How to Package Your LaTeX Package

Scott Pakin <pakin@uiuc.edu>

30 April 2002

Abstract

This tutorial is intended for advanced LaTeX 2_ε 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_ε for Class and Package Writers", which is available from CTAN (`http://www.ctan.org`) and comes with most LaTeX 2_ε distributions. Finally, you should know how to install packages that are shipped as a `.dtx` file plus a `.ins` file.

Terminology A *style file* (`.sty`) is primarily a collection of macro and environment definitions. One or more style files (e.g., a main style file that `\input`s or `\RequirePackage`s multiple helper files) is called a *package*. Packages are loaded into a document with `\usepackage{`⟨*main .sty file*⟩`}`. In the rest of this document, we use the notation "⟨*package*⟩" to represent the name of your package.

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

Before writing any code, the first step is to write an *installer* (`.ins`) file. An installer file extracts the code from a `.dtx` file, strips off the comments and documentation, and outputs a `.sty` file. The good news is that a `.ins` file is 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⟩ by ⟨your name⟩
%%
%% This file may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either
%% version 1.2 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 1.2 or later is part of all distributions of
%% LaTeX version 1999/12/01 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 change the name of the package if he modifies anything.

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⟩ by ⟨your name⟩

This file may be distributed and/or modified under the
conditions of the LaTeX Project Public License, either
version 1.2 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 1.2 or later is part of all distributions of
LaTeX version 1999/12/01 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⟩ by ⟨your name⟩
%%
%% This file may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either
%% version 1.2 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 1.2 or later is part of all distributions of
%% LaTeX version 1999/12/01 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:

```
\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`

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

```
\endbatchfile
```

Appendix A.1 lists a complete, skeleton `.ins` file.

3 The `.dtx` file

A `.dtx` file contains both the commented source code and the user documentation for the package. DocStrip normally extracts the source code proper—no user documentation and no comments—into the package's `.sty` file(s). 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 usually begin with a copyright and license comment:

```
% \iffalse meta-comment
%
% Copyright (C) ⟨year⟩ by ⟨your name⟩
%
% This file may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either
% version 1.2 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 1.2 or later is part of all distributions of
% LaTeX version 1999/12/01 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}[1999/12/01]
%<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 begin with the following:

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\ProvidesPackage{⟨package⟩}
   [⟨YYYY⟩/⟨MM⟩/⟨DD⟩ v⟨version⟩ ⟨description⟩]
```

For example:

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\ProvidesPackage{skeleton}
   [2002/03/25 v1.0 .dtx skeleton file]
```

The `\NeedsTeXFormat` line ensures that the package won't run under a version of LaTeX 2ε older than what the package was tested with. 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.

```
\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
```

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 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 package. 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 you need to typeset the user documentation.

4. `\EnableCrossrefs` tells Doc that you want it to construct an index for your code—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 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 consistent (because descriptions of macros and environments are always indexed by page number).

6. On page 11, we'll see how to log the changes made in each revision of the package. `\RecordChanges` tells Doc that it should keep 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 20.

`\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 can always add it manually or even enable `\OnlyDescription` for *all* `.dtx` files by adding the following to his `ltxdoc.cfg` file:

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

`\CheckSum {`⟨*number*⟩`}`

Doc supports a very simplistic form of document checksumming, to help ensure that a package didn't get corrupted in transport. Doc merely counts the number of backslashes that occur in the code. If the number matches the checksum, Doc gives a success message:

```
******************
* Checksum passed *
******************
```

Otherwise, it says what the correct checksum should be:

```
! Package doc Error: Checksum not passed (⟨incorrect⟩<>⟨correct⟩).
```

To specify the checksum in a `.dtx` file, merely add a `\CheckSum` statement:

```
% \CheckSum{⟨number⟩}
```

If ⟨*number*⟩ is `0`, or if the `.dtx` file lacks a `\CheckSum` line entirely, then Doc outputs the following warning message:

```
**********************************
* This macro file has no checksum!
* The checksum should be ⟨number⟩!
**********************************
```

During code development, it is convenient to specify `\CheckSum{0}`, so you don't receive an error message every time you run `latex`. But don't forget to replace "`0`" with the correct number before releasing your package!

