The autodoc–Option*

B Hamilton Kelly

Abstract

This style option is used as an adjunct to the doc style option, and facilitates the documentation of style and other files, by making it unnecessary to have a separate driver file for each file being documented.

Contents

1 Introduction

Frank Mittelbach's [1] excellent doc style option has one slight drawback: it is necessary to write a small LaTeX "driver" file for each file being documented. The \documentstyle used for this will always be article, and the last style option on the command will always be doc. However, because the documentation of a *style option* file should obviously include examples of the use of the commands which it provides, it is necessary also to include as a further option the name of the style option being documented, in order that its commands may be available as the documentation is generated.

The style option presented here removes the necessity for writing a separate driver file; it works by prompting the user for the name(s) of the file(s) being documented, then (optionally) reading in those files *and* the doc style file, so that the driver file, which may be common to all style options, just has to issue a single command to cause all the referenced style files to be processed. As each file is processed, this style file opens and closes various auxiliary files[1] appropriate to the file being documented; the names of these files are taken from \docname, which gets redefined as necessary, and, in fact, these macros also redefine \jobname as the documenting process progresses. Unfortunately, autodoc is unable to do anything about the fact that the .dvi and log files will have already been opened with the name of the driver file. For this it will be necessary to use the operating system's facilities for renaming files.

\processdocs This is the only macro that the user needs to know about; the driver file for use with the autodoc option should look something like this:

```
\documentstyle[autodoc]{article}
```

* This is version v2.1f dated 30-Apr-1989

[1] aux, idx, ind, toc, etc.

```
\begin{document}
\processdocs
\end{document}
```

All the work and interaction with the user is performed when the autodoc option is first read in. At present, it firstly asks the user the question:

`What file(s) are you documenting?`

to which the user should respond with the name(s) of the file(s) to be documented. If there is more than one such file, the names should be separated by commas, not spaces.

If any of the files are `.sty` files, which are *not* `doc.sty` or `autodoc.sty`, then the user is further prompted for each file to specify whether the file shall be read as an *option*, with:

`Does the description of` ⟨*filename*⟩ `use its macros?`

The user should answer `yes`[2] or `no`. If an affirmative answer is given, the user is further prompted:

`Can reading of` ⟨*filename*⟩ `be deferred until it is processed?`

It will be found that most style option files don't actually use any commands which may only be executed in the preamble, so stack save space can be conserved by postponing the reading of the file until just before it is processed. However, style options which *do* utilize such commands must be read now, before `doc.sty` is input.

For producing a summary of all style files, etc., available at a site, the user may want to typeset just the descriptions of the files. Therefore, he is prompted:

`Should` ⟨*filename*⟩ `be fully documented?`

A negative response will lead to that file being processed by `doc` with `\OnlyDescription` in force.

Once a file has been fully cross-referenced, it's pointless to keep on processing it with `\EnableCrossrefs` in effect, so the user is given the option of suppressing this for each file; the user is prompted with:

`Should` ⟨*filename*⟩ `be cross referenced?`

Having decided what files are to be documented, and how, the user is given the option of reading in additional style options *before* the `doc` option is read; this permits inclusion of such options as `german.sty`. The user prompt is:

`Give the names of any further options (WITHOUT .sty):`

If the user doesn't wish to use any such additional options, a response of just pressing the RETURN key will terminate user interaction; otherwise, the specified options will all be read.

After user interaction is completed, the specified files will be read and their documentation produced. Each file will generate its own auxiliary files (`.idx`, `.toc`, etc.), including an `.aux` file; it will be necessary to make the customary two or three passes through the files to complete all cross-references, indices, etc. The recommended sequence is:

1. `LATEX`, to generate the first `.idx` and `.toc` files.
2. `makeindex`, to generate an `.ind` file (which will have the wrong page numbers, since the `.toc` file has yet to be read).

[2] In fact, any response starting with the letter 'Y' or 'y' is interpreted as 'yes, and *anything* else as 'no.'

