The doc–Option[*]

Frank Mittelbach[†]
Gutenberg Universität Mainz

Abstract

This style option contains the definitions that are necessary to format the documentation of style files. The style file was developed in Mainz in cooperation with the Royal Military College of Science.

Contents

1 Introduction

The TeX macros which are described here allow definitions and documentation to be held in one and the same file. This has the advantage that normally very complicated instructions are made simpler to understand by comments inside the definition. In addition to this, updates are easier and only one source file needs to be changed. On the other hand, because of this, the style files are considerably longer: thus TeX takes longer to load them. If this is a problem, there is an easy remedy: one needs only to write a small Pascal program that removes all lines that begin with a % sign.

The idea of integrated documentation was born with the development of the TeX program; it was crystallized in Pascal with the WEB system. The advantages of this method are plain to

[*] This file has version number v1.5j dated 89/06/07. The documentation was last revised on 89/06/09.

[†] Further commentary added at Royal Military College of Science by B. Hamilton Kelly; English translation of parts of the original German commentary provided by Andrew Mills.

see (it's easy to make comparisons [2]). Since this development, systems similar to WEB have been developed for other programming languages. But for one of the most complicated programming languages (TeX) the documentation has however been neglected. The TeX world seems to be divided between:—

- a couple of "wizards", who produce many lines of completely unreadable code "off the cuff", and
- many users who are amazed that it works just how they want it to do. Or rather, who despair that certain macros refuse to do what is expected of them.

I do not think that the WEB system is *the* reference work; on the contrary, it is a prototype which suffices for the development of programs within the TeX world. It is sufficient, but not totally sufficient.[1] As a result of WEB, new programming perspectives have been demonstrated; unfortunately, though, they haven't been developed further for other programming languages.

The method of documentation of TeX macros which I have introduced here should also only be taken as a first sketch. It is designed explicitly to run under LaTeX alone. Not because I was of the opinion that this was the best starting point, but because from this starting point it was the quickest to develop.[2] As a result of this design decision, I had to move away from the concept of modularization; this was certainly a step backward.

I would be happy if this article could spark off discussion over TeX documentation. I can only advise anyone who thinks that they can cope without documentation to "Stop Time" until he or she completely understands the AMS-TeX source code.

1.1 Using the doc style option

Just like any other option, invoke it by including it amongst the style options in the optional parameter list for the `\documentstyle` command.

N.B. Because this style option makes the % character ignorable, which may interfere with the reading of other style options, doc should be the *last* style option invoked.

2 The User Interface

2.1 General conventions

A TeX file prepared to be used with the 'doc' style option consists of 'documentation parts' intermixed with 'definition parts'.

Every line of a 'documentation part' starts with a percent sign (`%`) in column one. It may contain arbitrary TeX or LaTeX commands except that the character '`%`' cannot be used as a comment character. To allow user comments, the `^^A` character is defined as a comment character later on.

All other parts of the file are called 'definition parts'. They contain fractions of the macros described in the 'documentation parts'.

If the file is used to define new macros (e.g. as a style file in the `\documentstyle` macro), the 'documentation parts' are bypassed at high speed and the macro definitions are pasted together, even if they are split into several 'definition parts'.

`macrocode` On the other hand, if the documentation of these macros is to be produced, the 'definition parts' should be typeset verbatim. To achieve this, these parts are surrounded by the macrocode environment. More exactly: before a 'definition part' there should be a line containing

[1] I know that this will be seen differently by a few people, but this product should not be seen as the finished product, at least as far as applications concerning TeX are concerned. The long-standing debate over 'multiple change files' shows this well.

[2] This argument is a bad one; however, it is all too often trotted out.

```
%␣␣␣␣\begin{macrocode}
```

and after this part a line

```
%␣␣␣␣\end{macrocode}
```

There must be *exactly* four spaces between the `%` and `\end{macrocode}` — TEX is looking for this string and not for the macro while processing a 'definition part'.

Inside a 'definition part' all TEX commands are allowed; even the percent sign could be used to suppress unwanted spaces etc.

`macrocode*` Instead of the macrocode environment one can also use the macrocode* environment which produces the same results except that spaces are printed as ␣ characters.

2.2 Describing the usage of new macros

`\DescribeMacro` When you describe a new macro you may use `\DescribeMacro` to indicate that at this point the usage of a specific macro is explained. It takes one argument which will be printed in the margin and also produces a special index entry. For example, I used `\DescribeMacro{\DescribeMacro}` to make clear that this is the point where the usage of `\DescribeMacro` is explained.

`\DescribeEnv` An analogous macro `\DescribeEnv` should be used to indicate that a LATEX environment is explained. It will produce a somewhat different index entry. Below I used `\DescribeEnv{verbatim}`.

`verbatim` `verbatim*` It is often a good idea to include examples of the usage of new macros in the text. Because of the `%` sign in the first column of every row, the verbatim environment is slightly altered to suppress those characters.[3] The verbatim* environment is changed in the same way.

2.3 Describing the definition of new macros

`macro` To describe the definition of a new macro we use the macro environment. It has one argument: the name of the new macro.[4] This argument is also used to print the name in the margin and to produce an index entry. Actually the index entries for usage and definition are different to allow an easy reference. This environment might be nested. In this case the labels in the margin are placed under each other.

`\MacrocodeTopsep` `\MacroTopsep` `\MacroIndent` `\MacroFont` There also exist four style parameters: `\MacrocodeTopsep` and `\MacroTopsep` are used to control the vertical spacing above and below the macrocode and the macro environment, `\MacroIndent` is used to indent the lines of code and `\MacroFont` holds the font and a possible size change command for the code lines. If you want to change their default values in a style file (like `ltugbot.sty`) use the `\DocstyleParms` command described below.

2.4 Formatting the margins

`\PrintDescribeMacro` `\PrintDescribeEnv` `\PrintMacroName` As mentioned earlier, some macros and the macro environment print their arguments in the margin. This is actually done by three macros which are user definable.[5] They are named `\PrintDescribeMacro`, `\PrintDescribeEnv` and `\PrintMacroName` (called by the macro environment).

[3] These macros were written by Rainer Schöpf. He also provided a new verbatim environment which can be used inside of other macros. This will be documented elsewhere.

[4] This is a change to the style design I described in TUGboat 10#1 (Jan. 89). We finally decided that it would be better to use the macro name *with* the backslash as an argument.

[5] You may place the changed definitions in a separate style file or at the beginning of the documentation file. For example, if you don't like any names in the margin but want a fine index you can simply `\let` these macros equal `\@gobble`. The doc style option won't redefine any existing definition of these macros.

2.5 Using a special escape character

`\SpecialEscapechar` If one defines complicated macros it is sometimes necessary to introduce a new escape character because the '\' has got a special `\catcode`. In this case one can use `\SpecialEscapechar` to indicate which character is actually used to play the rôle of the '\'. A scheme like this is needed because the macrocode environment and its counterpart macrocode* produce an index entry for every occurrence of a macro name. They would be very confused if you didn't tell them that you'd changed `\catcode`s. The argument to `\SpecialEscapechar` is a single-letter control sequence, that is, one has to use `\|` for example to denote that '|' is used as an escape character. `\SpecialEscapechar` only changes the behavior of the next macrocode or macrocode* environment.

The actual index entries created will all be printed with \ rather than |, but this probably reflects their usage, if not their definition, and anyway must be preferable to not having any entry at all. The entries *could* be formatted appropriately, but the effort is hardly worth it, and the resulting index might be more confusing (it would certainly be longer!).

2.6 Cross-referencing all macros used

`\DisableCrossrefs` `\EnableCrossrefs` As already mentioned, every new macro name used within a macrocode or macrocode* environment will produce an index entry. In this way one can easily find out where a specific macro is used. Since TeX is considerably slower when it has to produce such a bulk of index entries one can turn off this feature by using `\DisableCrossrefs` in the driver file. To turn it on again just use `\EnableCrossrefs`.[6]

`\DoNotIndex` But also finer control is provided. The `\DoNotIndex` macro takes a list of macro names separated by commas. Those names won't show up in the index. You might use several `\DoNotIndex` commands: their lists will be concatenated. In this article I used `\DoNotIndex` for all[7] macros which are already defined in LaTeX.

All three declarations are local to the current group.

2.7 Producing the actual index entries

Several of the aforementioned macros will produce some sort of index entries. These entries have to be sorted by an external program—the current implementation assumes that the makeindex program by Chen [4] is used.

But this isn't built in: one has only to redefine some of the following macros to be able to use any other index program. Since the doc style option has to be the last option in the `\documentstyle` macro, all macros which are installation dependent are defined in such a way that they won't overwrite a previous definition. Therefore it is safe to put the changed versions in a style file which might be read in before the doc style option.

To allow the user to change the specific characters recognized by his or her index program all characters which have special meaning in the makeindex program are given symbolic names.[8] However, all characters used should be of `\catcode` other than 'letter' (11).

`\actualchar` `\quotechar` `\encapchar` The `\actualchar` is used to separate the 'key' and the actual index entry. The `\quotechar` is used before a special index program character to suppress its special meaning. The `\encapchar` separates the indexing information from a letter string

[6] Actually, `\EnableCrossrefs` changes things more drastically; any following `\DisableCrossrefs` which might be present in the source will be ignored.

[7] In this implementation there is one exception: you can't use `\par` in the argument of `\DoNotIndex`. This will be fixed in a later version.