`\CharacterTable {⟨text⟩}`

The second mechanism that Doc uses to ensure that a `.dtx` file is uncorrupted is a character table. If you put the following command verbatim into your `.dtx` file, then Doc will ensure that no unexpected character translation took place in transport:[1]

```
% \CharacterTable
%  {Upper-case    \A\B\C\D\E\F\G\H\I\J\K\L\M\N\O\P\Q\R\S\T\U\V\W\X\Y\Z
%   Lower-case    \a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z
%   Digits        \0\1\2\3\4\5\6\7\8\9
%   Exclamation   \!     Double quote  \"     Hash (number) \#
%   Dollar        \$     Percent       \%     Ampersand     \&
%   Acute accent  \'     Left paren    \(     Right paren   \)
%   Asterisk      \*     Plus          \+     Comma         \,
%   Minus         \-     Point         \.     Solidus       \/
%   Colon         \:     Semicolon     \;     Less than     \<
%   Equals        \=     Greater than  \>     Question mark \?
%   Commercial at \@     Left bracket  \[     Backslash     \\
%   Right bracket \]     Circumflex    \^     Underscore    \_
%   Grave accent  \`     Left brace    \{     Vertical bar  \|
%   Right brace   \}     Tilde         \~}
```

[1] The character table is commonly prefixed with double percent signs so that it gets written to the `.sty` file. This seems unnecessary and is therefore shown here with single percent signs.

A success message looks like this:

```
**************************
* Character table correct *
**************************
```

and an error message looks like this:

```
! Package doc Error: Character table corrupted.
```

`\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 pieces, respectively, "`\filedate`", "`\fileversion`", and "`\fileinfo`":

```
% \GetFileInfo{⟨package⟩.sty}
```

For instance, the `\ProvidesPackage` example shown on page 7 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 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:

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

⋮

3.2 User documentation

We can finally 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 can certainly 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 will also 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.

`\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` is also useful on its own. For example, "`This needs a \meta{dimen}.`" is typeset as "This needs a ⟨*dimen*⟩."

In addition to those commands, Doc facilitates the typesetting of macro descriptions by automatically loading the `shortvrb` package. That lets you use

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

|...| 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.

Table 1: Argument-formatting commands

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

3.3 Code and commentary

\StopEventually {⟨*text*⟩}
\Finale

The package's source code is delineated by putting it between \StopEventually and \Finale. Note that \CheckSum (page 9) applies only to the package's source code. \StopEventually takes an argument, which is a block of text to typeset after the code. If \OnlyDescription (page 9) is specified, then nothing after the \StopEventually will be output—including text that follows \Finale. \StopEventually's ⟨*text*⟩ parameter is therefore the way to provide 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 more of the following commands.

\PrintChanges
\PrintIndex

\PrintChanges produces an unnumbered section called "Change History". (See Figure 1 on page 11). 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 actual 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 are additionally 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.[3] Note that definitions and uses are included in the index only if the document includes a code listing (i.e., `\OnlyDescription` was not specified).

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*
Other (i.e., an explicit `\index`)		myItem 27

Running `latex` on ⟨*package*⟩`.dtx` produces index data in ⟨*package*⟩`.idx`. To produce the actual 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 to find macro definitions and see what other macros a package depends upon.

[3] If `\CodelineIndex` (page 8) is not used, then "123" also refers to a page number.

```
\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.[4]
2. The lines of code within `\begin{macrocode}`...`\end{macrocode}` should not begin with "`%`". The code gets written exactly as 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 local to a single page). If `\PrintIndex` is used, the index will automatically include entries for `\newcommand`, `\mymacro`, and `\LaTeX`, unless any of these are `\DoNotIndex`'ed.

[4] 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.

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

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

The `macro` and `environment` environments are used to delineate a complete macro or environment definition. `macro`/`environment` 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.

```
1 \newcommand{\mymacro}{This is
2   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 LaTeX datatypes, 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}
```

Descriptionless `macro` environments should generally 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}
% Do some really complicated processing.
%    \begin{macrocode}

                ⋮


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

Appendix A.2 lists a complete, skeleton `.dtx` file.

4 Tips, tricks, and recommendations

- Write lots of good documentation! It really helps others understand your code and the package as a whole.
- 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).

- Be sure to read "The DocStrip Program" and "The Doc and `shortvrb` Packages", the documentation for DocStrip and Doc, respectively. 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 a Perl script) from a `.dtx` file.
  - Changing the formatting of the typeset documentation.
- You can use `\index` in the normal way to index things other than macros and environments.
- If you use Emacs as your text editor, look up the `string-rectangle` command. This helps a great deal with adding a "`%`" to the beginning of every line in a region.
- 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

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*⟩

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.3 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 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}
```

- 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 inadvertently using such names in his own document.[5] 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.

[5] Within a LaTeX document, "`@`" is marked as catcode "other", not "letter".

