Xerdi's Documentation Project (XDP): Intertwined documents

Erik Nijenhuis

Abstract

This article presents Xerdi's Documentation Project (XDP), a modular framework for scaling and simplifying LaTeX-based document production. XDP separates template logic, content, and configuration into structured components, such as Executable Document Templates (EDTs), Document Portfolios, Data Definition Repositories, and Workspaces. Leveraging LaTeX features like `TEXMFHOME` and `TEXMFDOTDIR`, it enables reusable and extensible document workflows across teams or clients.

Through practical examples, this article demonstrates how intertwined documents can simplify use cases like legal documents, invoices, and academic materials. It highlights how XDP tackles challenges such as responsive branding, agile development, and user-friendly input for minimal coding authors.

The framework incorporates tools such as `bib2-gls` [9], `attachfile2` [5], and `zref` [6] to automate complex workflows and improve document scalability. Readers will gain actionable insights and practical tools to better structure and manage their LaTeX projects efficiently and collaboratively.

1 Introduction

Xerdi, my sole proprietorship founded in 2021, is a software development company focused on creating (Free) software. I'm actively engaging Free Software communities like NTG, TUG, and the FSF, and I'm partnered with Red Hat for clients in need of a secure and robust platform to work on. When it comes to LaTeX, I'm package maintainer of several packages: `gitinfo-lua` [4], `lua-placeholders` [2], `regulatory` [3], and `fmtcount` [10].

2 Life cycle management

I submit packages to CTAN to get them widely distributed. For my document solutions, I solely trust on the TeX Live distribution. MiKTeX and MacTeX aren't tested by me, and for clients who use Windows NT, I also encourage using TeX Live instead.

Figure 1 is an illustration of my package management. For more information on how I manage my packages, you can read my article XDP Packaging Guidelines.[1]

[1] XDP Packaging Guidelines: `xerdi.com/static/public/xdp-packaging-guidelines.pdf`

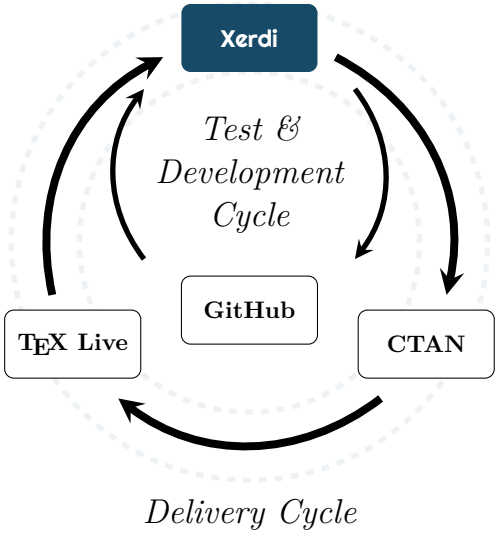


Figure 1: Life cycle of package management at Xerdi

3 Project goals

The project (XDP) consists of three main goals. It should provide straightforward macros for subject matter experts, like package `regulatory` for contract makers. Every useful component should be extracted from the project and be made widely available by extracting and submitting them as LaTeX packages. And lastly, it should be suitable for integration purposes. The latter isn't a hard requirement for every use case.

For example, scientific or technical reports are done individually by subject matter experts. So there is no need for integration, since there isn't an *End User* or a system trigger involved.

Some other use cases I practise are office documents, legal documents, music scores and diagrams.

I'm also encouraging my suppliers to use LaTeX, so that will also affect the challenges of the project. For example,

1. managing multiple company brands, which I do with the `TEXMFHOME` variable;
2. for agile development, it's invaluable to be able to directly change the code of my own packages so that document generation isn't delayed by the release process;
3. for some use cases it's necessary to separate some content from the author's code, e.g., legal authors shouldn't know any financial details of their customers. More about that in the final subject;
4. supporting user-friendly data input formats for easy document generation;
5. and finally, though not the most challenging, is having a full audit log of all changes, which is