[8] I don't know if there exists a program which needs more command characters, but I hope not.

which makeindex uses as a TeX command to format the page number associated with a special entry. It is used in this style to apply the `\main` and the `\usage` commands.

`\levelchar` Additionally `\levelchar` is used to separate 'item', 'subitem' and 'subsubitem' entries.

It is a good idea to stick to these symbolic names even if you know which index program is used. In this way your files will be portable.

`\SpecialMainIndex` `\SpecialIndex` `\SpecialUsageIndex` `\SpecialEnvIndex` `\SortIndex` To produce a main index entry for a macro the `\SpecialMainIndex` macro[9] may be used. It is called 'special' because it has to print its argument verbatim. If you want a normal index entry for a macro name `\SpecialIndex` might be used.[10] To index the usage of a macro or an environment `\SpecialUsageIndex` and `\SpecialEnvIndex` may be used. Additionally a `\SortIndex` command is provided. It takes two arguments—the sort key and the actual index entry.

All these macros are normally used by other macros; you will need them only in an emergency.

`\verbatimchar` But there is one characteristic worth mentioning: all macro names in the index are typeset with the `\verb*` command. Therefore one special character is needed to act as a delimiter for this command. To allow a change in this respect, again this character is referenced indirectly, by the macro `\verbatimchar`. It expands by default to `+` but if your code lines contain macros with '+' characters in their names (e.g. when you use `\+`) you will end up with an index entry containing `\verb+\++` which will be typeset as '\+' and not as '\+'. In this case you should redefine `\verbatimchar` globally or locally to overcome this problem.

`\*` We also provide a `\*` macro. This is intended to be used for index entries like

index entries
    Special macros for ~

Such an entry might be produced with the line:

```
\index{index entries\levelchar Special macros for \*}
```

2.8 Setting the index entries

`theindex` `\IndexMin` Contrary to standard LaTeX, the index is typeset in three columns by default. This is controlled by the LaTeX counter 'IndexColumns' and can therefore be changed with a `\setcounter` declaration. Additionally one doesn't want to start a new page unnecessarily. Therefore the theindex environment is redefined. When the theindex environment starts it will measure how much space is left on the current page. If this is more than `\IndexMin` then the index will start on this page. Otherwise `\newpage` is called.

`\IndexPrologue` Then a short introduction about the meaning of several index entries is typeset (still in onecolumn mode). Afterwards the actual index entries follow in multi-column mode. You can change this prologue with the help of the `\IndexPrologue` macro. Actually the section heading is also produced in this way, so you'd better write something like:

```
\IndexPrologue{\section*{Index} The index entries underlined ...}
```

`\IndexParms` When the theindex environment is finished the last page will be reformatted to produce balanced columns. This improves the layout and allows the next article to start on the same page. Formatting of the index columns (values for `\columnssep` etc.) is controlled by the `\IndexParms` macro. It assigns the following values:

`\parindent`	= 0.0pt	`\columnsep`	= 15.0pt
`\parskip`	= 0.0pt plus 1.0pt	`\rightskip`	= 15.0pt
`\mathsurround`	= 0.0pt	`\parfillskip`	= −15.0pt

`\@idxitem` Additionally it defines `\@idxitem` (which will be used when an `\item` command is

[9] This macro is called by the macro environment.

[10] This macro is called within the macrocode environment when encountering a macro name.

encountered) and selects `\small` size. If you want to change any of these values you have to define them all.

`\main` `\usage` The page numbers for main index entries are encapsulated by the `\main` macro (underlining its argument) and the numbers denoting the description are encapsulated by the `\usage` macro (which produces *italics*). As usual these commands are user definable.

2.9 Changing the default values of style parameters

`\DocstyleParms` If you want to overwrite some default settings made by the doc style, you can either put your declarations in the driver file (that is after `doc.sty` is read in) or use a separate style file for doing this work. In the latter case you have to define the macro `\DocstyleParms` which should contain all assignments. This indirect approach is necessary because your style file will be read before the `doc.sty`, thus some of the registers are not then allocated. If you don't define this macro its default definition `\makeindex` will be used which just starts the index process by calling `\makeindex`.

The doc style option currently assigns values to the following registers:

`\IndexMin`	= 80.0pt	`\MacroTopsep`	= 7.0pt plus 2.0pt minus 2.0pt
`\marginparwidth`	= 96.0pt	`\MacroIndent`	= 10.0pt
`\marginparpush`	= 0.0pt	`\MacrocodeTopsep`	= 3.0pt plus 1.2pt minus 1.0pt
`\tolerance`	= 1000		

2.10 Additional bells and whistles

We provide macros for logos such as WEB, $\mathcal{A}\kern-.1667em\lower.5ex\hbox{$\mathcal{M}$}\kern-.125em\mathcal{S}$-TEX, BIBTEX, SLITEX and PLAIN TEX. Just type `\Web`, `\AmSTeX`, `\BibTeX`, `\SliTeX` or `\PlainTeX`, respectively. LATEX and TEX are already defined in `latex.tex`.

`\meta` Another useful macro is `\meta` which has one argument and produces something like ⟨*dimen parameter*⟩.

`\OnlyDescription` `\StopEventually` You can use the `\OnlyDescription` declaration in the driver file to suppress the last part of your document. To make this work you have to place the command `\StopEventually` at a suitable point in your file. This macro has one argument in which you put all information you want to see printed if your document ends at this point (for example a bibliography which is normally printed at the very end). When `\Finale` the `\OnlyDescription` declaration is missing the `\StopEventually` macro saves its argument in a macro called `\Finale` which can afterwards be used to get things back (usually at the very end). Such a scheme makes changes in two places unnecessary.

Thus you can use this feature to produce a local guide for the TEX users which describes only the usage of macros (most of them won't be interested in your definitions anyway). For the same reason the `\maketitle` command is slightly changed to allow multiple titles in one document. So you can make one driver file reading in several articles at once.

`\IndexListing` Last but not least I defined an `\IndexListing` macro which takes a file name as an argument and produces a verbatim listing of the file, indexing every command as it goes along. This might be handy, if you want to learn something about macros without enough documentation. I used this feature to cross-reference `latex.tex` getting a verbatim copy with about 15 pages index.[11]

`\changes` To maintain a change history within the file, the `\changes` command may be placed amongst the description part of the changed code. It takes three arguments, thus:

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

[11] It took quite a long time and the resulting `.idx` file was longer than the `.dvi` file. Actually too long to be handled by the makeindex program directly (on our MicroVAX), but the final result was worth the trouble.

The changes may be used to produce an auxiliary file (LaTeX's `\glossary` mechanism is used for this) which may be printed after suitable formatting. The `\changes` macro encloses the ⟨*date*⟩ in parentheses and appends the ⟨*text*⟩ to form the printed entry in such a change history; because the makeindex program limits such fields to 64 characters, care should be taken not to exceed this limit when describing the change.

2.11 Acknowledgements

I would like to thank all folks at Mainz and at the Royal Military College of Science for their help in this project. Especially Brian and Rainer who pushed everything with their suggestions, bug fixes, etc.

3 The Description of Macros

As always, we begin by identifying the latest version of this file on the VDU and in the transcript file. But only if the macros are unkown to the system.

```
\@ifundefined{macro@cnt}{}{\endinput}
\typeout{Style-Option: `doc' \fileversion\space <\filedate>  (FMI)}
\typeout{English Documentation \@spaces <\docdate>  (RMCS and FMI)}
```

This time we also add a warning for the user.

```
\typeout{\@spaces Warning: This style option  should be used as last option}
\typeout{\@spaces Warning: in the \protect\documentstyle\space command !}
```

`\fileversion` `\filedate` `\docdate` As you can see I used macros like `\fileversion` to denote the version number and the date. They are defined at the very beginning of the style file (without a surrounding macrocode environment), so I don't have to search for this place here when I change the version number. You can see their actual outcome in a footnote to the title.

The first thing that we do next is to get ourselves a new comment sign. Because all sensible signs are already occupied, we will choose one that can only be entered indirectly:

```
\catcode`\^^A=14
```

3.1 Macros surrounding the 'definition parts'

`\macrocode` Parts of the macro definition will be surrounded by the environment macrocode. Put more precisely, they will be enclosed by a macro whose argument (the text to be set 'verbatim') is terminated by the string `%␣␣␣␣\end{macrocode}`. Carefully note the number of spaces. `\macrocode` is defined completely analogously to `\verbatim`, but because a few small changes were carried out, almost all internal macros have got new names. We start by calling the macro `\macro@code`, the macro which bears the brunt of most of the work, such as `\catcode` reassignments, etc.

```
\def\macrocode{\macro@code
```

Then we take care that all spaces have the same width, and that they are not discarded.

```
    \frenchspacing \@vobeyspaces
```

Before closing, we need to call `\xmacro@code`. It is this macro that expects an argument which is terminated by the above string. This way it is possible to keep the `\catcode` changes local.

```
    \xmacro@code}
```

`\macro@code` We will now begin with the macro that does the actual work:

```
\def\macro@code{%
```

In theory it should consist of a trivlist environment, but the empty space before and after the environment should not be too large.

```
    \topsep \MacrocodeTopsep
```

The next parameter we set is `\@beginparpenalty`, in order to prevent a page break before such an environment.

```
    \@beginparpenalty \predisplaypenalty
```

