How to Package Your LATEX Package

Scott Pakin `<scott+dtx@pakin.org>`

21 January 2024

Abstract

This tutorial is intended for advanced LATEX 2$_\varepsilon$ users who want to learn how to create `.ins` and `.dtx` files for distributing their home-brewed classes and style files.

1 Introduction

Requirements We assume that you already know how to *program* in LATEX. That is, you should know how to use `\newcommand`, `\newenvironment`, and preferably a smidgen of TEX. You should also be familiar with "LATEX 2$_\varepsilon$ for Class and Package Writers", which is available from CTAN (`http://www.ctan.org`) and comes with most LATEX 2$_\varepsilon$ distributions in a file called `clsguide.pdf`. Finally, you should know how to install packages that are shipped as a `.dtx` file plus a `.ins` file.

Terminology A *class* (`.cls`) *file* specifies a document's basic formatting: text block size and positioning on the page, typesetting of structural elements such as `\section`, header and footer style, etc. Exactly only class file is used in a single document (with `\documentclass`).

A *style* (`.sty`) *file* is primarily a collection of macro and environment definitions. Style files can override the class file's formatting decisions, provide new functionality, or change existing functionality. A document can load any number of style files (with `\usepackage`).

One or more class or style files (e.g., a main style file that uses `\input` or `\RequirePackage` to load multiple helper files) and their documentation is called a *package*. In the rest of this document, we use the notation

"⟨*package*⟩" to represent the name of your package. Warning: In the LaTeX community, the term "package" is sometimes also used to mean "style file". You may need to discern from context which definition is being used.

Motivation The important parts of a package are the code, the documentation of the code, and the user documentation. Using the Doc and DocStrip programs, it's possible to combine all three of these into a single, *documented LaTeX* (`.dtx`) *file.* The primary advantage of a `.dtx` file is that it enables you to use arbitrary LaTeX constructs to comment your code. Hence, macros, environments, code stanzas, variables, and so forth can be explained using tables, figures, mathematics, and font changes. Code can be organized into sections using LaTeX's sectioning commands. Doc even facilitates generating a unified index that indexes both macro definitions (in the LaTeX code) and macro descriptions (in the user documentation). This emphasis on writing verbose, nicely typeset comments for code—essentially treating a program as a book that describes a set of algorithms—is known as *literate programming* [2] and has been in use since the early days of TeX.

This tutorial will teach you how to write basic `.dtx` files and the `.ins` files that manipulate them. Although there is much overlap with chapter 14 of *The LaTeX Companion* [1], this document is structured as a step-by-step tutorial, while *The LaTeX Companion* is more reference-like. Furthermore, this tutorial shows how to write a single file that serves as both documentation and driver file, which is a more typical usage of the Doc system than using separate files.

2 The `.ins` file

The first step in preparing a package for distribution is to write an *installer* (`.ins`) *file.* An installer file extracts the code from a `.dtx` file, uses DocStrip to strip off the comments and documentation, and outputs a `.sty` file. The good news is that a `.ins` file is typically fairly short and doesn't change significantly from one package to another.

`.ins` files usually start with comments specifying the copyright and license information:

```
%%
%% Copyright (C) ⟨year⟩ ⟨your name⟩
```

```
%%
%% This file may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either
%% version 1.3 of this license or (at your option) any later
%% version.  The latest version of this license is in:
%%
%%    http://www.latex-project.org/lppl.txt
%%
%% and version 1.3c or later is part of all distributions of
%% LaTeX version 2008-05-04 or later.
%%
```

The LaTeX Project Public License (LPPL) is the license under which most packages—and LaTeX itself—are distributed. Of course, you can release your package under any license you want; the LPPL is merely the most common license for LaTeX packages. The LPPL specifies that a user can do whatever he wants with your package—including sell it and give you nothing in return. The only restrictions are that he must give you credit for your work, and he must unambiguously identify modified versions of the code as such to avoid versioning confusion.

The next step is to load DocStrip:

```
\input docstrip.tex
```

`\keepsilent`

By default, DocStrip gives a line-by-line account of its activity. These messages aren't terribly useful, so most people turn them off:

```
\keepsilent
```

`\usedir {⟨directory⟩}`

A system administrator can specify the base directory under which all TeX-related files should be installed, e.g., `/usr/share/texmf`. (See "`\BaseDirectory`" in the DocStrip manual.) The `.ins` file specifies where its files should be installed relative to that. The following is typical:

```
\usedir{tex/latex/⟨package⟩}
```

```
\preamble
⟨text⟩
\endpreamble
```

The next step is to specify a *preamble*, which is a block of commentary that will be written to the top of every generated file:

```
\preamble

This is a generated file.

Copyright (C) ⟨year⟩ ⟨your name⟩

This file may be distributed and/or modified under the
conditions of the LaTeX Project Public License, either
version 1.3 of this license or (at your option) any later
version.  The latest version of this license is in:

   http://www.latex-project.org/lppl.txt

and version 1.3c or later is part of all distributions of
LaTeX version 2008-05-04 or later.

\endpreamble
```

The preceding preamble would cause ⟨*package*⟩`.sty` to begin as follows:

```
%%
%% This is file `⟨package⟩.sty',
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% ⟨package⟩.dtx  (with options: `package')
%%
%% This is a generated file.
%%
%% Copyright (C) ⟨year⟩ ⟨your name⟩
%%
%% This file may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either
%% version 1.3 of this license or (at your option) any later
%% version.  The latest version of this license is in:
%%
```

```
%%    http://www.latex-project.org/lppl.txt
%%
%% and version 1.3c or later is part of all distributions of
%% LaTeX version 2008-05-04 or later.
%%
```

`\generate {\file {⟨style-file⟩} {\from {⟨dtx-file⟩} {⟨tag⟩}}}`

We now reach the most important part of a `.ins` file: the specification of what files to generate from the `.dtx` file. The following tells DocStrip to generate ⟨*package*⟩`.sty` from ⟨*package*⟩`.dtx` by extracting only those parts marked as "`package`" in the `.dtx` file. (Marking parts of a `.dtx` file is described in Section 3.)

```
\generate{\file{⟨package⟩.sty}{\from{⟨package⟩.dtx}{package}}}
```

`\generate` can extract any number of files from a given `.dtx` file. It can even extract a single file from multiple `.dtx` files. See the DocStrip manual for details.