3. LaTeX, to regenerate the auxiliary files, this time with correct page references in them.
4. makeindex, to generate a correct .ind file.
5. LaTeX, which hopefully will have correct page numbers throughout.

2 Description of the Macros

As with all style options, we commence by identifying ourselves on the terminal and in the log file:

```
\typeout{Style-Option: `autodoc' \fileversion\space\space
                                    \filedate\space (BHK)}
```

\docnames This style file is to be used (in conjunction with the doc style option) for documenting style *option* files (including itself), (main) style files, and even any other LaTeX document.

Our first action, therefore, is to prompt the user for the name(s) of the file(s) being documented.

```
\typein[\docnames]{What style file(s) are you documenting? }
```

Let's allow the user the luxury of not having to remember to type in the correct case...

```
\edef\next{\def\noexpand\docnames{\docnames}}
\lowercase\expandafter{\next}
```

2.1 Reading in additional style options

\style@ption When we come to document each of the files in \docnames, style option files need to be treated specially, because the description of such a file is likely to want to give examples of the facilities which it defines, so we ask the user if it needs to be read as an option before the documentation pass through the file. It's more convenient for the user to answer all such questions at the beginning, so we build, in this macro, a token list which contains the letters 'y' or 'n', in order corresponding with the file names, with 'y' indicating that the file *does* need to be read. Therefore, this macro needs to be initialized to \@empty.

```
\let\style@ption=\@empty
```

\doc@ption This macro similarly records whether the file is being fully documented, or whether the \OnlyDescription command should be invoked.

```
\let\doc@ption=\@empty
```

\cross@ption And this one records whether cross-referencing shall be enabled during the processing of the document.

```
\let\cross@ption=\@empty
```

\if@yes This is a pseudo-if, which takes one argument, a command into which a user response has been read by \typein. It behaves like \iftrue only if the response commences with the letter 'y', in either lower- or upper-case.

Our first action, therefore, is to force all the characters of the response to be lower-case.

```
\def\if@yes#1{%
  \edef\next{\def\noexpand #1{#1}}%
  \lowercase\expandafter{\next}%
```

Now we strip the response down to just its first character, so that we can make the comparison correctly.

```
  \edef #1{\expandafter\firstch@r #1\p@ramend}%
```

We finish up by making the comparison.

```
\if #1y}
```

`\firstch@r` `\otherch@rs` These macros yield, respectively, the first (or only) character, and the remaining characters, in the token string provided as argument. This list has to be terminated by the token `\p@ramend`.

```
\def\firstch@r#1#2\p@ramend{#1}
\def\otherch@rs#1#2\p@ramend{#2}
```

2.1.1 Recognizing parts of a file specification

As mentioned above, it is necessary to be able to recognize files whose file type is given as '`.sty`'; `autodoc` adopts a very simplistic approach to this, attempting to recognize the characters which follow the first period in the `\docname`, so cannot handle, for example, complicated *VAX/VMS* directory specifications. At the expense of making this style file operating system specific, it would be possible to extend this code to be able to strip off parts of the file specification which precede the file name itself, but it is recommended instead that all the files being documented should either be in the current directory or in the directory where TEX expects to find standard inputs (`TeXinputs:`), so that an explicit directory or path does not need to be given.

`\filetype` This macro yields (in `\ext`) the characters (if any) which follow the *first* period in its argument; the latter has to be terminated by `\p@ramend`.

```
\def\filetype#1.#2\p@ramend{\def\ext{#2}}
```

`\filen@me` This macro yields whatever *precedes* the first period in its argument, which again is terminated by the token `\p@ramend`.

```
\def\filen@me #1.#2\p@ramend{#1}
```

`\@ifextsty` This macro firstly tests to see if its argument includes the characters '`.sty`' at its end. If so, it interacts with the user to determine whether this is a *style option* which needs to be read (to establish its macros) before it may be documented. If this is the case, the macro sets `\@tempswatrue`.

