

The `doc` and `shortvrb` Packages*

Frank Mittelbach[†]

February 22, 2017

This file is maintained by the L^AT_EX Project team.
Bug reports can be opened (category `latex`) at
<http://latex-project.org/bugs.html>.

Abstract

This package contains the definitions that are necessary to format the documentation of package files. The package was developed in Mainz in cooperation with the Royal Military College of Science. This is an update which documents various changes and new features in `doc` and integrates the features of `newdoc`.

Contents

1	Introduction	3		
1.1	Using the <code>doc</code> package . . .	4	2.10	Changing the default values of style parameters . . . 10
2	The User Interface	4	2.11	Short input of verbatim text pieces 10
2.1	The driver file	4	2.12	Additional bells and whistles 10
2.2	General conventions	5	2.13	Basic usage summary . . . 13
2.3	Describing the usage of new macros	6	2.14	Acknowledgements 14
2.4	Describing the definition of new macros	6	3	The Description of Macros 14
2.5	Formatting the margins	6	3.1	Options supported by <code>doc</code> 15
2.6	Using a special escape character	7	3.2	Macros surrounding the ‘definition parts’ 15
2.7	Cross-referencing all macros used	7	3.3	Macros for the ‘documentation parts’ 22
2.8	Producing the actual index entries	8	3.4	Formatting the margin . . . 27
2.9	Setting the index entries	9	3.5	Creating index entries by scanning ‘macrocode’ 27

*This file has version number v2.1g dated 2016/02/15.

[†]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; fairly substantial additions, particularly from `newdoc`, and documentation of post-v1.5q features added at v1.7a by Dave Love (SERC Daresbury Lab). Extraction of `shortvrb` package added by Joachim Schrod (TU Darmstadt).

3.6	Macros for scanning macro names	29	3.12	Providing a checksum and character table	50
3.7	The index exclude list	33	3.13	Attaching line numbers to code lines	52
3.8	Macros for generating index entries	35	3.14	Layout Parameters for documenting package files	53
3.9	Redefining the index en- vironment	38	3.15	Changing the <code>\catcode</code> of %	54
3.10	Dealing with the change history	42	3.16	<code>GetFileInfo</code>	54
3.11	Bells and whistles	45			

Preface to version 1.7

This version of `doc.dtx` documents changes which have occurred since the last published version [5] but which have been present in distributed versions of `doc.sty` for some time. It also integrates the (undocumented) features of the distributed `newdoc.sty`.

The following changes and additions have been made to the user interface since the published version [5]. See §2 for more details.

Driver mechanism `\DocInput` is now used in the driver file to input possibly multiple independent `doc` files and `doc` no longer has to be the last package. `\IndexListing` is replaced by `\IndexInput`;

Indexing is controlled by `\PageIndex` and `\CodelineIndex`, one of which must be specified to produce an index—there is no longer a `\makeindex` in the default `\DocstyleParms`;

The macro environment now takes as argument the macro name *with* the backslash;

Verbatim text Newlines are now forbidden inside `\verb` and commands `\MakeShortVerb` and `\DeleteShortVerb` are provided for verbatim shorthand;

`\par` can now be used in `\DoNotIndex`;

Checksum/character table support for ensuring the integrity of distributions is added;

`\printindex` becomes `\PrintIndex`;

`multicol.sty` is no longer necessary to use `doc` or print the documentation (although it is recommended);

‘Docstrip’ modules are recognised and formatted specially.

As well as adding some completely new stuff, the opportunity has been taken to add some commentary to the code formerly in `newdoc.sty` and that added after version 1.5k of `doc.sty`. Since (as noted in the sections concerned) this commentary wasn’t written by Frank Mittelbach but the code was, it is probably *not* true in this case that “if the code and comments disagree both are probably wrong”!

Bugs

There are some known bugs in this version:

- The `\DoNotIndex` command doesn’t work for some single character commands most noticeable `\%`.
- The ‘General changes’ glossary entry would come out after macro

names with a leading ! and possibly a leading ";

- If you have an old version of `make-index` long `\changes` entries will come out strangely and you may find the section header amalgamated with the first changes entry. Try to get an up-to-date one (see p. 9);
- Because the accompanying `make-index` style files support the inconsistent attribute specifications of older and newer versions `make-index` always complains about three ‘unknown specifier’s when sorting the index and changes entries.
- If `\MakeShortVerb` and `\DeleteShortVerb` are used with single character arguments, e.g., `{|}` instead of `{\|}` chaos may happen.

(Some ‘features’ are documented below.)

Wish list

- Hooks to allow `\DescribeMacro` and `\DescribeEnv` to write out to a special file information about the package’s ‘exported’ definitions which they describe. This could subsequently be included in the `docstripped.sty` file in a suitable form for use by smart editors in command completion, spelling checking etc., based on the packages used in a document. This would need agreement on a ‘suitable form’.
- Indexing of the modules used in `docstrip`’s `%<` directives. I’m not sure how to index directives containing module combinations;
- Writing out bibliographic information about the package;
- Allow turning off use of the special font for, say, the next guarded block.

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 package 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 run the `docstrip.tex` program that removes nearly all lines that begin with a percent sign.

The idea of integrated documenta-

tion 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 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 T_EX 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 T_EX macros which I have introduced here should also only be taken as a first sketch. It is designed explicitly to run under L^AT_EX alone. Not because I was of the opinion that this was the best start-

ing 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 T_EX documentation. I can only advise anyone who thinks that they can cope without documentation to "Stop Time" until he or she completely understands the A_MS-T_EX source code.

1.1 Using the doc package

Just like any other package, invoke it by requesting it with a `\usepackage` command in the preamble. Doc's use of `\reversemarginpars` may make it incompatible with some classes.

2 The User Interface

2.1 The driver file

If one is going to document a set of macros with the `doc` package one has to prepare a special driver file which produces the formatted document. This driver file has the following characteristics:

```
\documentclass[options]{document-class}
\usepackage{doc}
  preamble
\begin{document}
  special input commands
\end{document}
```

The *document-class* might be any document class, I normally use `article`.

