The file `phddoc.dtx` for use with $\LaTeX\,2_\varepsilon$.[*] It contains the code for `phddoc.cls`

Yannis Lazarides

2001/08/06

[*]This file has version number v0.02, dated 2001/08/06.

Fusce adipiscing justo nec ante. Nullam in enim. Pellentesque felis orci, sagittis ac, malesuada et, facilisis in, ligula. Nunc non magna sit amet mi aliquam dictum. In mi. Curabitur sollicitudin justo sed quam et quadd.

Contents

@tocrmarg=4em
@pnumwidth=1.5em
section
indent=1.5em
subsection
indent=3.8em
subsubsection
indent=7.0em

1 How to Develop your Own Class or Package **3**

1.1 Introduction **5**

1.2 Identifying your class **5**

1.3 Class Options **6**

1.4 Set-up the font sizes **7**

1.4.1 Setting up the normalsize **[8]**

1.5 Adjusting paragraph parameters **9**

1.6 Formatting Chapters and Sections **10**

1.7 Adjusting the Index **11**

1.8 Provide some hooks **11**

1.9 The final act of kindness to your users **12**

2 How to Package Your Class 12

2.0.1 The .ins file [16]

2.0.2 Generating messages [18]

2.1 The .dtx file 19

2.1.1 User documentation [22]

2.2 General tips for defining a Class 22

2.3 Declaring Options 23

2.4 Executing Options 23

2.5 Special Commands for class files 24

2.6 A minimal class 24

3 User Manual 24

3.1 Documentation of the LaTeX sources 26

3.2 Specification 26

3.3 Customisation 27

4 Implementation Code 29

4.1 Options 30

4.2 Configuration 30

4.3 Option Processing 30

4.4 Loading book and doc 30

4.5 Useful abbreviations 31

4.6 DocInclude 32

RULES

1

CHAPTER

HOW TO DEVELOP YOUR OWN CLASS OR PACKAGE

First there was one user and I took a lot of time to satisfy myself. Then I had 10 users, and a whole new level of difficulties arose. Then I had a hundred users and another level of things happened. I had a thousand users, I had ten thousand each of those were special phases in the development, important. I couldn't have gone with ten thousand until I'd done it with a thousand. But each time a new wave of changes came along, the idea was to have TEX get better, and not get more diverse as it needed to handle new things.

Donald Knuth

1.1 Introduction

To *make* a book is an interesting and somewhat involved process[1]. The text is set in type and printed on pages, the pages are gathered and folded into signatures and these are gathered and folded into signatures and these are then bound and covered. Many of the aspects of this process that has passed down to us by previous generations is discussed extensively in other sections of this book. Class authors have to distill this knowledge in a set of typographical rules to be described in a class file. The first thing such an author must do is to describe the *rationale* of developing such a class. The `octavo`[2] class was developed to enable printing books in dimensions that follow traditional styles. The[3] class to offer a flexible system on which other classes could be based and so does[4]. The `tufte-book` and `tufte-handout` classes to provide a style that resembles those found in Tufte books. Many Universities offer *Thesis* classes to standardize the way these are produced. Many of these Universities, translated the styles previously typed and the results are a typographical disaster, only mitigated by the ability to display beautiful mathematics. As these are printed on standard *photocopy paper* one cannot do much with the layout.

1.2 Identifying your class

The first thing a class must do is to identify any other formats it needs and to announce its name. This is accomplished using the two commands **??**→ P. ?? and **??**→ P. ??.

The following example, delares the version of LaTeX 2_ε that it requires and then gives the class name. It can be found in the preable of most well

[1]Town, L (1951). *Bookbinding by hand*. 1st Edition. London: Faber & Faber.
[2]Revets, Stefan A. (2007). *The Octavo Package*. CTAN.
[3]Wilson, P. (2001). *The Memoir class for configurable typesetting*. CTAN.
[4]**koma**.

written classes. You should also put some remarks to identify you as the author, the version number and other similar details. These are discussed in more detail in the next Chapter, where you will see how to automate documentation for your class.

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesClass{myclass—book}[2010/12/11 v3.5.0 myclass—book]
```

The above syntax must be followed exactly so that this information can be used by `LoadClass` or `documentclass` (for classes) or `\RequirePackage` or`\sepackage` (for packages) to test that the release is not too old. The whole of this $< release - info >$ information is displayed by `\listfiles` and should therefore not be too long.

```
% Load the common style elements
\input{myclass—common.def}
```

Another command that can be used is `\ProvidesFile`. This is similar to the two previous commands except that here the fullname, including the extension, must be given. It is used for declaring any files other than main class and package files.

This is useful, if you decide to have your main definitions in a separate file.

1.3 Class Options

Before we see in detail how to add options to a class, we need to review a package called **xkeyval** . Unless you are in the business of re-discovering wheels, this is an absolute must for developing, readable and maintenable code and your class is to provide many options.