We then start a `\trivlist`, set `\parskip` back to zero and start an empty `\item`.

```
    \trivlist \parskip \z@ \item[]%
```

Additionally, everything should be set in `typewriter` font. Some people might prefer it somewhat differently; because of this the font choice is macro-driven.[12]

```
    \MacroFont
```

Because `\item` sets various parameters, we have found it necessary to alter some of these retrospectively.

```
    \leftskip\@totalleftmargin \advance\leftskip\MacroIndent
    \rightskip\z@ \parindent\z@ \parfillskip\@flushglue
```

The next line consists of the LaTeX definition of `\par` used in `\verbatim` and should result in blank lines being shown as blank lines.

```
    \blank@linefalse \def\par{\ifblank@line\hbox{}\fi\blank@linetrue\@@par}
```

What use is this definition of `\par`? We use the macro `\obeylines` of [3] which changes `^^M` to `\par` so that each line can control its own indentation. Next we must also ensure that all special signs are normalized; that is, they must be given `\catcode` 12.

```
    \obeylines \let\do\@makeother \catcode`\`\active \@noligs \dospecials
```

We also initialize the cross-referencing feature by calling `\init@crossref`. This will start the scanning mechanism when encountering an escape character.

```
    \init@crossref}
```

`\ifblank@line` `\blank@linetrue` `\blank@linefalse`

`\ifblank@line` is the switch used in the definition above. In the original verbatim environment the `\if@tempswa` switch is used. This is dangerous because its value may change while processing lines in the macrocode environment.

```
  \newif\ifblank@line
```

`\endmacrocode`

Because we have begun a trivlist environment in the macrocode environment, we must also end it. This is easily done using the following line of code:

```
  \def\endmacrocode{\endtrivlist
```

Additionally `\close@crossref` is used to do anything needed to end the cross-referencing mechanism.

```
                    \close@crossref}
```

`\MacrocodeTopsep` `\MacroIndent`

In the code above, we have used two registers. Therefore we have to allocate them. The default values might be overwritten with the help of the `\DocstyleParms` macro.

```
  \newskip\MacrocodeTopsep \MacrocodeTopsep = 3pt plus 1.2pt minus 1pt
  \newdimen\MacroIndent    \MacroIndent     = 10pt
```

`\MacroFont`

Here is the default definition for this macro:

```
  \@ifundefined{MacroFont}{\def\MacroFont{\small\tt}}{}
```

[12] The font change has to be placed *after* the `\item`. Otherwise a change to `\baselineskip` will affect the paragraph above.

`\macrocode*`
`\endmacrocode*`

Just as with the verbatim environment, there is also a 'star' variant of the macrocode environment in which a space is shown by the symbol ␣. Until this moment, I have not yet used it (it will be used in the description of the definition of `\xmacro@code` below) but it's exactly on this one occasion *here* that you can't use it (cf. Münchhausen's Marsh problem)[13] directly. Because of this, on this one occasion we'll cheat around the problem with an additional comment character. But now back to `\macrocode*`. We start with the macro `\macro@code` which prepares everything and then call the macro `\sxmacro@code` whose argument is terminated by the string `%␣␣␣␣\end{macrocode*}`.

```
\@namedef{macrocode*}{\macro@code\sxmacro@code}
```

As we know, `\sxmacro@code` and then `\end{macrocode*}` (the macro, not the string), will be executed, so that for a happy ending we still need to define the macro `\endmacrocode*`.

```
\expandafter\let\csname endmacrocode*\endcsname = \endmacrocode
```

`\xmacro@code`

As already mentioned, the macro `\xmacro@code` expects an argument delimited by the string `%␣␣␣␣\end{macrocode}`. At the moment that this macro is called, the `\catcode` of TEX's special characters are 12 ('other') or 13 ('active'). Because of this we need to utilize a different escape character during the definition. This happens locally.

```
\begingroup
\catcode`\|=\z@␣\catcode`\[=\@ne␣\catcode`\]=\tw@
```

Additionally, we need to ensure that the symbols in the above string contain the `\catcode`s which are available within the macrocode environment.

```
\catcode`\{=12␣\catcode`\}=12
\catcode`\%=12␣\catcode`\␣=\active␣\catcode`\\=\active
```

Next follows the actual definition of `\macro@code`; notice the use of the new escape character. We manage to get the argument surrounded by the string `\end{macrocode}`, but at the end however, in spite of the actual characters used during the definition of this macro, `\end` with the argument `{macrocode}` will be executed, to ensure a balanced environment.

```
|gdef|xmacro@code#1%␣␣␣␣\end{macrocode}[#1|end[macrocode]]
```

`\sxmacro@code`

The definition of `\sxmacro@code` is completely analogous, only here a slightly different terminating string will be used. Note that the space is not active in this environment.

```
|catcode`| =12
|gdef|sxmacro@code#1%    \end{macrocode*}[#1|end[macrocode*]]
```

Because the `\catcode` changes have been made local by commencing a new group, there now follows the matching `\endgroup` in a rather unusual style of writing.

```
|endgroup
```

3.2 Macros for the 'documentation parts'

`\DescribeMacro`
`\DescribeEnv`

The `\DescribeMacro` and `\DescribeEnv` macros should print their arguments in the margin and produce an index entry. We simply use `\marginpar` to get the desired result. This is however not the best solution because the labels might be slightly misplaced. One also might get a lot of 'marginpar moved' messages which are hard-wired into the LATEX output routine.[14] First we change to horizontal mode if necessary.

[13] Karl Friedrich Hieronymus Frhr. v. Münchhausen (*1720, †1797). Several books were written about fantastic adventures supposedly told by him (see [5] or [1]). In one story he escaped from the marsh by pulling himself out by his hair.

[14] It might be better to change these macros into environments like the macro environment.

The LaTeX macros `\@bsphack` and `\@esphack` are used to make those commands invisible (i.e. to normalize the surrounding space and to make the `\spacefactor` transparent).

```
\def\DescribeMacro#1{\leavevmode\@bsphack
            \marginpar{\raggedleft\PrintDescribeMacro{#1}}%
```

Note the use of `\raggedleft` to place the output flushed right. Finally we call a macro which produces the actual index entry and finish with `\@esphack` to leave no trace.[15]

```
            \SpecialUsageIndex{#1}\@esphack\ignorespaces}
```

The `\DescribeEnv` macro is completely analogous.

```
\def\DescribeEnv#1{\leavevmode\@bsphack
            \marginpar{\raggedleft\PrintDescribeEnv{#1}}%
            \SpecialEnvIndex{#1}\@esphack\ignorespaces}
```

To put the labels in the left margin we have to use the `\reversemarginpar` declaration. (This means that the `doc.sty` can't be used with all style options.) We also make the `\marginparpush` zero and `\marginparwidth` suitably wide.

```
\reversemarginpar
\setlength\marginparpush{0pt}  \setlength\marginparwidth{8pc}
```

`\bslash` From time to time, it is necessary to print a \ without being able to use the `\verb` command because the `\catcode` s of the symbols are already firmly established. In this instance we can use the command `\bslash` presupposing, of course, that the actual font in use at this point contains a 'backslash' as a symbol. Note that this definition of `\bslash` is expandable; it inserts a $\backslash_{12}$. This means that you have to `\protect` it if it is used in 'moving arguments'.

We start a new group in which to hide the alteration of `\catcode` s, and make `|` introduce commands, whilst \ becomes an 'other' character.

```
{\catcode`\|=\z@ \catcode`\\=12
```

Now we are able to define `\bslash` (globally) to generate a backslash of `\catcode` 'other'. We then close this group, restoring original `\catcode` s.

```
|gdef|bslash{\}}
```

`\verbatim` The verbatim environment holds no secrets; it consists of the normal LaTeX environment. We also set the `\@beginparpenalty` and change to the font given by `\MacroFont`.

```
\def\verbatim{\@beginparpenalty \predisplaypenalty \@verbatim
            \MacroFont \frenchspacing \@vobeyspaces \@xverbatim}
```

`\@verbatim` Additionally we redefine the `\@verbatim` macro so that it suppresses `%` characters at the beginning of the line. The first lines are copied literally from `latex.tex`.

```
\def\@verbatim{\trivlist \item[]\if@minipage\else\vskip\parskip\fi
      \leftskip\@totalleftmargin\rightskip\z@
      \parindent\z@\parfillskip\@flushglue\parskip\z@
      \@tempswafalse
```

`\@verbatim` sets `^^M`, the end of line character, to be equal to `\par`. This control sequence is redefined here; `\@@par` is the paragraph primitive of TeX.

```
\def\par{\if@tempswa\hbox{}\fi\@tempswatrue\@@par
```

[15] The whole mechanism won't work because of the `\leavevmode` in front. As a temporary change `\ignorespaces` is added.

