The l3doc class – experimental*

The LaTeX Project team†

Released 2026-11-26

Contents

1 Introduction **1**

2 Features of other packages **2**
2.1 The hypdoc package . . . 2
2.2 The docmfp package . . . 2
2.3 The xdoc2 package . . . 2
2.4 The gmdoc package . . . 3

3 Problems & Todo **3**

4 Documentation **3**
4.1 Configuration . . . 3
4.2 Class options . . . 4
4.3 Partitioning documentation and implementation . . . 4
4.4 General text markup . . . 5
4.5 Describing functions in the documentation . . . 6
4.6 Describing functions in the implementation . . . 7
4.7 Keeping things consistent . . . 8
4.8 Documenting templates . . . 8

Index **9**

1 Introduction

Code and documentation for this class have been written prior to the change of doc from version 2 to version 3, which already shows how far behind this class currently is. So take the following warning seriously please:

> **It is much less stable than the main expl3 packages.**
> **Use at own risk!**

*On popular request we now distribute the document for this experimental class. However, please note that it is by no means in final state and is *likely* to undergo modifications, even *incompatible ones*! Thus, using it might therefore require you to do updates, if the class changes.

†https://www.latex-project.org/latex3/

This is an ad-hoc class for documenting the expl3 bundle, a collection of modules or packages that make up LaTeX3's programming environment. Eventually it will replace the ltxdoc class for LaTeX3, but not before the good ideas in hypdoc, xdoc2, docmfp, and gmdoc are incorporated.

It is written as a "self-contained" docstrip file: executing `latex l3doc.dtx` generates the `l3doc.cls` file and typesets this documentation; execute `tex l3doc.dtx` to only generate `l3doc.cls`.

2 Features of other packages

This class builds on the ltxdoc class and the doc package, but in the time since they were originally written some improvements and replacements have appeared that we would like to use as inspiration.

These packages or classes are hypdoc, docmfp, gmdoc, and xdoc. I have summarized them below in order to work out what sort of features we should aim at a minimum for l3doc.

2.1 The hypdoc package

This package provides hyperlink support for the doc package. I have included it in this list to remind me that cross-referencing between documentation and implementation of methods is not very good. (E.g., it would be nice to be able to automatically hyperlink the documentation for a function from its implementation and vice-versa.)

2.2 The docmfp package

- Provides `\DescribeRoutine` and the `routine` environment (etc.) for MetaFont and MetaPost code.
- Provides `\DescribeVariable` and the `variable` environment (etc.) for more general code.
- Provides `\Describe` and the `Code` environment (etc.) as a generalization of the above two instantiations.
- Small tweaks to the DocStrip system to aid non-LaTeX use.

2.3 The xdoc2 package

- Two-sided printing.
- `\NewMacroEnvironment`, `\NewDescribeEnvironment`; similar idea to docmfp but more comprehensive.
- Tons of small improvements.

2.4 The gmdoc package

Radical re-implementation of doc as a package or class.

- Requires no `\begin{macrocode}` blocks!
- Automatically inserts `\begin{macro}` blocks!
- And a whole bunch of other little things.

3 Problems & Todo

Problems at the moment: (1) not flexible in the types of things that can be documented; (2) no obvious link between the `\begin{function}` environment for documenting things to the `\begin{macro}` function that's used analogously in the implementation.

The `macro` should probably be renamed to `function` when it is used within an implementation section. But they should have the same syntax before that happens!

Furthermore, we need another "layer" of documentation commands to account for "user-macro" as opposed to "code-functions"; the expl3 functions should be documented differently, probably, to the ltcmd user macros (at least in terms of indexing).

In no particular order, a list of things to do:

- Rename `function`/`macro` environments to better describe their use.
- Generalize `function`/`macro` for documenting "other things", such as environment names, package options, even keyval options.
- New function like `\part` but for files (remove awkward "File" as `\partname`).
- Something better to replace `\StopEventually`; I'm thinking two environments `documentation` and `implementation` that can conditionally typeset/ignore their material. (This has been implemented but needs further consideration.)
- Hyperlink documentation and implementation of macros (see the DTX file of svn-multi v2 as an example). This is partially done, now, but should be improved.