```
\usepackage[textcolor=red,font=times]{mypack}
```

Class options are best set by using booleans`\ewboolean`.

We first set a new boolean that we `name@myclass@afourpaper`. This is used using the package `ifthen`[5] Then we can `DecalareOptionX` and we set the boolean to default to true. If the user then types

myclass[a4paper]

The a4paper options will be set. This is a much better and concise way of defining options. `\ewboolean`

```
\newboolean{@myclass@afourpaper}
\DeclareOptionX[myclass]<common>{a4paper}
  {
   \setboolean{@myclass@afourpaper}
   {true}
```

[5]The ifthen package was developed by David Carlisle, can be downloaded at http://www.ifi.uio.no/it/latex-links/ifthen.pdf

```
  }
```

Note that the command provide by `ifthen` `\setboolean` takes true or false, as #2, and sets #1 accordingly. In the above code we set the option as true.

It is much easier and most programmers use the `ifthen` package to check for option booleans

```
\ifthenelse{\boolean{@myclass@afourpaper}}
  {\geometry{
        a4paper,
        left=24.8mm,
        top=27.4mm,
        headsep=2\baselineskip,
        textwidth=107mm,
        marginparsep=8.2mm,
        marginparwidth=49.4mm,
        textheight=49\baselineskip,
        headheight=\baselineskip
    }
  }
 {}
```

1.4 Set-up the font sizes

LaTeX does not provide definitions of all the font-sizes. Unless you are extending an existing class, this is one of the first tasks you need to do in your new class.

Normally class authors will define all the commonly defined size commands, such as `\mall`, `\ormalsize` and other similar commands.

In the example shown below, we first start by defining the `\ormalsize` font size. In this book the `\normalsize` is defined as 14pt. We also define the vertical spaces that we need to have abovedisplay and belowdisplayskip. These are all very difficult to remember and once you have something you are happy with, just copy from class to class or even define a samll definition file to keep them all together.

Helvetica looks like this and Palatino looks like this.

The user has access to a number of commands which change the size of the fount, relative to the 'main' size used for the bulk of the text.

These `\ize` commands issue a `\setfontsize` command.

```
  \@setfontsize\size\font—size{baselineskip} where:
```

font-size The absolute size of the fount to use from now on.

baselineskip The normal value of `\aselineskip` for the size of the fount selected. (The actual value will be

A number of commands, defined in the LaTeXkernel, shorten the following definitions and are used throughout. These are:

`\@vpt`	5	`\@vipt`	6	`\@viipt`	7
`\@viiipt`	8	`\@ixpt`	9	`\@xpt`	10
`\@xipt`	10.95	`\@xiipt`	12	`\@xivpt`	14.4
...					

1.4.1 Setting up the normalsize

The user command to obtain the 'main' size is `\ormalsize`. LaTeX uses `\normalsize` when referring to the main size and maintains this value even if `\ormalsize` is redefined. The `\ormalsize` macro also sets values for `\bovedisplayskip`, `\bovedisplayshortskip` and `\elowdisplayshortskip`.

```
%%
% Set the font sizes and baselines to match Tufte's books
% normalsize
%%
\renewcommand\normalsize{%
   \@setfontsize\normalsize\@xpt{14}%
   \abovedisplayskip 10\p@ \@plus2\p@ \@minus5\p@
   \abovedisplayshortskip \z@ \@plus3\p@
   \belowdisplayshortskip 6\p@ \@plus3\p@ \@minus3\p@
   \belowdisplayskip \abovedisplayskip
   \let\@listi\@listI}

\normalbaselineskip=14pt
\normalsize
```

```
\renewcommand\small{%
   \@setfontsize\small\@ixpt{12}%
   \abovedisplayskip 8.5\p@ \@plus3\p@ \@minus4\p@
   \abovedisplayshortskip \z@ \@plus2\p@
   \belowdisplayshortskip 4\p@ \@plus2\p@ \@minus2\p@
   \def\@listi{\leftmargin\leftmargini
               \topsep 4\p@ \@plus2\p@ \@minus2\p@
               \parsep 2\p@ \@plus\p@ \@minus\p@
               \itemsep \parsep}%
   \belowdisplayskip \abovedisplayskip
}
\renewcommand\footnotesize{%
   \@setfontsize\footnotesize\@viiipt{10}%
   \abovedisplayskip 6\p@ \@plus2\p@ \@minus4\p@
   \abovedisplayshortskip \z@ \@plus\p@
   \belowdisplayshortskip 3\p@ \@plus\p@ \@minus2\p@
   \def\@listi{\leftmargin\leftmargini
               \topsep 3\p@ \@plus\p@ \@minus\p@
               \parsep 2\p@ \@plus\p@ \@minus\p@
               \itemsep \parsep}%
   \belowdisplayskip \abovedisplayskip
}
\renewcommand\scriptsize{\@setfontsize\scriptsize\@viipt\@viiipt}
\renewcommand\tiny{\@setfontsize\tiny\@vpt\@vipt}
\renewcommand\large{\@setfontsize\large\@xipt{15}}
\renewcommand\Large{\@setfontsize\Large\@xiipt{16}}
\renewcommand\LARGE{\@setfontsize\LARGE\@xivpt{18}}
\renewcommand\huge{\@setfontsize\huge\@xxpt{30}}
\renewcommand\Huge{\@setfontsize\Huge{24}{36}}

