如何打包你的 LaTeX 宏包

Scott Pakin <scott+dtx@pakin.org>

13 September 2015

张泓知　翻译

2023 年 12 月 13 日

Abstract

本教程适用于高级 LaTeX 2$_\varepsilon$ 用户，他们希望学习如何创建 `.ins` 和 `.dtx` 文件，以便分发他们自己编写的类和样式文件。

1　介绍

要求　我们假设您已经了解如何在 LaTeX 中编程。也就是说，您应该知道如何使用 `\newcommand`、`\newenvironment`，最好还懂一点 TeX。您还应该熟悉 “LaTeX 2$_\varepsilon$ for Class and Package Writers” ，它可以在 CTAN (http://www.ctan.org) 上获取，并且大多数 LaTeX 2$_\varepsilon$ 发行版中都包含一个名为 `clsguide.dvi` 的文件。最后，您应该知道如何安装由 `.dtx` 文件和 `.ins` 文件组成的软件包。

术语　一个宏包 （`.sty`）文件主要是一组宏和环境的定义。一个或多个样式文件（例如一个主样式文件，`\input` 或 `\RequirePackage` 多个辅助文件）称为一个 *package*。包可以用`\usepackage{`⟨*main .sty file*⟩`}` 载入文档中。在本文档的其余部分，我们使用符号 “⟨*package*⟩” 代表您的包的名称。

动机　一个包的重要部分包括代码、代码的文档和用户文档。使用 Doc和DocStrip程序，可以将这三者合并为一个单一的，带有说明的 *说明 LaTeX*（.dtx）文件。.dtx文件的主要优势在于，它允许您使用任意的 LaTeX 构造来注释您的代码。因此，宏、环境、代码段、变量等都可以使用表格、图形、数学公式和字体变化来解释。代码可以使用 LaTeX的分段命令进行组织。Doc甚至可以生成一个统一的索引，对宏定义（在 LaTeX 代码中）和宏描述（在用户文档中）进行索引。这种注重为代码编写详细的、漂亮排版的注释的方法——本质上将程序视为描述一组算法的书——被称为 *文学编程* [2]，并自早期的 TeX 开始就被使用。

本教程将教您如何编写基本的 .dtx 文件和操作它们的 .ins 文件。虽然与《LaTeX Companion》的第 14 章存在许多重叠 [1]，但本文档结构更像是一步一步的教程，而《LaTeX Companion》更像是参考资料。此外，本教程展示了如何编写一个单一文件，既作为文档又作为驱动文件，这是 Doc 系统的一种更典型的用法，而不是使用分开的文件。

2　.ins 文件

为了准备一个包用于发布，第一步是编写一个*安装*（.ins）文件。安装文件从 .dtx 文件中提取代码，使用 DocStrip 去掉注释和文档，然后输出一个 .sty 文件。好消息是，.ins 文件通常相当简短，并且在一个包到另一个包之间没有明显变化。

.ins 文件通常以注释开始，指定版权 和许可信息：

```
%%
%% 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 1.3 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 1.3 or later is part of all distributions of
%% LaTeX version 2005/12/01 or later.
%%
```

LaTeX 项目公共许可证（LPPL）是大多数包——以及 LaTeX 本身——所使用的许可证。当然，您可以根据您想要的任何许可证发布您的包；LPPL 只是 LaTeX 包中最常见的许可证。LPPL 规定用户可以对您的包做任何事情——包括出售它，并且无需向您支付任何费用。唯一的限制是他必须为您的工作给予您信用，并且如果他修改了任何内容以避免版本混淆，他必须更改包的名称。

下一步是加载 DocStrip:

```
\input docstrip.tex
```

`\keepsilent`

默认情况下，DocStrip 会详细列出其活动情况。这些消息并不是特别有用，所以大多数人会将其关闭：

```
\keepsilent
```

`\usedir {⟨directory⟩}`

系统管理员可以指定所有与 TeX 相关文件应安装在其下的基本目录，例如 /usr/share/texmf。（请参阅 DocStrip 手册中的 “\BaseDirectory”。）.ins 文件指定其文件相对于该目录应安装的位置。以下是典型的设置：

```
\usedir{tex/latex/⟨package⟩}
```

```
\preamble
⟨text⟩
\endpreamble
```