We start by assuming that the given argument doesn't have any file extension. The macro `\ext` will be set to the string of characters which follow the first period in `\docname`, and if there is no such period, to '`tex`'.

```
\def\@ifextsty#1{%
  \expandafter\filetype #1.tex\p@ramend
```

If there *is* an explicit file extension, then `\ext` will *not* be '`tex`'. This next bit gets any such actual extension into `\ext`.

```
  \ifx\ext\ext@is@tex
  \else
    \expandafter\filetype #1\p@ramend
  \fi
```

We can now set `\@tempswatrue` if those last three characters are '`sty`'.

```
  \ifx\ext\ext@is@sty \@tempswatrue \fi
```

If the file type *is* '`.sty`', we ask the user whether the file should be read as a style option. This has to be done by a separate macro, because the pseudo-if `\if@yes` will not be correctly matched with its `\fi` if the file type *isn't* `.sty`.

```
  \if@tempswa
    \testif@ption
  \fi
}
```

\testif@ption This macro is invoked if \fulldocname has been found to end in .sty. It asks the user whether it needs to be read as a style option.

```
\def\testif@ption{%
  \typein[\is@option]{Does the description of \fulldocname\space
                      use its macros? }
```

Now we use \if@yes to determine whether \is@option was an affirmative response. If a negative response was given, we cancel \@tempswatrue.

```
  \if@yes \is@option \else
    \@tempswafalse
  \fi}
```

\ext@is@tex \ext@is@sty These macros expand to the strings 'tex' and 'sty', respectively, and are used in recognizing these strings if they form part of the current argument of \@ifextsty.

```
\def\ext@is@tex{tex}
\def\ext@is@sty{sty}
```

\SaveAnswer This macro takes two arguments; the first is whatever the user has typed in as a response to a \typein command, whilst the second gives the name of a command which is to have the letter 'y' or 'n' appended, depending upon whether the first argument is an affirmative answer.

```
\def\SaveAnswer#1#2{%
```

We use \if@yes to extract from the response its first character, converted to lowercase, and to compare this with 'y'.

```
  \if@yes #1
```

If this response is affirmative, we append a 'y' to the second argument.

```
    \edef #2{#2y}%
```

Alternatively, we append an 'n'.

```
  \else
    \edef #2{#2n}%
  \fi}
```

\Check@ption Again, this pseudo-if test has to be placed inside a separate macro if TEX is not to become confused when skipping conditional text.

```
\def\Check@ption{%
  \typein[\is@option]{Can reading of \fulldocname\space be deferred
          until it is processed? }
  \if@yes \is@option
  \else
    \input \fulldocname\relax
  \fi}
```

2.2 Scanning the file names in \docnames

\thisdoc \thedoc Now we are ready to cycle through all the file names in \docnames, during which we shall establish whether the file name ends in .sty, and if so, whether the user wants it to be read as a style option. However, we *know* that the files doc.sty and autodoc.sty should *not* be read in as options, so we establish here two macros which expand to those names, so that they may be used in \ifx tests.

```
\def\thisdoc{autodoc.sty}
\def\thedoc{doc.sty}
```

\fulldocname At last we start to cycle through the filenames and get the user's responses.

```
\@for\fulldocname:=\docnames\do{%
```

We assume initially that the file *shouldn't* be read as an option...

```
\@tempswafalse
```

Now we don't want to read **this** file again (otherwise we'll recurse and annoy the user by repeatedly asking for the name of the option to be documented)! Therefore we avoid that here...

```
\ifx\fulldocname\thisdoc
\else
```

Nor do we want to read the doc style option yet, because we're going to read it in at the end of this file.

```
  \ifx\fulldocname\thedoc
  \else
```

Otherwise, we use `\@ifextsty` to establish whether it *is* a file with suffix `.sty`, and, if so, whether it should be read as an option.

```
    \@ifextsty\fulldocname
  \fi
\fi
```

At this point, we know whether the file should be read in later as a style option: we put the appropriate character into `\style@ption`.

```
\if@tempswa
  \Check@ption
\else
  \def\is@option{n}
\fi
\SaveAnswer{\is@option}{\style@ption}
```

Now we ask whether the file is to be fully documented.

```
\typein[\is@full]{Should \fulldocname\space
                                  be fully documented? }%
\SaveAnswer{\is@full}{\doc@ption}
```