doi.org/10.47397/tb/46-2/tb143nijenhuis-doceng

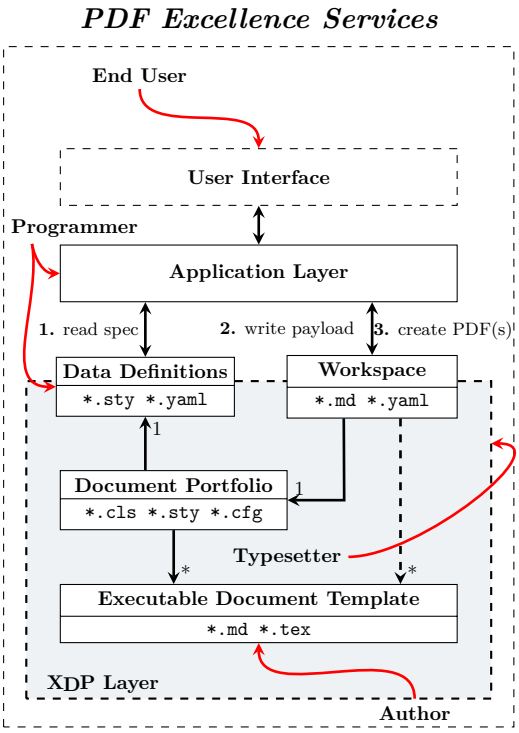


Figure 2: XDP overview of components

a bit off-topic. If curious, check out `git` and especially Git submodules to handle multiple directories within the same tree.

4 XDP components

To overcome all challenges, I came up with the following distinctions:

The Workspace, which contains all user-related files. Ideally, it contains no LaTeX-related files, but rather YAML or JSON;

Data Definitions (DD), which contain the input specification. This is typically done in YAML, and the configuration can be simplified using a LaTeX package file (`.sty`);

The Document Portfolio (DP), which will be the `TEXMFHOME` directory; and

The Executable Document Template (EDT) contains the complex LaTeX files. Together with input data, the sources can be compiled to PDF.

This setup is specifically for the worst case scenarios.

To illustrate the relations between those components, see the diagram in figure 2. As you can see, there are four actors: the end-user, who typically fills in an HTML form; the programmer or system administrator, who specifies the input specification and programs the software accordingly; the typesetter, who manages all LaTeX-related files; and the author, who writes most of the content in the Executable Document Template.

With all those roles separated, we end up with the following workflow (also see figure 3):

1. the author drafts an EDT;
2. the author submits it;
3. the typesetter reviews it;
4. optionally, the typesetter improves the EDT and or the DP;
5. the programmer reviews the DDs used;
6. optionally, the programmer improves the DDs;
7. finally, the programmer releases the entire DP and upgrades the system with it.

5 Directory management

As mentioned in the previous Section, the Document Portfolio is nothing more or less than the `TEXMFHOME` directory. The directory structure is as follows:

```
xerdi-portfolio/ ..... TDS compliant
├─ bibtex/bib/
│  └─ *.bib
├─ packages/ ..... managed with TEXMFDOTDIR
│  ├─ gitinfo-lua/
│  ├─ lua-placeholders/
│  └─ regulatory/
└─ tex/
   ├─ generic/images/
   │  └─ *.(png|jpg|svg)
   ├─ defs/
   │  └─ ⟨data definitions⟩/
   │     └─ *.(yaml|sty)
   ├─ edts/
   │  └─ ⟨executable document template⟩/
   │     └─ *.(tex|md)
   └─ latex/
      └─ *.(cls|sty|cfg)
```

In this example, the EDTs are stored within the DP, as well as the DDs. Document class files or package files are stored typically in the LaTeX directory, as one would do normally. To briefly come back to the agile programming challenge, in this example I've included it in the `TEXMFHOME` directory as well. It doesn't necessarily have to be in there, however, it's quite convenient to have it next to your global LaTeX files if you also have test projects in the same directory.

So the environment setup for this scenario is: `TEXMFHOME=`⟨*document portfolio*⟩ and `TEXMFDOTDIR=.:`⟨*document portfolio*⟩`/packages//`

If you're using only one branding, it's highly advisable to configure this globally in your `texmf.cnf`

