Xerdi's Documentation Project Packaging Guidelines*

This document offers extended guidelines and best practices for LaTeX package development, bundling, submission and version control, within the scope of Xerdi's Documentation Project.

Introduction

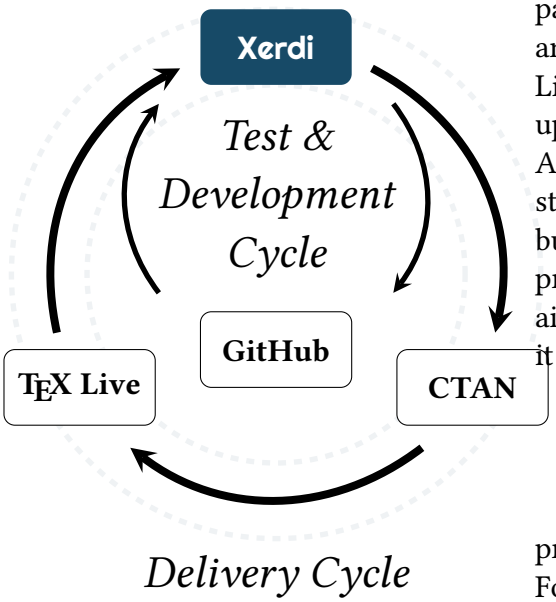

Figure 1: Test & Development- and Delivery Cycle of Xerdi's Documentation Project

Xerdi's Documentation Project (XDP) is part of an Open Source initiative of Xerdi that aims to bring free software to the world of corporate businesses. The guidelines provided by this document state guidelines for working on the XDP project itself or any packages derived from the XDP project.

However, most parts of these guidelines can be of guidance for any LaTeX package development project, since any package should aim for the TeX Live distribution. One must do so by uploading it to the Comprehensive TeX Archive Network (CTAN), as explicitly stated by the TeX Live package contributions [5]. The guidelines will never prevail over their guidelines, but rather aims to extend it, yet not interfere with it in any way.

The guidelines also contain best practices for version control with Git. For collaboration, Xerdi uses GitHub for repository hosting and GitHub Actions for continuous integration and delivery purposes. However, this isn't necessary for every package, as there are plenty of sufficient alternatives, like GitLab with GitLab CI, which would be more cost-efficient for in-house solutions.

*Version 1.0, 2026-09-01.

1 Operating Systems and TEX Distributions

XDP is designed and tested to run exclusively on GNU/Linux-based systems, with a specific focus on Ubuntu 22.04 and RHEL9 environments. This strategic decision aligns with Xerdi's commitment to open-source principles and robust computing platforms. Do note the fact that these distributions are not seen as free distros by the Free Software Foundation (FSF) [2].

The reason for Xerdi to focus on lesser free distributions is the mission to create a bridge between the world of free software and corporate businesses, which typically require robust and secure computing platforms.

However, it's essential to acknowledge that developing a LATEX package on a GNU/Linux platform doesn't limit its compatibility solely to that environment. LATEX by its nature is cross-platform and packages should strive to maintain this characteristic. Therefore, package maintainers should avoid introducing platform-dependent libraries and binaries. Alternatively, if such components are necessary, they should be included in a way that facilitates cross-compilation, ensuring compatibility across different operating systems. For instance, leveraging tools like GCC and Automake can aid in achieving this goal.

2 Programs and Compilers

To ensure consistency and efficiency in package development, all deliverables of a package should be creatable with the `make` program. All documentation sources should be compiled with LuaLATEX. For managing glossaries, acronyms and definitions with BIBTEX, program `bib2gls` [6] is used, which requires Java version 8 or higher. Additionally, the `latexmk` [1] program can be used as a compiler utility.

3 Project Directory Structure

To decide which project directory structure suits best, there are in total three available options to choose from as far as I know, which are:

1. Top-level only; which is the case for most of the LATEX packages I came acros so far. It simply means a single directory and works out of the box thanks to the `TEXMFDOTDIR` variable. This is also the preferred choice of CTAN and TEX Live.
2. Single layered; which became the most relevant for XDP packages in the end. It's suitable for *'large packages'* as stated by the CTAN Team [9], which means *'improved by a single extra layer of directories'* as seen in figure 2.
3. TDS-packaged layout [8]; which

is only suitable for complex packages [5]. The structure simply implies a texmf tree [10], which is normally overkill.