We add to the definition of `\par` a control sequence, `\check@percent`, whose task it is to check for a percent character.

```
    \check@percent}%
```

The rest is again copied literally from `latex.tex`.

```
  \obeylines \tt \catcode`\`\active \@noligs \let\do\@makeother \dospecials}
```

`\check@percent` Finally we define `\check@percent`. Since this must compare a character with a percent sign we must first (locally) change percent's `\catcode` so that it is seen by TeX. The definition itself is nearly trivial: grab the following character, check if it is a `%`, and insert it again if not. At the end of the verbatim environment this macro will peek at the next input line. In the case the argument to `\check@percent` might be a `\par` or a macro with arguments. Therefore we make the definition `\long` (`\par` allowed) and use the normal `\next` mechanism to reinsert the argument after the `\fi` if necessary.

```
{\catcode`\%=12
 \long\gdef\check@percent#1{\ifx #1%\let\next\@empty \else
                                     \let\next#1\fi \next}}
```

`\macro` `\macro@` `\macro@cnt` The macro environment is implemented as a trivlist environment, whereby in order that the macro names can be placed under one another in the margin (corresponding to the macro's nesting depth), the macro `\makelabel` must be altered. In order to store the nesting depth, we use a counter.

```
 \newcount\macro@cnt \macro@cnt=0
```

The environment takes an argument—the macro name to be described. Since this name may contain special 'letters' we have to re-`\catcode` them before scanning the argument. This is done by the `\MakePrivateLetters` macro.

```
 \def\macro{\begingroup \MakePrivateLetters \macro@}
```

After scanning the argument we close the group to get the normal `\catcode`s back. Then we assign a special value to `\topsep` and start a trivlist environment.

```
 \long\def\macro@#1{\endgroup \topsep\MacroTopsep \trivlist
```

We also save the name being described in `\saved@macroname` for use in conjunction with the `\changes` macro.

```
    \edef\saved@macroname{\string#1}%
```

Now there follows a variation of `\makelabel` which is used should the environment not be nested, or should it lie between two successive `\begin{macro}` instructions or explanatory text. One can recognize this with the switch `\if@inlabel` which will be `true` in the case of successive `\item` commands.

```
    \def\makelabel##1{\llap{##1}}%
```

If `@inlabel` is `true` and if `\macro@cnt` > 0 then the above definition needs to be changed, because in this case LaTeX would otherwise put the labels all on the same line and this would lead to them being overprinted on top of each other. Because of this `\makelabel` needs to be redefined in this case.

```
    \if@inlabel
```

If `\macro@cnt` has the value 1, then we redefine `\makelabel` so that the label will be positioned in the second line of the margin. As a result of this, two macro names appear correctly, one under the other. It's important whilst doing this that the generated label box is not allowed to have more depth than a normal line since otherwise the distance between the first two text lines of TeX will be incorrectly calculated. The definition should then look like:

```
    \def\makelabel##1{\llap{\vtop to \baselineskip
         {\hbox{\strut}\hbox{##1}\vss}}}
```

Completely analogous to this is the case where labels need to be placed one under the other. The lines above are only an example typeset with the verbatim environment. To produce the real definition we save the value of \macro@cnt in \count@ and empty the temp macro \@tempa for later use.

```
    \let\@tempa\@empty \count@\macro@cnt
```

In the following loop we append for every already typeset label an \hbox{\strut} to the definition of \@tempa.

```
    \loop \ifnum\count@>\z@
      \edef\@tempa{\@tempa\hbox{\strut}}\advance\count@\m@ne \repeat
```

Now be put the definition of \makelabel together.

```
    \edef\makelabel##1{\llap{\vtop to\baselineskip
                                    {\@tempa\hbox{##1}\vss}}}%
```

Next we increment the value of the nesting depth counter. This value inside the macro environment is always at least one after this point, but its toplevel definition is zero. Provided this environment has been used correctly, \macro@cnt = 0 should not occur when @inlabel = true. It is however possible if this environment is used within other list environments (but this would have little point).

```
    \advance \macro@cnt \@ne
```

If @inlabel is false we reset \macro@cnt assuming that there is enough room to print the macro name without shifting.

```
  \else  \macro@cnt\@ne  \fi
```

Now the label will be produced using \item. The following line is only a hack saving the day until a better solution is implemented. We have to face two problems: the argument might be a \par which is forbidden in the argument of other macros if they are not defined as \long, or it is something like \iffalse or \else, i.e. something which will be misinterpreted when TeX is skipping conditional text. In both cases \item will bomb, so we protect the argument by using \string.

```
  \edef\@tempa{\noexpand\item[\noexpand\PrintMacroName{\string#1}]}\@tempa
```

At this point we also produce an index entry. Because it is not known which index sorting program will be used, we do not use the command \index, but rather a command \SpecialMainIndex. This may be redefined by the user in order to generate an index entry which will be understood by the index program in use (note the definition of \SpecialMainIndex for our installation).

```
  \SpecialMainIndex{#1}\nobreak
```

The \nobreak is needed to prevent a page break after the \write produced by the \SpecialMainIndex macro. We exclude the new macro in the cross-referencing feature, to prevent spurious non-main entry references. Again we have to watch out for problematic arguments. In case of \par we wait for a new implementation; the conditionals are uncritical.

```
  \def\@tempa{#1}%
  \ifx\@tempa\@defpar \else \DoNotIndex{#1}\fi
```

Because the space symbol should be ignored between the \begin{macro}{...} and the following text we must take care of this with \ignorespaces.

```
  \ignorespaces}
```

\endmacro At this command nothing special needs to happen. However, the trivlist environment must still be ended. Because of the \endgroup which is used by \end, the changes to \macro@cnt stay local to the environment.

```
\let\endmacro\endtrivlist
```

`\MacroTopsep` Here is the default value for the `\MacroTopsep` parameter used above.

```
\newskip\MacroTopsep     \MacroTopsep = 7pt plus 2pt minus 2pt
```

3.3 Formatting the margin

The following three macros should be user definable. Therefore we define those macros only if they have not already been defined.

`\PrintMacroName` `\PrintDescribeMacro` `\PrintDescribeEnv` The formatting of the macro name in the left margin is done by these macros. We first set a `\strut` to get the height and depth of the normal lines. Then we change to the `\MacroFont` using `\string` to `\catcode` the argument to other (assuming that it is a macro name). Finally we print a space. The font change remains local since this macro will be called inside an `\hbox`.

```
\@ifundefined{PrintMacroName}
   {\def\PrintMacroName#1{\strut \MacroFont \string #1\ }}{}
```

We use the same formatting conventions when describing a macro.

```
\@ifundefined{PrintDescribeMacro}
   {\def\PrintDescribeMacro#1{\strut \MacroFont \string #1\ }}{}
```

To format the name of a new environment there is no need to use `\string`.

```
\@ifundefined{PrintDescribeEnv}
   {\def\PrintDescribeEnv#1{\strut \MacroFont #1\ }}{}
```

3.4 Creating index entries by scanning 'macrocode'

When this doc option is used, it automatically invokes `\makeindex` to cause an `.idx` file to be generated. The following macros ensure that index entries are created for each occurrence of a TEX-like command (something starting with '\'). With the default definitions of `\SpecialMainIndex`, etc., the index file generated is intended to be processed by Chen's makeindex program [4].

Of course, in *this* style file itself we've sometimes had to make | take the rôle of TEX's escape character to introduce command names at places where \ has to belong to some other category. Therefore, we may also need to recognize | as the introducer for a command when setting the text inside the macrocode environment. Other users may have the need to make similar reassignments for their macros.

`\SpecialEscapechar` `\active@escape@char` `\special@escape@char` The macro `\SpecialEscapechar` is used to denote a special escape character for the next macrocode environment. It has one argument—the new escape character given as a 'single-letter' control sequence. Its main purpose is defining `\special@escape@char` to produce the chosen escape character `\catcode`d to 12 and `\active@escape@char` to produce the same character but with `\catcode` 13.

The macro `\special@escape@char` is used to *print* the escape character while `\active@escape@char` is needed in the definition of `\init@crossref` to start the scanning mechanism.

In the definition of `\SpecialEscapechar` we need an arbitrary character with `\catcode` 13. We use '~' and ensure that it is active. The `\begingroup` is used to make a possible change local to the expansion of `\SpecialEscapechar`.

```
\def\SpecialEscapechar#1{%
    \begingroup \catcode`\~\active
```

Now we are ready for the definition of `\active@escape@char`. It's a little tricky: we first define locally the uppercase code of '~' to be the new escape character.

```
        \uccode`\~`#1%
```

Around the definition of \active@escape@char we place an \uppercase command. Recall that the expansion of \uppercase changes characters according to their \uccode, but leaves their \catcode s untouched (cf. TEXbook page 41).

```
      \uppercase{\gdef\active@escape@char{~}}%
```

The definition of \special@escape@char is easier; we use \string to \catcode the argument of \SpecialEscapechar to 12 and suppress the preceding \escapechar.

```
      \escapechar\m@ne  \xdef\special@escape@char{\string#1}%
```

Now we close the group and end the definition: the value of \escapechar as well as the \uccode and \catcode of '~' will be restored.

```
    \endgroup}
```

\init@crossref The replacement text of \init@crossref should fulfil the following tasks:

1) \catcode all characters used in macro names to 11 (i.e. 'letter').
2) \catcode the '\' character to 13 (i.e. 'active').

3a) \let the '\' equal \scan@macro (i.e. start the macro scanning mechanism) if there is no special escape character (i.e. the \special@escape@char is '\').

3b) Otherwise \let it equal \bslash, i.e. produce a printable \.

4) Make the ⟨*special escape character*⟩ active.
5) \let the active version of the special escape character (i.e. the expansion of \active@escape@char) equal \scan@macro.