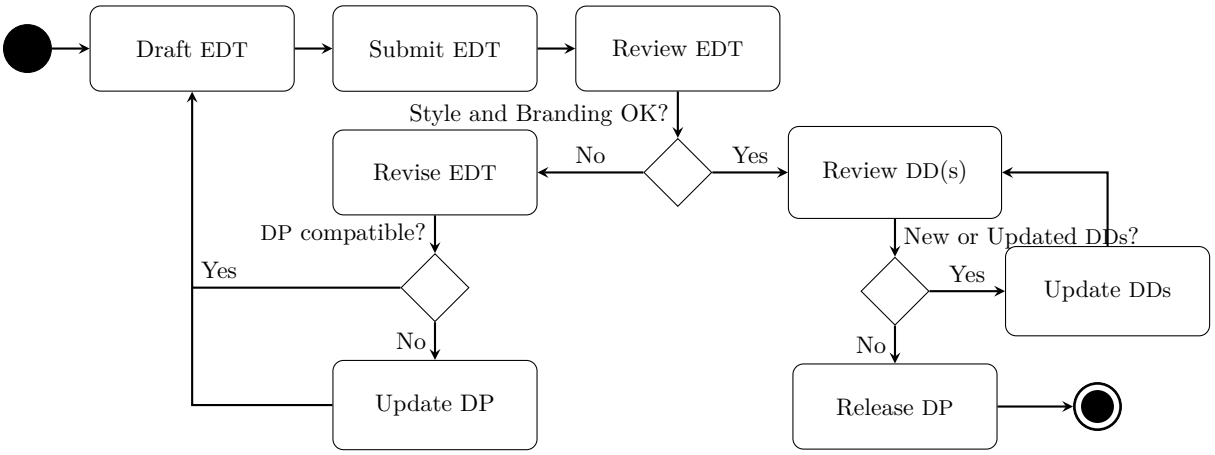


Figure 3: XDP workflow

file. This ensures that your IDE of choice finds the correct files, which could be beneficial for e.g. autocompletion. In my case, I have to set those variables per client project before opening an IDE, to get such benefits.

To ensure you get the right execution environment for every use case, like executing test files, or executing EDTs without values for drafting purposes, I highly recommend using GNU Make.

```
1 export PROJECT_ROOT=$(shell pwd)
2 export TEXMFCNF=$(PROJECT_ROOT):
3 export TEXMFHOME=$(PROJECT_ROOT)
4 export TEXMFDOTDIR=.:$(PROJECT_ROOT)/packages ↩
    //
5
6 test-env:
7     @echo TEXMFHOME: $$TEXMFHOME
8     @echo TEXMFHOME \(kpsewhich\): $$( ↩
        kpsewhich --var-value=TEXMFHOME)
9     @echo TEXMFCNF: $$(kpsewhich --var-value ↩
        TEXMFCNF)
10    @echo TEXMFDOTDIR: $$(kpsewhich --var- ↩
        value TEXMFDOTDIR)
11    @echo lua-placeholders.sty: $$(kpsewhich ↩
        lua-placeholders.sty)
12    @echo lua-placeholders.lua: $$(kpsewhich ↩
        lua-placeholders.lua)
```

The listing demonstrates how you can test if it finds the correct `.sty` and `.lua` files with the `kpsewhich` program, which is also used internally for TEX programs.

6 Intertwined documents

Xerdi offers clients an assignment and or maintenance agreement, which is summarized in the quote. The goal here is to specify only what's necessary. So to come up with the full offer, there are two requirements: the message in Markdown format, which briefly describes what the intention is of the agreement(s); and some project-specific variables in YAML. For the end solution, end-users wouldn't manually provide a YAML file to the contract maker, but would typically use an HTML user interface instead. For integration purposes, `lua-placeholders` supports both YAML and JSON.

- `message.md`
- `project.yaml`

⇒

- `quote.pdf`
- `assignment.pdf`
- `maintenance.pdf`
- `terms-conditions.pdf`

The two files will result in a complete set of documents, which are fully embedded as well. They also reference each other, which is illustrated in figure 4.

6.1 Input files

The sample message shown in listing 1 can be of any Markdown-compliant content. For some quotes, it would be preferred to use TEX instead to leverage all features of the `regulatory` package.

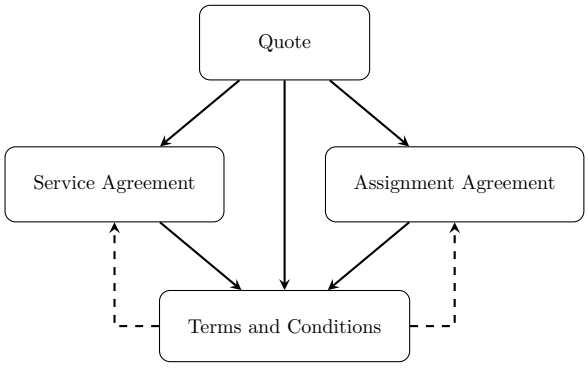


Figure 4: Overview of the end result

Listing 1: Message in Markdown

```
The proposal before you was drafted following ↩
   the conversation on 27 March 2024 and ↩
  revised based on the communications between ↩
   13 and 14 April 2024.
The following main points were discussed:

- Grapefruit Inc. has developed its own ↩
  administrative system.
- Grapefruit Inc. desires further development ↩
   of the administrative system.
- Xerdi concludes that research and ↩
  development is essential to enable further ↩
  development.
```

The YAML file in listing 2 specifies all necessary information, which is used in every document. The `client` variable refers to a client label name. The most important variables are `activities` and `maintenance`. When there are any activities specified, an assignment agreement will be embedded in the quote. Also, when any maintenance activities are specified, a maintenance agreement will be attached to the quote. So now we'll take a look at how this input specification is defined.