%% Define a HUGE for fun
\newcommand\HUGE{\@setfontsize\Huge{38}{47}}
```

1.5 Adjusting paragraph parameters

The parameters which control TEX's behaviour when typesetting paragraphs receive a bit of a tweak here. Contrary to the usual behaviour of modifying the grid with glue when difficulties are encountered with vertical space, here we shall try to counteract these tendencies and enforce as much as possible uniformity of the grid of lines.

A good value for paragraph indentation is `parindent 0.5pt`, for vertical spacing between paragraphs that are indented use 0pt. At this point if you are using any marginals it is a good idea to allow hyphenation with the `ragged2e` package. Since marginals use very narrow paragraphs you may get a very funny looking marginal text. Using the package, adjustments can be made to hyphenate the marginal text.

```
%%
% \RaggedRight allows hyphenation

\RequirePackage{ragged2e}
\setlength{\RaggedRightRightskip}{\z@ plus 0.08\hsize}
\setlength{\RaggedRightParindent}{1pc}

```

```
% Paragraph indentation and separation for normal text
\newcommand{\@tufte@reset@par}{%
  \setlength{\RaggedRightParindent}{1.0pc}%
  \setlength{\parindent}{1pc}%
  \setlength{\parskip}{0pt}%
}
\@tufte@reset@par

% Paragraph indentation and separation for marginal text
\newcommand{\@tufte@margin@par}{%
  \setlength{\RaggedRightParindent}{0.5pc}%
  \setlength{\parindent}{0.5pc}%
  \setlength{\parskip}{0pt}%
}
```

1.6 Formatting Chapters and Sections

The section on Chapters etc, has more on this, but we will touch on it briefly. Most recent class developerss use the **titlesec** and **titletoc** package to handle the complexity of these commands. With the `phd` package this is unecessary.

```
\titleformat{\subsection}%
  [hang]% shape
  {\normalfont\large}% format applied to label+text removed \itshape
  {\thesubsection}% label
  {1em}% horizontal separation between label and title body
  {}% before the title body
  []% after the title body
```

These are normally followed by the “titlespacing” commands to define the space around these sections.

```
%% We set the titlespacing using the package titlesec and titletoc
%
\titlespacing*{\chapter}{0pt}{20pt}{40pt}
\titlespacing*{\section}{0pt}{3.5ex plus 1ex minus .2ex}{2.3ex plus .2ex}
\titlespacing*{\subsection}{0pt}{3.25ex plus 1ex minus .2ex}{1.5ex plus.2ex}
```

1.7 Adjusting the Index

For classes representing books, the index is treated like a chapter whereas for others it is normally treated like a section. Whatever your document ends up like, indices are best done in a multi-column environment. One possibility is shown below, using the package ”multcol”.

```
\RequirePackage{multicol}
\renewenvironment{theindex}
  {\begin{fullwidth}%
    \small%
    \ifthenelse{\equal{\@tufte@class}{book}}%
      {\chapter{\indexname}}%
      {\section*{\indexname}}%
    \parskip0pt%
    \parindent0pt%
    \let\item\@idxitem%
    \begin{multicols}{3}%
  }
  {\end{multicols}%

\renewcommand\@idxitem{\par\hangindent 2em}
\renewcommand\subitem{\par\hangindent 3em\hspace*{1em}}
\renewcommand\subsubitem{
    \par\hangindent 4em\hspace*{2em}
}
\renewcommand\indexspace{
    \par\addvspace{
       1.0\baselineskip plus 0.5ex minus 0.2ex}\relax
    }%
%we now  swallow the letter heading in the index
\newcommand{\lettergroup}[1]{}
```

The code, renews the ”theindex” environment, with minor tweaks and defines it as a three column layout at ”fullwidth”.

1.8 Provide some hooks

It is useful at the end of the class to allow for localization of the class by importing a local file. This is easily achieved by checking if the file exists and then loading it. If there is a `myclass-book-local.sty` file, load it.

```
\IfFileExists{myclass-book-local.tex}
  {input{myclass-book-local}
  \MyClassInfoNL{Loading myclass-book-local.tex}}
 {}
```

If you intent to publish your class, you may also want to consider adding a hook for a patch-file.

1.9 The final act of kindness to your users

Many common classes, such as the `memoir` use such a tactic to avoid breaking old code.