4 Documentation

4.1 Configuration

Before class options are processed, l3doc loads a configuration file `l3doc.cfg` if it exists, allowing you to customize the behavior of the class without having to change the documentation source files.

For example, to produce documentation on letter-sized paper instead of the default A4 size, create `l3doc.cfg` and include the line

```
\PassOptionsToClass{letterpaper}{l3doc}
```

By default, l3doc selects the `T1` font encoding and loads the Latin Modern fonts. To prevent this, use the class option `cm-default`.

4.2 Class options

The class recognizes a number of options, some of which are generally useful and some of which are aimed squarely at use by the kernel team only.

`full` / `onlydoc` When the `full` option is set (the standard setting), both the documentation and implementation parts of the source are typeset. If on the other hand the `onlydoc` option is set, only the documentation part is typeset.

`lm-default` Selects whether the standard font set up is Latin Modern in the `T1` encoding (the standard setting) or leaves the font setup unchanged.

`kernel` Determines whether l3doc treats internal functions and variables belonging to `kernel` module as allowable in code, for instance `\__kernel_tl_to_str:w`, `\c__kernel_expl_-date_tl`, and `\l__kernel_expl_bool`. In general, *no* internal material from outside the current module is allowed. However, for bootstrapping the expl3 kernel, a small number of cross-module functions are needed. To suppress the error message that would otherwise arise, the class option `kernel` may be given.

`check` When the `check` option is given, the class will record all commands defined and documented in a ⟨*name*⟩`.cmds` file. This will show which are both documented and defined, which are only documented and which are only defined. (Here, "defined" means listed using a `macro` or `variable` environment in the implementation part of the source file).

`checktest` When `checktest` is given as an option, the class will check that each function entry in the implementation part of the source is marked using `\UnitTest`.

`show-notes` / `hide-notes` These complementary options determine if the information given using the `\NB` and `\NOTE` commands is printed.

`cs-break` / `cs-break-nohyphen` The commands `\cmd` and `\cs` allow hyphenation of control sequences after (most) underscores. By default, a hyphen is used to mark the hyphenation, but this can be changed with the `cs-break-nohyphen` class option. To disable hyphenation of control sequences entirely, use `cs-break = false`.

By default, class options

```
full , check = false , checktest = false , lm-default
```

are set.

4.3 Partitioning documentation and implementation

doc uses the `\OnlyDocumentation`/`\AlsoImplementation` macros to guide the use of `\StopEventually{}`, which is intended to be placed to partition the documentation and implementation within a single `.dtx` file.

This isn't very flexible, since it assumes that we *always* want to print the documentation. For the expl3 sources, I wanted to be able to input `.dtx` files in two modes: only displaying the documentation, and only displaying the implementation. For example:

```
\DisableImplementation
\DocInput{l3basics,l3prg,...}
\EnableImplementation
\DisableDocumentation
\DocInputAgain
```

The idea being that the entire expl3 bundle can be documented, with the implementation included at the back. Now, this isn't perfect, but it's a start.

Use `\begin{documentation}...\end{documentation}` around the documentation, and `\begin{implementation}...\end{implementation}` around the implementation. The `\EnableDocumentation`/`\EnableImplementation` causes them to be typeset when the `.dtx` file is `\DocInput`; use `\DisableDocumentation`/`\DisableImplementation` to omit the contents of those environments.

Note that `\DocInput` now takes comma-separated arguments, and `\DocInputAgain` can be used to re-input all `.dtx` files previously input in this way.

4.4 General text markup

Many of the commands in this section come from ltxdoc with some improvements.

`\cmd` `\cmd [⟨options⟩] ⟨control sequence⟩`
`\cs` `\cs [⟨options⟩] {⟨csname⟩}`

These commands are provided to typeset control sequences. `\cmd\foo` produces "`\foo`" and `\cs{foo}` produces the same. In general, `\cs` is more robust since it doesn't rely on catcodes being "correct" and is therefore recommended.