Listing 2: Project settings in YAML

```
client: grapefruit
start date: April 22, 2024
# ... other project options
price: 3216
effort: 8 hours a week
activities:
  - Research and development for a real-time ↩
    billing system.
  - Writing and delivering of technical ↩
    documentation in PDF.
maintenance:
  - name: CI/CD
    desc: Facilitation of GitLab Runners.
  - name: Migration Support
    desc: Upgrades for test- and production ↩
      systems.
    price: 85
```

6.2 Input specification

The first specifications in the `project.yaml` specification are quite straightforward. Those values will load other recipes and values correspondingly. For simplicity's sake, we'll skip those recipes.

The project recipe is typically located in the DD directory, `tex/defs/legal/project.yaml`.

Listing 3: Project file specification in YAML

```
activities:
  type: list
  item type: string
maintenance:
  type: table
  columns:
    name:
      type: string
      placeholder: Service name
    desc:
      type: string
      placeholder: Service description
    price:
      type: number
      default: 67
```

For the activities we simply use a list with strings, and for the maintenance activities we'll use a complex table. The reason to have the maintenance activities in a table is that each maintenance activity can have a different pricing. This is dependent on your contractual preferences.

6.3 Loading input-related files

It's useful to place the input-related file loading in the DD directory as well. For our case, let's say, `tex/defs/legal/project.sty`.

Listing 4: Load input-related files

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{project}[2024/03/29 Xerdi's ↩
  Project Data Package]
\RequirePackage{lua-placeholders}

\setnamespace{project}

\loadrecipe[supplier]{defs/legal/supplier. ↩
  yaml}
\loadrecipe[client]{defs/legal/client.yaml}
\loadrecipe{defs/legal/project.yaml}

\IfFileExists{./project.yaml}{%
   \loadpayload[project]{./project.yaml}
   \loadpayload[supplier] ↩
     {suppliers/\rawparam{project}{supplier ↩
     }.yaml}
   \hasparam{client}{%
      \loadpayload[client]{clients/\rawparam ↩
        {project}{client}.yaml}
   }{\PackageWarning{project.sty}{Warning: ↩
     client not loaded}}
}{}
```

The core dependency for these YAML files is the package `lua-placeholders`. For convenience, the namespace is set to `project`, because that's the most used namespace in all files. Then all recipes need to be loaded in any case. All value files are guarded with the `\IfFileExists` command, so that the document can always be compiled containing placeholders instead of actual values.

7 Contextual content

The contextual content starts with `\hasparam` on line 2, where some text is given, including a command `\documentfootnote`, which is a macro from the `regulatory` package. It ensures that the actual file is embedded and described in the footnote. An automatic external reference is shown on line 4, using `\aref`, also a macro from `regulatory`.

Listing 5: Contextual assignment agreement content

```
1 \newcommand\formatservice[1]{\item #1}%
2 \hasparam{activities}{%
3    \noindent\documentorg{} proposes an ↩
       Assignment Agreement\documentfootnote{ ↩
       ovo} for a duration of ↩
       \underline{\param{duration}} with an ↩
       effort of \underline{\param{effort}}, ↩
       at a fixed fee of \underline{€\param{ ↩
       price}} (\vatinclusion{}) starting on ↩
       \underline{\param{start date}}.
4    Below is an overview of the non- ↩
       exhaustive activities as outlined in ↩
       \aref{ovo-lid:werkzaamheden}:
5    \begin{enumerate}[label=\alph*)]
6       \forlistitem{activities}{formatservice}
7    \end{enumerate}
8 }{}%
```

8 Embedding

To get files embedded, or suitable for referencing, you can either use macro `\refdocument` for referencing only, or `\masterdocument` for embedding as well. This also includes sharing glossaries, which I use for legal definitions, which are typically listed in the terms and conditions, or "Algemene Voorwaarden" in Dutch.