The reader might ask why we bother to \catcode the '\' first to 12 (at the end of \macro@code) then re-\catcode it to 13 in order to produce a $\backslash_{12}$ in case 3b) above. This is done because we have to ensure that '\' has \catcode 13 within the macrocode environment. Otherwise the delimiter for the argument of \xmacro@code would not be found (parameter matching depends on \catcode s).
Therefore we first re-\catcode some characters.

```
  \begingroup   \catcode`\|=\z@  \catcode`\\=\active
```

We carry out tasks 2) and 3b) first.

```
  |gdef|init@crossref{|catcode`|\|active    |let\|bslash
```

Because of the popularity of the '@' character as a 'letter' in macros, we normally have to change its \catcode here, and thus fulfil task 1). But the macro designer might use other characters as private letters as well, so we use a macro to do the \catcode switching.

```
      |MakePrivateLetters
```

Now we \catcode the special escape character to 13 and \let it equal \scan@macro, i.e. fulfil tasks 4) and 5). Note the use of \expandafter to insert the chosen escape character saved in \special@escape@char and \active@escape@char.

```
      |catcode|expandafter`|special@escape@char|active
      |expandafter|let|active@escape@char|scan@macro}
  |endgroup
```

If there is no special escape character, i.e. if \SpecialEscapechar is \\, the second last line will overwrite the previous definition of $\backslash_{13}$. In this way all tasks are fulfilled.
For happy documenting we give default values to \special@escape@char and \active@escape@char with the following line:

```
  \SpecialEscapechar{\\}
```

`\MakePrivateLetters` Here is the default definition of this command, which makes just the `@` into a letter. The user may change it if he/she needs more or other characters masquerading as letters.

```
\@ifundefined{MakePrivateLetters}
    {\let\MakePrivateLetters\makeatletter}{}
```

`\close@crossref` At the end of a cross-referencing part we prepare ourselves for the next one by setting the escape character to '\'.

```
\def\close@crossref{\SpecialEscapechar\\}
```

3.5 Macros for scanning macro names

`\scan@macro` `\macro@namepart` The `\init@crossref` will have made `\active` our `\special@escape@char`, so that each `\active@escape@char` will invoke `\scan@macro` when within the macrocode environment. By this means, we can automatically add index entries for every TEX-like command which is met whilst setting (in verbatim) the contents of macrocode environments.

```
\def\scan@macro{%
```

First we output the character which triggered this macro. Its version `\catcode`d to 12 is saved in `\special@escape@char`.

```
  \special@escape@char
```

If the macrocode environment contains, for example, the command \\, the second \ should not start the scanning mechanism. Therefore we use a switch to decide if scanning of macro names is allowed.

```
  \ifscan@allowed
```

The macro assembles the letters forming a TEX command in `\macro@namepart` so this is initially cleared; we then set `\next` to the *first* character following the \ and call `\macro@switch` to determine whether that character is a letter or not.

```
    \let\macro@namepart\@empty
    \def\next{\futurelet\next\macro@switch}%
```

As you recognize, we actually did something else, because we have to defer the `\futurelet` call until after the final `\fi`. If, on the other hand, the scanning is disabled we simply `\let \next` equal 'empty'.

```
  \else \let\next\@empty \fi
```

Now we invoke `\next` to carry out what's needed.

```
  \next}
```

`\ifscan@allowed` `\scan@allowedtrue` `\scan@allowedfalse` `\ifscan@allowed` is the switch used above to determine if the `\active@escape@char` should start the macro scanning mechanism.

```
\newif\ifscan@allowed    \scan@allowedtrue
```

`\EnableCrossrefs` `\DisableCrossrefs` At this point we might define two macros which allow the user to disable or enable the cross-referencing mechanism. Processing of files will be faster if only main index entries are generated (i.e., if `\DisableCrossrefs` is in force).

```
\def\DisableCrossrefs{\@bsphack\scan@allowedfalse\@esphack}
```

The macro `\EnableCrossrefs` will also disable any `\DisableCrossrefs` command encountered afterwards.

```
\def\EnableCrossrefs{\@bsphack\scan@allowedtrue
                     \def\DisableCrossrefs{\@bsphack\@esphack}\@esphack}
```

`\macro@switch` Now that we have the character which follows the escape character (in `\next`), we can determine whether it's a 'letter' (which category probably includes `@`).

If it is, we let `\next` invoke a macro which assembles the full command name.

```
\def\macro@switch{\ifcat\noexpand\next a%
     \let\next\macro@name
```

Otherwise, we have a 'single-character' command name. For all such single-character names, we use `\short@macro` to process them into suitable index entries.

```
     \else \let\next\short@macro  \fi
```

Now that we know what macro to use to process the macro name, we invoke it ...

```
    \next}
```

`\short@macro` This macro will be invoked (with a single character as parameter) when a single-character macro name has been spotted whilst scanning within the macrocode environment.

First we take a look at the `\index@excludelist` to see whether this macro name should produce an index entry. This is done by the `\ifnot@exlcuded` macro which assumes that the macro name is saved in `\macro@namepart`. Since the argument might be an active character, `\string` is used to normalize it.

```
\def\short@macro#1{\edef\macro@namepart{\string#1}%
   \ifnot@excluded
```

If necessary the index entry is produced by the macro `\produce@index`. Depending on the actual character seen, this macro has to do different things, so we pass the character as an argument.

```
     \produce@index{#1}\fi
```

Then we disable the cross-referencing mechanism with `\scan@allowedfalse` and print the actual character. The index entry was generated first to ensure that no page break intervenes (recall that a `^^M` will start a new line).

```
     \scan@allowedfalse#1%
```

After typesetting the character we can safely enable the cross-referencing feature again. Note that this macro won't be called (since `\macro@switch` won't be called) if cross-referencing is globally disabled.

```
     \scan@allowedtrue }
```

`\produce@index` This macro is supposed to generate a suitable `\SortIndex` command for a given single-character control sequence. We test first for the cases which involve active characters (i.e. the backslash, the special escape character (if any), the space and the `^^M`). Using the `\if` test (testing for character codes), we have to ensure that the argument isn't expanded.

```
\def\produce@index#1{%
  \if\noexpand#1\special@escape@char
```

If the character is the special escape character (or the '\' in case there was none) the `\it@is@a` macro is used to produce the actual `\SortIndex` call.

```
     \scan@allowedfalse \it@is@a\special@escape@char \else
```

Next comes the test for a '\' which must be the $\backslash_{13}$ expanding to `\bslash`.

```
     \if\noexpand#1\bslash \it@is@a\bslash \else
```

Another possibility is $\sqcup_{13}$. Recall that `\space` produces a $\sqcup_{10}$.

```
       \if\noexpand#1\space \it@is@a\space \else
```

The last[16] possibility of an active character is `^^M`. In this case we don't test for character codes, since it is easier to look if the character is equal to `\par`. (We are inside the macrocode environment.)

```
\ifx#1\par
```

If we end up here we have just scanned a `\^^M` or something similar. Since this will be treated like `\␣` by TeX we produce a corresponding index entry.

```
\it@is@a\space \else
```

The next three branches are needed because of bugs in our makeindex program. You can't produce unbalanced index entries[17] and you have to double a percent character. To get around these restrictions we use special macros to produce the `\index` calls.[18]

```
\if\noexpand#1\bgroup \LeftBraceIndex \else
  \if\noexpand#1\egroup \RightBraceIndex \else
    \if\noexpand#1\percentchar \PercentIndex \else
```

All remaining characters are used directly to produce their index entries. This is possible even for the characters which have special meanings in the index program, provided we quote the characters. (This is correctly done in `\it@is@a`.)

```
\it@is@a{\string#1}%
```

We now need a whole pile of `\fi`s to match up with the `\if`s.

```
\fi \fi \fi \fi \fi \fi \fi}
```

`\macro@name` We now come to the macro which assembles command names which consist of one or more 'letters' (which might well include `@` symbols, or anything else which has a `\catcode` of 11).

To do this we add the letter to the existing definition of `\macro@namepart` (which you will recall was originally set to `\@empty`).

```
\def\macro@name#1{\edef\macro@namepart{\macro@namepart#1}%
```

Then we grab hold of the *next* single character and let `\more@macroname` determine whether it belongs to the letter string forming the command name or is a non-letter.

```
\futurelet\next\more@macroname}
```

`\more@macroname` This causes another call of `\macro@name` to add in the next character, if it is indeed a letter.

```
\def\more@macroname{\ifcat\noexpand\next a%
    \let\next\macro@name
```

Otherwise, it finishes off the index entry by invoking `\macro@finish`.

```
\else \let\next\macro@finish \fi
```

Here's where we invoke whatever macro was `\let` equal to `\next`.

```
\next}
```

[16] Well, it isn't the last active character after all. I added `\@noligs` some days ago and now ' too is active. So we have to make sure that such characters don't get expanded in the index.

[17] This is possible for TeX if you use $\{_{12}$ or $\}_{12}$, but makeindex will complain.

[18] Brian HAMILTON KELLY has written fixes for all three bugs. When they've found their way through all installations, the lines above will be removed. See page 265 if you already have them.