These commands are aware of the `@@` DocStrip syntax and replace such instances correctly in the typeset documentation. This only happens after a `%<@@=⟨module⟩>` declaration.

Additionally, commands can be used in the argument of `\cs`. For instance, `\cs{\meta{name}:\meta{signature}}` produces `\⟨name⟩:⟨signature⟩`.

The ⟨*options*⟩ are a key–value list which can contain the following keys:

- `index=`⟨*name*⟩: the ⟨*csname*⟩ is indexed as if one had written `\cs{`⟨*name*⟩`}`.
- `no-index`: the ⟨*csname*⟩ is not indexed.
- `module=`⟨*module*⟩: the ⟨*csname*⟩ is indexed in the list of commands from the ⟨*module*⟩; the ⟨*module*⟩ can in particular be `TeX` for "TEX and LATEX 2ε" commands, or empty for commands which should be placed in the main index. By default, the ⟨*module*⟩ is deduced automatically from the command name.
- `replace` is a boolean key (`true` by default) which indicates whether to replace `@@` as DocStrip does.

These commands allow hyphenation of control sequences after (most) underscores. By default, a hyphen is used to mark the hyphenation, but this can be changed with the `cs-break-nohyphen` class option. To disable hyphenation of control sequences entirely, use `cs-break = false`.

`\tn` `\tn [⟨options⟩] {⟨csname⟩}`

Analogous to `\cs` but intended for "traditional" TEX or LATEX 2ε commands; they are indexed accordingly. This is in fact equivalent to `\cs [module=TeX, replace=false,` ⟨*options*⟩`] {`⟨*csname*⟩`}`.

\meta `\meta {⟨name⟩}`

`\meta` typesets the ⟨*name*⟩ italicized in ⟨*angle brackets*⟩. Within a `function` environment or similar, angle brackets `<...>` are set up to be a shorthand for `\meta{...}`.

This function has additional functionality over its ltxdoc versions; underscores can be used to subscript material as in math mode. For example, `\meta{arg_{xy}}` produces "⟨arg_{xy}⟩".

\Arg `\Arg {⟨name⟩}`
\marg
\oarg
\parg

Typesets the ⟨*name*⟩ as for `\meta` and wraps it in braces.

The `\marg`/`\oarg`/`\parg` versions follow from ltxdoc in being used for "mandatory" or "optional" or "picture" brackets as per LaTeX 2_ε syntax.

\file `\pkg {⟨name⟩}`
\env
\pkg
\cls

These all take one argument and are intended to be used as semantic commands for representing files, environments, package names, and class names, respectively.

\NB `\NB {⟨tag⟩} {⟨comments⟩}`
\NOTE `\begin{NOTE} {⟨tag⟩}`
`    ⟨comments⟩`
`\end{NOTE}`

Make notes in the source that are not typeset by default. When the `show-notes` class option is active, the comments are typeset in a detokenized and verbatim mode, respectively.

4.5 Describing functions in the documentation

`function` (*env.*)
`variable` (*env.*)
`syntax` (*env.*)

Two heavily-used environments are defined to describe expl3 functions and variables. If describing a variable, use the latter environment; it behaves identically to the `function` environment. Both of the above environments are typically combined with the `syntax` environment, to describe their syntax.

```
\begin{function}{\package_function_one:N, \package_function_two:n}
  \begin{syntax}
    \cs{package_function_one:N} \meta{cs}
    \cs{package_function_two:n} \marg{Argument}
  \end{syntax}
Descriptive text here ...
\end{function}
```

\package_function_one:N `\package_function_one:N ⟨cs⟩`
\package_function_two:n `\package_function_two:n {⟨Argument⟩}`

Descriptive text here ...