```
\IfFileExists{mypatch.sty}{%
\RequirePackage{mypatch}}{}
```

RULE

2

CHAPTER

HOW TO PACKAGE YOUR CLASS

In the previous chapter we have outlined the main sections that you probably need to define in your class. In the examples we have used we just typed the examples as `example.cls` or `package.sty`.

In this chapter we will go over the packaging of the class and automating the generation of user documentation, using the `doc` and **DocStrip**[1] programs in files with an extension `.dtx`. The DocStrip program is an amazing piece of code that was originally created by Frank Mittelbach to accompany the `doc` package. The idea behind it was to remove comment lines in order to reduce the execution time of the program. Having created the DocStrip program to remove comment lines from programs it became feasible to do more than just strip comments. Wouldn't it be nice to have a way to include parts of the code only when some condition is set true? Wouldn't it be as nice to have the possibility to split the source of a TEX program into several smaller files and combine them later into one 'executable'? Both these wishes have been implemented in the DocStrip program.

You should also be familiar with "LaTeX2e" for Class and Package Writers", which is available from CTAN (`http://www.ctan.org`) and comes with most LaTeX2e" distributions in a file called clsguide.dvi.[2] Finally, you should know how to install packages that are shipped as a `.dtx` file plus a `.ins` file.

style (.sty) file is primarily a collection of macro and environment definitions. One or more style files (e.g., a main style file that `\input` or `\RequirePackages` multiple helper files) is called a package. Packages are loaded into a document with `\usepackage{`⟨*main .sty fille*⟩`}`. 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 **literate** and has been in use since the early days of TEX .

Furthermore, this tutorial shows how to write a single file that serves as

[1]**docstrip**.

[2]Pakin, Scott (2004). *How to Package Your LATEX Package*. `http://ctan.um.ac.ir/info/dtxtut/dtxtut.pdf`.

both documentation and driver file, which is a more typical usage of the Doc system than using separate files.

2.0.1 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.

License The 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 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, by using the command `\keepsilent`:

```
\keepsilent
```

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/packagename}
\preamble
htexti \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
----------------------------------------------------------------
phddoc --- A class to typeset LaTeX code.
E-mail: yannislaz@gmail.com
Released under the LaTeX Project Public License v1.3c or later
See http://www.latex-project.org/lppl.txt
----------------------------------------------------------------
\endpreamble
```

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

```
%%
%% This is file `phddoc.cls',
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% phddoc.dtx  (with options: `class')
%% ----------------------------------------------------------------
%% phddoc --- A class to typeset LaTeX code.
%% E-mail: yannislaz@gmail.com
%% Released under the LaTeX Project Public License v1.3c or later
%% See http://www.latex-project.org/lppl.txt
%% ----------------------------------------------------------------
```

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 hpackagei.sty from hpackagei.dtx by extracting only those parts marked as 'package' in the .dtx file. (Marking parts of a .dtx file is described later on.)

```
\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.

Personally I also generate README.md files in `markdown` format as well, so that when they get uploaded to `github` they can be rendered nicely.

```
\generate{\file{\jobname.md}{\from{\jobname.dtx}{readmemd}}}
```

The text has to be wriiten using 'guards' with the tag `readmd`

```
%<*readmemd>
# The `phddoc` LaTeX2e class

The `phd` latex package and the class with the same name provide
convenient methods to create new styles for books, reports
and articles. It also loads the most commonly used packages
and resolves conflicts.
%</readmemd>
```

2.0.2 Generating messages

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 `\sg` 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{* packagei.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
```

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

2.1 The .dtx file

We started describing the `.ins` install file first. The next file we will describe is the `.dtx` file. This holds both the code definitions as well as the user documentation.

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.

Guards If you open any .dtx file you will notice that the lines either start with a % sign or sometimes with a percentage sign and `<guard>`. The latter is called a guard and they are in a way like html tags. They have a starting and an ending tag. In the example below there are two different guards `<*10pt>...</10pt>` and `<*11pt></11pt>`. Unlike html tags guards are boolean expressions! You can use:

|! &

The |stands for disjunction (OR), the & stands for conjunction (AND) and the ! (NOT) stands for negation. The terminal is any sequence of letters and evaluates to true iff it occurs in the list of options that have to be included.

```
%<*10pt|11pt|12pt>
... code
%</10pt|11pt|12pt>
```

A longer example from KOMA shows the concept better.

```
%    \begin{macrocode}
\def\normalsize{%
%<*10pt>
  \@setfontsize\normalsize\@xpt\@xiipt
  \abovedisplayskip 10\p@ \@plus2\p@ \@minus5\p@
  \abovedisplayshortskip \z@ \@plus3\p@
  \belowdisplayshortskip 6\p@ \@plus3\p@ \@minus3\p@
%</10pt>
```

```
%<*11pt>
  \@setfontsize\normalsize\@xipt{13.6}%
  \abovedisplayskip 11\p@ \@plus3\p@ \@minus6\p@
  \abovedisplayshortskip \z@ \@plus3\p@
  \belowdisplayshortskip 6.5\p@ \@plus3.5\p@ \@minus3\p@
%</11pt>
...
%    end{macrocode}
```

If the guards only contain a one line of text, then a short form is provided as `<10pt>`. It is unecessary to provide a closing tag and the '*' is omitted. The example below from the KOMA classes shows a quite ingenious way of writing the `\ProvidesFile` macro in the different files; one for each tag. Two kinds of optional code are supported: one can either have optional code that '

ts' on one line of text, like the example above, or one can have blocks of optional code.