For our final question regarding each file being documented, we ask if full cross-referencing is required. Processing is *much* faster without this.

```
\typein[\is@cross]{Should \fulldocname\space
                                  be cross referenced? }%
\SaveAnswer{\is@cross}{\cross@ption}
```

And that completes the preliminary processing of the names in `\docnames`.

```
}
```

`\options` The user is asked one final question before the doc option is read in: whether additional options are required.

```
\typein[\options]{Give the names of any further
                               options (WITHOUT .sty): }%
```

If a non-null reply is given, each such option is read in...

```
\if \options \endlinechar
\else
  \@for\fulldocname:=\options\do{\input\fulldocname.sty\relax}%
```

Otherwise, there's nothing more to be done.

```
\fi
```

3 Processing the specified files

\processdocs This is the macro which, when invoked as the body of the document, will cause all the files to be input and documented.

We commence by cycling through each of the files to be processed. Any preliminary material (for instance, a description of the files being described) will use a setting of zero for TEX's \count1 register. Before we do this, however, we must remember the .toc file for the complete document, so that entries may be directed to it at the end of processing each document.

```
\def\processdocs{\count1=0
  \let\SavedTOC=\tf@toc
  \@for\fulldocname:=\docnames\do{%
```

\docname \Docname We then parse the filename to extract just the name into \docname, and generate an uppercase version of it for use in page numbering.

```
\edef\docname{\expandafter \filen@me \fulldocname .tex\p@ramend}%
\edef\next{\def\noexpand\Docname{\docname}}%
\uppercase\expandafter{\next}%
```

At this point, we start a new group for each file being documented, in order that any changes to macros defined in doc remain local to each processed file. We also ensure that any writes to a .toc file are discarded unless the individual document has opened one.

```
\begingroup
\let\tf@toc=\relax
```

Now we look at the first character of \style@ption to determine whether the file shall be read as a style option...

```
\expandafter\expandafter\expandafter \if
    \expandafter\firstch@r \style@ption\p@ramend y
```

If it is to be read, we suspend doc's ignorance of % whilst it is read, and treat @ as a letter, so that we are reproducing the conditions that pertain when style options are ordinarily read.

```
  \MakePercentComment\makeatletter
  \input\fulldocname\relax
  \MakePercentIgnore\makeatother
\fi
```

Throw away that first character of \style@ption, which will therefore now commence with the user's response relating to the *next* file to be processed.

```
\xdef\style@ption{\expandafter \otherch@rs \style@ption\p@ramend}%
```

3.1 Processing the file itself

We start by changing the \jobname appropriately, in order that the correct .toc, .ind, etc., files can be read. We also ensure that the next printed page will be *recto*, since it's nicer to start a new document on a right-hand page.

```
\edef\jobname{\docname}\clearpage
\ifodd\count0\else
  \vspace*{\fill}%
  \centerline{\small This page is left intentionally blank}%
  \vspace*{\fill}\clearpage
\fi
```

Pages are numbered individually within each separate document, so reset the counter and redefine \@oddfoot to include the document name in the page numbers. We also

use TEX's \count1 register to give us *different* page numbers, to facilitate the use of DVI processing programs. Note that we do *not* do this through \thepage, because that would also affect the table of contents and index entries.

```
\setcounter{page}{1}\global\advance\count1by1\relax
\def\@oddfoot{\hfil\Docname--\thepage\hfil}%
\let\@evenfoot=\@oddfoot
```

At this point, we perform most of the actions that would be undertaken by \include; the latter cannot be used for this task, because it is *only* capable of reading files with type .tex.