Function environments take an optional argument to indicate whether the function(s) it describes are expandable (use `EXP`) or restricted-expandable (use `rEXP`) or defined in

conditional forms (use `TF`, `pTF`, or `noTF`). Note that `pTF` implies `EXP` since predicates must always be expandable, and that `noTF` means that the function without `TF` should be documented in addition to `TF`. For the conditional forms `TF` and `pTF`, the argument of the `function` environment is *not* in fact a command that exists: in the example below, `\tl_if_empty:N` does not exist, but its conditional forms `\tl_if_empty:NT`, `\tl_if_empty:NF`, `\tl_if_empty:NTF` and predicate form `\tl_if_empty_p:N` exist:

```
\begin{function}[pTF]{\tl_if_empty:N, \tl_if_empty:c}
  \begin{syntax}
    \cs{tl_if_empty_p:N} \meta{tl~var}
    \cs{tl_if_empty:NTF} \meta{tl~var} \Arg{true code} \Arg{false code}
  \end{syntax}
  Tests if the \meta{tl~var} is entirely empty
  (i.e., contains no tokens at all).
\end{function}
```

`\tl_if_empty_p:N` ⋆ `\tl_if_empty_p:N` ⟨*tl var*⟩
`\tl_if_empty_p:c` ⋆ `\tl_if_empty:NTF` ⟨*tl var*⟩ {⟨*true code*⟩}
`\tl_if_empty:NTF` ⋆ {⟨*false code*⟩}
`\tl_if_empty:cTF` ⋆ Tests if the ⟨*tl var*⟩ is entirely empty (i.e., contains no tokens at all).

`texnote` (*env.*) This environment is used to call out sections within `function` and similar environments that are only of interest to seasoned TEX developers.

4.6 Describing functions in the implementation

`macro` (*env.*) The well-used environment from LATEX 2ε for marking up the implementation of macros/functions remains the `macro` environment. Some changes in l3doc: it now accepts comma-separated lists of functions, to avoid a very large number of consecutive `\end{macro}` statements. Spaces and new lines are ignored (the option `[verb]` prevents this).

```
% \begin{macro}{\foo:N, \foo:c}
%   \begin{macrocode}
... code for \foo:N and \foo:c ...
%   \end{macrocode}
% \end{macro}
```

If you are documenting an auxiliary macro, it's generally not necessary to highlight it as much and you also don't need to check it for, say, having a test function and having a documentation chunk earlier in a `function` environment. l3doc will pick up these cases from the presence of `__` in the name, or you may force marking as internal by using `\begin{macro}[int]` to mark it as such. The margin call-out is then printed in grey for such cases.

For documenting expl3-type conditionals, you may also pass this environment a `TF` option (and omit it from the function name) to denote that the function is provided with `T`, `F`, and `TF` suffixes. A similar `pTF` option prints both `TF` and `_p` predicate forms. An

option `noTF` prints both the `TF` forms and a form with neither `T` nor `F`, to document functions such as `\prop_get:NN` which also have conditional forms (`\prop_get:NNTF`).

In a very small number of cases, there is no user documentation for a "public" function. In these rare cases, the option `no-user-doc` may be added to suppress the undefined reference that would otherwise then arises.

`\TestFiles` `\TestFiles{`⟨*list of files*⟩`}` is used to indicate which test files are used for the current code; they are printed in the documentation.

`\UnitTested` Within a `macro` environment, it is a good idea to mark whether a unit test has been created for the commands it defines. This is indicated by writing `\UnitTested` anywhere within `\begin{macro}` ... `\end{macro}`.

If the class option `checktest` is enabled, then it is an *error* to have a `macro` environment without a call to `Testfiles`. This is intended for large packages such as expl3 that should have absolutely comprehensive tests suites and whose authors may not always be as sharp at adding new tests with new code as they should be.

`\TestMissing` If a function is missing a test, this may be flagged by writing (as many times as needed) `\TestMissing {`⟨*explanation of test required*⟩`}`. These missing tests are summarized in the listing printed at the end of the compilation run.

`variable` (*env.*) When documenting variable definitions, use the `variable` environment instead. Here it behaves identically to the `macro` environment, except that if the class option `checktest` is enabled, variables are not required to have a test file.