`\Msg {⟨text⟩}`

The next part of a `.ins` file consists of commands to output a message to the user, telling him what files need to be installed and reminding him how to produce the user documentation. The following set of `\Msg` commands is typical:

```
\obeyspaces
\Msg{****************************************************}
\Msg{*                                                  *}
\Msg{* To finish the installation you have to move the  *}
\Msg{* following file into a directory searched by TeX: *}
\Msg{*                                                  *}
\Msg{*     ⟨package⟩.sty                                *}
\Msg{*                                                  *}
\Msg{* To produce the documentation run the file        *}
\Msg{* ⟨package⟩.dtx through LaTeX.                     *}
\Msg{*                                                  *}
\Msg{* Happy TeXing!                                    *}
\Msg{*                                                  *}
\Msg{****************************************************}
```

Note the use of `\obeyspaces` to inhibit TeX from collapsing multiple spaces into one.

`\endbatchfile`

Finally, we tell DocStrip that we've reached the end of the `.ins` file:

```
\endbatchfile
```

Appendix A.1 lists a complete, skeleton `.ins` file. Appendix A.2 is similar but contains slight modifications intended to produce a class (`.cls`) file instead of a style (`.sty`) file.

3 The `.dtx` file

A `.dtx` file contains both the commented source code and the user documentation for the package. Running a `.dtx` file through `latex` typesets the user documentation, which usually also includes a nicely typeset version of the commented source code.

Due to some Doc trickery, a `.dtx` file is actually evaluated *twice*. The first time, only a small piece of LaTeX driver code is evaluated. The second time, *comments* in the `.dtx` file are evaluated, as if there were no "%" preceding them. This can lead to a good deal of confusion when writing `.dtx` files and occasionally leads to some awkward constructions. Fortunately, once the basic structure of a `.dtx` file is in place, filling in the code is fairly straightforward.

3.1 Prologue

`.dtx` files generally begin with a copyright and license comment:

```
% \iffalse meta-comment
%
% Copyright (C) ⟨year⟩ ⟨your name⟩
%
% This file may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either
% version 1.3 of this license or (at your option) any later
% version.  The latest version of this license is in:
%
%    http://www.latex-project.org/lppl.txt
%
% and version 1.3c or later is part of all distributions of
% LaTeX version 2008-05-04 or later.
%
% \fi
```

The `\iffalse` and `\fi` are needed because the second time the `.dtx` file is processed, `%` characters at the beginning of lines are ignored. To prevent the copyright/license from being evaluated as LaTeX code, we have to surround it with `\iffalse`…`\fi`. Adding “`meta-comment`” after “`\iffalse`” is nothing more than a convention for indicating that the comment is intended to be read by a human, not by Doc, DocStrip, or LaTeX.

```
\NeedsTeXFormat {⟨format-name⟩} [⟨release-date⟩]
\ProvidesPackage {⟨package-name⟩} [⟨release-info⟩]
```

The next few lines are also surrounded by `\iffalse`…`\fi` so as not to be processed by `latex` on the second pass through the `.dtx` file. However, these lines are intended not for a human reader, but for DocStrip (hence, no “`meta-comment`”):

```
% \iffalse
%<package>\NeedsTeXFormat{LaTeX2e}[2023-11-01]
%<package>\ProvidesPackage{⟨package⟩}
%<package>    [⟨YYYY⟩-⟨MM⟩-⟨DD⟩ v⟨version⟩ ⟨description⟩]
%
```

(We’ll encounter the `\fi` shortly.)

Remember the `\generate` line in the `.ins` file (page 5)? It ended with the tag “`package`”. This tells DocStrip to write lines that begin with “`%<package>`” to the `.sty` file, stripping off the “`%<package>`” in the process. Hence, our `.sty` file will include the following code right after the header comments:

```
\NeedsTeXFormat{LaTeX2e}[2023-11-01]
\ProvidesPackage{⟨package⟩}
   [⟨YYYY⟩-⟨MM⟩-⟨DD⟩ v⟨version⟩ ⟨description⟩]
```

For example:

```
\NeedsTeXFormat{LaTeX2e}[2023-11-01]
\ProvidesPackage{skeleton}
   [2002-03-25 v1.0 .dtx skeleton file]
```

The `\NeedsTeXFormat` line ensures that the package won't run under a TeX format other than LaTeX 2ε. The optional date argument causes `latex` to output a warning message if the version of LaTeX 2ε (which can be found via LaTeX 2ε's `\fmtversion` macro) is older than what the package was tested with:

```
LaTeX Warning: You have requested release `2239-09-30' of LaTeX,
               but only release `2023-11-01' is available.
```

The date and version strings in the `\ProvidesPackage` line are used by Doc to set the `\filedate` and `\fileversion` macros. Note the date format; *YYYY-MM-DD* is used throughout LaTeX 2ε and should be used in your packages as well.[1]

```
\EnableCrossrefs
\CodelineIndex
\RecordChanges
\DocInput {⟨filename⟩}
```

Next comes the only part of the `.dtx` file that isn't commented out (i.e., doesn't begin each line with "%"):

```
%<*driver>
\documentclass{ltxdoc}
\usepackage{⟨package⟩}
\EnableCrossrefs
\CodelineIndex
\RecordChanges
\begin{document}
  \DocInput{⟨package⟩.dtx}
\end{document}
%</driver>
% \fi
```

---

[1] Older versions of LaTeX 2ε expected the *YYYY/MM/DD* format, but the latest package/class author guide [3] specifies that *YYYY-MM-DD* should be used from now on.