```
\newcommand\footnotelabel[1]{#1, vindplaats: ↩
  \href{https://xerdi.com/terms-and- ↩
  conditions.pdf}{xerdi.com/terms-and- ↩
  conditions.pdf}}
\newcommand\definitionlabel[1]{~(zie #1)}
\masterdocument[av-]{av}{
   defs=definities,
   author=E. Nijenhuis,
   subject=Algemene Voorwaarden van \gls*{ ↩
     mycompany},
   description={De Algemene Voorwaarden van ↩
     \rawparam{bedrijf}{naam}, vind u ↩
     aangehecht in dit document, of is te ↩
     verkrijgen via https://\rawparam{ ↩
     bedrijf}{website}/terms-and-conditions. ↩
     pdf},
   footnote label=\footnotelabel,
   ref label={van de \gls{terms} van \gls{ ↩
     mycompany}},
   def label=\definitionlabel,
   referred=true
}
```

Many settings are available. For example, using `referred=true` will prevent setting footnotes at the first occurrence, because I have done it manually for this setup. All options are described in detail in the `regulatory` package manual.

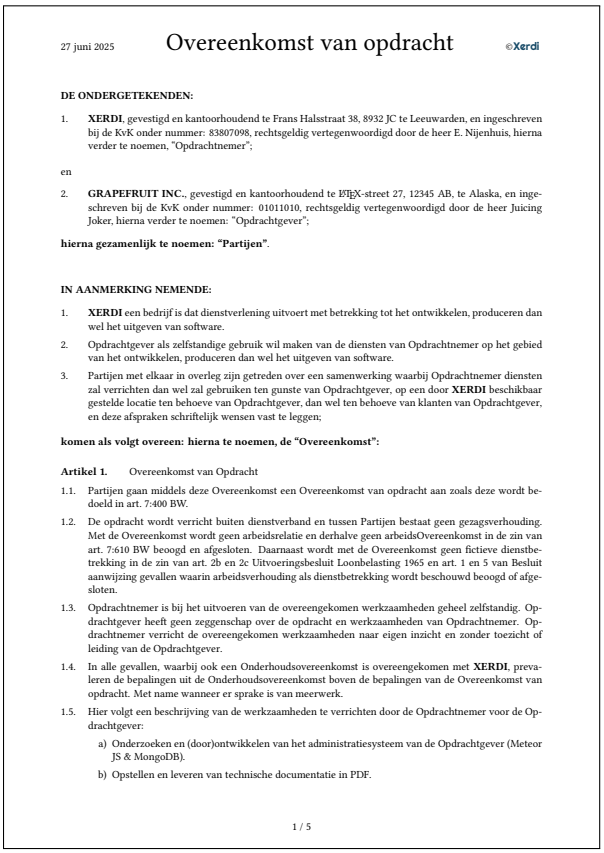
In the above, you can also see some `\rawparam` usages, since I haven't yet made the configuration fully supplier-independent.

9 The result

Figure 5 shows the resulting quote. It's basically a standard letter, followed by our custom message. Then based on what values were given, activities are displayed, including a footnote which contains a reference to the PDF file.

The assignment agreement shown in figure 6 also has an attachment, in this case of the terms and conditions. You can again see the client details

Figure 5: The resulting quote shown in Evince



27 juni 2025 Overeenkomst van opdracht ©Xerdi

DE ONDERGETEKENDEN:

1. **XERDI**, gevestigd en kantoorhoudend te Frans Halsstraat 38, 8932 JC te Leeuwarden, en ingeschreven bij de KvK onder nummer: 83807098, rechtsgeldig vertegenwoordigd door de heer E. Nijenhuis, hierna verder te noemen, "Opdrachtnemer";

en

2. **GRAPEFRUIT INC.**, gevestigd en kantoorhoudend te LaTeX-street 27, 12345 AB, te Alaska, en ingeschreven bij de KvK onder nummer: 01011010, rechtsgeldig vertegenwoordigd door de heer Juicing Joker, hierna verder te noemen: "Opdrachtgever";

hierna gezamenlijk te noemen: "Partijen".

IN AANMERKING NEMENDE:

1. **XERDI** een bedrijf is dat dienstverlening uitvoert met betrekking tot het ontwikkelen, produceren dan wel het uitgeven van software.
2. Opdrachtgever als zelfstandige gebruik wil maken van de diensten van Opdrachtnemer op het gebied van het ontwikkelen, produceren dan wel het uitgeven van software.