In the *preamble* one should place declarations which manipulate the behavior of the `doc` package like `\DisableCrossrefs` or `\OnlyDescription`.

Finally the *special input commands* part should contain one or more `\DocInput`*<file name>* and/or `\IndexInput`*<file name>* commands. The `\DocInput` command is used for files prepared for the `doc` package whereas `\IndexInput` can be used for all kinds of macro files. See page 11 for more details of `\IndexInput`. Multiple `\DocInputs` can be used with a number of included files

`\DocInput`
`\IndexInput`

¹I 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 T_EX are concerned. The long-standing debate over 'multiple change files' shows this well.

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

which are each self-contained self-documenting packages—for instance, each containing `\maketitle`.

As an example, the driver file for the `doc` package itself is the following text surrounded by `%<*driver>` and `%</driver>`. To produce the documentation you can simply run the `.dtx` file through \LaTeX in which case this code will be executed (loading the document class `ltxdoc`, etc.) or you can extract this into a separate file by using the `docstrip` program. The line numbers below are added by `doc`'s formatting. Note that the class `ltxdoc` has the `doc` package preloaded.

```

1 (*driver)
2 \documentclass{ltxdoc}
3 \EnableCrossrefs
4 %\DisableCrossrefs % Say \DisableCrossrefs if index is ready
5 \CodelineIndex
6 \RecordChanges % Gather update information
7 %\OnlyDescription % comment out for implementation details
8 %\OldMakeindex % use if your MakeIndex is pre-v2.9
9 \setlength\hfuzz{15pt} % dont make so many
10 \hbadness=7000 % over and under full box warnings
11 \begin{document}
12 \DocInput{doc.dtx}
13 \end{document}
14 </driver>
```

2.2 General conventions

A \TeX file prepared to be used with the ‘`doc`’ package 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. Such ‘metacomments’ may be also be included simply by surrounding them with `\iffalse ... \fi`.

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 package file in the `\usepackage` 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

```
%\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.

2.3 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 L^AT_EX environment is explained. It will produce a somewhat different index entry. Below I used `\DescribeEnv{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. The `\verb` command is re-implemented to give an error report if a newline appears in its argument. The `verbatim` and `verbatim*` environments set text in the style defined by `\MacroFont` (§2.4).

2.4 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. There should be some text—even if it’s just an empty `\mbox{}`—in this environment before `\begin{macrocode}` or the marginal label won’t print in the right place.

`\MacrocodeTopsep` 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, the `verbatim[*]` environment and the macro names printed in the margin. If you want to change their default values in a class file (like `ltugboat.cls`) use the `\DocstyleParms` command described below. Starting with release 2.0a it can now be changed directly as long as the redefinition happens before the `\begin{document}`.

2.5 Formatting the margins

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

³These macros were written by Rainer Schöpf [8]. He also provided a new `verbatim` environment which can be used inside of other macros.

⁴This 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.

⁵You may place the changed definitions in a separate package 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` package won’t redefine any existing definitions of these macros.

`\PrintMacroName` (called by the `macro` environment) and `\PrintEnvName` (called by the `environment` environment).

2.6 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 `\catcodes`. 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!).

2.7 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 macros which are already defined in \LaTeX .

All three above declarations are local to the current group.

`\PageIndex`
`\CodelineIndex` Production (or not) of the index (via the `\makeindex` command) is controlled by using or omitting the following declarations in the driver file preamble; if neither is used, no index is produced. Using `\PageIndex` makes all index entries refer to their page number; with `\CodelineIndex`, index entries produced by `\DescribeMacro` and `\DescribeEnv` refer to page number but those produced by the `macro` environment refer to the code lines, which will be numbered automatically.⁷ The style of this numbering can be controlled by defining the macro `\theCodelineNo`. Its default definition is to use scriptsize arabic numerals; a user-supplied definition won’t be overwritten.

`\theCodelineNo`
`\CodelineNumbered` When you don’t wish to get an index but want your code lines numbered use

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

⁷The line number is actually that of the first line of the first `macrocode` environment in the `macro` environment.

`\CodelineNumbered` instead of `\CodelineIndex`. This prevents the generation of an unnecessary `.idx` file.

2.8 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. 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 package file which might be read in before the doc package.

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` The `\actualchar` is used to separate the 'key' and the actual index entry. The
`\quotechar` `\quotechar` is used before a special index program character to suppress its special
`\encapchar` meaning. The `\encapchar` separates the indexing information from a letter string
 which `makeindex` uses as a `TEX` command to format the page number associated
 with a special entry. It is used in this package to apply the `\main` and the `\usage`
`\levelchar` commands. 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` To produce a main index entry for a macro the `\SpecialMainIndex` macro⁹
`\SpecialMainEnvIndex` may be used. It is called 'special' because it has to print its argument verbatim.
 A similar macro, called `\SpecialMainEnvIndex` is used for indexing the main
`\SpecialIndex` definition point of an environment.¹⁰ If you want a normal index entry for a
`\SpecialUsageIndex` macro name `\SpecialIndex` might be used.¹¹ To index the usage of a macro
`\SpecialEnvIndex` or an environment `\SpecialUsageIndex` and `\SpecialEnvIndex` may be used.
`\SortIndex` 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

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

⁹This macro is called by the `macro` environment.

¹⁰This macro is called by the `environment` environment.

¹¹This macro is called within the `macrocode` environment when encountering a macro name.

index entries
 Special macros for ~

Such an entry might be produced with the line:

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

`\OldMakeindex` Versions of `makeindex` prior to 2.9 had some bugs affecting `doc`. One of these, pertaining to the `%` character doesn't have a work-around appropriate for versions with and without the bug. If you have an old version, invoke `\OldMakeindex` in a package file or the driver file to prevent problems with index entries such as `\%`, although you'll probably normally want to turn off indexing of `\%` anyway. Try to get an up-to-date `makeindex` from one of the `TEX` repositories.

2.9 Setting the index entries

After the first formatting pass through the `.dtx` file you need to sort the index entries written to the `.idx` file using `makeindex` or your favourite alternative. You need a suitable style file for `makeindex` (specified by the `-s` switch). A suitable one is supplied with `doc`, called `gind.ist`.

`\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 package 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.

`theindex` 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.

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

`\main` The page numbers for main index entries are encapsulated by the `\main` macro
`\usage` (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.

2.10 Changing the default values of style parameters

`\DocstyleParms` If you want to overwrite some default settings made by the `doc` package, you can either put your declarations in the driver file (that is after `doc.sty` is read in) or use a separate package file for doing this work. In the latter case you can define the macro `\DocstyleParms` to contain all assignments. This indirect approach is necessary if your package file might be read before the `doc.sty`, when some of the registers are not allocated. Its default definition is null.

The `doc` package currently assigns values to the following registers:

<code>\IndexMin</code>	= 80.0pt	<code>\MacroTopsep</code>	= 7.0pt plus 2.0pt minus 2.0pt
<code>\marginparwidth</code>	= 126.0pt	<code>\MacroIndent</code>	= 14.65285pt
<code>\marginparpush</code>	= 0.0pt	<code>\MacrocodeTopsep</code>	= 3.0pt plus 1.2pt minus 1.0pt
<code>\tolerance</code>	= 1000		

2.11 Short input of verbatim text pieces

`\MakeShortVerb` It is awkward to have to type, say, `\verb|...|` continually when quoting verbatim bits (like macro names) in the text, so an abbreviation mechanism is provided. Pick a character `<c>`—one which normally has catcode ‘other’ unless you have very good reason not to—which you don’t envisage using in the text, or not using often. (I like `"`, but you may prefer `|` if you have `"` active to do umlauts, for instance.) Then if you say `\MakeShortVerb{\<c>}` you can subsequently use `<c><text><c>` as the equivalent of `\verb<c><text><c>`; analogously, the `*-form` `\MakeShortVerb*{\<c>}` gives you the equivalent of `\verb*<c><text><c>`. Use `\DeleteShortVerb{\<c>}` if you subsequently want `<c>` to revert to its previous meaning—you can always turn it on again after the unusual section. The ‘short verb’ commands make global changes. The abbreviated `\verb` may not appear in the argument of another command just like `\verb`. However the ‘short verb’ character may be used freely in the `verbatim` and `macrocode` environments without ill effect. `\DeleteShortVerb` is silently ignored if its argument does not currently represent a short verb character. Both commands type a message to tell you the meaning of the character is being changed.

Please remember that the command `\verb` cannot be used in arguments of other commands. Therefore abbreviation characters for `\verb` cannot be used there either.

This feature is also available as a sole package, `shortvrb`.

2.12 Additional bells and whistles

We provide macros for logos such as `WEB`, `AMS-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` You can use the `\OnlyDescription` declaration in the driver file to suppress the last part of your document (which presumably exhibits the code). 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 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.

`\Finale` 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

`\maketitle`

`\ps@titlepage`

reading in several articles at once. To avoid an unwanted `pagestyle` on the title page the `\maketitle` command issues a `\thispagestyle{titlepage}` declaration which produces a `plain` page if the `titlepage` page style is undefined. This allows class files like `ltugboat.cls` to define their own page styles for title pages.

`\AlsoImplementation`

Typesetting the whole document is the default. However, this default can also be explicitly selected using the declaration `\AlsoImplementation`. This overwrites any previous `\OnlyDescription` declaration. The $\LaTeX 2_{\epsilon}$ distribution, for example, is documented using the `ltxdoc` class which allows for a configuration file `ltxdoc.cfg`. In such a file one could then add the statement