`arguments` (*env.*) Within a `macro` environment, you may use the `arguments` environment to describe the arguments taken by the function(s). It behaves like a modified enumerate environment.

```
% \begin{macro}{\foo:nn, \foo:VV}
% \begin{arguments}
%   \item Name of froozle to be frazzled
%   \item Name of muble to be jubled
% \end{arguments}
%   \begin{macrocode}
... code for \foo:nn and \foo:VV ...
%   \end{macrocode}
% \end{macro}
```

4.7 Keeping things consistent

Whenever a function is either documented or defined with `function` and `macro` respectively, its name is stored in a sequence for later processing.

At the end of the document (i.e., after the `.dtx` file has finished processing), the list of names is analyzed to check whether all defined functions have been documented and vice versa. The results are printed in the console output.

If you need to do more serious work with these lists of names, take a look at the implementation for the data structures and methods used to store and access them directly.

4.8 Documenting templates

The following macros are provided for documenting templates; might end up being something completely different but who knows.

```
\begin{TemplateInterfaceDescription} {⟨template type name⟩}
  \TemplateArgument{none}{---}
OR ONE OR MORE OF THESE:
  \TemplateArgument {⟨arg no⟩} {⟨meaning⟩}
AND
\TemplateSemantics
  ⟨text describing the template type semantics⟩
\end{TemplateInterfaceDescription}

\begin{TemplateDescription} {⟨template type name⟩} {⟨name⟩}
ONE OR MORE OF THESE:
  \TemplateKey {⟨key name⟩} {⟨type of key⟩}
    {⟨textual description of meaning⟩}
    {⟨default value if any⟩}
AND
\TemplateSemantics
  ⟨text describing special additional semantics of the template⟩
\end{TemplateDescription}

\begin{InstanceDescription} [⟨text to specify key column width (optional)⟩]
            {⟨template type name⟩}{⟨instance name⟩}{⟨template name⟩}
ONE OR MORE OF THESE:
  \InstanceKey {⟨key name⟩} {⟨value⟩}
AND
\InstanceSemantics
  ⟨text describing the result of this instance⟩
\end{InstanceDescription}
```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

A

\AlsoImplementation 4
\Arg 6
arguments (env.) 8

C

check (option) 4
checktest (option) 4
\cls 6
\cmd 4, 5
\cs 4, 5
cs-break (option) 4
cs-break-nohyphen (option) 4

D

\Describe 2
\DescribeRoutine 2
\DescribeVariable 2
\DisableDocumentation 5
\DisableImplementation 5
\DocInput 5
\DocInputAgain 5

E

\EnableDocumentation 5
\EnableImplementation 5
\env 6
environments:
  arguments 8
  function 6
  macro 7
  syntax 6
  texnote 7
  variable 6, 8

F

\file 6
\foo 5
full (option) 4
function (env.) 6

H

hide-notes (option) 4

K

kernel (option) 4
kernel internal commands:
\l__kernel_expl_bool 4
\c__kernel_expl_date_tl 4
__kernel_tl_to_str:w 4

L

lm-default (option) 4

M

macro (env.) 7
\marg 6
\meta 6

N

\NB 4, 6
\NewDescribeEnvironment 2
\NewMacroEnvironment 2
\NOTE 4, 6

O

\oarg 6
onlydoc (option) 4
\OnlyDocumentation 4
options:
check 4
checktest 4
cs-break 4
cs-break-nohyphen 4
full 4
hide-notes 4
kernel 4
lm-default 4
onlydoc 4
show-notes 4

P

package commands:
\package_function_one:N 6
\package_function_two:n 6
\parg 6
\pkg 6

S

show-notes (option) 4
\StopEventually 3, 4
syntax (env.) 6

T

\TestFiles 8
\TestFiles 8
\TestMissing 8
\TestMissing 8
TEX and LATEX 2ε commands:
\part 3
\partname 3
texnote (env.) 7
tl commands:
\tl_if_empty:NTF 7
\tl_if_empty_p:N 7
\tn 5

U

\UnitTest 4
\UnitTested 8
\UnitTested 8

V

variable (env.) 6, 8