3. Partijen met elkaar in overleg zijn getreden over een samenwerking waarbij Opdrachtnemer diensten zal verrichten dan wel zal gebruiken ten gunste van Opdrachtgever, op een door **XERDI** beschikbaar gestelde locatie ten behoeve van Opdrachtgever, dan wel ten behoeve van klanten van Opdrachtgever, en deze afspraken schriftelijk wensen vast te leggen;

komen als volgt overeen: hierna te noemen, de "Overeenkomst":

Artikel 1. Overeenkomst van Opdracht

1.1. Partijen gaan middels deze Overeenkomst een Overeenkomst van opdracht aan zoals deze wordt bedoeld in art. 7:400 BW.

1.2. De opdracht wordt verricht buiten dienstverband en tussen Partijen bestaat geen gezagsverhouding. Met de Overeenkomst wordt geen arbeidsrelatie en derhalve geen arbeidsOvereenkomst in de zin van art. 7:610 BW beoogd en afgesloten. Daarnaast wordt met de Overeenkomst geen fictieve dienstbetrekking in de zin van art. 2b en 2c Uitvoeringsbesluit Loonbelasting 1965 en art. 1 en 5 van Besluit aanwijzing gevallen waarin arbeidsverhouding als dienstbetrekking wordt beschouwd beoogd of afgesloten.

1.3. Opdrachtnemer is bij het uitvoeren van de overeengekomen werkzaamheden geheel zelfstandig. Opdrachtgever heeft geen zeggenschap over de opdracht en werkzaamheden van Opdrachtnemer. Opdrachtnemer verricht de overeengekomen werkzaamheden naar eigen inzicht en zonder toezicht of leiding van de Opdrachtgever.

1.4. In alle gevallen, waarbij ook een Onderhoudsovereenkomst is overeengekomen met **XERDI**, prevaleren de bepalingen uit de Onderhoudsovereenkomst boven de bepalingen van de Overeenkomst van opdracht. Met name wanneer er sprake is van meerwerk.

1.5. Hier volgt een beschrijving van de werkzaamheden te verrichten door de Opdrachtnemer voor de Opdrachtgever:

a) Onderzoeken en (door)ontwikkelen van het administratiesysteem van de Opdrachtgever (Meteor JS & MongoDB).

b) Opstellen en leveren van technische documentatie in PDF.

1 / 5

Figure 6: The resulting assignment agreement

and project values are used in the document's content.

10 Compilation of the setup

It's not easy to refer to a document which doesn't exist yet. To make this work out, I used the *draft* option of the LuaLaTeX compiler. So in order to compile these documents, there needs to be a *draft* version of both agreements. There's also another compilation step, which is done with `bib2gls` for the definitions, which are used in almost all documents. The Bash script in listing 6 eventually does several compilation cycles.

I've used Bash for this specific use case, where I normally would use `latexmk`,[2] so that I can use continuous compilation. However, in this case, `latexmk` will do several unnecessary compilations, which is a bit distracting while changing one of the documents.

[2] `ctan.org/pkg/latexmk`

Listing 6: Bash compilation script