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

```
%%
%% Copyright (C) ⟨year⟩ by ⟨your name⟩
%%
%% This file may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either
%% version 1.2 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 1.2 or later is part of all distributions of
%% LaTeX version 1999/12/01 or later.
%%

\input docstrip.tex
\keepsilent

\usedir{tex/latex/⟨package⟩}

\preamble

This is a generated file.

Copyright (C) ⟨year⟩ by ⟨your name⟩

This file may be distributed and/or modified under the
conditions of the LaTeX Project Public License, either
version 1.2 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 1.2 or later is part of all distributions of
LaTeX version 1999/12/01 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 `.dtx` file

```
% \iffalse meta-comment
%
% Copyright (C) ⟨year⟩ by ⟨your name⟩
% ----------------------------------
%
% This file may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.2
% 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 1.2 or later is part of all distributions of LaTeX
% version 1999/12/01 or later.
%
% \fi
%
% \iffalse
%<package>\NeedsTeXFormat{LaTeX2e}[1999/12/01]
%<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
%
% \CheckSum{0}
%
% \CharacterTable
%  {Upper-case    \A\B\C\D\E\F\G\H\I\J\K\L\M\N\O\P\Q\R\S\T\U\V\W\X\Y\Z
%   Lower-case    \a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z
%   Digits        \0\1\2\3\4\5\6\7\8\9
%   Exclamation   \!     Double quote  \"     Hash (number) \#
%   Dollar        \$     Percent       \%     Ampersand     \&
%   Acute accent  \'     Left paren    \(     Right paren   \)
%   Asterisk      \*     Plus          \+     Comma         \,
%   Minus         \-     Point         \.     Solidus       \/
%   Colon         \:     Semicolon     \;     Less than     \<
%   Equals        \=     Greater than  \>     Question mark \?
%   Commercial at \@     Left bracket  \[     Backslash     \\
%   Right bracket \]     Circumflex    \^     Underscore    \_
%   Grave accent  \`     Left brace    \{     Vertical bar  \|
%   Right brace   \}     Tilde         \~}
%
%
% \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{skeleton}~\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 |\YOURMACRO| here.
%
% \DescribeEnv{YOURENV}
% Put description of |YOURENV| here.
%
% \StopEventually{\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.3 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}
    ⟨abstract⟩
  \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`.

Index

@
    in macro names, 21

\AtBeginDocument, 9
\author, 13

\BaseDirectory, 3

change history, 11, 14–15
\changes, 11, 14
\chapter, 13
Character table corrupted, 11
\CharacterTable, 10–11
\CheckSum, 9–10, 14
Checksum not passed, 10
checksumming, 9–10
\CodelineIndex, 7–9
comments, 2–4, 6–7, 9, 18–20
control sequences, 12
copyright, 2, 6

date format, 7
\DescribeEnv, 13
\DescribeMacro, 13
\DisableCrossrefs, 8
Doc, 1, 2, 6–13, 16, 17, 20, 21
\DocInclude, 20, 21
\DocInput, 7–9, 21
DocStrip, 1, 3–7, 20
\documentclass, 8
documented LaTeX file, 1, 5–19, 23–25
\DoNotIndex, 12, 16
driver code, 8
.dtx, *see* documented LaTeX file

\EnableCrossrefs, 7–9
\endbatchfile, 5
\endpreamble, 3–4
environment, 16–19

\file, 4–5
\filedate, 7, 8, 11–12
\fileinfo, 11–12
\fileversion, 7, 8, 11–12
\Finale, 14
\footnote, 14
\from, 4–5

\generate, 4–5, 7
\GetFileInfo, 11–12

\iffalse, 6, 7
\index, 20
indexing, 2, 9, 12, 15, 21
\input, 1
.ins, *see* installer file
installer file, 1–5, 22–23

\keepsilent, 3

LaTeX Project Public License, 3
license, 2–3, 6
literate programming, 2, 13, 15
LPPL, *see* LaTeX Project Public License
ltxdoc, 8, 13, 20
ltxdoc.cfg, 9

macro, 16–19
macrocode, 15–19
makeindex, 15
\maketitle, 13
\marg, 13–14
\meta, 13–14
meta-comment, 6, 7
\Msg, 5

\NeedsTeXFormat, 7
\newcommand, 1
\newenvironment, 1

\oarg, 13–14
\OnlyDescription, 9, 14, 15

package, 1–3, 6–11, 13–15, 19, 21–22
\PageIndex, 8
\parg, 13–14
preamble, 3
\preamble, 3–4
\PrintChanges, 14–15
\PrintIndex, 14–16, 21
\ProvidesPackage, 7–8, 11–13

\RecordChanges, 7–9
\RequirePackage, 1
roman numerals, 21

\section, 13
\space, 5
\StopEventually, 14
.sty, *see* style file
style file, 1, 2, 4–7, 10, 12, 16, 19, 20

\tableofcontents, 21
\textsf, 13
\thanks, 13
\title, 13

\usedir, 3
\usepackage, 1, 8

\verb, 14