```
\AtBeginDocument{\AlsoImplementation}
```

to make sure that all documents will show the code part.

`\IndexInput`

Last but not least I defined an `\IndexInput` 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.¹²

`\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>}
```

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 generates the printed entry in such a change history; because old versions¹³ of the `makeindex` program limit such fields to 64 characters, care should be taken not to exceed this limit when describing the change. The actual entry consists of the `<version>`, the `\actualchar`, the current macro name, a colon, the `\levelchar`, and, finally, the `<text>`. The result is a `glossaryentry` for the `<version>`, with the name of the current macro as subitem. Outside the `macro` environment, the text `\generalname` is used instead of the macro name. When referring to macros in change descriptions it is conventional to use `\cs{<macroname>}` rather

¹²It 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.

¹³Before 2.6.

than attempting to format it properly and using up valuable characters in the entry with old `makeindex` versions.

`\RecordChanges` To cause the change information to be written out, include `\RecordChanges`
`\PrintChanges` in the driver file. To read in and print the sorted change history (in two columns), just put the `\PrintChanges` command as the last (commented-out, and thus executed during the documentation pass through the file) command in your package 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. You need a special `makeindex` style file; a suitable one is supplied with `doc`, called `glo.ist`. The `\GlossaryMin`, `\GlossaryPrologue` and `\GlossaryParms` macros are analogous to the `\Index...` versions. (The L^AT_EX ‘glossary’ mechanism is used for the change entries.)

`\GlossaryMin`
`\GlossaryPrologue`
`\GlossaryParms`
`\CharacterTable`
`\Checksum`

To overcome some of the problems of sending files over the networks we developed two macros which should detect corrupted files. If one places the lines

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

at the beginning of the file then character translation failures will be detected, provided of course, that the used `doc` package has a correct default table. The percent signs¹⁴ at the beginning of the lines should be typed in, since only the `doc` package should look at this command.

Another problem of mailing files is possible truncation. To detect these sort of errors we provide a `\Checksum` macro. The check-sum of a file is simply the number of backslashes in the code, i.e. all lines between the macrocode environments. But don't be afraid: you don't have count the code-lines yourself; this is done by the `doc` package for you. You simply have add

```
% \Checksum{0}
```

near the beginning of the file and use the `\StopEventually` (which starts looking for backslashes) and the `\Finale` command. The latter will inform you either that your file has no check-sum (telling you the right number) or that your number is incorrect if you put in anything other than zero but guessed wrong (this time

¹⁴There are two percent signs in each line. This has the effect that these lines are not removed by the `docstrip.tex` program.

telling you both the correct and the incorrect one). Then you go to the top of your file again and change the line to the right number, i.e. line

```
% \Checksum{<number>}
```

and that's all.