The preceding code stanza is what `latex` evaluates on its first pass through the `.dtx` file. We'll now examine that stanza line-by-line:

1. Putting code between "`%<*driver>`" and "`%</driver>`" is a DocStrip shorthand for prefixing each line with "`%<driver>`". This demarcates the Doc driver code.
2. The `\documentclass` should almost always be `ltxdoc` as that loads Doc and provides a few useful macros for formatting program documentation.
3. You should always `\usepackage` your style file. If you don't, Doc won't see the package's `\ProvidesPackage` line and won't know how to set `\filedate` and `\fileversion` (see page 11). This is also where you should `\usepackage` any other packages needed to typeset the user documentation (as opposed to packages needed by your package).
4. `\EnableCrossrefs` tells Doc that you want it to construct an index for your code—normally a good idea. The alternative is `\DisableCrossrefs`, which speeds up processing by a negligible amount.
5. `\CodelineIndex` tells Doc that the index should refer to program line numbers instead of page numbers. (The alternative is `\PageIndex`.) `\CodelineIndex` makes index entries easier to find at the expense of making the index less self-consistent (because descriptions of macros and environments are always indexed by page number). The index does, however, begin with a note of explanation.
6. Page 10 discusses how to log the changes made in each revision of the package. `\RecordChanges` tells Doc that it should keep and aggregate the log entries.
7. There should be only one command between the `\begin{document}` and `\end{document}`: a `\DocInput` call with which the `.dtx` file inputs itself. This enables a master file to `\DocInput` multiple files in order to produce a single document that covers more than one package but contains a unified index. Master documentation files are described on page 22.

`\OnlyDescription`

Another command that sometimes appears in the preamble (i.e., before the `\begin{document}`) is `\OnlyDescription`, which tells `Doc` to typeset only the user documentation, not the package code/comments. It's usually best to omit `\OnlyDescription` (or add it commented out). A user always can add it manually or even enable `\OnlyDescription` for *all* `.dtx` files by adding the following to his `ltxdoc.cfg` file:

```
\AtBeginDocument{\OnlyDescription}
```

The remainder of this section covers `latex`'s second pass through the `.dtx` file. Consequently, all subsequent examples are prefixed with percent signs.

`\changes` {⟨*version*⟩} {⟨*date*⟩} {⟨*description*⟩}

On page 9 we learned that `Doc` has a mechanism for recording changes to the package. The command is "`\changes{`⟨*version*⟩`}{`⟨*date*⟩`}{`⟨*description*⟩`}`", and it's common to use `\changes` for the initial version of the package to log the package's creation date:

```
% \changes{v1.0}{2002/03/25}{Initial version}
```

One nice feature of the `\changes` command is that it knows whether it was used internally to a macro/environment definition. As Figure 1 shows, top-level changes are prefixed with "General:", and internal changes are prefixed with the name of the enclosing macro or environment.

Change History

v1.0
    General: Top-level comment 1
v1.2j
    `myMacro`: Internal macro comment 5

Figure 1: Sample change history

```
\GetFileInfo {⟨style-file⟩}
\filedate
\fileversion
\fileinfo
```

Next, we tell Doc to parse the `\ProvidesPackage` command (page 7), calling the three components of `\ProvidesPackage`'s argument, respectively, `\filedate`, `\fileversion`, and `\fileinfo`:

```
% \GetFileInfo{⟨package⟩.sty}
```

For instance, the `\ProvidesPackage` example shown on page 8 would be parsed as follows:

`\filedate` ≡ 2002-03-25
`\fileversion` ≡ v1.0
`\fileinfo` ≡ .dtx skeleton file

```
\DoNotIndex {⟨macro-name , ...⟩}
```

When producing an index, Doc normally indexes *every* control sequence (i.e., backslashed word or symbol) in the code. The problem with this level of automation is that many control sequences are uninteresting from the perspective of understanding the code. For example, a reader probably doesn't want to see every location where `\if` is used—or `\the` or `\let` or `\begin` or any of numerous other control sequences.

As its name implies, the `\DoNotIndex` command gives Doc a list of control sequences that should not be indexed. `\DoNotIndex` can be used any number of times, and it accepts any number of control sequence names per invocation:

```
% \DoNotIndex{\#,\$,\%,\&,\@,\\,\{,\},\^,\_,\~,\ }
% \DoNotIndex{\@ne}
% \DoNotIndex{\advance,\begingroup,\catcode,\closein}
% \DoNotIndex{\closeout,\day,\def,\edef,\else,\empty,\endgroup}
```

⋮

3.2 User documentation

We finally can start writing the user documentation. A typical beginning looks like this:

```
% \title{The \textsf{⟨package⟩} package\thanks{This document
%   corresponds to \textsf{⟨package⟩}~\fileversion,
%   dated~\filedate.}}
% \author{⟨your name⟩ \\ \texttt{⟨your e-mail address⟩}}
%
% \maketitle
```

The title certainly can be more creative, but note that it's common for package names to be typeset with `\textsf` and for `\thanks` to be used to specify the package version and date. This yields one of the advantages of literate programming: Whenever you change the package version (the optional second argument to `\ProvidesPackage`), the user documentation is updated accordingly. Of course, you still have to ensure manually that the user documentation accurately describes the updated package.

Write the user documentation as you would any LaTeX document, except that you have to precede each line with a "`%`". Note that the `ltxdoc` document class is derived from `article` so the top-level sectioning command is `\section`, not `\chapter`.

```
\DescribeMacro {⟨macro⟩}
\DescribeEnv {⟨environment⟩}
```

Doc provides a couple of commands to help format user documentation. If you include "`\DescribeMacro{`⟨*macro*⟩`}`"[2] within a paragraph, Doc will stick "⟨*macro*⟩" in the margin to make it easy for a reader to see. Doc also will add ⟨*macro*⟩ to the index and format the corresponding page number to indicate that this is where the macro is described (as opposed to the place in the source code where the macro is defined).