To distinguish both kinds of optional code the 'guard modi

er' has been in- troduced. The 'guard modifier' is one character that immediately follows the < of the guard. It can be either * for the beginning of a block of code, or / for the end of a block of code. The beginning and ending guards for a block of code have to be on a line by themselves.

When a block of code is not included, any guards that occur within that block are not evaluated.

```
%    \begin{macrocode}
\ProvidesFile{%
%<10pt>  scrsize10pt.clo%
%<11pt>  scrsize11pt.clo%
%<12pt>  scrsize12pt.clo%
}[\KOMAScriptVersion\space font size class option %
%<10pt>  (10pt)%
%<11pt>  (11pt)%
%<12pt>  (12pt)%
]
%    \end{macrocode}
```

In the `.ins` file one could write to generate the various `.clo` files.:

```
\generate{\usepreamble\defaultpreamble
  \file{scrsize10pt.clo}{%
    \from{scrkernel-version.dtx}{clo,10pt}%
    \from{scrkernel-fonts.dtx}{clo,10pt}%
    \from{scrkernel-paragraphs.dtx}{clo,10pt}%
  }%
```

```
    \file{scrsize11pt.clo}{%
      \from{scrkernel-version.dtx}{clo,11pt}%
      \from{scrkernel-fonts.dtx}{clo,11pt}%
      \from{scrkernel-paragraphs.dtx}{clo,11pt}%
    }%
    \file{scrsize12pt.clo}{%
      \from{scrkernel-version.dtx}{clo,12pt}%
      \from{scrkernel-fonts.dtx}{clo,12pt}%
      \from{scrkernel-paragraphs.dtx}{clo,12pt}%
    }%
  }%
```

The character table check 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:

```
% \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 \~}
A success message looks like this:
***************************
* Character table correct *
***************************

and an error message looks like this:
! Package doc Error: Character table corrupted.
```

DoNotIndex 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}
```

2.1.1 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`.

2.2 General tips for defining a Class

Evaluate, if there is a class that is nearer to what you wish to achive. If not do a set of requirements.

Book structure - start with book or `Octavo` if you need to hack extensively. If not use memoir, `koma` or `tufte-book`.

Paragraph looks
Lists
Figures
Bibliography and citations
Footnotes
Index
Titel pages
Book Cover
Language support
Mathematics
Graphs and figures
Typography - fonts, indentations fontsize etc
headers and footers

2.3 Declaring Options

Most classes or packages will have a good deal of options. These are declared using the `\DeclareOption` command. In this part no package loading should take place.

`\DeclareOption` {⟨*option*⟩} {⟨*code*⟩}

The argument option is the name of the option being declared and the {⟨*code*⟩} is the code that will execute if this option is requested.

`\DeclareOption*` {⟨*code*⟩}

The argument ⟨*code*⟩ in the star version of the command specifies the action to be taken if an unknown option is specified. Within this argument the `\CurrentOption` refers to the name of the option in question.

For example one could pass all such options to another package, using:

```
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{A}}
```

2.4 Executing Options

Normally after the options have been defined, one would need to provide default values and the options need to be executed.

`\ExecuteOptions` {⟨*option list*⟩}

You can also `\ExecuteOptions` when declaring other options. There is one caveat. This command can only be executed prior to executing the `\ProcessOptions` command because, as one of its last actions, the latter command reclaims all of the memory taken up by the code for the declared options.

\ProcessOptions*

For some packages it is preferable or essential to process options in the order they appear in the usepackage commands rather than using the order given through the sequence of the \DeclareOption$^{\rightarrow \sigma 23}$ commands. In this case it one has to use the star version of the command, i.e, \ProcessOptions* rather than \ProcessOptions.

2.5 Special Commands for class files

It is sometimes preferable to define a new class based on another and hence to extend it. To support this concept the LaTeX kernel provides two commands, \LoadClass and \PassOptionsToClass. These two commands can then be used to develop a new class, by adding and extending the functionality of the loaded class.

??$^{\rightarrow \text{P. ??}}$ [⟨*option list*⟩]{⟨*class*⟩}[⟨*release*⟩]

For example the ltxdoc class loads the standard article class. The **tufte-book** class loads the book class. The best way to understand the concepts discussed here is to study these classes.

2.6 A minimal class

Example 6.1 Model Class

```

\begin{filecontents}{phdexampleclass.cls}
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{phdexampleclass}[2015/07/07]
\renewcommand\normalsize{\fontsize{}{10pt}{12pt}\selectfont}
\setlength\textwidth{6.5in}
\setlength\textheight{5in}
\pagenumbering{arabic}
\end{filecontents}
```

3 User Manual

Fusce adipiscing justo nec ante. Nullam in enim. Pellentesque felis orci, sagittis ac, malesuada et, facilisis in, ligula. Nunc non magna sit amet mi aliquam dictum. In mi. Curabitur sollicitudin justo sed quam et quadd.

Fusce adipiscing justo nec ante. Nullam in enim. Pellentesque felis orci, sagittis ac, malesuada et, facilisis in, ligula. Nunc non magna sit amet mi aliquam dictum. In mi. Curabitur sollicitudin justo sed quam et quadd.

3.1 Documentation of the LaTeX sources

This is a class for documenting the **phd** bundle, a collection of packages and classes that enables the typesetting of documents using a flexible user interface,

You may however find it generally useful as a class for typesetting the documentation of files produced in 'doc' format.

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

Each documented file in the standard distribution comes with extension `dtx`. The appropriate class package or initex file will be extracted from the source by the docstrip system. Each `dtx` file may be directly processed with LaTeX 2$_\varepsilon$, for example