Note that cases one and three should already work from *any* directory if the package directory is added to the `TEXMFAUXTREES` variable. Case two, however, requires an additional configuration in order for it to work, due to its incompatibility with texmf trees (case three).

```
⟨project dir⟩
├── .github
│   └── workflows
│       └── ⟨workflow⟩.yaml
├── doc
│   ├── ⟨package name⟩.tex
│   ├── ⟨package name⟩.pdf
│   ├── init.lua
│   └── latexmkrc
├── scripts
│   ├── ⟨package name⟩.lua
│   └── ⟨package name⟩-⟨class
│       name⟩.lua
├── tex
│   └── ⟨package name⟩.sty
├── .gitignore
├── Makefile
├── README.md
└── LICENSE.txt
```

Figure 2: Project Directory Structure

If you keep all LaTeX projects in a certain directory, let's say `~/src/latex/a-package`, it can be easily configured for all packages in that directory like so:

```
TEXMFDOTDIR = .:~/src/latex//
```

This approach came out as a very generic solution of managing LaTeX package sources locally and was hinted by Karl Berry while discussing previous approaches with both local development practices and CI/CD practices in mind. It can be configured in your custom `texmf.cnf` file, generally found under:

```
/usr/local/texlive/2025/texmf. ↩
  cnf
```

Or you can find it by consulting:

```
kpsewhich -a texmf.cnf
```

Note that the second response always happens to be the configuration shipped by the TeX Live distribution and shouldn't be tampered with.

If you favor to 'whitelist' packages one by one, instead of adding all packages in a certain directory, a single package can be added like so:

```
tlgmr conf auxtrees add \
   ~/src/latex/a-package
```

Or use [... `auxtrees remove` ...] instead, to prevent a package from being found. This way you can manage which development packages are active or not on your local development environment, which in some complex scenarios might be necessary.

4 Documented Sources (.dtx)

Packages whose source is meant to be read alongside the implementation should be authored as `.dtx` files with a

companion `.ins` driver. The `.dtx` carries prose and code together; `docstrip` produces the runnable `.sty`/`.cls` from the same file. The package manual loads the same `.dtx` through `\DocInput`, so the manual and the code can never drift apart.

The package bundle ships *both* the `.dtx` and the generated `.sty`/`.cls`. The `.dtx` preserves the documented sources; the generated files let TeX Live install the package without running `docstrip` on every user's machine. Shipping only the `.dtx` would shift the docstrip step onto every installer, which is exactly the wrong place to do it: the installer is not the author and has no reason to carry a build dependency.

`.dtx` stays in version control, generated files do not

The `.dtx` is the source. The `.sty`/`.cls` are build artefacts of the `.dtx`. Committing both invites silent drift between them; the release pipeline regenerates the artefacts at packaging time, and that is where they belong.

`xdp.ins` should be non-interactive (`\askforoverwritefalse`) so the package's Makefile can regenerate the outputs in a single command, and the generated files should be excluded from version control (they are derived artefacts of the `.dtx`, not sources). The release pipeline regenerates them before bundling.

5 Package Bundling

Packages must be bundled with either `zip` or `tar.gz` extension. XDP prefers the `tar.gz` extension, because `tar` is by default present on GNU/Linux platforms. The archive layout must be one of the layouts described in section 3. An additional `tds.zip` archive should be created, however, not to be included in the toplevel of the package archive as some guidelines may mention. The additional TDS archive is helpful for users that want to manually install the package without the support of the `tlmgr` program. The TDS archive may be included a GitHub release page (see section 9). All archiving procedures should be available as targets in the projects' Makefile. The `doc` directory typically contains auxiliary files after compilation of the manual. These auxiliary files must be excluded in any of the described directory structures in section 3. This can be avoided by either cleaning your output directory with `latexmk -c` ⟨*package name*⟩ after the compilation of the manual or by explicitly picking files with the `tar` command when bundling the package.