A command is put into the main document's auxiliary file to read the .aux file for the document being processed; when the main .aux file is being read, we prevent entries being written to the main document's table of contents, etc, which actually relate to the tables of the included file(s).

```
\if@filesw
  \immediate\write\@mainaux{\string\let\string\tf@toc
                                          =\string\relax}%
  \immediate\write\@mainaux{\string\@input{\docname.aux}}%
  \immediate\write\@mainaux{\string\let\string\tf@toc=
                                        \string\SavedTOC}%
```

We arrange for output to an auxiliary file to go to the appropriate file:

```
\immediate\openout\@partaux \docname.aux
\let\@auxout=\@partaux
```

We need to start that off tidily...

```
\immediate\write\@partaux{\relax}\fi
```

Now we examine \cross@ption to decide whether cross-referencing shall be enabled...

```
\expandafter\expandafter\expandafter
\if \expandafter \firstch@r \cross@ption \p@ramend y
  \EnableCrossrefs
\else
  \DisableCrossrefs
\fi
\xdef\cross@ption{\expandafter\otherch@rs\cross@ption\p@ramend}%
```

And \doc@ption determines whether the entire document is to be processed, or whether \OnlyDescription is required.

```
\expandafter\expandafter\expandafter
\if \expandafter \firstch@r \doc@ption \p@ramend y
\else
  \OnlyDescription
\fi
\xdef\doc@ption{\expandafter\otherch@rs\doc@ption\p@ramend}%
```

We activate the recording of modification records and of an index. Now we are in a position to read the next file being documented. Afterwards, we revert to single-column setting again (in case an index has been printed).

```
\RecordChanges
\MakeIndex
\input{\fulldocname}\clearpage \onecolumn
```

By reverting to the original auxiliary file and reading back that just written, we complete the toc entries, etc. When performing this read, we do it under the same conditions as for an \end{document}, which redefines a number of commands; however, we're already inside a group that is about to be ended, so there's no need to start another.

```
\let\@auxout=\@mainaux
```

```
\if@filesw \immediate\closeout\@partaux
  \def\global\@namedef##1##2{}\def\newlabel{\@testdef r}%
  \def\bibcite{\@testdef b}\@tempswafalse
  \makeatletter\input \docname.aux
\fi
```

After this, we'd better close any `.toc` files, etc. We close the group first, because these changes should apply "globally" to each file being documented.

```
\endgroup
\@for\ext:=\extensions\do{\@closetoc{\ext}}%
```

We also close any glossary and index files that may be open, because TEX can only manage a limited number of simultaneously open files.

```
\if@filesw \immediate\closeout\@glossaryfile
           \immediate\closeout\@indexfile \fi
```

Now LATEX's `\makeindex` and doc's `\RecordChanges` commands both allocate the files' "handles" through the `\newwrite` command, and it would be wasteful to keep on allocating new files through this mechanism. So we redefine `\makeindex` and `\RecordChanges` to open new files, appropriate to the file being processed, but reusing the same "handle" (providing we are actually generating auxiliary files).

```
\let\makeindex=\LaterMakeIndex
\let\RecordChanges=\LaterRecordChanges
```

We are now able to write (to the driver file's `.aux` file, and thence to its `.toc` file) an entry for the complete document's table of contents. We use LATEX's `\addtocontents` for this, but being impatient sorts, we need the `\write` to take place `\immediate`ly (actually, we're not really impatient; without this the contents for the last document processed would be stuck in a "whatsit", and never written to the file, because `\output` will not be called again). Therefore, we make local redeclarations for `\write`.

```
{\let\Write=\write \def\write{\immediate\Write}%
 \addtocontents{toc}{\protect \contentsline{section}%
   {\protect\numberline{\number\count1}\savedtitle}{\Docname--1}}}%
```

Finally, before reading any further files, we'd better reset the section counters, and cancel the effect of any `\appendix` command. We zero the footnote counter as well, in case a subsequent `\title` has a `\thanks` on it.

```
\setcounter{section}{0}%
\setcounter{subsection}{0}%
\def\thesection{\arabic{section}}%
\setcounter{footnote}{0}%
```

And that's all there is to it!!!! As our parting shot, we ensure that any `\@writefile` commands that may be read from the `.aux` files will be directed to the correct `toc` file.

```
    \let\tf@toc=\SavedTOC
  }% end of the \do
}
```