```
OUT_DIR=build

function prepare() {
  for document in "$@"; do
    lualatex --draft --output-directory ↩
      $OUT_DIR $document
  done
}

function compile() {
  for document in "$@"; do
    lualatex --output-directory $OUT_DIR ↩
      $document
  done
}

function defs() {
  for document in "$@"; do
    bib2gls --tex-encoding utf8 --log- ↩
      encoding utf8 --dir $OUT_DIR $document
  done
}

function av() {
  prepare oo ovo
  compile av
  defs av
  compile av
}

function quote() {
  av
  defs oo ovo
  compile oo ovo quote
  defs quote
  compile quote
}

if [ ! -f ./project.yaml ]; then
  echo "Warning: no project.yaml"
  echo "Creating templates instead"
fi

if [ ! -f ./message.md ]; then
  echo "Skipping quote"
  av
else
  quote
fi
```

11 Discussion

11.1 Indirectly used packages

The `regulatory` package in Xerdi's Documentation Project (XDP) helps manage complex documents by making it easy to embed and link related files. It uses the `attachfile2` package to include files, like agreements or notes, directly into a PDF, keeping everything in one place. The `zref` package is used for advanced references, making it easier to link sections and ensure everything connects properly. With the `xr` package, you can reference content from other documents, which is helpful when multiple files are related.

It also partially supports the Markdown format, which is simpler for non-technical users to create text for documents. For sharing documents online, `regulatory` works well with tools like `tex4ht` to convert files into HTML. These features make the package a great tool for creating organized, professional documents that combine multiple elements and are straightforward to manage.

11.2 Markdown support

While I've tried my best to have Markdown as an input method, there are some areas where the simple format is lacking. For the referencing system, you'd like to have labels on list items, which is quite hard. However, Vít Starý Novotný recently added support for `bracketedSpans`,[3] which is quite useful for the referencing in Markdown [8].

Other challenges for Markdown as input are the use of glossaries for definitions and specifying placeholders. In summary, Markdown is quite a nice format, but for the legal document purposes it's questionable if it ends up being more straightforward than LaTeX.

11.3 HTML support

For the following image (Figure 7), I've chosen to demonstrate the documentation of the `regulatory` package, which is a far greater challenge than a contract. The figure illustrates the compatibility with `fmtcount` [10]. The foremost challenge for HTML is the hyperlinks behind the references. However, it's questionable whether the hyperlinks should exist in the first place, since from the terms and conditions perspective it can't know the correct contract to point to.

This specific HTML support for the legal documents is made possible thanks to Michal Hoftich. The actual tool used is `make4ht` [1], which makes it quite easy to execute for my case.

[3] PR: `github.com/Witiko/markdown/pull/528`

4 References

For referencing articles, members, and parts, `zref` is used behind the scenes. All components mentioned in section 2 are configured for this purpose. However, `zref` does not provide as many formatting adjustments as `cleveref`.

`\rref` `\Rref` Due to this limitation, it was decided to develop entirely new variants, including `\rref{`⟨*label*⟩`}`. With `\rref`, one can refer to articles just as is customary for `\section`, `\subsection`, and so on. The `\rref` family consists of a total of four different macros: `\rref` Starting with a lowercase letter and with a hyperlink.

Article	Paragraph	Subparagraph
2	(1)	first

`\rref*` Starting with a lowercase letter and without a hyperlink.

Article	Paragraph	Subparagraph
2	(1)	first

`\Rref` Starting with an uppercase letter and with a hyperlink.

Article	Paragraph	Subparagraph
2	(1)	First

`\Rref*` Starting with an uppercase letter and without a hyperlink.

Article	Paragraph	Subparagraph
2	(1)	First

In the examples above, a notable difference is already apparent in the alternative `\zref`, namely the presentation of the reference number/letter/word and a distinct type in the title. For instance, for `ex1-lid:lorem`, the title is '2.1,' and it is referenced as '(1)'.

`\nref` `\Nref` To reference with the corresponding designation, the macro family `\nref{`⟨*label*⟩`}` has been developed. This family, like `\rref`, has four variants. In the following example, for simplicity, we'll only consider `\Nref`.

	Article	Paragraph	Subparagraph
EN	Article 2	(1)	First subparagraph
NL	Artikel 2	Eerste lid	Onderdeel a

A notable difference with the alternative `\zcref` is that `\nref` can take into account the position of the designation. For example, consider the outcome for the subparagraph (⟨*written ordinal*⟩⟨*designation*⟩).

With the macro families `\rref` and `\nref`, a lot is already possible; however, there are still

Figure 7: An illustration of conversion to HTML

11.4 Music experience

At the conference, I was approached by Julin Shaji, who asked about my experience with using LaTeX for music-related projects. Over the years, I have explored two significant tools in this domain: Ti*k*Z

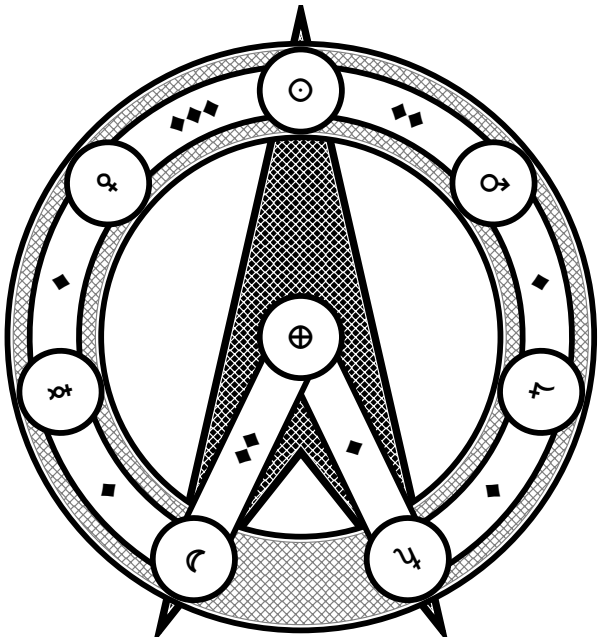


Figure 8: Mode Compass — Sol / Dorian I

for creating illustrative and descriptive musical diagrams, and `lyluatex` [7], which seamlessly integrates LilyPond to generate music scores.

Firstly, I use Ti*k*Z for crafting circular diagrams that visually represent musical scales and relationships. These diagrams are particularly helpful in both educational and research settings as they allow the viewer to grasp complex musical structures at a glance. For instance, Figure 8 shows a "Mode Compass", which illustrates an ancient scale (predating Pythagoras) in a compass-like form. This diagram relies on planetary positions from astrological systems to organize the scale, with the mode centered on the Sun. The diamonds located between the planetary symbols visually represent semitone distances, which provide critical insight into the scale's intervallic structure. Diagrams like these are powerful tools for discussing music theory in a clear and engaging format.

Beyond diagrams, I frequently use the `lyluatex` package to integrate the capabilities of LilyPond for generating music notation directly within LaTeX documents. LilyPond is an advanced program for engraving high-quality music scores, and `lyluatex` serves as a bridge, making it easy to include beautifully formatted sheet music alongside textual descriptions. The following snippet demonstrates an example of LilyPond code:

