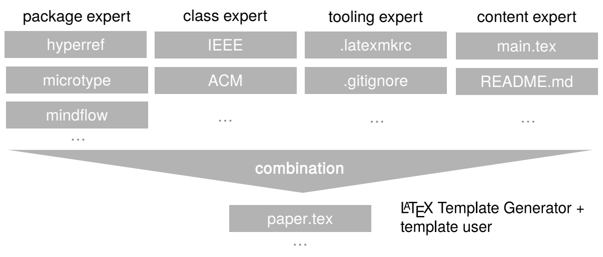


Figure 1: Roles in the process of template generation

The task of combining the different inputs into a template file (“combination” in Figure 1) is done by the LaTeX Template Generator.

The basic idea of the LTG is to offer configuration possibilities where required and to assume sensible defaults where possible. For instance, the template user is offered a choice for the overall template to target (e.g., IEEE or a master’s thesis), the language to be used (e.g., English or German), but does not need to choose anything for packages such as `microtype`, because sensible defaults are provided.

In the following, details of the LTG are provided. First, Section 2 presents reasoning on the chosen prompting and generation framework. Section 3 outlines the general concept. Section 4 presents the usage of the LaTeX template generator. Finally, Section 5 provides a discussion and presents an outlook on future work.

doi.org/10.47397/tb/44-2/tb137kopp-microtemplates

2 The choice of yeoman as the basis for the generator

To guide the template user through different options and to generate the final template, an existing framework should be used. The basic requirements are: i) being able to mix multiple micro-templates into larger templates (e.g., the `hyperref` configuration should be stored once in the repository and used by multiple templates) and ii) offering dependent prompts. For instance, if "Overleaf compatibility" is chosen, the choices of the TEX Live variants should be constrained to versions before 2023.

In 2019, the following frameworks were evaluated: Yeoman,[1] Cookiecutter,[2] copier,[3] Jinja2,[4] Cheetah,[5] Apache Velocity,[6] and LuaLATEX. Some of these options are templating engines only (Jinja2, Cheetah, Apache Velocity). Thus, the prompting would have been required to be hardwired. LuaLATEX is a very general "framework", which is not commonly used for templating. Since I was not proficient enough to use it, the final choice was between Python-based tooling (Cookiecutter and copier) and JavaScript-based tooling Yeoman. Since both Cookiecutter and copier require the choices to be made available in text-based configuration files and it seemed to be impossible to craft choices dependent on previous choices, I opted for Yeoman.

In 2020, TEXplate was released on CTAN.[7] The current version has a different structure than the LTG. TEXplate relies on TOML[8] files to define the LATEX file to be generated. LTG builds upon `.tex` files which are "enriched" by templating commands. The claim of LTG is that it is easier for contributors to edit `.tex` files with their editor of choice than to edit TOML files.

All in all, Yeoman has been chosen as the generator framework. It is based on JavaScript and fulfills both requirements: The package preambles and examples are stored in different files (details in Section 3) and offers built-in prompting. Prompt choices can be modified on the fly to enable dependent prompts.

To make the knowledge about the decision sustainable, Markdown Any Decision Records [4] have been written. All the aforementioned options as well as their pros and cons are included as Markdown files inside the path `docs/decisions/`[9] inside the repository.

[1] https://yeoman.io/
[2] https://github.com/cookiecutter/cookiecutter
[3] https://github.com/copier-org/copier
[4] http://jinja.pocoo.org/
[5] http://cheetahtemplate.org/
[6] http://velocity.apache.org/
[7] https://ctan.org/pkg/texplate
[8] https://toml.io/en/

```
/
└─generators/
   └─app/
      ├─templates/
      │  ├─...
      │  ├─Dockerfile.iot
      │  ├─Texlivefile
      │  ├─...
      │  ├─main.en.tex
      │  ├─mindflow.example.en.tex
      │  ├─mindflow.preamble.en.tex
      │  └─...
      ├─index.js
      └─options.js
```

Figure 2: Directory tree of the generator

3 The concept of the LTG

This section presents the concept of LTG. Thereby, the file structure of LTG's source repository[10] is used. The most important files are presented in Figure 2.

The pair of files `mindflow.example.en.tex` and `mindflow.preamble.en.tex` show the basic concept of LTG's micro templates using the `mindflow` package. The `mindflow` package[11] is a basic LATEX package enabling a) quickly noting down thoughts and b) having LATEX marking these thoughts visually.

A micro template consists of i) a preamble file and ii) optionally an example file. The content of the preamble file is put into the preamble and the example is put as LATEX example in the document body. The filename is always the package name, followed by either `preamble` or `example` and then `en` for English or `de` for German.