```
% latex2e docclass.dtx
%
```

would produce the documentation of the Class and package interface.

Each file that is used in producing the LaTeX 2$_\varepsilon$ format (ie not including the standard class and packages) will be printed together in one document if you LaTeX the file `sources2e.tex`. This has the advantage that one can produce a full index of macro usage across all the source files.

If you need to customise the typesetting of any of these files, there are two options:

- You can use DOCSTRIP with the module 'driver' to extract a small LaTeX file that you may edit to use whatever class or package options you require, before inputting the source file.
- You can create a file `phddoc.cfg`. This configuration file will be read whenever the `phddoc` class is used, and so can be used to customise the typesetting of all the source files, without having to edit lots of small driver files.

The second option is usually more convenient. Various possibilities are discussed in the next section.

3.2 Specification

The class builds on the **ltxdoc**[1] class and the **doc**[2] package, but since they were written many authors have come up with different ideas, as to how these documents should be produced.

The LaTeX3 Team has also more recently developed the `l3doc` class and `l3docstrip` package for documenting the l3 sources. Othe Teams such as

[1] Carlisle, David (Mar. 2018). *The file ltxdoc.dtx for use with LaTeX2e.*

[2] Mittelbach, Frank (May 2018). *The doc and shortvrb package.*

the developers of **pgf** prefer not to use `docstrip` and document the code and user manuals in a more traditional way, as normal documents in conjuction with external scripts written in python.

My objectives in writing this package, was to integrate the ability of the other packages in this series to document code in a flexible way. For longer books, such as a thesis, where the author might use their own developed macros, it also enables one to use such a method.

The objectives are as follows:

Flexibilty Provide flexibility to use one of the standard LaTeX 2e classes `article`, `book` `report` or the `KOMA` classes `scrartcl`, `scrbook`, `scrreprt` as the main class.

Style Enable the use of a fully featured key value interface for documenting the code.

Tools Provide a series of tools to create new documents, formatting and scaffolding. Currently LaTeX distributions have a plethora of tools, mostly using perl and lately l3build using Lua. Perl tools have served the community well for many years. One such tool `ctanify` does not work using normal dstributions as the Perl bundled in the distributions has some missiong modules. Go is a cross-compliation systems language enabling scripts to be bundled for different operating systems easily, hence the choice here.[3] Some of these problems with Perl on Windows can be overcome using `Strawberry Perl`[4] For any conflicts follow the guidelines in penwatch.[5]

```
phd ctanify  myclass.dtx  myclass.ins  README
```

3.3 Customisation

The simplest form of customisation is to pass more options to the `article` class which is loaded by **phddoc** . For instance if you wish all the documentation to be formated for A4 paper, add the following line to `phddoc.cfg`:

```
% \PassOptionsToClass{a4paper}{article}
%
```

All the source files are in two parts, separated by `\StopEventually`. The first part (should) contain ‘user’ documentation. The second part is a full

[3]See for example https://tex.stackexchange.com/questions/256096/which-perl-to-install-for-xindy-with-miktex-on-windows

[4]Donload at http://strawberryperl.com/. This will also enable xindy to work on a MikTeX distribution.

[5]https://www.penwatch.net/cms/pip_conflict/

documented listing of the source code. The `doc` package provides the command `\OnlyDescription` which suppresses the code listings. This may also be used in the configuration file, but as the `doc` package is read later, you must delay the execution of `\OnlyDescription` until after the `doc` package has been read. The simplest way is to use `\AtBeginDocument`. Thus you could put the following in your `phddoc.cfg`.

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

If the full source listing `sources2e.tex` is processed, then an index and change history are produced by default, however indices are not normally produced for individual files.

As an example, consider `ltclass.dtx`, which contains the sources for the new class and package interface commands. With no `cfg` file, a 19 page document is produced. With the above configuration a slightly more readable document (4 pages) is produced.

Conversely, if you really want to read the source listings in detail, you will want to have an index. Again the index commands provided by the doc package may be used, but their execution must be delayed.