`\macro@finish` When we've assembled the full letter-string which forms the command name, we set the characters forming the entire command name, and generate an appropriate `\index` command (provided the command name is not on the list of exclusions). The '\' is already typeset; therefore we only have to output all letters saved in `\macro@namepart`.

```
\def\macro@finish{%
  \macro@namepart
```

Then we call `\ifnot@excluded` to decide whether we have to produce an index entry. The construction with `\@tempa` is needed because we want the expansion of `\macro@namepart` in the `\index` command.[19]

```
   \ifnot@excluded
     \edef\@tempa{\noexpand\SpecialIndex{\bslash\macro@namepart}}%
     \@tempa  \fi}
```

3.6 The index exclude list

The internal form of the index exclude list is

`\@elt` ⟨*macro name*⟩ `\@elt` ⟨*macro name*⟩ `\@elt` ⟨*macro name*⟩ ...

where ⟨*macro name*⟩ is a macro name like `\@tempa`. To test if a given macro name is on the list we only have to assign `\@elt` a proper meaning and then call `\index@excludelist`. This is faster than looping through the list and looking for the last element.

`\DoNotIndex` This macro is used to suppress macro names in the index. It starts off with a new group because we have to change the `\catcode`s of all characters which belong to 'letters' while macros are defined.

```
\def\DoNotIndex{\begingroup \MakePrivateLetters
```

Then we call the macro which actually reads the argument given by the user.

```
    \do@not@index}
```

`\do@not@index` We make the `\do@not@index` macro `\long` since the user might want to exclude the `\par` macro.[20]

```
\long\def\do@not@index#1{%
```

Now we have to face the problem that `\index@excludelist` should be changed only locally (and we are already in a new group). Therefore we pass its contents to `\@gtempa`.

```
    \global\let\@gtempa\index@excludelist
```

Then we define `\@elt` to allow expanding `\@gtempa` without damaging its contents.

```
    \def\@elt{\noexpand\@elt\noexpand}%
```

The argument to `\do@not@index` is a set of macros separated by commas, so we use `\@for` to extract individual entries. Since `\@for` expands its argument, we hide it in another macro.[21]

```
    \def\@tempa{#1}\@for\@tempb:=\@tempa\do
```

Now we can safely add new entries to `\@gtempa` (i.e. `\index@excludelist`).

```
      {\xdef\@gtempa{\@gtempa \expandafter \@elt \@tempb}}%
```

[19] The `\index` command will expand its argument in the `\output` routine. At this time `\macro@namepart` might have a new value.

[20] Actually this doesn't work either because the argument is passed to `\@forloop` which isn't a `\long` macro. This will be fixed in a later version.

[21] Instead of using `\@for` one can make the comma active (expanding into `\@elt`) and then simply putting `\@gtempa` and `,#1` together. This will probably change.

After this we close the group and assign the retained value of `\@gtempa` to `\index@excludelist`.

```
    \endgroup \let\index@excludelist\@gtempa}
```

`\index@excludelist` `\@for` To get things going we have to initialize `\index@excludelist` and fix a bug in the LaTeX `\@for` macro.

```
  \def\index@excludelist{}
```

In the original `\@for` macro, `\@fortmp` gets its value using `\edef`. This means that it bombs if the contents of the second argument (i.e. the list A,B,C,...) contains undefined macros. Since this is possible if we use the `doc.sty` file to document a macro package which isn't loaded, we change the definition a little bit. Now it is tested only if the second argument expands to 'empty' when it is expanded once.[22]

```
  \def\@for#1:=#2\do#3{\expandafter\def\expandafter\@fortmp\expandafter{#2}%
         \ifx\@fortmp\@empty \else
             \expandafter\@forloop#2,\@nil,\@nil,\@@#1{#3}\fi}
```

`\ifnot@excluded` Now we take a look at the `\index@excludelist` to see whether a macro name saved in `\macro@namepart` should produce an index entry. This macro is a pseudo `\if`; it should expand to `\iftrue` or `\iffalse` depending on the contents of `\index@excludelist`. We use `\if@tempswa` for this purpose and initialize it with true.

```
  \def\ifnot@excluded{\@tempswatrue
```

Then we `\let` the macro `\@elt` equal to a test macro (which is supposed to change the switch if the `\macro@namepart` is on the list) and call `\index@excludelist`. This is all done in an `\hbox` so any garbage produced by calling `\index@includelist` will vanish. The test macro uses `\aftergroup` to avoid global changes while communicating with the outside world.

```
      \setbox\z@\hbox{\let\@elt\exclude@test \index@excludelist}%
```

Finally we call `\if@tempswa`.

```
      \if@tempswa}
```

Note however that since we have called `\if@tempswa` inside this macro, such a construction can't be used inside a conditional at the same expansion level [3, p211].

`\exclude@test` Strictly speaking, `\macro@namepart` contains only the name without the backslash. So we use `\expandafter` and `\csname` to produce the actual macro name (arguments to `\ifx` are not expanded).

```
  \def\exclude@test#1{%
      \expandafter \ifx \csname\macro@namepart\endcsname #1%
```

The `\ifx` test will be true if the argument and the constructed macro are the same. In this case we have to change the switch. We also change the definition of `\@elt` because our goal is reached and we can gobble up the tail of the list. As mentioned above, switch changing is deferred until after the current group by using `\aftergroup`.

```
          \aftergroup\@tempswafalse \let\@elt\@gobble \fi}
```

[22] This is exactly the way in which the argument is used in the second part of the definition (in the actual loop). Therefore I am inclined to call it a bug and not a feature.

3.7 Macros for generating index entries

Here we provide default definitions for the macros invoked to create index entries; these are either invoked explicitly, or automatically by `\scan@macro`. As already mentioned, the definitions given here presuppose that the `.idx` file will be processed by Chen's makeindex program — they may be redefined for use with the user's favourite such program.

To assist the reader in locating items in the index, all such entries are sorted alphabetically *ignoring* the initial '\'; this is achieved by issuing an `\index` command which contains the 'actual' operator for makeindex. The default value for the latter operator is '`@`', but the latter character is so popular in LaTeX style files that it is necessary to substitute another character. This is indicated to makeindex by means of an 'index style file'[23]; the character selected for this function is `=`, and therefore this character too must be specially treated when it is met in a TeX command.

`\actualchar` **`\quotechar`** **`\levelchar`**

First come the definitions of `\actualchar`, `\quotechar` and `\levelchar`. Note, that our defaults are not the ones used by the makeindex program without a style file.

```
\@ifundefined{actualchar}{\def\actualchar{=}}{}
\@ifundefined{quotechar}{\def\quotechar{!}}{}
\@ifundefined{levelchar}{\def\levelchar{>}}{}
```

`\encapchar`

The makeindex default for the `\encapchar` isn't changed.

```
\@ifundefined{encapchar}{\def\encapchar{|}}{}
```

`\verbatimchar`

We also need a special character to be used as a delimiter for the `\verb*` command used in the definitions below.

```
\@ifundefined{verbatimchar}{\def\verbatimchar{+}}{}
```

`\SpecialIndex`

The `\SpecialIndex` command creates index entries for macros. If the argument is `\`*xyz*, the command produces `\indexentry{`*xyz*`=\verb!*+\`*xyz*`+}{`*n*`}` given the above defined defaults for `\actualchar`, `\quotechar` and `\verbatimchar`. We first remove the initial '\' to get a better index.

```
\def\SpecialIndex#1{\@bsphack\index{\expandafter\@gobble\string#1\actualchar
```

Then follows the actual entry. A `\quotechar` is placed before the `*` to allow its use as a special makeindex character. Again `\@bsphack` and `\@esphack` are used to make the macros invisible.

```
        \string\verb\quotechar*\verbatimchar\string#1\verbatimchar}\@esphack}
```

`\SpecialMainIndex` **`\SpecialUsageIndex`**

The `\SpecialMainIndex` macro is used to cross-reference the names introduced by the macro environment. The action is as for `\SpecialIndex`, except that makeindex is instructed to 'encap'sulate the entry with the string `|main` to cause it to generate a call of the `\main` macro.

```
\def\SpecialMainIndex#1{\@bsphack\index{\expandafter\@gobble
                                         \string#1\actualchar
                                         \string\verb
                                         \quotechar*\verbatimchar
                                         \string#1\verbatimchar
                                         \encapchar main}%
                          \@esphack}
```

The `\SpecialUsageIndex` is literally the same—only we use `usage` instead of `main`.

```
\def\SpecialUsageIndex#1{\@bsphack\index{\expandafter\@gobble\string#1%
    \actualchar\string\verb\quotechar*\verbatimchar\string#1\verbatimchar
    \encapchar usage}\@esphack}
```

[23] A file suitable to this task is provided amongst the supporting files for this style file in `gind.ist`.

`\SpecialEnvIndex` Indexing environments is done a little bit differently; we produce two index entries with the \SpecialEnvIndex macro:

```
\def\SpecialEnvIndex#1{\@bsphack
```

First we sort the environment under its own name stating in the actual entry that this is an environment.

```
    \index{#1\actualchar{\tt #1} (environment)\encapchar usage}%
```

The second entry is sorted as a subitem under the key 'environments:'.

```
    \index{environments:\levelchar{\tt #1}\encapchar usage}\@esphack}
```

Because both entries corresponds to 'descriptions' of the environment, we encapsulate the page numbers with the \usage macro.