`mindflow.preamble.en.tex` looks as follows:

```
<% switch (documentclass) { case "acmart":
  case "ieee": -%>
\usepackage[incolumn]{mindflow}
<% break; default: -%>
\usepackage{mindflow}
<% break; } -%>
```

On the first line, one sees the templating language "Embedded JavaScript templating" (EJS[12]) in use. In general, template commands are enclosed in

[9] https://github.com/latextemplates/generator-latex-template/tree/main/docs/decisions
[10] https://github.com/latextemplates/generator-latex-template
[11] https://ctan.org/pkg/mindflow
[12] https://ejs.co/

`<% ... %>`. The minus sign before the closing `%>` indicates that the following newline should be removed. The intention of the EJS code is that in the case of the document class being a two-column document class (as it is for ACM and IEEE), the `mindflow` package is passed the option `incolumn`. In all other cases, just `\usepackage{mindflow}` is written out.

`mindflow.example.en.tex` looks as follows:

```
<%= heading2 %>{Notes separated from
  the text}

The package mindflow enables writing down
notes and annotations in a way so that they
are separated from the main text.

<%- bexample %>
\begin{mindflow}
This is a small note.
\end{mindflow}
<%- eexample %>
```

The template command `<%= heading2 %>` instructs Yeoman to put the content of the `heading2` variable at that place. In the case of an IEEE template, this is `\subsection`; in the case of, for instance, the provided `scientific-thesis` template, this is `\section`, because the latter's main structuring element is `\chapter`.

The markers `bexample` and `eexample` are markers for LaTeX commands for beginning and ending an example. The LTG defines its own environment for examples to output both rendered LaTeX code as well as the source code. It makes use of the capabilities of the `tcolorbox` package.[13]

The preamble is included as follows in the file `main.en.tex`:

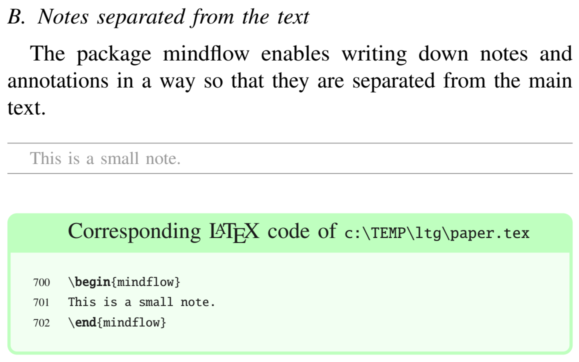
```
<% if (texlive >= 2021) { %><%- include
  ('mindflow.preamble.en.tex', this); } -%>
```

The reason for the guard with the TeX Live version is that `mindflow` was released in 2021, and a template may require support of earlier TeX Live versions.

The file `options.js` contains all options offered to the user. The following excerpt presents the option for TeX Live:

```
{
  type: "list",
  name: "texlive",
  message: "Which TeXLive compatibility?",
  choices(state) {
    const res = [
      {
        name: "TeXLive 2021",
        value: 2021,
      },
      {
        name: "TeXLive 2022",
        value: 2022,
      },
    ];
    if (!state.overleaf) {
      res.push({
        name: "TeXLive 2023",
        value: 2023,
      });
    }
    return res;
  }
},
```

[13] https://ctan.org/pkg/tcolorbox



B. Notes separated from the text

The package mindflow enables writing down notes and annotations in a way so that they are separated from the main text.

This is a small note.

Corresponding LaTeX code of `c:\TEMP\ltg\paper.tex`

```
\begin{mindflow}
This is a small note.
\end{mindflow}
```

Figure 3: Mindflow example section in the rendered IEEE template

It starts by defining the option to be a list and instructs that the answer should end up in a JSON object property named `texlive`. The message used for prompting is given in `message`. The choices are created dynamically, based on previous choices. As an example, the user can choose whether the template should be usable on Overleaf. If they opted for "yes", the option to choose TeX Live 2023 is not included, since (at the time of release of the LTG), version 2023 is not supported by Overleaf. (Overleaf typically adds support for new TeX Live releases in the fall [2].) Coming back to the options, there are in total 18 possible options to choose from. Some are dependent on the chosen document class and thus not all are shown to the user.

The file `index.js` calls the prompting, derives internal variables based on the result, and finally creates the resulting files. One internal variable is the final file name. In the case of journals and conferences, this is `paper.tex`. In case of a scientific thesis, it is `main.tex`.

Figure 3 shows the rendered output of this `mindflow` example, when the IEEE template is selected.

4 Usage

The LTG requires a recent Node.js installation. There, the command

```
npm install -g generator-latex-template
```

installs the generator and makes it globally accessible on the target machine. Then, the user can invoke the generator using `yo latex-template`. After issuing that command, LTG outputs following:

```
$ yo latex-template

? Which template should be generated?
    (Use arrow keys)
> Scientic Thesis
  Association for Computing Machinery (ACM)
  Institute of Electrical and Electronics
    Engineers (IEEE)
  Springer's Lecture Notes in Computer
    Science (LNCS)
```

The user is first asked which template they want to create. Currently, a scientific thesis template, ACM, IEEE, and LNCS are supported. More templates are part of future work.