`\DescribeEnv` is the analogous command for describing an environment. Both `\DescribeMacro` and `\DescribeEnv` can be used multiple times within a paragraph.

```
\NewDocElement [⟨options⟩] {⟨element-name⟩} {⟨env-name⟩}
```

While LaTeX documentation most commonly documents macros and environments, `\NewDocElement` facilitates documenting arbitrary package components. It defines a `\Describe`⟨*element-name*⟩ macro analogous to `\DescribeMacro` and `\DescribeEnv` above. It then lets you

---

[2] "⟨*macro*⟩" should include the backslash.

use `\begin{`⟨*env-name*⟩`}`...`\end{`⟨*env-name*⟩`}` to demarcate instances of your new element, analogously to `\begin{macro}`...`\end{macro}` and `\begin{environment}`...`\end{environment}`, which are described on page 16. The ⟨*options*⟩ argument specifies, among other things, whether the new element is `macrolike` (begins with a backslash) or `envlike` (does not begin with a backslash).

`\marg {`⟨*argument*⟩`}`
`\oarg {`⟨*argument*⟩`}`
`\parg {`⟨*argument*⟩`}`
`\meta {`⟨*text*⟩`}`

The `ltxdoc` document class provides three commands to help typeset macro and environment syntax (Table 1). `\marg` formats mandatory arguments, `\oarg` formats optional arguments, and `\parg` formats picture arguments. All three of these utilize `\meta` to typeset the argument proper. `\meta` also is useful on its own. For example, "`This needs a \meta{dimen}`." is typeset as "This needs a ⟨*dimen*⟩."

Table 1: Argument-formatting commands

Command	Result
`\marg{text}`	`{`⟨*text*⟩`}`
`\oarg{text}`	`[`⟨*text*⟩`]`
`\parg{text}`	`(`⟨*text*⟩`)`

In addition to those commands, Doc facilitates the typesetting of macro descriptions by automatically loading the `shortvrb` package. `shortvrb` lets you use `|`...`|` as a convenient shorthand for `\verb|`...`|`. For instance, "`|\mymacro| \oarg{pos} \marg{width} \marg{text}`" is typeset as follows:

`\mymacro [`⟨*pos*⟩`] {`⟨*width*⟩`} {`⟨*text*⟩`}`

Like `\verb`, the `|`...`|` shorthand does not work within `\footnote` or other fragile macros.

3.3 Code and commentary

```
\MaybeStop {⟨text⟩}
\Finale
```

The package's source code is delineated by putting it between `\MaybeStop`[3] and `\Finale`. `\MaybeStop` takes an argument, which is a block of text to typeset after the code. If `\OnlyDescription` (page 10) is specified, then nothing after the `\MaybeStop` will be output—including text that follows `\Finale`. `\MaybeStop`'s ⟨*text*⟩ parameter is therefore the mechanism for providing a piece of text that should be output regardless of whether or not a code listing is typeset. It commonly includes a bibliography section and/or one or both of the following two commands:

```
\PrintChanges
\PrintIndex
```

`\PrintChanges` produces an unnumbered section called "Change History". (See Figure 1 on page 10.) The Change History section aggregates all of the `\changes` commands in the `.dtx` file into a single list of per-version modifications. This makes it easy to keep track of what changed from version to version.

`\PrintChanges` uses LaTeX's glossary mechanism. Running `latex` on ⟨*package*⟩`.dtx` produces change-history data in ⟨*package*⟩`.glo`. To produce the typeset change history (⟨*package*⟩`.gls`), the user should run the `makeindex` program as follows:

```
makeindex -s gglo.ist -o ⟨package⟩.gls ⟨package⟩.glo
```

`\PrintIndex` produces an unnumbered section called "Index". The index automatically includes entries for all macros and environments that are used, defined, or described in the document. All environments additionally are listed under "environments". Table 2 illustrates the way that various entries are formatted. In that table, "27" refers to a page number, and "123" refers to a line number.[4] Note that macro/environment definitions and uses are included in the index only if the document includes a code listing (i.e., if `\OnlyDescription` was not specified).

---

[3] In older versions of Doc, `\MaybeStop` was called `\StopEventually`. The latter remains defined but is deprecated.

[4] If `\CodelineIndex` (page 8) were not used then "123" would refer to a page number.

Table 2: Formatting of entries in the index

Item	Function	Formatting in index
Macro	Used	`\myMacro` 123
Macro	Defined	`\myMacro` 123
Macro	Described	`\myMacro` *27*
Environment	Defined	`myEnv` (environment) 123
Environment	Described	`myEnv` (environment) *27*
Explicit `\index`	—	myItem 27

The default formatting for an explicit `\index` command uses a roman page number. This leads to confusion, as roman page numbers otherwise indicate line numbers in the package source code. The solution is to specify "`usage`" formatting to the `\index` command, which typesets the page number in italics:

```
\index{explicit indexing|usage}
```

Running `latex` on ⟨*package*⟩`.dtx` produces index data in ⟨*package*⟩`.idx`. To produce the typeset index (⟨*package*⟩`.ind`), the user should run the `makeindex` program as follows:

```
makeindex -s gind.ist -o ⟨package⟩.ind ⟨package⟩.idx
```

A code index is a nice "value added" made possible by literate programming. It requires virtually no extra effort and greatly helps code maintainers find macro definitions and see what other macros a package depends upon.

```
\begin{macrocode}
⟨code⟩
\end{macrocode}
```

Code fragments listed between `\begin{macrocode}` and `\end{macrocode}` are extracted verbatim into the `.sty` file. When typeset, the code fragments are shown with a running line counter to make it easy to refer to a specific line. Here are some key points to remember about the `macrocode` environment:

1. There must be *exactly* four spaces between the "`%`" and the "`\begin{macrocode}`" or "`\end{macrocode}`". Otherwise, Doc won't

detect the end of the code fragment.[5]

2. The lines of code within `\begin{macrocode}`...`\end{macrocode}` should not begin with "`%`". The code gets written exactly as it is to the `.ins` file, with no `%`-stripping.