While `\CharacterTable` and `\Checksum` have been important features in the early days of the public internet when `doc` was written as the mail gateways back then were rather unreliable and often mangled files they are these days more a nuisance than any help. They are therefore now fully optional and no longer recommended for use with new files.

`\bslash` From time to time, it is necessary to print a `\` without being able to use the `\verb` command because the `\catcodes` 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 `_12`. This means that you have to `\protect` it if it is used in 'moving arguments'.

`\MakePrivateLetters` If your macros `\catcode` anything other than `@` to 'letter', you should redefine `\MakePrivateLetters` so that it also makes the relevant characters 'letters' for the benefit of the indexing. The default definition is just `\makeatletter`.

`\DontCheckModules` The 'module' directives of the `docstrip` system [6] are normally recognised and invoke special formatting. This can be turned on and off in the `.dtx` file or the driver file using `\CheckModules` and `\DontCheckModules`. If checking for module directives is on (the default) then code in the scope of the directives is set as determined by the hook `\AltMacroFont`, which gives *small italic typewriter* by default in the New Font Selection Scheme but just ordinary *small typewriter* in the old one, where a font such as *italic typewriter* can't be used portably (plug for NFSS); you will need to override this if you don't have the *italic typewriter* font available. Code is in such a scope if it's on a line beginning with `%<` or is between lines starting with `%<*<name list>` and `%</<name list>`. The directive is formatted by the macro `\Module` whose single argument is the text of the directive between, but not including, the angle brackets; this macro may be re-defined in the driver or package file and by default produces results like `<+foo | bar>` with no following space.

`StandardModuleDepth` Sometimes (as in this file) the whole code is surrounded by modules to produce several files from a single source. In this case it is clearly not appropriate to format all code lines in a special `\AltMacroFont`. For this reason a counter `StandardModuleDepth` is provided which defines the level of module nesting which is still supposed to be formatted in `\MacroFont` rather than `\AltMacroFont`. The default setting is 0, for this documentation it was set to

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

at the beginning of the file.

2.13 Basic usage summary

To sum up, the basic structure of a `.dtx` file without any refinements is like this:

```
% <waffle>...
...
% \DescribeMacro{\fred}
```

```

% <description of fred's use>
...
% \StopEventually{<finale code>}
...
% \begin{macro}{\fred}
% <commentary on macro fred>
%UUUU\begin{macrocode}
<code for macro fred>
%UUUU\end{macrocode}
% \end{macro}
...
% \Finale \PrintIndex \PrintChanges

```

For examples of the use of most—if not all—of the features described above consult the `doc.dtx` source itself.

2.14 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.

A big thank you to David Love who brought the documentation up-to-date again, after I neglected this file for more than two years. This was most certainly a tough job as many features added to `doc.dtx` after its publication in *TUGboat* have been never properly described. Beside this splendid work he kindly provided additional code (like “docstrip” module formatting) which I think every `doc.dtx` user will be grateful for.

3 The Description of Macros

Most of the following code is destined for `doc.sty` after processing with `docstrip` to include the module `style` indicated here. (All code in this file not appropriate to `doc.sty` has to be included explicitly by `docstrip` so that this `.dtx` file can be used as directly as a package file rather than the stripped version.) The usual font change for the conditionally-included lines between the `<*style>` and `</style>` directives is suppressed since only the lines with an explicit directive are special in this file.

```
15 <*package>
```

Under $\text{\LaTeX} 2_{\epsilon}$ the test to avoid reading `doc` in twice is normally unnecessary. It was kept to only to stay compatible with $\text{\LaTeX} 209$ styles that `\input doc` directly.

```
16 \@ifundefined{macro@cnt}{-}{\endinput}
```

```

\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 package 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:

```
17 \catcode'\^^A=14
```

We repeat this statement at the beginning of the document in case the `inputenc` package is used disabling it again.

```
18 \AtBeginDocument{\catcode'\^^A=14\relax}
```

3.1 Options supported by doc

Not options available at the moment

3.2 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.

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

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

```
20 \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.

```
21 \xmacro@code}
```

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

```
22 \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.

```
23 \topsep \MacrocodeTopsep
```

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

```
24 \@beginparpenalty \predisplaypenalty
```

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

```
25 \if@inlabel\leavevmode\fi
```

```
26 \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.¹⁵

```
27 \macro@font
```

¹⁵The font change has to be placed *after* the `\item`. Otherwise a change to `\baselineskip` will affect the paragraph above.

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

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

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

```
30 \blank@linefalse \def\par{\ifblank@line
31 \leavevmode\fi
32 \blank@linetrue\@par
33 \penalty\interlinepenalty}
```

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

```
34 \obeylines
35 \let\do\do@noligs \verbatim@nolig@list
36 \let\do\@makeother \dospecials
```

If indexing by code lines is switched on the line number is incremented and set appropriately. We also check whether the start of the next line indicates a `docstrip` module directive and process it appropriately if so using `\check@module`.

```
37 \global\@newlistfalse
38 \global\@minipagefalse
39 \ifcodeline@index
40 \everypar{\global\advance\c@CodelineNo\@ne
41 \llap{\theCodelineNo\ \hskip\@totalleftmargin}%
42 \check@module}%
43 \else \everypar{\check@module}%
44 \fi
```

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

```
45 \init@crossref}
```

`\ifblank@line` `\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.