The user navigates through the options using arrow keys. Once a choice is made, the system proceeds to the next question, continuing this iterative process until all questions have been answered.

The following presents the result of an example complete process of selections:

```
? Which template should be generated? IEEE
? Which variant of IEEE paper? conference paper
? Which paper size to use? A4
? Overleaf compatibility? yes
? Which TeXLive compatibility? TeXLive 2022
? Should a Dockerfile be generated?
  yes (Island of TeX)
? Which language should the document be?
  English
? Which package to typeset listings? listings
? Which package to use to "enquote" text?
  csquotes (\enquote{...} command)
? Which package to mark TODOs? pdfcomment
? Include hints on text
  (e.g., how to write an abstract)? Yes
? Include minimal LaTeX examples? Yes
```

After all questions have been answered, Yeoman outputs the files it creates:

```
  create .gitignore
  create .editorconfig
  create paper.bib
  create _latexmkrc
  create localSettings.yaml
  create LICENSE
  create Makefile
  create paper.tex
  create README.md
  create .dockerignore
  create Dockerfile
  create Texlivefile
  create .github\workflows\check.yml
```

Note that the file `latexmkrc` is prefixed by an underscore. This enables uploading the whole repository to Overleaf without any error shown in the user interface. A `Dockerfile` is also generated. In this example, the file is generated based on `Dockerfile.iot` and uses a minimal Island of TeX Docker image [3]. The image installs all LaTeX packages listed in `Texlivefile` into the image. This way, the image size is kept to a minimum.

Finally, a GitHub workflow[14] file is generated. When publishing the repository on GitHub, GitHub's CI will build a docker image based on the `Dockerfile` and build the LaTeX file using `latexmk`.

5 Discussion and outlook

This paper presented the LaTeX Template Generator as one solution to collect knowledge about best practices of packages and a way to include them in rich templates for authors. To add support for a new class, the class expert has to adapt `main.en.tex` and `options.js` to include the class and add proper conditions for packages which should be included or excluded. Then, a new template file is generated. No work for the package experts is caused: Their templates can (most probably) just be used by the class expert. Vice versa, if an update on the package examples are made, the class expert (most probably) does not need to do anything, because the contents are directly available in their template.

For end users, installing Node.js can be tedious. Therefore, for each supported template, a separate GitHub repository is offered. In that repository, default `paper-*.tex` files are offered. For the LNCS template,[15] `paper.tex` uses the Computer Modern font, `microtype` configuration, `listings` configuration (including JSON support), `pdfcomment`[16] for TODO marking, and LaTeX examples. To reduce the size of the `.tex` file, no hints on writing a paper are included.

The repository also offers other `paper-*.tex` files. For instance, `paper-en-times-minted.tex` provides a template where Times New Roman is used for the font and `minted`[17] as the package for listings. A template user can just download the ZIP

[14] https://github.com/features/actions
[15] https://github.com/latextemplates/LNCS
[16] https://ctan.org/pkg/pdfcomment
[17] https://ctan.org/pkg/minted

archive of the repository or even use GitHub's template feature[18] to create a new git repository hosted on GitHub containing the latest template files as single commit.

The most impactful design decision is to have the choices encoded in the templating language. For instance, if LuaLaTeX is chosen by the user, the font configuration is generated for LuaLaTeX. In case the author wants to switch to pdfLaTeX, they must regenerate the whole template: There is no "dynamic" LaTeX if/else construct for a pdfLaTeX fallback. Future work will investigate this further and possibly add an additional user option to generate a more flexible template.

The LTG project itself is a true open source project and calls for contributions of examples of common classes, packages and practices. Currently, around 20 packages and examples are offered. A good start are the hints given by Beeton [1]. As the new *The LaTeX Companion*, third edition, discusses more than 500 examples [5], there is lots of room to include examples. Certainly, a careful selection of discussed packages needs to be made. The LTG focuses on providing only one example per topic. Thus, these examples will surely be enriched by references to TLC3 for the interested readers.

References

[1] B. Beeton. What every (La)TeX newbie should know. *TUGboat* 44(2):164–169, 2023.

[2] T. Hejda. TeX Live and Overleaf revisited. *TUGboat* 44(2):256–256, 2023.

[3] Island of TeX. Living in containers — on TeX live in a docker setting. *TUGboat* 44(2):249–252, 2023.

[4] O. Kopp, A. Armbruster, O. Zimmermann. Markdown architectural decision records: Format and tool support. In *ZEUS*, vol. 2072 of *CEUR Workshop Proceedings*, pp. 55–62, 2018. https://ceur-ws.org/Vol-2072/paper9.pdf

[5] F. Mittelbach, U. Fischer. *The LaTeX Companion: Parts I & II.* Addison-Wesley, third ed., 2023.

◇ Oliver Kopp
Sindelfingen, Germany
https://github.com/koppor
ORCID 0000-0001-6962-4290

[18] https://docs.github.com/en/repositories/creating-and-managing-repositories/creating-a-repository-from-a-template