6 Package Versioning

Packages must at least have a distinct date and should additionally have a version for every release to come. Despite having a second variable; the version, packages must not create multiple releases on one day. Both the main Lua file, likely called ⟨*package name*⟩`.lua`,

and all TEX related files, like `.cls` or `.sty` files, should have included an updated version and date in its sources, prior to creating a release (see section 7). The version and date must correspond with the administration of the Version Control system (see section 8).

7 Package Submission

Before uploading the package bundle to CTAN, the bundle should be reviewed by a *human* first. For submitting an update to a package, it's best to first browse to the package url on CTAN and secondly hit the upload button. That way, earlier provided information will be automatically filled in, i.e. the suggested CTAN directory. For more information on uploading packages to CTAN, read the upload addendum of CTAN [7].

8 Version Control

All package sources must be versioned with Git for documented change management. All package sources should be held publicly on GitHub as a central point of collaboration. Every commit should be digitally signed either using GPG, SSH or S/MIME by any contributor. Package versions should be managed using Git *signed* tags. The Git tag must point to a revision where hardcoded versions and dates are updated in all applicable sources of the package. Package manuals should address the package versions and dates with `gitinfo-lua` [4], instead of inserting a hardcoded version and date.

9 Continuous Integration and Delivery

Any GitHub Action should use the Docker image originating from `xdp-docker` [3]. The Docker image contains most LATEX packages and some additional binaries. If the Docker image isn't working for your project, you can either contact Xerdi, or fork the repository, make the necessary changes and host it on:

`https://hub.docker.com`

Any GitHub Action should use the appropriate `TEXMFHOME` repository. For XDP packages, this typically is:

`https://github.com/Xerdi/texmf-packaging`

A project should have a 'build' GitHub Action. The build action should trigger for the master/main branch and for every opened or synchronized pull-request. The build action should test the successful creation of the package manual and bundles. The build action should archive the build directory for troubleshooting purposes.

A project should have a 'publish' GitHub Action. The publishing action should trigger for any tag. The publishing action should create all deliverables. The publishing action should create a GitHub Release as a `draft`. Every deliverable should be attached to

the GitHub Release. The GitHub Release message must be reviewed and corrected by the package maintainer. The GitHub Release must either be made publicly or deleted by the package maintainer.

10 Licensing

All package sources must be licensed as Free Software, as described in the TEX Live package contributions [5]. All LATEX related files within XDP and its derived packages are all to be licensed under the LPPLv1.3c license. Either platform-dependent program's sources and scripts should instead be licensed under the GPLv2 license or any higher version of the GPL, because the LPPL license is specifically for LATEX related sources.

References

[1] John Collins, Evan McLean, and David J. Musliner. *The latexmk package. Fully automated LATEX document generation*. Version 4.85. Apr. 7, 2024. URL: https://www.cantab.net/users/johncollins/latexmk/index.html (visited on 04/22/2024).

[2] *Free GNU/Linux Distributions*. Free Software Foundation. URL: https://www.gnu.org/distros/free-distros.html (visited on 05/23/2024).

[3] Erik Nijenhuis. *TEX Live Docker. Dockerfile for creating TEX Live distributions with Ubuntu 22.04*. Xerdi. 2024. URL: https://github.com/Xerdi/xdp-docker (visited on 04/25/2024).

[4] Erik Nijenhuis. *The gitinfo-lua package. Display git project information in your LATEX projects*. Version 1.1.0. 2024. URL: https://ctan.org/pkg/gitinfo-lua (visited on 04/25/2024).

[5] *TEX Live Package Contributions*. URL: https://tug.org/texlive/pkgcontrib.html (visited on 05/23/2024).

[6] Nicola Talbot. *The bib2gls package. Command line application to convert .bib files to glossaries-extra.sty resource files*. Version 3.9. Jan. 30, 2024. URL: https://www.dickimaw-books.com/software/bib2gls/ (visited on 04/22/2024).

[7] The CTAN Team. *Additional Information for CTAN Uploaders*. URL: https://ctan.org/file/help/ctan/CTAN-upload-addendum (visited on 04/23/2024).

[8] The CTAN Team. *Guidelines for uploading TDS-Packaged materials to CTAN*. URL: https://ctan.org/TDS-guidelines (visited on 04/23/2024).