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

`\endmacrocode` Because we have begun a `trivlist` environment in the macrocode environment, we must also end it. We must also act on the value of the `pm@module` flag (see below) and empty `\everypar`.

```
47 \def\endmacrocode{%
48 \ifpm@module \endgroup \pm@modulefalse \fi
49 \everypar{}%
50 \global\@inlabelfalse
51 \endtrivlist
```

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

```
52 \close@crossref}
```

`\MacroFont` Here is the default definition for the `\MacroFont` macro. With the new math font handling in NFSS2 it isn't any longer correct to suppress the math font setup since this is now handled differently. But to keep the font change fast we use only a single `\selectfont` (in `\small`) and do the rest by hand.

```
53 \@ifundefined{MacroFont}{%
54   \if@compatibility
```

Despite the above statement we will call `\small` first if somebody is using a \LaTeX 2.09 document with doc. I wouldn't have bothered since doc-sources should be up-to-date but since the request came from someone called David Carlisle ... :-)

```
55   \def\MacroFont{\small
56     \usefont\encodingdefault
57       \ttdefault
58       \mddefault
59       \updefault
60   }%
61   \else
62     \def\MacroFont{\fontencoding\encodingdefault
63       \fontfamily\ttdefault
64       \fontseries\mddefault
65       \fontshape\updefault
66       \small}%
67   \fi
68   }{}
```

`\AltMacroFont` Although most of the macro code is set in `\MacroFont` we want to be able to switch to indicate module code set in `\AltMacroFont`. `\macro@font` keeps track of which one we're using. We can't do the same thing sensibly in OFSS as in NFSS.

```
69 \@ifundefined{AltMacroFont}{%
70   \if@compatibility
```

Again have `\small` first if we are in compat mode.

```
71   \def\AltMacroFont{\small
72     \usefont\encodingdefault
73       \ttdefault
74       \mddefault
75       \sldefault
76   }%
77   \else
78     \def\AltMacroFont{\fontencoding\encodingdefault
79       \fontfamily\ttdefault
80       \fontseries\mddefault
81       \fontshape\sldefault
82       \small
83     }%
84   \fi
85   }{}
```

To allow changing the `\MacroFont` in the preamble we defer defining the internally used `\macro@font` until after the preamble.

```
86 \AtBeginDocument{\let\macro@font\MacroFont}
```

`\check@module` This is inserted by `\everypar` at the start of each macrocode line to check whether it starts with module information. (Such information is of the form `%<switch>`, where the `%` must be at the start of the line and `<switch>` comprises names with various possible separators and a possible leading `+`, `-`, `*` or `/` [6]. All that concerns us here is what the first character of `<switch>` is.) First it checks the `pm@module` flag in case the previous line had a non-block module directive i.e., not `%<*` or `%</`; if it did we need to close the group it started and unset the flag. `\check@module` looks ahead at the next token and then calls `\ch@percent` to take action depending on whether or not it's a `%`; we don't want to expand the token at this stage. This is all done conditionally so it can be turned off if it causes problems with code that wasn't designed to be `docstripped`.

```

87 \def\check@module{%
88   \ifcheck@modules
89     \ifpm@module \endgroup \pm@modulefalse \fi
90     \expandafter\futurelet\expandafter\next\expandafter\ch@percent
91   \fi}
92 \newif\ifpm@module

```

`\DontCheckModules` Here are two driver-file interface macros for turning the module checking on and off using the `check@modules` switch.

```

\CheckModules
\ifcheck@modules
93 \def\DontCheckModules{\check@modulesfalse}
94 \def\CheckModules{\check@modulestrue}
95 \newif\ifcheck@modules \check@modulestrue

```

`\ch@percent` If the lookahead token in `\next` is `%12` we go on to check whether the following one is `<` and otherwise do nothing. Note the `\expandafter` to get past the `\fi`.

```

96 \def\ch@percent{%
97   \if \percentchar\next
98     \expandafter\check@angle
99   \fi}

```

`\check@angle` Before looking ahead for the `<` the `%` is gobbled by the argument here.

```

100 \def\check@angle#1{\futurelet\next\ch@angle}

```

`\ch@angle` If the current lookahead token is `<` we are defined to be processing a module directive can go on to look for `+` etc.; otherwise we must put back the gobbled `%`. With $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ `<` is active so we have to be a bit careful.

```

101 \begingroup
102 \catcode'\<\active
103 \gdef\ch@angle{\ifx<\next
104   \expandafter\ch@plus@etc
105   \else \percentchar \fi}

```

`\ch@plus@etc` We now have to decide what sort of a directive we're dealing with and do the right thing with it.

```

\check@plus@etc
106 \gdef\ch@plus@etc<{\futurelet\next\check@plus@etc}
107 \gdef\check@plus@etc{%
108   \if +\next
109     \let\next\pm@module
110   \else\if -\next
111     \let\next\pm@module
112   \else\if *\next

```

```

113     \let\next\star@module
114     \else\if /\next
115     \let\next\slash@module

```

At some point in the past the `docstrip` program was partly rewritten and at that time it also got support for a very special directive of the form `%<<` followed by an arbitrary string. This is used for “verbatim” inclusion in case of certain problem. We do not really attempt to pretty print that case but we need at least account for it since otherwise we get an error message since this is the only case where we will not have a closing `>`.

```

116     \else\ifx <\next
117     \percentchar
118     \else
119     \let\next\pm@module
120     \fi\fi\fi\fi\fi
121     \next}
122 \endgroup

```

`\pm@module` If we’re not dealing with a block directive (`*` or `/`) i.e., it’s a single special line, we set everything up to the next `>` appropriately and then change to the special macro font inside a group which will be ended at the start of the next line. If the apparent module directive is missing the terminating `>` this will lose, but then so will the `docstrip` implementation. An alternative strategy would be to have `\pm@module` make `>` active and clear a flag set here to indicate processing the directive. Appropriate action could then be taken if the flag was found still to be set when processing the next line.

```

123 \begingroup
124 \catcode'\>=\active
125 \lccode'\>='\>
126 \lowercase{\gdef\pm@module#1~}{\pm@moduletrue
127   \Module{#1}\begingroup

```

We switch to a special font as soon the nesting is higher than the current value of `\c@StandardModuleDepth`. We do a local update to the `\guard@level` here which will be restored after the current input line.

```

128     \advance\guard@level\@ne
129     \ifnum\guard@level>\c@StandardModuleDepth\AltMacroFont\fi
130 }