The following is a sample code fragment. It happens to be a complete macro definition, but this is not necessary; any fragment of LaTeX code can appear within a `macrocode` environment.

```
%    \begin{macrocode}
\newcommand{\mymacro}{This is
  a \LaTeX{} macro.}
%    \end{macrocode}
```

Doc formats the preceding code fragment as follows:

```
1 \newcommand{\mymacro}{This is
2   a \LaTeX{} macro.}
```

Note that line numbers are unique across the entire program (as opposed to being reset at the top of each page). If `\PrintIndex` is used in the `.dtx` file containing the preceding definition of `\mymacro`, the index will automatically include entries for `\newcommand`, `\mymacro`, and `\LaTeX`, unless any of these are `\DoNotIndex`'ed.

```
\begin{macro}{⟨macro⟩}
    ⋮
\end{macro}

\begin{environment}{⟨environment⟩}
    ⋮
\end{environment}
```

---

[5] Trivia: Only the `\end{macrocode}` needs this precise spacing and then only for typesetting the documentation. Nevertheless, it's good practice to use "`%␣␣␣␣`" for the `\begin{macrocode}`, as well.

The `macro` and `environment` environments are used to delineate a complete macro or environment definition. `macro`/`environment` environments generally contain one or more `macrocode` environments interspersed with code documentation. The following is a more complete version of the `macrocode` example shown on the preceding page.

```
% \begin{macro}{\mymacro}
% We define a trivial macro, |\mymacro|, to illustrate
% the use of the |macro| environment.
%    \begin{macrocode}
\newcommand{\mymacro}{This is
  a \LaTeX{} macro.}
%    \end{macrocode}
% \end{macro}
```

The typeset version is shown below:

`\mymacro` We define a trivial macro, `\mymacro`, to illustrate the use of the `macro` environment.

```
\newcommand{\mymacro}{This is
  a \LaTeX{} macro.}
```

Doc typesets the macro/environment name in the margin for increased visibility. Doc also adds the appropriate entries to the index. (See Table 2 on page 15 for examples of how these entries are formatted.) Note that `\begin{macro}`...`\end{macro}` is not required to indicate a macro definition. It can also be used to indicate definitions of other control sequences such as counters, lengths, and boxes:

```
% \begin{macro}{myCounter}
% This is an example of using the |macro| environment to format
% something other than a macro.
%    \begin{macrocode}
\newcounter{myCounter}
%    \end{macrocode}
% \end{macro}
```

`macro` and `environment` environments can be nested. This capability is useful not only for macros that define other macros, but also when defining a group of related datatypes that share a description:

```
% \begin{macro}{\thingheight}
% \begin{macro}{\thingwidth}
% \begin{macro}{\thingdepth}
% These lengths keep track of the dimensions of our |\thing|
% box.  (Actually, we're just trying to show how to nest
% |macro| environments.)
%    \begin{macrocode}
\newlength{\thingheight}
\newlength{\thingwidth}
\newlength{\thingdepth}
%    \end{macrocode}
% \end{macro}
% \end{macro}
% \end{macro}
```

As a convenience, recent versions of Doc enable a single `\begin{macro}` to list more than one macro to be defined:

```
% \begin{macro}{\thingheight,\thingwidth,\thingdepth}
% These lengths keep track of the dimensions of our |\thing|
% box.  (Actually, we're just trying to show how to nest
% |macro| environments.)
%    \begin{macrocode}
\newlength{\thingheight}
\newlength{\thingwidth}
\newlength{\thingdepth}
%    \end{macrocode}
% \end{macro}
```

Descriptionless `macro` environments generally should be avoided, as the formatting is a little ugly: the macro name appears on its own line, to the left of an "empty" description, but the code doesn't start until the next line.

There can be multiple `macrocode` environments within a `\begin{macro}`... `\end{macro}` or `\begin{environment}`... `\end{environment}` block. This is the mechanism by which code can be commented internally to a macro/environment. (It's considered bad style to use "`%`" for comments within a `macrocode` block.) Here's an example of the way that a nontrivial macro might be commented:

```
% \begin{macro}{\complexMacro}
% Pretend that this is a very complex macro that needs
% to have its various pieces documented.
%    \begin{macrocode}
\newcommand{\complexMacro}{%
%    \end{macrocode}
% Initialize all of our counters to zero.
%    \begin{macrocode}
  \setcounter{count@i}{0}%
  \setcounter{count@ii}{0}%
  \setcounter{count@iii}{0}%
  \setcounter{count@iv}{0}%
%    \end{macrocode}
% \begin{macro}{\helperMacro}
% Define a helper macro for |\complexMacro| to use.
%    \begin{macrocode}
  \def\helperMacro#1,#2,\relax{%
    \someOtherMacro{#1}{#2}%
  }%
%    \end{macrocode}
% Do some really complicated processing.
%    \begin{macrocode}

                ⋮

%    \end{macrocode}
% We're all finished now.
%    \begin{macrocode}
}
%    \end{macrocode}
% \end{macro}
% \end{macro}
```

Note how the above defines `\helperMacro` within `\complexMacro`. The document introduces `\helperMacro` with a nested `\begin{macro}` that ends, as is customary, with an `\end{macro}` placed before the outer macro's `\end{macro}` as opposed to immediately after `\helperMacro's` final “`}`”.

Appendix A.3 lists a complete, skeleton `.dtx` file that encapsulates a `.sty` file and its documentation.

Class files The procedure to produce a class file from a `.dtx` file is far less straightforward than the procedure to produce a style file. The problem is that `\DocInput` relies on the `\usepackage{`⟨*package*⟩`}` line (more precisely, the

`\ProvidesPackage` line within ⟨*package*⟩`.sty`) to set the `\fileversion` and `\filedate` macros. However, a class file can't be loaded with `\usepackage`. Nor can we simply load it with `\documentclass{`⟨*package*⟩`}` because only one class can be loaded per document and we need that class to be `ltxdoc`.