[9] The CTAN Team. *How can I upload a package? Help on CTAN*. URL: https://ctan.org/help/upload-pkg (visited on 04/23/2024).

[10] TUG TWG-TDS. *The tds package. The TEX Directory Structure standard*. Version 1.1. 2004. URL: https://tug.org/tds/ (visited on 04/22/2024).

11 Examples

11.1 Package Manual Example

Listing 1: Example of a package manual file `doc/my-package.tex`

```
% Suitable document classes are ltxdoc and xdpdoc
\documentclass{ltxdoc}
\usepackage[english]{babel}

% Use the titlepage option for setting \author and \date automatically
% Use the authors option in order to base author and date on the last commit
% The rootdir option is an optimization for continuous compilation
% with latexmk's -pvc option
\usepackage[titlepage,authors,rootdir]{gitinfo-lua}

% Decide wether you'd like to use multicols.
% This would only be useful for sections containing a lot of texts, not code
\usepackage{multicol}

\begin{document}
   % Setting the title just before showing it
   \title{My Package Manual%
      % Add a footnote which states package version and date
      \thanks{%
         This manual corresponds to \texttt{my-package}
         version \gitversion\ written on \gitdate.}}%
   \maketitle
   % A summary for this document
   \begin{abstract}
      ...
   \end{abstract}
   % Show the table of contents and macro index in a two-sided column view
   \begin{multicols}{2}
      \tableofcontents \columnbreak \printindex
   \end{multicols}
\end{document}
```

11.2 Latexmkrc Example

Listing 2: Example of a latemk configuration file `doc/latexmkrc`

```
# Use Evince the GNOME Document Viewer as PDF previewer
$pdf_previewer = "start evince";

# Adds the gind.ist style, which is required
$makeindex = "makeindex %O -o %D -s gind.ist %S";

# Used for removing the .bbl file
$bibtex_use = 2;

# Cleanup extra auxiliary files
push @generated_exts, "atfi", "glo", "glstex", "glg";

# Watch bib2gls files
push @file_not_found, '^Package .* No file `([^\\\']*)\\\'';

add_cus_dep('aux', 'glstex', 0, 'run_bib2gls');

sub run_bib2gls {
  my $job = $_[0];

  # First-run / no-glossary guard: if the .aux has no resource directive yet,
  # skip silently so latexmk proceeds (it reruns once the resource is written
    ).
  # This keeps documents without a glossary (or on their first pass) from
  # aborting the build.
  my $aux = "$job.aux";
  return 0 unless -e $aux;
  {
    local $/;
    open my $fh, '<', $aux or return 0;
    my $contents = <$fh>;
    close $fh;
    return 0 unless $contents =~ /\\glsxtr\@resource/;
  }

  my $ret = $silent
    ? system( "bib2gls --silent --group '$job'" )
    : system( "bib2gls --group '$job'" );

  my ($base, $path) = fileparse( $job );
  if ($path && -e "$base.glstex") {
    rename "$base.glstex", "$path$base.glstex";
  }
```

```
  # Track the .bib files bib2gls read so edits trigger a rerun.
  my $glg = "$job.glg";
  if (!$ret && -e $glg) {
    open my $log, '<', $glg or return 0;
    while (<$log>) {
      rdb_ensure_file( $rule, $1 ) if /^Reading (.*\.bib)\s$/;
    }
    close $log;
  }

  # Never abort the build from this cus-dep.
  return 0;
}
```