```

`\star@module` `\slash@module` If the start or end of a module *block* is indicated, after setting the guard we have to check whether a change in the macrocode font should be done. This will be the case if we are already inside a block or are ending the outermost block. If so, we globally toggle the font for subsequent macrocode sections between the normal and special form, switching to the new one immediately.

```

131 \lowercase{\gdef\star@module#1~}{%
132   \Module{#1}%
133   \global \advance \guard@level\@ne
134   \ifnum \guard@level>\c@StandardModuleDepth
135     \global\let\macro@font=\AltMacroFont \macro@font
136   \fi}
137 \catcode'\>=\active
138 \gdef\slash@module#1>{%
139   \Module{#1}%

```

```

140 \global \advance \guard@level\m@ne
141 \ifnum \guard@level=\c@StandardModuleDepth
142   \global\let\macro@font\MacroFont \macro@font
143   \fi
144 }
145 \endgroup

```

`\c@StandardModuleDepth` Counter defining up to which level modules are considered part of the main code. If, for example, the whole code is surrounded by a `%<*package>` module we better set this counter to 1 to avoid getting the whole code be displayed in typewriter italic.

```
146 \newcounter{StandardModuleDepth}
```

`\guard@level` We need a counter to keep track of the guard nesting.

```
147 \newcount \guard@level
```

`\Module` This provides a hook to determine the way the module directive is set. It gets as argument everything between the angle brackets. The default is to set the contents in sans serif text between `<>` with the special characters suitably `\mathcoded` by `\mod@math@codes`. (You can't just set it in a sans text font because normally `|` will print as an em-dash.) This is done differently depending on whether we have the NFSS or the old one. In the latter case we can easily change `\fam` appropriately.

```
148 \@ifundefined{Module}{%
```

With NFSS what we probably *should* do is change to a new `\mathversion` but I (Dave Love) haven't spotted an easy way to do so correctly if the document uses a version other than `normal`. (We need to know in what font to set the other groups.) This uses a new math alphabet rather than `version` and consequently has to worry about whether we're using `oldlfn` or not. I expect there's a better way...

```
149   \def\Module#1{\mod@math@codes$\langle\mathsf{#1}\rangle$}
150 }{}
```

`\mod@math@codes` As well as 'words', the module directive text might contain any of the characters `*/+-,&!()` for the current version of `docstrip`. We only need special action for two of them in the math code changing required above: `|` is changed to a `\mathop` (it's normally "026A) and `&` is also made a `\mathop`, but in family 0. Remember that `&` will not have a special catcode when it's encountered.

```
151 \def\mod@math@codes{\mathcode'\|= "226A \mathcode'\&="2026
152   \mathcode'\-="702D \mathcode'\+="702B
153   \mathcode'\:="703A \mathcode'\=="703D }
```

`\mathsf` If NFSS is in use we need a new math alphabet which uses a sans serif font. To support both the release one and two of NFSS the alphabet was renamed to `\mathsf` which is defined in NFSS2.

```
154 %\ifx\selectfont\undefined
155 %\else
156 %   \ifx\mathsf\undefined
157 %     \newmathalphabet*\mathsf{\sfdefault}{m}{n}\fi
158 %\fi
```

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

```
159 \newskip\MacrocodeTopsep \MacrocodeTopsep = 3pt plus 1.2pt minus 1pt
160 \newdimen\MacroIndent
161 \settowidth\MacroIndent{\rmfamily\scriptsize 00\ }
```

`\macrocode*` Just as in 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. Münchhausens Marsh problem)¹⁶ 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*}`.

```
162 \@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*`.

```
163 \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.

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

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

```
166 \catcode‘\{=12␣\catcode‘\}=12
167 \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.

```
168 |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.

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

¹⁶Karl Friedrich Hieronymus Frhr. v. Münchhausen (*1720, †1797). Several books were written about fantastic adventures supposedly told by him (see [7] or [1]). In one story he escaped from the marsh by pulling himself out by his hair.

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.

```
171 |endgroup
```

3.3 Macros for the ‘documentation parts’

`\DescribeMacro` 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 L^AT_EX output routine.¹⁷ First we change to horizontal mode if necessary. The L^AT_EX macros `\@bsphack` and `\@esphack` are used to make those commands invisible (i.e. to normalize the surrounding space and to make the `\spacefactor` transparent).

```
172 \def\DescribeMacro{\leavevmode\@bsphack
```

When documenting the code for the `amstex.sty` option we encountered a bug: the `\catcode` of `@` was active and therefore couldn’t be used in command names. So we first have to make sure that we get all `\catcodes` right by calling `\MakePrivateLetters` inside a group. Then we call `\Describe@Macro` to do the work.

```
173 \begingroup\MakePrivateLetters\Describe@Macro}
174 \def\Describe@Macro#1{\endgroup
175 \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.¹⁸

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

The `\DescribeEnv` macro is completely analogous.

```
177 \def\DescribeEnv{\leavevmode\@bsphack\begingroup\MakePrivateLetters
178 \Describe@Env}
179 \def\Describe@Env#1{\endgroup
180 \marginpar{\raggedleft\PrintDescribeEnv{#1}}%
181 \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 classes or packages.) We also make the `\marginparpush` zero and `\marginparwidth` suitably wide.

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

`\bslash` We start a new group in which to hide the alteration of `\catcodes`, and make `|` introduce commands, whilst `\` becomes an ‘other’ character.

```
184 {\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 `\catcodes`.

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

¹⁷It might be better to change these macros into environments like the `macro` environment.

¹⁸The whole mechanism won’t work because of the `\leavevmode` in front. As a temporary change `\ignorespaces` is added.

`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`.

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

We deal in a similar way with the star form of this environment.

```
188 \@namedef{verbatim*}{\@beginparpenalty \predisplaypenalty \@verbatim
189           \MacroFont \@sxverbatim}
```

`\@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`.

```
190 \def\@verbatim{\trivlist \item[]\if@minipage\else\vskip\parskip\fi
191   \leftskip\@totalleftmargin\rightskip\z@
192   \parindent\z@\parfillskip\@flushglue\parskip\z@
193   \@@par
194   \@tempwafalse
```

`\@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 .