```
% \AtBeginDocument{\CodelineIndex\EnableCrossrefs}
% \AtEndDocument{\PrintIndex}
%
```

The `doc` package writes index files to be sorted using MakeIndex with the `gind` style, so one would then use a command such as

```
% makeindex -s gind.ist ltclass.idx
%
```

and re-run LaTeX.

Similarly to print a Change history, you would add

```
% \AtBeginDocument{\RecordChanges}
% \AtEndDocument{\PrintChanges}
%
```

to `phddoc.cfg`, and use MakeIndex with a comand such as

```
% makeindex -s gglo.ist -o ltclass.gls ltclass.glo
%
```

Finally if you do not want to list all the sections of `source2e.tex`, you can use `\includeonly` in the `cfg` file:

```
% \includeonly{ltvers,ltboxes}
%
```

4 Implementation Code

Fusce adipiscing justo nec ante. Nullam in enim. Pellentesque felis orci, sagittis ac, malesuada et, facilisis in, ligula. Nunc non magna sit amet mi aliquam dictum. In mi. Curabitur sollicitudin justo sed quam et quadd.

Fusce adipiscing justo nec ante. Nullam in enim. Pellentesque felis orci, sagittis ac, malesuada et, facilisis in, ligula. Nunc non magna sit amet mi aliquam dictum. In mi. Curabitur sollicitudin justo sed quam et quadd.

4.1 Options

```
⟨*class⟩
\DeclareOption{a5paper}{\@latexerr{Option not supported}%
  {}}
\newif\if@book
\newif\if@article
\newif\if@report
\newif\if@scrbook
\newif\if@scrartcl
\newif\if@scrreprt

\DeclareOption{article}{\@articletrue}
\DeclareOption{book}{\@booktrue}
\DeclareOption{report}{\@reporttrue}
\DeclareOption{scrbook}{\@scrbooktrue}
\DeclareOption{scrartcl}{\@scrartcltrue}
\DeclareOption{scrreprt}{\@scrreprttrue}

\DeclareOption*{%
    \PassOptionsToClass  {\CurrentOption}{book}}
```

4.2 Configuration

Input a local configuration file, if it exists.

```
\InputIfFileExists{phddoc.cfg}
          {\typeout{************************************^^J%
                    * Local config file phddoc.cfg used^^J%
                    ************************************}}
          {}
```

4.3 Option Processing

```
\ProcessOptions
```

4.4 Loading book and doc

The original `phddoc` uses the article class. For longer documentation it is preferable to use the book. This means we might get an error on the production of a title, which we will fix very soon.

```
\if@book
  \LoadClass{book}
    \else
      \if@article
        \LoadClass{article}
        \cxset{section number prefix=,}
        \else
          \if@report
            \LoadClass{report}
          \else
             \if@scrbook
             \LoadClass{scrbook}
             \else
             \if@scrartcl
                 %\cxset{section number prefix=,}
                 \LoadClass{scrartcl}
                 \else
                 \if@scrreprt
                 \LoadClass{scrreprt}
              \fi
           \fi
      \fi
    \fi
  \fi
\fi
```

```
\RequirePackage{doc}
\RequirePackage{expl3}
```

Make | be a ‘short verb’ character, but not in the document preamble, where an active character may interfere with packages that are loaded.

```
\AtBeginDocument{\MakeShortVerb{\|}}
```

As ‘doc’ documents tend to have a lot of monospaced material, Set up some `tt` substitutions to occur silently.

```
\DeclareFontShape{OT1}{cmtt}{bx}{n}{<-> ssub * cmtt/m/n}{}
\DeclareFontFamily{OMS}{cmtt}{\skewchar\font 48}  % '60
\DeclareFontShape{OMS}{cmtt}{m}{n}{<-> ssub * cmsy/m/n}{}
\DeclareFontShape{OMS}{cmtt}{bx}{n}{<-> ssub * cmsy/b/n}{}
```

This substitution is in the standard fd file, but not silent.

```
\DeclareFontShape{OT1}{cmss}{m}{it}{<->ssub*cmss/m/sl}{}
```

```
\CodelineNumbered
\DisableCrossrefs
```

Increase the text width slightly so that width the standard fonts 72 columns of code may appear in a `macrocode` environment.

```
\setlength{\textwidth}{355pt}
```

Increase the marginpar width slightly, for long command names. And increase the left margin by a similar amount

```
\addtolength\marginparwidth{30pt}
\addtolength\oddsidemargin{20pt}
\addtolength\evensidemargin{20pt}
```

```
\setcounter{StandardModuleDepth}{1}
```

4.5 Useful abbreviations

The **phd-documentation** provides numerous commands for typesetting LaTeX code. It is imported automatically by the `phddoc` class and hence the following macros are described here for convenience.

`\cmd{\foo}` Prints `\foo` verbatim. It may be used inside moving arguments. It can *not* be use to record commands that are defined as "`\outer`" nor is it possible to use it on conditionals such as `\iftrue` or defined by `\newif`. `\cs{foo}` also prints `\foo`, for those who prefer that syntax. (This second form can be used to record all type of commends so the above restrictions do not apply.

`\cmd`
`\cs`

```
\def\cmd#1{\cs{\expandafter\cmd@to@cs\string#1}}
\def\cmd@to@cs#1#2{\char\number`#2\relax}
%\newcommand\cs[1]{\color{blue}{\texttt{\char`\ #1}}}
```

`\marg` `\marg{text}` prints {⟨*text*⟩}, 'mandatory argument'.

```
%\providecommand\marg[1]{%
%  {\ttfamily\char`\{}\meta{#1}{\ttfamily\char`\}}}
```

`\oarg` `\oarg{text}` prints [⟨*text*⟩], 'optional argument'.