11.3 Makefile Example

Listing 3: Example of a Makefile `Makefile` (toplevel)

```
CONTRIBUTION = my-package
MANUAL = example-manual
VERSION = "$(shell git describe --tags --always)"

# Let latexmk decide whether to recompile the docs or not
# Also the bundling targets, as you'd typically expect them to be updated
.PHONY: manual tds_archive ctan_upload

all: manual tds_archive ctan_upload

# Either named manual, build or docs is fine
manual:
    cd doc && latexmk -lualatex "$(MANUAL)"
    cd doc && latexmk -c "$(MANUAL)"

# Every TeX directory needs to be post-fixed with:
#   <program>/<package name>
# For example,
#   doc/example-manual.tex
# becomes
#   doc/lualatex/my-package/example-manual.tex
# with the exception for the scripts directory,
# which only takes the package name as post-fix
tds_archive:
```

```
    tar --transform "s,^doc,doc/lualatex/$(CONTRIBUTION)," \
        --transform "s,scripts,scripts/$(CONTRIBUTION)," \
        --transform "s,^tex,tex/lualatex/$(CONTRIBUTION)," \
        -czvf "$(CONTRIBUTION)-$(VERSION).tds.tar.gz" doc scripts tex

# Every directory is given preceded with a dot,
# so that the current working directory (.)
# will be replaced with the package name,
# which must be the first directory level in the tar
ctan_upload:
    tar --transform "s,^\.,$(CONTRIBUTION)," \
        --exclude=doc/Makefile.example \
        -czvf "$(CONTRIBUTION)-$(VERSION).tar.gz" ./doc ./scripts ./tex
```

11.4 Build Action Example

Listing 4: Example of continuous integration `.github/worflows/build.yaml`

```yaml
name: build

on:
  pull_request:
    types: [ opened,synchronize ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Check out the repository
        uses: actions/checkout@v4
        with:
          path: my-package
          fetch-tags: true
          fetch-depth: 0
      - name: Check out texmf
        uses: actions/checkout@v4
        with:
          repository: Xerdi/texmf-packaging
          path: texmf
      - name: Run the build process with Docker
        uses: addnab/docker-run-action@v3
        with:
          image: maclotsen/texlive:with-gf
          shell: bash
          options: --rm -i -v ${{ github.workspace }}/texmf:/root/texmf -v $ ↩
            {{ github.workspace }}:/build
          run: |
            git config --global --add safe.directory /build/my-package
            make -C my-package manual
      - name: Archive Documentation
        uses: actions/upload-artifact@v3
        with:
          name: build
          path: ${{ github.workspace }}/my-package/doc
```

11.5 Publish Action Example

Listing 5: Example of continuous delivery `.github/workflows/publish.yaml`

```yaml
name: publish

on:
  push:
    tags:
      - '*'

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Check out the repository
        uses: actions/checkout@v4
        with:
          path: my-package
          fetch-tags: true
          fetch-depth: 0
      - name: Check out texmf
        uses: actions/checkout@v4
        with:
          repository: Xerdi/texmf-packaging
          path: texmf
      - name: Run the build process with Docker
        uses: addnab/docker-run-action@v3
        with:
          image: maclotsen/texlive:with-gf
          shell: bash
          options: --rm -i -v ${{ github.workspace }}/texmf:/root/texmf -v $ ↩
            {{ github.workspace }}:/build
          run: |
            git config --global --add safe.directory /build/my-package
            make -C my-package all
      - name: Create Release
        id: create_release
        uses: actions/create-release@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          tag_name: ${{ github.ref }}
          release_name: Release ${{ github.ref_name }}
          draft: true
          body: |
```

```
          Release for version ${{ github.ref_name }}
      - name: 'Upload Release Asset: CTAN Upload'
        id: upload_release_asset_ctan_upload
        uses: actions/upload-release-asset@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          upload_url: ${{ steps.create_release.outputs.upload_url }}
          asset_path: ${{ github.workspace }}/my-package/my-package-${{ ↩
            github.ref_name }}.tar.gz
          asset_name: my-package-${{ github.ref_name }}.tar.gz
          asset_content_type: application/gzip
      - name: 'Upload Release Asset: TDS Archive'
        id: upload_release_asset_tds_archive
        uses: actions/upload-release-asset@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          upload_url: ${{ steps.create_release.outputs.upload_url }}
          asset_path: ${{ github.workspace }}/my-package/my-package-${{ ↩
            github.ref_name }}.tds.tar.gz
          asset_name: my-package-${{ github.ref_name }}.tds.tar.gz
          asset_content_type: application/gzip
      - name: 'Upload Release Asset: Manual'
        id: upload_release_asset_manual
        uses: actions/upload-release-asset@v1
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        with:
          upload_url: ${{ steps.create_release.outputs.upload_url }}
          asset_path: ${{ github.workspace }}/my-package/doc/my-package.pdf
          asset_name: my-package-${{ github.ref_name }}.pdf
          asset_content_type: application/pdf
```