`\SortIndex` This macro is used to generate the index entries for any single-character command that \scan@macro encounters. The first parameter specifies the lexical order for the character, whilst the second gives the actual characters to be printed in the entry. It can also be used directly to generate index entries which differ in sort key and actual entry.

```
\def\SortIndex#1#2{\index{#1\actualchar#2}}
```

`\it@is@a` This macro is supposed to produce a correct \SortIndex entry for a given character. Since this character might be recognised as a 'command' character by the index program used, all characters are quoted with the \quotechar.

```
\def\it@is@a#1{\SortIndex{\quotechar #1}%
                         {\string\verb\quotechar*\verbatimchar
                          \quotechar\bslash\quotechar#1\verbatimchar}}
```

`\LeftBraceIndex` `\RightBraceIndex` These two macros fix the problems with makeindex. Note the 'hack' with \iffalse}\fi to satisfy both TeX and the makeindex program. When this is written to the .idx file TeX will see both braces (so we get a balanced text). makeindex will also see balanced braces but when the actual index entry is again processed by TeX the brace in between \iffalse \fi will vanish.

```
\@ifundefined{LeftBraceIndex}{\def\LeftBraceIndex{%
   \index{\bgroup\actualchar\string\verb\quotechar*\verbatimchar
         \quotechar\bslash{\verbatimchar\string\iffalse}\string\fi}}}{}

\@ifundefined{RightBraceIndex}{\def\RightBraceIndex{%
   \index{\egroup\actualchar\string\iffalse{\string\fi\string\verb
         \quotechar*\verbatimchar\quotechar\bslash}\verbatimchar}}}{}
```

`\PercentIndex` `\percentchar` Here is one solution for the percent bug in makeindex. The macro \percentchar denotes a $\%_{12}$.

```
\@ifundefined{PercentIndex}{\def\PercentIndex{%
   \index{\quotechar\percentchar\actualchar\string\verb
         \quotechar*\verbatimchar\quotechar\bslash
         \percentchar\percentchar\verbatimchar}}}{}

{\catcode'\%=12 \gdef\percentchar{%}}
```

If you've got a newer makeindex program which handles the percents correctly you have to uncomment the next three lines.[24] Otherwise you will get \%% entries in your index.

```
% \def\PercentIndex{\it@is@a\percentchar}
% \typeout{The doc style option assumes that a \percentchar\space
%          will be processed as a \percentchar\space by the index program!}
```

[24] This is the only change which is allowed in this file! All other changes should be made through a style file, predefining internal quantities.

3.8 Redefining the index environment

The index is set in three columns, and will start on the same page as, and underneath, the last part of the text of the documented style file, if possible. The last page will be reformatted with balanced columns. We make use of the multicols environment which is described elsewhere (in an article scheduled to appear in the next issue of TUGboat).

```
\input{multicol.sty}
```

\IndexMin **\c@IndexColumns** When the index is started we compute the remaining space on the current page; if it is greater than \IndexMin, the first part of the index will then be placed in the available space. The number of columns set is controlled by the counter \c@IndexColumns which can be changed with a \setcounter declaration.

```
\newdimen\IndexMin          \IndexMin       = 80pt
\newcount\c@IndexColumns    \c@IndexColumns = 3
```

\theindex Now we start the multi-column mechanism. We use the \c@IndexColumns LaTeX counter declared above to denote the number of columns and insert the 'index prologue' text (which might contain a \section call, etc.). See the default definition for an example.

```
\def\theindex{\multicols\c@IndexColumns[\index@prologue][\IndexMin]%
```

Then we make a few last minute assignments to read the individual index \items and finish off by ignoring any initial space.

```
      \IndexParms \let\item\@idxitem \ignorespaces}
```

\endtheindex At the end of the index, we have only to end the multicols environment.

```
\let\endtheindex=\endmulticols
```

\IndexPrologue **\index@prologue** The \IndexPrologue macro is used to place a short message into the document above the index. It is implemented by redefining \index@prologue, a macro which holds the default text. We'd better make it a \long macro to allow \par commands in its argument.

```
\long\def\IndexPrologue#1{\@bsphack\def\index@prologue{#1}\@esphack}
```

Now we test whether the default is already defined by another style file. If not we define it.

```
\@ifundefined{index@prologue}
     {\def\index@prologue{\section*{Index}%
                 \markboth{Index}{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.
                 }}{}
```

\IndexParms These are some last-minute assignments for formatting the index entries. They are defined in a separate macro so that a user can substitute different definitions. We start by defining the various parameters controlling leading and the separation between the two columns. The entire index is set in \small size.

```
\@ifundefined{IndexParms}
    {\def\IndexParms{%
       \parindent \z@
       \columnsep 15pt
       \parskip 0pt plus 1pt
       \rightskip 15pt
```

```
        \mathsurround \z@
        \parfillskip=-15pt
         \small
```

`\@idxitem` `\subitem` `\subsubitem` Index items are formatted with hanging indentation for any items which may require more than one line.

```
        \def\@idxitem{\par\hangindent 30pt}%
```

Any sub-item in the index is formatted with a 15pt indentation under its main heading.

```
        \def\subitem{\@idxitem\hspace*{15pt}}%
```

Whilst sub-sub-items go in a further 10pt.

```
        \def\subsubitem{\@idxitem\hspace*{25pt}}%
```

`\indexspace` The makeindex program generates an \indexspace before each new alphabetic section commences. After this final definition we end the \@ifundefined and the definition of \IndexParms.

```
        \def\indexspace{\par\vspace{10pt plus 2pt minus 3pt}}%
       }}{}
```

`\efill` This definition of \efill is intended to be used after index items which have no following text (for example, "*see*" entries). It just ensures that the current line is filled, preventing "Underfull \hbox" messages.

```
\def\efill{\hfill\nopagebreak}%
```

`\pfill` `\dotfil` `\dotfill` The following definitions provide the \pfill command; if this is specified in the index style file to makeindex as the delimiter to appear after index items, then the intervening space before the referenced page numbers will be filled with dots, with a little white space interpolated at each end of the dots. If the line is broken the dots will show up on both lines.

```
\def\dotfill{\leaders\hbox to.6em{\hss .\hss}\hskip\z@ plus  1fill}%
\def\dotfil{\leaders\hbox to.6em{\hss .\hss}\hfil}%
\def\pfill{\unskip~\dotfill\penalty500\strut\nobreak
             \dotfil~\ignorespaces}%
```

`\*` Here is the definition for the * macro. It isn't used in this set of macros.

```
\def\*{\leavevmode\lower.8ex\hbox{$\,\widetilde{\ }\,$}}
```

`\main` The *defining* entry for a macro name is flagged with the string |main[25] in the \index command; makeindex processes this so that the \main macro will be invoked to underline the page number(s) on which the *definition* of the macro will be found.

```
\@ifundefined{main}{\def\main#1{\underline{#1}}}{}
```

`\usage` The \usage macro is used to indicate entries describing the usage of a macro. The corresponding page number(s) will be set in *italics*.

```
\@ifundefined{usage}{\def\usage#1{{\it #1}}}{}
```

`\printindex` To read in and print the sorted index, just put the \printindex command as the last (commented-out, and thus executed during the documentation pass through the file) command in your style file. Precede it by any bibliography commands necessary for your citations.

Alternatively, it may be more convenient to put all such calls amongst the arguments of the \StopEventually macro, in which case a \Finale command should appear at the end of your file.

```
\def\printindex{\@input{\jobname.ind}}
```

[25] With the current definition of \encapchar substituted for |

3.9 Dealing with the change history[26]

To provide a change history log, the \changes command has been introduced. This takes three arguments, respectively, the version number of the file, the date of the change, and some detail regarding what change has been made. The first of these arguments is otherwise ignored, but the others are written out and may be used to generate a history of changes, to be printed at the end of the document. However, note that Chen's standard makeindex program limits any textual field to just 64 characters; therefore, is important that the number of characters in the second and third parameters should not exceed 61 altogether (to allow for the parentheses placed around the date.

\changes The output of the \changes command goes into the ⟨*Glossary_File*⟩ and therefore uses the normal \indexentry commands. Thus makeindex or a similar program can be used to process the output into a sorted "glossary". The \changes command commences by taking the usual measures to hide its spacing, and then redefines \protect for use within the argument of the generated \indexentry command.

```
\def\changes#1#2#3{\@bsphack{%
  \def\protect##1{\string##1\space}%
```

We now create the requisite \glossary command, and output it.

```
  \edef\@tempa{\noexpand\glossary{\expandafter\@gobble
                                  \saved@macroname\actualchar
                                  \string\verb\quotechar*%
                                  \verbatimchar\saved@macroname
                                  \verbatimchar\levelchar#2%
                                  \actualchar(#2) #3}}%
  \@tempa}\@esphack}
```

\saved@macroname The entries are sorted for convenience by the name of the most recently introduced macroname (i.e., that in the most recent \begin{macro} command). We therefore provide \saved@macroname to record that argument, and provide a default definition in case \changes is used outside a macro environment. (This is a *wicked* hack to get such entries at the beginning of the sorted list!)

```
\def\saved@macroname{' General Changes '}
```

\RecordChanges To cause the changes to be written (to a .glo) file, we define \RecordChanges to invoke LaTeX's usual \makeglossary command.

```
\let\RecordChanges\makeglossary
```

\GlossaryMin
\c@GlossaryColumns The remaining macros are all analogues of those used for the theindex environment. When the glossary is started we compute the space which remains at the bottom of the current page; if this is greater than \GlossaryMin then the first part of the glossary will be placed in the available space. The number of columns set are controlled by the counter \c@GlossaryColumns which can be changed with a \setcounter declaration.

```
\newdimen\GlossaryMin         \GlossaryMin        = 80pt
\newcount\c@GlossaryColumns   \c@GlossaryColumns = 2
```

\theglossary We start with a few last minute assignments to read the individual glossary \items. Note the \par at the beginning. If we leave it out, parameter changes done by the \GlossaryParms macro might affect the paragraph above the glossary.

```
\def\theglossary{\par \GlossaryParms \let\item\@idxitem
```

[26] The whole section was proposed by Brian HAMILTON KELLY. He also documented and debugged the macros as well as many other parts of this style option.