3.2 Handling toc files, etc

`\@closetoc` We need to be able to close `toc` files, etc., for each file being documented. Files are only closed if they exist, and have been used for the current document!

```
\def\@closetoc#1{\@ifundefined{tf@#1}{}{\expandafter
  \ifx \csname tf@#1\endcsname \relax \else
    \expandafter \immediate \closeout \csname tf@#1\endcsname
    \typeout{Closing \docname.#1 file}\fi
```

Since it is not possible to cancel the effect of the \newwrite which allocated the toc file, we merely make LaTeX's internal reference to the file be \relax.

```
    \global\expandafter\let\csname tf@#1\endcsname=\relax}%
}
```

\extensions Here is the list of possible file extensions used for table of contents files, etc.

```
\def\extensions{toc,lof,lot}
```

\@writefile Because toc files, etc., are allocated by \newwrite, which defines the "name" globally, we need a modified \@writefile that suppresses output if the "name" has been "undefined" (by letting it be \relax). This can happen

- if no file has *ever* been opened; or
- if the file has been closed, and not re-opened.

```
\def\@writefile#1#2{\@ifundefined{tf@#1}{}{\expandafter
   \ifx\csname tf@#1\endcsname \relax \else
     \expandafter\immediate\write\csname tf@#1\endcsname{#2}\fi}}
```

\savedtitle When documenting a number of files, it's pleasant to be able to put a table of contents before all the documented files, listing the titles of the files documented. Therefore, we need to save each document's title until after it has been processed. It is saved in this macro, which is here given a default definition.

```
\def\savedtitle{\hbox{}}
```

\@maketitle \orig@maketitle We give it a value when \@maketitle is called[3], so we save here the original definition of that macro, and define a new one which makes the necessary save operation inside a group: when doing that, we want to discard any \thanks that might be in the user's \title command.

```
\let\orig@maketitle=\@maketitle
\def\@maketitle{{\def\protect{\noexpand\protect\noexpand}%
                 \let\thanks=\@gobble
                 \xdef\savedtitle{\@title}}%
  \orig@maketitle}
```

\LaterMakeIndex \LaterRecordChanges Here are the redefinitions for \makeindex and \RecordChanges. They are similar to the ordinary \makeindex, but re-use the same file "handle" (providing we are actually generating auxiliary files). Note that we have to make the appropriate definitions for \index and glossary

```
\def\LaterMakeIndex{\if@filesw
  \immediate\openout\@indexfile=\docname.idx\relax
  \def\index{\@bsphack\begingroup
     \def\protect####1{\string####1\space}\@sanitize
     \@wrindex\@indexfile}\typeout
  {Writing index file \docname.idx}%
\fi}%
\def\LaterRecordChanges{\if@filesw
  \immediate\openout\@glossaryfile=\docname.glo\relax
  \def\glossary{\@bsphack\begingroup\@sanitize
                \@wrindex\@glossaryfile}\typeout
  {Writing glossary file \docname.glo}%
\fi}%
```

[3] It would have been cleaner to have redefined \maketitle to have done the work, but that gets redefined in doc, which we haven't yet read!

4 Reading in the doc Style Option

Finally, as our parting shot, we read in the doc style option, which will set up everything for creating the documentation. Before we do so, however, we "remember" the \makeindex command (because ordinarily that may be issued only in the preamble), and define \DocstyleParms to do nothing; this will prevent the doc style from invoking \makeindex unnecessarily for the root file.

```
\let\MakeIndex=\makeindex
\def\DocstyleParms{\relax}
```

We now read in doc.sty; that will leave the % character ignorable, so we'll set it back to it's usual state before tidying up this file, which ends, (you'll have to take my word for it!) with a call of \MakePercentIgnore so that documentation can proceed in the usual way.

```
\input{doc.sty}
\MakePercentComment
```

References

[1] F. MITTELBACH The doc-Option. (see page 245 of this issue of TUGboat.).

Index

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

◇ B Hamilton Kelly
Royal Military College of Science
Shrivenham
SWINDON
SN6 8LA
United Kingdom
Janet: tex@uk.ac.cranfield.rmcs