The solution is to use `\ProvidesFile` to make the file version and date available to the `.dtx` file. Appendix A.4 lists a complete, skeleton `.dtx` file that encapsulates a `.cls` file and its documentation. It resembles the skeleton file shown in Appendix A.3 but has a differently structured header section.

4 Tips, tricks, and recommendations

- Write lots of good documentation! It really helps others understand your code and the package as a whole.
- If you believe the LaTeX community at large would be interested in your package then you should upload it to CTAN at `http://www.ctan.org/upload`. As a central repository of all things TeX-related, CTAN makes it easier for others to find your LaTeX package than if it were located on your personal home page.
- When distributing your package, be sure to include a `README` file describing what your package does as well as *prebuilt* documentation, preferably as a PDF file. Prebuilt documentation saves users the bother of having to download your package, install it, and build the documentation before even knowing what the package is supposed to do or if it meets their needs.
- Use LaTeX's sectioning commands to organize the code and clarify its structure (e.g., `\subsection{Initialization macros}`, `\subsection{Helper functions}`, `\subsection{Exported macros and environments}`, . . .).
- Although commentary really belongs only in the typeset documentation, it is also possible to write comments that are visible only in the `.sty` file, in both the typeset documentation and the `.sty` file, or only in the `.dtx` source. Table 3 shows how to control comment visibility.
- All lines between `<*package>` and `</package>`, except those within a `macrocode` environment, should begin with "`%`". Don't use any blank lines; these would get written to the `.sty` file (and oughtn't).

- It is good practice for LaTeX programs to use "`@`" within the names of macros, lengths, counters, etc. that are declared globally, but intended to be used only internally to the package. This prevents a user from corrupting package state by inadvertently redefining package internals.[6] Another good practice is to prefix all global names that are internal to the package with the name of the package (e.g., "`\`⟨*package*⟩`@thing`" instead of "`\@thing`" or—even worse—just "`\thing`"). This helps avoid inter-package naming conflicts. Finally, because decimal digits are not normally allowed in macro names, it is common to use roman numerals instead, for example: `\arg@i`, `\arg@ii`, `\arg@iii`, `\arg@iv`, etc.

- You can use `\index` in the normal way to index things other than macros and environments.

- Because macro names can be long, consider using the `idxlayout` package to reduce the number of columns in the index. (It provides control over other aspects of index formatting, as well.)

- If you use Emacs as your text editor, try out `swiftex.el`'s `doctex-mode`, an Emacs mode designed specifically for writing `.dtx` files. `swiftex.el` is available from CTAN.

  As a more primitive alternative, look up Emacs's `string-rectangle` and `kill-rectangle` commands. These help a great deal with adding and removing a "`%`" at the beginning of every line in a region.

Table 3: Comment visibility

Appears in docs	Appears in `.sty`	Mechanism
N	N	`% ^^A` ⟨*comment*⟩

N	Y	`% \iffalse`
		`%%` ⟨*comment*⟩
		`% \fi`

Y	N	`%` ⟨*comment*⟩
Y	Y	`%%` ⟨*comment*⟩

---

[6] Within a LaTeX document, "`@`" is set to category code 12 ("other"), not category code 11 ("letter"), so the user can't easily define or use a macro with "`@`" in its name.

- Be sure to read "The DocStrip Program" and "The `doc` and `shortvrb` Packages", the documentation for DocStrip and Doc, respectively (provided in `.dtx` format, of course). These explain how to do more advanced things with `.ins` and `.dtx` files than this tutorial covered. Some advanced topics include the following:
  - Extracting multiple `.sty` files from a single `.dtx` file.
  - Putting different preambles in different `.sty` files.
  - Extracting something other than a `.sty` file (e.g., a configuration file or Python code) from a `.dtx` file.
  - Changing the formatting of the typeset documentation.

5 Advanced packaging techniques

This section describes various bits of wizardry that can be accomplished with Doc and DocStrip. Few packages require these techniques but they are included here for convenient reference.

5.1 Master documentation files

Doc supports "master" documentation files that typeset multiple `.dtx` files. The advantage is that a set of related `.dtx` files can be typeset with continuous section numbering and a single, unified index. In fact, the LaTeX 2_ε source code itself is typeset using a master document (`source2e.tex`) that includes all of the myriad `.dtx` files that comprise LaTeX 2_ε.

To help produce master documents, the `ltxdoc` class provides a command called "`\DocInclude`". `ltxdoc`'s `\DocInclude` is much like Doc's `\DocInput`—it even uses it internally—but has the following additional features.

- `\PrintIndex` is automatically handled properly.
- Every `\DocInclude`'d file is given a title page.
- `\tableofcontents` works as expected. `.dtx` filenames are used as "chapter" names.

Note that `\DocInclude`, unlike `\DocInput`, assumes a `.dtx` extension.

Appendix A.5 presents a master-document skeleton that uses `\DocInclude` to typeset ⟨*file1*⟩`.dtx`, ⟨*file2*⟩`.dtx`, and ⟨*file3*⟩`.dtx` as a single document. If you prefer a more manual approach (e.g., if you dislike `\DocInclude`'s per-file title pages), you can still use `\DocInput`. Just make sure to redefine `\PrintIndex` to do nothing; otherwise, each file will get its own index. After all of the `.dtx` files have been typeset, call the original `\PrintIndex` command to print a unified index:

```
\begin{document}
  \let\origPrintIndex=\PrintIndex \let\PrintIndex=\relax
  \DocInput{⟨file1⟩.dtx}
  \DocInput{⟨file2⟩.dtx}
  \DocInput{⟨file3⟩.dtx}
  \origPrintIndex
\end{document}
```

5.2 Single-file package distributions

Although LaTeX packages are typically distributed as both a `.ins` and a `.dtx` file, it is possible to distribute a package as a single file. The trick is to include the entire `.ins` at the top of the `.dtx` file, right after the `%`⟨*package*⟩ lines:

```
%<*batchfile>
\begingroup
        ⋮
⟨Entire contents of the .ins file⟩
        ⋮
\endgroup
%</batchfile>
```