接下来的步骤是指定一个 *preamble*，即将写入到每个生成文件顶部的一段注释：

```
\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 1.3 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 1.3 or later is part of all distributions of
LaTeX version 2005/12/01 or later.

\endpreamble
```

前述的前言会导致 ⟨*package*⟩`.sty` 文件开头如下：

```
%%
%% 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 1.3 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 1.3 or later is part of all distributions of
%% LaTeX version 2005/12/01 or later.
%%
```

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

现在我们来到一个.ins文件中最重要的部分：指定从.dtx文件生成哪些文件。以下告诉 DocStrip从 ⟨*package*⟩.dtx 中仅提取标记为 “package” 的部分，生成 ⟨*package*⟩.sty。（如何标记 .dtx 文件的部分在第 3节中描述。）

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

\generate 可以从给定的 .dtx 文件中提取任意数量的文件。它甚至可以从多个 .dtx 文件中提取单个文件。详细信息请参阅 DocStrip 手册。

`\Msg {⟨text⟩}`

.ins 文件的下一部分包括命令，用于向用户输出消息，告诉他需要安装哪些文件，并提醒他如何生成用户文档。以下一组 \Msg 命令是典型的：

```
\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{****************************************************}
```

请注意使用 `\obeyspaces` 来阻止 TeX 合并多个空格为一个。

`\endbatchfile`

最后，我们告诉 DocStrip 已经到达 `.ins` 文件的末尾：

```
\endbatchfile
```

附录 A.1 列出了一个完整的骨架 `.ins` 文件。附录 A.2 类似，但包含了一些微小的修改，旨在生成一个类（`.cls`）文件，而不是样式（`.sty`）文件。

3 .dtx 文件

一个 `.dtx` 文件包含了包的有注释源代码和用户文档。通过运行 `latex` 命令来处理一个 `.dtx` 文件，可以排版出用户文档，通常还包括一个漂亮排版的有注释 源代码版本。

由于一些 Doc 的技巧，一个 `.dtx` 文件实际上被评估了*两次*。第一次，只评估了一小部分 LaTeX 驱动代码。第二次，*comments* 在 `.dtx` 文件中被评估，就好像它们前面没有 “%”。这可能会导致写 `.dtx` 文件时产生许多混乱，并偶尔导致一些笨拙的构造。幸运的是，一旦 `.dtx` 文件的基本结构就位，填写代码就相当简单。

3.1 序言

.dtx 文件通常以版权和许可的注释开始：

```
% \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 1.3 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 1.3 or later is part of all distributions of
% LaTeX version 2005/12/01 or later.
%
% \fi
```

由于第二次处理 .dtx 文件时，行首的 % 字符会被忽略，所以需要使用 \iffalse 和 \fi。为了防止版权/ 许可被解释为 LaTeX 代码，我们必须将其用\iffalse…\fi 括起来。在 “\iffalse” 后添加 “meta-comment” 只是一种约定，表示这个注释是为人类阅读而非 Doc、DocStrip 或 LaTeX 的。

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

接下来的几行同样被\iffalse…\fi包围，以防止在第二次通过 .dtx 文件时被 latex 处理。不过，这些行不是为人类读者准备的，而是为了 DocStrip（因此没有 “meta-comment”）：

```
% \iffalse
%<package>\NeedsTeXFormat{LaTeX2e}[2005/12/01]
%<package>\ProvidesPackage{⟨package⟩}
```

```
%<package>    [⟨YYYY⟩/⟨MM⟩/⟨DD⟩ v⟨version⟩ ⟨description⟩]
%
```

（我们很快就会遇到 \fi。）

还记得 .ins 文件（第 5 页）中的 \generate 行吗？它以标签 “package” 结束。这告诉 DocStrip 将以 “%<package>” 开头的行写入到 .sty 文件中，并在此过程中剥离 “%<package>”。因此，我们的 .sty 文件将以以下内容开头：

```
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{⟨package⟩}
   [⟨YYYY⟩/⟨MM⟩/⟨DD⟩ v⟨version⟩ ⟨description⟩]
```