```
195 \def\par{\if@tempwa\hbox{} \fi \@tempwattrue \@@par
196   \penalty\interlinepenalty
```

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

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

The rest is again copied literally from `latex.tex` (less `\tt`).

```
198   \obeylines
199   \let\do\do@noligs \verbatim@nolig@list
200   \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 that 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. There is a subtle problem here, the equal sign between `\next` and `#1` is actually necessary. Do you see why? The omission of this token once caused a funny error.

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

`\verb` We re-define `\verb` to check for newlines in its argument since a missing delimiter is difficult to detect in `doc` source. The code is the same as in `latex.tex` of September 19, 1993. Perhaps there should be a font-changing hook rather than just using `\ttfamily`, but if so it probably should be different from `\MacroFont` since that normally includes `\small` and would look wrong inline.

```
204 \def\verb{\relax\ifmmode\hbox\else\leavevmode\null\fi
205   \bgroup \let\do\do@noligs \verbatim@nolig@list
```

```

206 \ttfamily \verb@eol@error \let\do@makeother \dospecials
207 \@ifstar{\@sverb}{\@vobeyspaces \frenchspacing \@sverb}}

\verb@balance@group
  \verb@egroup 208 \let\verb@balance@group\@empty
  \verb@eol@error 209
210 \def\verb@egroup{\global\let\verb@balance@group\@empty\egroup}
211
212 \begingroup
213 \obeylines%
214 \gdef\verb@eol@error{\obeylines%
215 \def~M{\verb@egroup\@latex@error{%
216 Text for \noexpand\verb command ended by end of line}\@ehc}}%
217 \endgroup

  \@sverb See [8] for commentary.
218 \def\@sverb#1{%
219 \catcode'#1\active \lccode'\~'#1%
220 \gdef\verb@balance@group{\verb@egroup
221 \@latex@error{Illegal use of \noexpand\verb command}\@ehc}%
222 \aftergroup\verb@balance@group
223 \lowercase{\let~\verb@egroup}}

\verbatim@nolig@list These macros replace the old \@noligs mechanism by an extensible version to
  \do@noligs allow more ligatures to be added.
224 \def\verbatim@nolig@list{\do\~\do\<\do\>\do\,\do\'\do\~}
225 \def\do@noligs#1{%
226 \catcode'#1\active
227 \begingroup
228 \lccode'\~='#1\relax
229 \lowercase{\endgroup\def~{\leavevmode\kern\z@\char'#1}}

  \macro The macro environment is implemented as a trivlist environment, whereby in order
  \m@cro@ that the macro names can be placed under one another in the margin (corres-
  \macro@cnt sponding to the macro's nesting depth), the macro \makelabel must be altered.
  In order to store the nesting depth, we use a counter. We also need a counter to
  count the number of nested macro environments.
230 \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.
231 \def\macro{\begingroup
232 \catcode'\12
233 \MakePrivateLetters \m@cro@ \iftrue}

environment The “environment” environment will be implemented just like the “macro” envi-
  ronment flagging any differences in the code by passing \iffalse or \iftrue to
  the \m@cro@ environment doing the actual work.
234 \def\environment{\begingroup
235 \catcode'\12
236 \MakePrivateLetters \m@cro@ \iffalse}

```

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

```
237 \long\def\macro@#1#2{\endgroup \topsep\MacroTopsep \trivlist
```

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

```
238 \edef\saved@macroname{\string#2}%
239 #1
```

If documenting an environment we put its name in `\saved@indexname` otherwise the name without the backslash.

```
240 \let\saved@indexname\saved@macroname
241 \else
242 \edef\saved@indexname{\expandafter@gobble\string#2}%
243 \fi
```

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.

```
244 \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 L^AT_EX 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.

```
245 \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 T_EX 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.

```
246 \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`.

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

Now we put the definition of `\makelabel` together.

```
249 \edef\makelabel##1{\llap{\vtop to\baselineskip
250 {\@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).

```
251 \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.

```
252 \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`.

```
253 \edef\@tempa{\noexpand\item[%
```

Depending on whether we are inside a “macro” or “environment” environment we use `\PrintMacroName` or `\PrintEnvName` to display the name.

```
254 #1%
255 \noexpand\PrintMacroName
256 \else
257 \noexpand\PrintEnvName
258 \fi
259 {\string#2}]}%
260 \@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` after advancing the counter for indexing by line number. 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). We advance the current codeline number and after producing an index entry revert to the original value

```
261 \global\advance\c@CodelineNo\@ne
```

Again the macro to call depends on the environment we are actually in.

```
262 #1%
263 \SpecialMainIndex{#2}\nobreak
264 \DoNotIndex{#2}%
265 \else
266 \SpecialMainEnvIndex{#2}\nobreak
267 \fi
268 \global\advance\c@CodelineNo\m@ne
```

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. Regarding possibly problematic arguments, the implementation takes care of `\par` and the conditionals are uncritical.

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

```
269 \ignorespaces}
```

`\endmacro` Older releases of this environment omit the `\endgroup` token, when being nested.
`\endenvironment` This was done to avoid unnecessary stack usage. However it does not work if macro and environment environments are mixed, therefore we now use a simpler approach.

```
270 \let\endmacro \endtrivlist
271 \let\endenvironment\endmacro
```

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

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

3.4 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` The formatting of the macro name in the left margin is done by these macros. We
`\PrintEnvName` first set a `\strut` to get the height and depth of the normal lines. Then we change
`\PrintDescribeMacro` to the `\MacroFont` using `\string` to `\catcode` the argument to other (assuming
`\PrintDescribeEnv` 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`.

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

We use the same formatting conventions when describing a macro.

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

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

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

```
279 \@ifundefined{PrintEnvName}
280   {\def\PrintEnvName#1{\strut \MacroFont #1\ }}{}
```

3.5 Creating index entries by scanning ‘macrocode’

The following macros ensure that index entries are created for each occurrence of a T_EX-like command (something starting with ‘\’) providing indexing has been turned on with `\PageIndex` or `\CodelineIndex`. 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* package file itself we’ve sometimes had to make | take the rôle of T_EX’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` The macro `\SpecialEscapechar` is used to denote a special escape character for
`\active@escape@char` the next macrocode environment. It has one argument—the new escape char-
`\special@escape@char` acter given as a ‘single-letter’ control sequence. Its main purpose is defining
`\special@escape@char` `\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`.