Omit the `\endbatchfile` to allow LaTeX to continue on with the rest of the `.dtx` file. Also, to avoid the "`File` ⟨*sty-file*⟩ `already exists on the system. Overwrite it? [y/n]`" message you can put "`\askforoverwritefalse`" before the first `\generate` command. (This will automatically overwrite the existing `.sty` file. Wrapping the `\generate` command(s) within “`\IfFileExists{⟨sty-file⟩}{}{...}`” will suppress the overwriting.) You should also move the `.sty` installation instructions to the end of the `.dtx` file so they don’t scroll off the user’s screen. You’ll need to use `\typeout` as `\Msg` won’t be defined:

```
% \Finale
%
% \typeout{**************************************************}
% \typeout{*}
% \typeout{* To finish the installation you have to move the}
% \typeout{* following file into a directory searched by TeX:}
% \typeout{*}
% \typeout{* \space\space skeleton.sty}
% \typeout{*}
% \typeout{* Documentation is in skeleton.dvi.}
% \typeout{*}
% \typeout{* Happy TeXing!}
% \typeout{**************************************************}
\endinput
```

5.3 Class and style files with shared versioning information

Some packages contain both a `.cls` and `.sty` file. It may be desirable to have these extracted from the same `.dtx` file and share the same versioning string. The DocStrip documentation explains how to extract multiple files from a single `\generate` call:

```
\generate{\file{⟨package⟩.cls}{\from{⟨package⟩.dtx}{class}}
          \file{⟨package⟩.sty}{\from{⟨package⟩.dtx}{package}}}
```

Using a single versioning string for both the `.cls` and `.sty` files can be accomplished by changing the following lines in the `.dtx` file shown in Appendix A.4:

```
%<class>\NeedsTeXFormat{LaTeX2e}[2023-11-01]
%<class>\ProvidesClass{⟨package⟩}
%<*class>
    [⟨YYYY⟩-⟨MM⟩-⟨DD⟩ v⟨version⟩ ⟨brief description⟩]
%</class>
```

The replacement code specifies which lines belong to the class file and which belong to the style file:

```
%<class|package>\NeedsTeXFormat{LaTeX2e}[2023-11-01]
%<class>\ProvidesClass{⟨package⟩}
%<package>\ProvidesPackage{⟨package⟩}
%<*class|package>
    [⟨YYYY⟩-⟨MM⟩-⟨DD⟩ v⟨version⟩ ⟨brief description⟩]
%</class|package>
```

5.4 Gallery of advanced packaging techniques

See the `.dtx` gallery on CTAN `https://www.ctan.org/tex-archive/info/dtxgallery` for examples of various packaging possibilities, including the following:

- single-file package distributions (cf. Section 5.2)
- conditional code inclusion (cf. Table 3)
- rearranging code for presentation in the documentation

A Skeleton files

This section contains complete skeletons of the types of files discussed in the rest of the document. These skeletons can be used as templates for creating your own packages.

A.1 A skeleton `.ins` file to generate a `.sty` file

```
%%
%% Copyright (C) ⟨year⟩ ⟨your name⟩
%%
%% This file may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either
%% version 1.3 of this license or (at your option) any later
%% version.  The latest version of this license is in:
%%
%%    http://www.latex-project.org/lppl.txt
%%
%% and version 1.3c or later is part of all distributions of
%% LaTeX version 2008-05-04 or later.
%%
```

```
\input docstrip.tex
\keepsilent

\usedir{tex/latex/⟨package⟩}

\preamble

This is a generated file.

Copyright (C) ⟨year⟩ ⟨your name⟩

This file may be distributed and/or modified under the
conditions of the LaTeX Project Public License, either
version 1.3 of this license or (at your option) any later
version.  The latest version of this license is in:

   http://www.latex-project.org/lppl.txt

and version 1.3c or later is part of all distributions of
LaTeX version 2008-05-04 or later.

\endpreamble

\generate{\file{⟨package⟩.sty}{\from{⟨package⟩.dtx}{package}}}

\Msg{*********************************************************}
\Msg{*}
\Msg{* To finish the installation you have to move the}
\Msg{* following file into a directory searched by TeX:}
\Msg{*}
\Msg{* \space\space ⟨package⟩.sty}
\Msg{*}
\Msg{* To produce the documentation run the file ⟨package⟩.dtx}
\Msg{* through LaTeX.}
\Msg{*}
\Msg{* Happy TeXing!}
\Msg{*********************************************************}

\endbatchfile
```

A.2 A skeleton `.ins` file to generate a `.cls` file

```
%%
```

```
%% Copyright (C) ⟨year⟩ ⟨your name⟩
%%
%% This file may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either
%% version 1.3 of this license or (at your option) any later
%% version.  The latest version of this license is in:
%%
%%    http://www.latex-project.org/lppl.txt
%%
%% and version 1.3c or later is part of all distributions of
%% LaTeX version 2008-05-04 or later.
%%

\input docstrip.tex
\keepsilent

\usedir{tex/latex/⟨class⟩}

\preamble

This is a generated file.

Copyright (C) ⟨year⟩ ⟨your name⟩

This file may be distributed and/or modified under the
conditions of the LaTeX Project Public License, either
version 1.3 of this license or (at your option) any later
version.  The latest version of this license is in:

   http://www.latex-project.org/lppl.txt

and version 1.3c or later is part of all distributions of
LaTeX version 2008-05-04 or later.

\endpreamble

\generate{\file{⟨class⟩.cls}{\from{⟨class⟩.dtx}{class}}}

\Msg{*********************************************************}
\Msg{*}
\Msg{* To finish the installation you have to move the}
\Msg{* following file into a directory searched by TeX:}
\Msg{*}
\Msg{* \space\space ⟨class⟩.cls}
\Msg{*}
```

```
\Msg{* To produce the documentation run the file ⟨class⟩.dtx}
\Msg{* through LaTeX.}
\Msg{*}
\Msg{* Happy TeXing!}
\Msg{*********************************************************}

\endbatchfile
```