```
\begin{lilypond}
  \omit Staff.TimeSignature
  \relative c' { \cadenzaOn d e f g a b c d }
\end{lilypond}
```

This LilyPond code generates a music score representing the Greek Dorian scale in the key of D, as depicted in Figure 9. By omitting the time signature, I am able to focus solely on the melodic content. This is an example of how `lyluatex` allows for flexibility in formatting, making it an ideal choice for both educational materials and professional music applications.

To summarize, my approach to LaTeX for music spans two distinct yet complementary areas. Ti*k*Z allows for the creation of visually captivating diagrams that simplify the understanding of abstract musical relationships, while `lyluatex` provides the ability to generate polished music notation directly within the context of a LaTeX document. Together, these tools enable me to tackle a wide range of music-related documentation needs, from theoretical explorations to performance-ready materials.

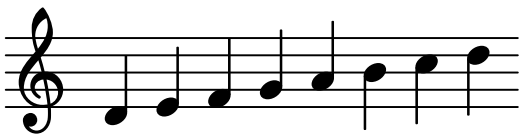


Figure 9: Greek Dorian scale in D

References

[1] M. Hoftich. *The `make4ht` package*, 2025-04-14. A build system for `tex4ht`. `ctan.org/pkg/make4ht`

[2] E. Nijenhuis. *The `lua-placeholders` package*, 2024-04-03. Specifying placeholders for demonstration purposes. `ctan.org/pkg/lua-placeholders`

[3] E. Nijenhuis. *The `regulatory` package*, 2024-09-05. Flexible drafting of legal documents, especially in Dutch. `ctan.org/pkg/regulatory`

[4] E. Nijenhuis. *The `gitinfo-lua` package*, 2024-09-14. Display git project information in your LaTeX projects. `ctan.org/pkg/gitinfo-lua`

[5] H. Oberdiek. *The `attachfile2` package*, 2024-01-20. Attach files into PDF output. `ctan.org/pkg/attachfile2`

[6] H. Oberdiek. *The `zref` package*, 2025-06-08. A new reference scheme for LaTeX. `ctan.org/pkg/zref`

[7] J. Peron, The Gregorio Project. *The `lyluatex` package*, 2023-04-18. Commands to include Lilypond scores within a (Lua)LaTeX document. `ctan.org/pkg/lyluatex`

[8] V. Starý Novotný. *The `markdown` package*, 2025-06-24. Converting and rendering Markdown documents inside TeX. `ctan.org/pkg/markdown`

[9] N. Talbot. *The `bib2gls` package*, 2025-05-15. Command line application to convert `.bib` files to `glossaries-extra.sty` resource files. `ctan.org/pkg/bib2gls`

[10] N. Talbot, V. Belaïche, E. Nijenhuis. *The `fmtcount` package*, 2025-02-09. Display the value of a LaTeX counter in a variety of formats. `ctan.org/pkg/fmtcount`

◇ Erik Nijenhuis
Frans Halsstraat 38
Leeuwarden, 8932JC
The Netherlands
`erik (at) xerdi dot com`
`https://xerdi.com/about?lang=en`