Now we start the multi-column mechanism. We use `\c@GlossaryColumns` to denote the number of columns and insert the 'glossary prologue' text which might contain a `\section` call etc. See the default definition for an example.

```
      \multicols\c@GlossaryColumns[\glossary@prologue][\GlossaryMin]}
```

`\endglossary` At the end of the glossary, we've only to end the multicols environment.

```
\let\endtheglossary=\endmulticols
```

`\GlossaryPrologue` `\glossary@prologue` The `\GlossaryPrologue` macro is used to place a short message above the glossary into the document. It is implemented by redefining `\glossary@prologue`, a macro which holds the default text. We better make it a long macro to allow `\par` commands in its argument.

```
\long\def\GlossaryPrologue#1{\@bsphack
                           \def\glossary@prologue{#1}%
                           \@esphack}
```

Now we test whether the default is already defined by another style file. If not we define it.

```
\@ifundefined{glossary@prologue}
     {\def\glossary@prologue{\begingroup\parfillskip=0pt plus 1fil
 \section*{{Change History}}\par
 \endgroup
                 \markboth{{Change History}}{{Change History}}%
                 }}{}
```

`\GlossaryParms` Unless the user specifies otherwise, we set the change history using the same parameters as for the index.

```
\@ifundefined{GlossaryParms}{\let\GlossaryParms=\IndexParms}{}
```

`\PrintChanges` To read in and print the sorted change history, just put the `\PrintChanges` command as the last (commented-out, and thus executed during the documentation pass through the file) command in your style file. Alternatively, this command may form one of the arguments of the `\StopEventually` command, although a change history is probably *not* required if only the description is being printed.

The command assumes that makeindex or some other program has processed the `.glo` file to generate a sorted `.gls` file.

```
\def\PrintChanges{\@input{\jobname.gls}}
```

3.10 Bells and whistles

`\StopEventually` `\Finale` `\OnlyDescription` Here is the default definition for `\StopEventually`, we simply save its argument in the macro `\Finale`.

```
\long\def\StopEventually#1{\@bsphack\def\Finale{#1}\@esphack}
```

When the user places an `\OnlyDescription` declaration in the driver file the document should only be typeset up to `\StopEventually`. We therefore have to redefine this macro.

```
\def\OnlyDescription{\@bsphack\long\def\StopEventually##1{%
```

In this case the argument of `\StopEventually` should be set and afterwards TEX should stop reading from this file. Therefore we finish this macro with

```
          ##1\endinput}\@esphack}
```

`\meta` The `\meta` macro is very elementary.

```
\def\meta#1{\mbox{$\langle$\it#1\/$\rangle$}}
```

`\IndexListing` This next macro may be used to read in a separate file (possibly a style file that is *not* documented by this means) and set it verbatim, whilst scanning for macro names and indexing the latter. This could be a useful first pass in preparing to generate documentation for the file read.

```
\def\IndexListing#1{%
```

We commence by setting up a group, and initializing a `\trivlist` as is normally done by a `\begin{macrocode}` command.

```
        \begingroup \macro@code
```

We also make spacing behave as in the macrocode environment, because otherwise all the spaces will be shown explicitly.

```
        \frenchspacing \@vobeyspaces
```

Then it only remains to read in the specified file, and finish off the `\trivlist`.

```
        \input{#1}\endtrivlist
```

Of course, we need to finish off the group as well.

```
        \endgroup}
```

`\maketitle` The macro to generate titles is easily altered in order that it can be used more than once (an article with many titles). In the original, diverse macros were concealed after use with `\relax`. We must cancel anything that may have been put into `\@thanks`, etc., otherwise *all* titles will carry forward any earlier such setting!

```
\def\maketitle{\par
        \begingroup \def \thefootnote {\fnsymbol {footnote}}%
        \setcounter {footnote}\z@
        \def \@makefnmark {\hbox to \z@{$^{\@thefnmark }$\hss }}%
        \if@twocolumn \twocolumn [\@maketitle ]
        \else \global \@topnum \z@ \@maketitle \fi
        \@thanks \endgroup
```

If the driver file documents many files, we don't want parts of a title of one to propagate to the next, so we have to cancel these:

```
        \setcounter {footnote}\z@
        \gdef\@date{\today}\gdef\@thanks{}%
        \gdef\@author{}\gdef\@title{}}
```

3.11 Layout parameters for documenting style files

`\tolerance` People documenting style files would probably rather have things "sticking out" in overfull `\hbox`es and poorish spacing, because they probably don't want to spend a lot of time on making all the line breaks perfect!

```
        \tolerance=1000\relax
```

The following `\mathcode` definitions allow the characters '`\`' and '`@`' to appear in `\tt` font when invoked in math mode;[27] particularly for something like `\@abc` = 1.

If an *old* version of the german style option is in force, then the '`"`' character is active and would upset the definition of the ⟨*16-bit number*⟩ quantities below, therefore we change the `\catcode` of " inside a group, and use `\global`.

```
{ \catcode`\"=12
  \global\mathcode`\\="705C \global\mathcode`\@="7040 }
```

[27] You may wonder why the definitions state that both characters belong to the *variable family* (i.e. the number 7 in front). The reason is this: Originally the `\mathcode` of \ was defined to be "075C, i.e. ordinary character number 92 (hex 5C) in math family number 7 which is the typewriter family in standard LaTeX. But this file should not depend on this specific setting, so I changed these `\mathcode`s to work with any family assignments. For an example see the article about the new font selection scheme.

`\DocstyleParms` This macro can be used, for example, to assign new values to `\MacrocodeTopsep` and `\MacroIndent` and some other internal registers. If it is already defined, the default definition won't be carried out. Note that it is necessary to assign new values via this macro if it should be done in a style file (like `ltugbot.sty` for example) since the registers are undefined before `doc.sty` is read in. The default values for the internal registers are scattered over this file. Here we only execute `\makeindex` because this declaration can't be overwritten otherwise.

```
\@ifundefined{DocstyleParms}{\makeindex}{}
```

Now we allow overwriting the values by calling `\DocstyleParms`.

```
\DocstyleParms      \let\DocstyleParms\relax
```

`\AmSTeX` `\BibTeX` `\SliTeX` Here are a few definitions which can usefully be employed when documenting style files: now we can readily refer to $\mathcal{A}\kern-.2em\lower.376ex\hbox{$\mathcal{M}$}\kern-.2em\mathcal{S}$-TeX, BibTeX and SliTeX, as well as the usual TeX and LaTeX.

```
\@ifundefined{AmSTeX}
   {\def\AmSTeX{\leavevmode\hbox{$\cal A\kern-.2em\lower.376ex%
        \hbox{$\cal M$}\kern-.2em\cal S$-\TeX}}}{}
\@ifundefined{BibTeX}
   {\def\BibTeX{{\rm B\kern-.05em{\sc i\kern-.025em b}\kern-.08em%
    T\kern-.1667em\lower.7ex\hbox{E}\kern-.125emX}}}{}
\@ifundefined{SliTeX}
   {\def\SliTeX{{\rm S\kern-.06em{\smc l\kern-.035emi}\kern-.06em\TeX}}}{}
```

`\PlainTeX` `\Web` There's even a PLAIN TeX and a WEB.

```
\@ifundefined{PlainTeX}{\def\PlainTeX{{\sc Plain}\kern2pt\TeX}}{}
\@ifundefined{Web}{\def\Web{{\sc Web}}}{}
```

3.12 Changing the `\catcode` of %

`\MakePercentIgnore` `\MakePercentComment` And finally the most important bit: we change the `\catcode` of '`%`' so that it is ignored (which is how we are able to produce this document!). We provide two commands to do the actual switching. Then `\MakePercentIgnore` is called as the last command in this file.

```
\def\MakePercentIgnore{\catcode'\%9\relax}
\def\MakePercentComment{\catcode'\%14\relax}
```

References

[1] G. A. BÜRGER. Wunderbare Reisen zu Wasser und zu Lande, Feldzüge und lustige Abenteuer des Freyherrn v. Münchhausen. London, 1786 & 1788.

[2] D. E. KNUTH. Literate Programming. Computer Journal, Vol. 27, *pp.* 97–111, May 1984.

[3] D. E. KNUTH. Computers & Typesetting (The TeXbook). Addison-Wesley, Vol. A, 1986.

[4] L. LAMPORT. MakeIndex: An Index Processor for LaTeX. 17 February 1987. (Taken from the file `makeindex.tex` provided with the program source code.)

[5] R. E. RASPE (*1737, †1797). Baron Münchhausens narrative of his marvellous travels and campaigns in Russia. Oxford, 1785.

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.

◇ Frank Mittelbach
Fachbereich Mathematik
Universitat Mainz
Staudinger Weg 9
D-6500 Mainz
Federal Republic of Germany
Bitnet: `schoepf@dmznat51`