```
%\providecommand\oarg[1]{%
%  {\ttfamily[}\meta{#1}{\ttfamily]}}
```

`\parg` `\parg{te,xt}` prints (⟨*te,xt*⟩), 'picture mode argument'.

```
\providecommand\parg[1]{%
  {\ttfamily(}\meta{#1}{\ttfamily)}}
```

4.6 DocInclude

```
\@addtoreset{CodelineNo}{part}
```

`\DocInclude` More or less exactly the same as `\include`, but uses `\DocInput` on a `dtx` file, not `\input` on a `tex` file.

```
\def\partname{File}
```

```
\newcommand*{\DocInclude}[1]{%
  \relax
  \clearpage
  \docincludeaux
  \IfFileExists{#1.fdd}{\def\currentfile{#1.fdd}}{\def\currentfile{#1.dtx}}%
  \ifnum\@auxout=\@partaux
    \@latexerr{\string\include\space cannot be nested}\@eha
  \else \@docinclude#1 \fi}
\def\@docinclude#1 {\clearpage
\if@filesw \immediate\write\@mainaux{\string\@input{#1.aux}}\fi
\@tempswatrue\if@partsw \@tempswafalse\edef\@tempb{#1}\@for
\@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswatrue\fi}\fi
```

```
\if@tempswa \let\@auxout\@partaux \if@filesw
\immediate\openout\@partaux #1.aux
\immediate\write\@partaux{\relax}\fi
```

We need to save (and later restore) various index-related commands which might be changed by the included file.

```
\let\@phddoc@PrintIndex\PrintIndex
\let\PrintIndex\relax
\let\@phddoc@PrintChanges\PrintChanges
\let\PrintChanges\relax
\let\@phddoc@theglossary\theglossary
\let\@phddoc@endtheglossary\endtheglossary
\part{\currentfile}%
  {\let\ttfamily\relax
  \xdef\filekey{\filekey, \thepart={\ttfamily\currentfile}}}%
\DocInput{\currentfile}%
\let\PrintIndex\@phddoc@PrintIndex
\let\PrintChanges\@phddoc@PrintChanges
\let\theglossary\@phddoc@theglossary
\let\endtheglossary\@phddoc@endtheglossary
\clearpage
\@writeckpt{#1}\if@filesw \immediate\closeout\@partaux \fi
\else\@nameuse{cp@#1}\fi\let\@auxout\@mainaux}
```

```
\gdef\codeline@wrindex#1{\if@filesw
        \immediate\write\@indexfile
            {\string\indexentry{#1}%
            {\filesep\number\c@CodelineNo}}\fi}%
```

```
\let\filesep\@empty
```

`\aalph` Special form of `\alph` as currently `source2e.tex` includes more than 26 files .

```
\def\aalph#1{\@aalph{\csname c@#1\endcsname}}
\def\@aalph#1{%
  \ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or
         j\or k\or l\or m\or n\or o\or p\or q\or r\or s\or
         t\or u\or v\or w\or x\or y\or z\or A\or B\or C\or
         D\or E\or F\or G\or H\or I\or J\or K\or L\or M\or
         N\or O\or P\or Q\or R\or S\or T\or U\or V\or W\or
         X\or Y\or Z\else\@ctrerr\fi}
```

`\docincludeaux`

```
\def\docincludeaux{%
  \def\thepart{\aalph{part}}\def\filesep{\thepart-}%
  \let\filekey\@gobble
  \g@addto@macro\index@prologue{%
    \gdef\@oddfoot{\parbox{\textwidth}{\strut\footnotesize
       \raggedright{\bfseries File Key:} \filekey}}%
    \let\@evenfoot\@oddfoot}%
  \global\let\docincludeaux\relax
\gdef\@oddfoot{%
  \expandafter\ifx\csname ver@\currentfile\endcsname\relax
   File \thepart: {\ttfamily\currentfile} %
  \else
   \GetFileInfo{\currentfile}%
   File \thepart: {\ttfamily\filename} %
   Date: \filedate\ %
   Version \fileversion
   \fi
   \hfill\thepage}%
\let\@evenfoot\@oddfoot}%
```

`ddangernote`

Use the double danger bend if there is something which could cause serious problems when used in a wrong way. Better the normal user does not know about such things.

```
\ExplSyntaxOn
\newenvironment {ddangernote}
  {
    \begin{trivlist}\item[]\noindent
    \begingroup\hangindent=3.5pc\hangafter=-2
    \cs_set:Npn \par{\endgraf\endgroup}
    \hbox to0pt{\hskip-\hangindent\dbend\kern2pt\dbend\hfill}\ignorespaces
  }{
      \par\end{trivlist}
  }
\ExplSyntaxOff
```

This is a double bend danger sign from **manfnt**[1]

```
\def\task#1#2{}
⟨/class⟩
```

[1]Kielhorn, Axel and Kosygin, Denis (July 1999). *The `Manfnt` package. A quick way to access the symbols in manfnt.*