比如：

```
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{skeleton}
   [2002/03/25 v1.0 .dtx skeleton file]
```

\NeedsTeXFormat 行确保该包不会在早于该包测试的 LaTeX 2_ε 版本下运行。\ProvidesPackage 行中的日期和版本字符串用于由 Doc 设置 \filedate 和 \fileversion 宏。请注意日期的格式：*YYYY/MM/DD* 在整个 LaTeX 2_ε 中都被使用，您的包中也应该使用这种格式。

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

接下来是 .dtx 文件中唯一不被注释掉的部分（即每行不以 % 开头）：

```
%<*driver>
\documentclass{ltxdoc}
\usepackage{⟨package⟩}
```

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

前述的代码块是 latex 在第一次处理 .dtx 文件时所评估的内容。现在我们逐行来看这段代码：

1. 将代码放置在 “%<*driver>” 和 “%</driver>” 之间是 DocStrip 的一种简写，表示在每一行前加上 “%<driver>”。这标示了 Doc 的驱动代码。
2. \documentclass 几乎总是应该使用 ltxdoc，因为这会加载 Doc 并提供一些有用的宏来格式化程序文档。
3. 您应该始终使用 \usepackage 导入您的包。如果不这样做，Doc 将无法看到包的 \ProvidesPackage 行，并且不知道如何设置 \filedate 和 \fileversion（参见第 13 页）。这也是您应该在此处使用 \usepackage 导入用于排版用户文档所需的其他任何包的地方。
4. \EnableCrossrefs 告诉 Doc 您希望它为您的代码构建索引——通常是个好主意。另一种选择是 \DisableCrossrefs，它可以在处理速度上稍微提升一些，但影响微乎其微。
5. \CodelineIndex 告诉 Doc 索引应该引用程序行号而不是页码。（另一种选择是 \PageIndex。）\CodelineIndex 使得索引条目更易于查找，但以索引的自洽性稍有损失（因为宏和环境的描述总是按页码索引）。索引，不过，会以一条说明性的注释开始。
6. 在第 12 页，我们将看到如何记录包每个版本的更改。\RecordChanges 告诉 Doc 应该保留并汇总日志条目。

7. 在 \begin{document} 和 \end{document} 之间应该只有一个命令：一个 \DocInput 调用，用于 .dtx 文件自身的输入。这使得主文件可以通过 \DocInput 来输入多个文件，从而生成一个单一文档，涵盖了多个包但包含了一个统一的索引。主文档文件在第 on page 25 页有描述。

\OnlyDescription

在前言（即 \begin{document} 之前）有时会出现的另一个命令是 \OnlyDescription，它告诉 Doc 仅排版用户文档，而不是包的代码或注释。最好通常省略 \OnlyDescription（或将其注释掉）。用户始终可以手动添加它，甚至可以通过将以下内容添加到他的 ltxdoc.cfg 文件中，为所有 .dtx 文件启用 \OnlyDescription：

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

本节剩余部分涵盖了 latex 对 .dtx 文件的第二次处理。因此，所有随后的示例都以百分号开头。

\CheckSum {⟨*number*⟩}

Doc 支持一种非常简单的文档校验机制，以确保包在传输过程中没有损坏。Doc 只是简单地计算代码中反斜杠的数量。如果数量与校验和匹配，Doc 会给出一个成功的消息：

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

否则，它会显示正确的校验和应该是多少：

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

要在 .dtx 文件中指定校验和，只需添加一个 \CheckSum 语句：

```
% \CheckSum{⟨number⟩}
```

当 ⟨*number*⟩ 为 0，或者 .dtx 文件完全缺少 \CheckSum 行时，Doc 会输出以下警告消息：

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

在代码开发过程中，指定 \CheckSum{0} 很方便，这样你每次运行 latex 时就不会收到错误消息。但在发布你的包之前，请不要忘记将 “0” 替换为正确的数字！