A.3 A skeleton `.dtx` file to generate a `.sty` file

```
% \iffalse meta-comment
%
% Copyright (C) ⟨year⟩ ⟨your name⟩
% --------------------------------
%
% This file may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in:
%
%    http://www.latex-project.org/lppl.txt
%
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008-05-04 or later.
%
% \fi
%
% \iffalse
%<package>\NeedsTeXFormat{LaTeX2e}[2023-11-01]
%<package>\ProvidesPackage{⟨package⟩}
%<package>   [⟨YYYY⟩-⟨MM⟩-⟨DD⟩ v⟨version⟩ ⟨brief description⟩]
%
%<*driver>
\documentclass{ltxdoc}
\usepackage{⟨package⟩}
\EnableCrossrefs
\CodelineIndex
\RecordChanges
\begin{document}
  \DocInput{⟨package⟩.dtx}
\end{document}
%</driver>
% \fi
```

```
%
% \changes{v1.0}{⟨YYYY⟩-⟨MM⟩-⟨DD⟩}{Initial version}
%
% \GetFileInfo{⟨package⟩.sty}
%
% \DoNotIndex{⟨list of control sequences⟩}
%
% \title{The \textsf{⟨package⟩} package\thanks{This document
%   corresponds to \textsf{⟨package⟩}~\fileversion,
%   dated \filedate.}}
% \author{⟨your name⟩ \\ \texttt{⟨your e-mail address⟩}}
%
% \maketitle
%
% \begin{abstract}
%   Put text here.
% \end{abstract}
%
% \section{Introduction}
%
% Put text here.
%
% \section{Usage}
%
% \DescribeMacro{\YOURMACRO}
% Put description of macro |\YOURMACRO| here.
%
% \DescribeEnv{YOURENV}
% Put description of environment |YOURENV| here.
%
% \MaybeStop{\PrintIndex}
%
% \section{Implementation}
%
% \begin{macro}{\YOURMACRO}
% Put explanation of |\YOURMACRO|'s implementation here.
%    \begin{macrocode}
\newcommand{\YOURMACRO}{}
%    \end{macrocode}
% \end{macro}
%
% \begin{environment}{YOURENV}
% Put explanation of |YOURENV|'s implementation here.
%    \begin{macrocode}
\newenvironment{YOURENV}{}{}
```

```
%    \end{macrocode}
% \end{environment}
%
% \Finale
\endinput
```

A.4 A skeleton `.dtx` file to generate a `.cls` file

```
% \iffalse meta-comment
%
% Copyright (C) ⟨year⟩ ⟨your name⟩
% -------------------------------
%
% This file may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in:
%
%    http://www.latex-project.org/lppl.txt
%
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008-05-04 or later.
%
% \fi
%
% \iffalse
%<*driver>
\ProvidesFile{⟨class⟩.dtx}
%</driver>
%<class>\NeedsTeXFormat{LaTeX2e}[2023-11-01]
%<class>\ProvidesClass{⟨class⟩}
%<*class>
    [⟨YYYY⟩-⟨MM⟩-⟨DD⟩ v⟨version⟩ ⟨brief description⟩]
%</class>
%
%<*driver>
\documentclass{ltxdoc}
\EnableCrossrefs
\CodelineIndex
\RecordChanges
\begin{document}
  \DocInput{⟨class⟩.dtx}
\end{document}
%</driver>
```

```
% \fi
%
% \changes{v1.0}{⟨YYYY⟩-⟨MM⟩-⟨DD⟩}{Initial version}
%
% \GetFileInfo{⟨class⟩.dtx}
%
% \DoNotIndex{⟨list of control sequences⟩}
%
% \title{The \textsf{⟨class⟩} class\thanks{This document
%   corresponds to \textsf{⟨class⟩}~\fileversion,
%   dated \filedate.}}
% \author{⟨your name⟩ \\ \texttt{⟨your e-mail address⟩}}
%
% \maketitle
%
% \begin{abstract}
%   Put text here.
% \end{abstract}
%
% \section{Introduction}
%
% Put text here.
%
% \section{Usage}
%
% \DescribeMacro{\YOURMACRO}
% Put description of macro |\YOURMACRO| here.
%
% \DescribeEnv{YOURENV}
% Put description of environment |YOURENV| here.
%
% \MaybeStop{\PrintIndex}
%
% \section{Implementation}
%
% \begin{macro}{\YOURMACRO}
% Put explanation of |\YOURMACRO|'s implementation here.
%    \begin{macrocode}
\newcommand{\YOURMACRO}{}
%    \end{macrocode}
% \end{macro}
%
% \begin{environment}{YOURENV}
% Put explanation of |YOURENV|'s implementation here.
%    \begin{macrocode}
```

```
\newenvironment{YOURENV}{}{}
%    \end{macrocode}
% \end{environment}
%
% \Finale
\endinput
```

A.5 A skeleton master-document file (`.tex`)

```
\documentclass{ltxdoc}
\usepackage{⟨file1⟩}
\usepackage{⟨file2⟩}
\usepackage{⟨file3⟩}

\title{⟨title⟩}
\author{⟨you⟩}

\EnableCrossrefs
\CodelineIndex
\RecordChanges

\begin{document}
  \maketitle

  \begin{abstract}
    Put text here.
  \end{abstract}

  \tableofcontents

  \DocInclude{⟨file1⟩}
  \DocInclude{⟨file2⟩}
  \DocInclude{⟨file3⟩}
\end{document}
```

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison Wesley, Reading, Massachusetts, October 1, 1994. ISBN 0-201-54199-8.

[2] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984. British Computer Society. Available from `http://www.literateprogramming.com/knuthweb.pdf`.

[3] LaTeX Project Team. LaTeX for package and class authors. Available from `https://ctan.org/pkg/clsguide`, October 24, 2023.

Index