\CharacterTable {⟨*text*⟩}

Doc 使用的第二种确保 .dtx 文件未损坏的机制是字符表。[1] 如果你将以下命令原样放入你的 .dtx 文件中，Doc 将确保在传输过程中没有发生意外的字符转换：[2]

```
% \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         \,
```

[1] 译者注：虽然 \CharacterTable 和 \CheckSum 在公共互联网早期是 Doc 写作时的重要的特性，因为当时的邮件网关相当不可靠并经常搞乱文件，但在今天，它们更多的是一个麻烦而不是帮助。因此，它们现在是完全可选的，并且不建议在新文件中使用。

[2] 字符表通常以双百分号作为前缀，这样它会被写入 .sty 文件。这似乎是不必要的，因此此处用单百分号显示。

```
%   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         \~}
```

A success message looks like this:

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

and an error message looks like this:

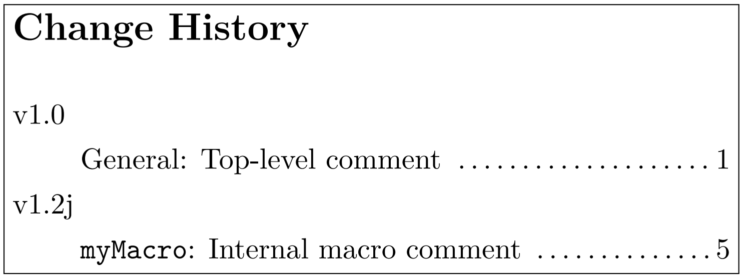
```
! Package doc Error: Character table corrupted.
```

`\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 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 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*⟩`}`”[3] 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. `shortvrb` lets you use `|…|` as a convenient shorthand for `\verb|…|`. For instance, “`|\mymacro| \oarg{pos} \marg{width} \marg{text}`” is typeset as follows:

[3] “⟨*macro*⟩” should include the backslash.

Table 1: Argument-formatting commands

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

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

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

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 10) 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 10) is specified, then nothing after the `\StopEventually` will be output—including text that follows `\Finale`. `\StopEventually`'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 more of the following commands.

`\PrintChanges`
`\PrintIndex`

`\PrintChanges` produces an unnumbered section called "Change History". (See Figure 1 on page 13). 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.[4] Note that macro/environment 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

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:

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

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

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.

```
\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.

[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 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}
```

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 above.

```
% \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 17 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.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.

- 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.

[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.

- 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.

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$_\varepsilon$ source code itself is typeset using a master document (`source2e.tex`) that includes all of the myriad `.dtx` files that comprise LaTeX 2$_\varepsilon$.

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 `.ins` 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}[2005/12/01]
%<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}[2005/12/01]
%<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⟩ by ⟨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 1.3 or later is part of all distributions of
%% LaTeX version 2005/12/01 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 1.3 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 1.3 or later is part of all distributions of
LaTeX version 2005/12/01 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 .ins file to generate a .cls 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 1.3 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 1.3 or later is part of all distributions of
%% LaTeX version 2005/12/01 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 1.3 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 1.3 or later is part of all distributions of
LaTeX version 2005/12/01 or later.

\endpreamble

\generate{\file{⟨package⟩.cls}{\from{⟨package⟩.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 ⟨package⟩.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⟩ by ⟨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 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% \fi
%
% \iffalse
%<package>\NeedsTeXFormat{LaTeX2e}[2005/12/01]
%<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{⟨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 |\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.4 A skeleton .dtx file to generate a .cls 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 1.3
% 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 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% \fi
%
% \iffalse
%<*driver>
\ProvidesFile{⟨package⟩.dtx}
%</driver>
%<class>\NeedsTeXFormat{LaTeX2e}[2005/12/01]
%<class>\ProvidesClass{⟨package⟩}
%<*class>
    [⟨YYYY⟩/⟨MM⟩/⟨DD⟩ v⟨version⟩ ⟨brief description⟩]
%</class>
%
%<*driver>
\documentclass{ltxdoc}
\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⟩.dtx}
%
% \DoNotIndex{⟨list of control sequences⟩}
%
% \title{The \textsf{⟨package⟩} class\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 |\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.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}
    ⟨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`.

Table 3: Comment visibility

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

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

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

Index