```
281 \begingroup
282 \catcode'\~\active
283 \gdef\SpecialEscapechar#1{%
284   \begingroup
```

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.

```
285   \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 `\catcodes` untouched (cf. *T_EXbook* page 41).

```
286   \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`.

```
287   \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.

```
288   \endgroup}
289 \endgroup
```

`\init@crossref` The replacement text of `\init@crossref` should fulfill 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 `\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 `\catcodes`).

Therefore we first re-`\catcode` some characters.

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

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

```
291 |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 fulfill 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.

```
292 |MakePrivateLetters
```

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

```
293 |catcode|expandafter'|special@escape@char|active
```

```
294 |expandafter|let|active@escape@char|scan@macro}
```

```
295 |endgroup
```

If there is no special escape character, i.e. if `\SpecialEscapechar` is `\`, the second last line will overwrite the previous definition of `_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:

```
296 \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.

```
297 \@ifundefined{MakePrivateLetters}
```

```
298 {\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 ‘\’.

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

3.6 Macros for scanning macro names

`\scan@macro` The `\init@crossref` will have made `\active` our `\special@escape@char`, so
`\macro@namepart` 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 T_EX-like command which is met whilst setting (in verbatim) the contents of `macrocode` environments.

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

First we output the character which triggered this macro. Its version `\catcode d` to 12 is saved in `\special@escape@char`. We also call `\step@checksum` to generate later on a proper check-sum (see section 2.12 for details).

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

```
302 \step@checksum
```

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.

```
303 \ifscan@allowed
```

The macro assembles the letters forming a T_EX 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.

```
304     \let\macro@namepart\@empty
305     \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’.

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

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

```
307     \next}
```

```
\ifscan@allowed \ifscan@allowed is the switch used above to determine if the \active@escape@char
\scan@allowedtrue should start the macro scanning mechanism.
\scan@allowedfalse 308 \newif\ifscan@allowed \scan@allowedtrue
```

```
\EnableCrossrefs At this point we might define two macros which allow the user to disable or
\DisableCrossrefs 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).
```

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

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

```
310 \def\EnableCrossrefs{\@bsphack\scan@allowedtrue
311     \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 probably includes @).
```

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

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

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

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

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

```
315     \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@excluded` macro which assumes that the macro name is saved in `\macro@namepart`. The character mustn’t be stored with a special category code or exclusion from the index won’t work, so we employ the case-changing trick used elsewhere. Since the argument might be an active character, `\string` is used to normalize it.

```
316 \begingroup
317 \catcode'\&=12
318 \gdef\short@macro#1{\begingroup
```

```

319 \uccode'\&=\expandafter'\string#1%
320 \uppercase{\def\x{\def\macro@namepart{&}}}%
321 \expandafter\endgroup\x
322 \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.

```
323 \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).

```
324 \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.

```
325 \scan@allowedtrue }
326 \endgroup

```

`\produce@index` This macro is supposed to generate a suitable `\SortIndex` command for a given single-letter 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.

```
327 \def\produce@index#1{%
328 \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.

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

```

Next comes the test for a `'\'` which must be the `_13` expanding to `\bslash`.

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

```

Another possibility is `_13`. Recall that `\space` produces a `_10`.

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

```

The last¹⁹ 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.)

```
332 \ifx#1\par

```

If we end up here we have just scanned a `^M` or something similar. Since this will be treated like `_1` by `TEX` we produce a corresponding index entry.

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

```

If it is the token `\relax` we do nothing. This can't happen when the 'doc' package is used in the way described here, but was added to allow extensions like the `idxverb` option.

```
334 \ifx#1\relax \else

```

¹⁹Well, 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.

The next three branches are needed because of bugs in our `makeindex` program. You can't produce unbalanced index entries²⁰ and you have to double a percent character. To get around these restrictions we use special macros to produce the `\index` calls.²¹

```
335         \if\noexpand#1\bgroup \LeftBraceIndex \else
336         \if\noexpand#1\egroup \RightBraceIndex \else
337         \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`.)

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

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

```
339         \fi \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`).

```
340 \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'.

```
341     \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'.

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

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

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

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

```
345     \next}
```

`\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`.

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

²⁰This is possible for T_EX if you use `{12` or `}12`, but `makeindex` will complain.

²¹Brian 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 38 if you already have them. (I'm not sure which versions incorporate these, but 2.11 is OK. See also 9.)

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.²²

```
348 \ifnot@excluded
349   \edef\@tempa{\noexpand\SpecialIndex{\backslash\macro@namepart}}%
350   \@tempa \fi}
```

3.7 The index exclude list²³

The internal form of the index exclude list is

$$\langle macro\ name\rangle, \langle macro\ name\rangle, \dots,$$

where $\langle macro\ name\rangle$ is a macro name like `_12p11@11` or `_12$11`. Note that the `\` has category ‘other’ and the other characters in the name are all ‘letter’, regardless of their normal category.

`\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 `\catcodes` of all characters which belong to ‘letters’ while macros are defined.

```
351 \def\DoNotIndex{\begingroup \MakePrivateLetters
352   \catcode'\_12
```

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

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

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

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

It just adds to a token list after finishing the group in which the catcodes were changed.

```
355   \endgroup
356   \addto@hook\index@excludelist{#1,}}
```

`\addto@hook` The code for adding tokens (the second argument) to a token list (the first argument) is taken from [8], but it needs to be `\long` in case `\par` is amongst the tokens.

```
357 \long\def\addto@hook#1#2{#1\expandafter{\the#1#2}}
```

`\index@excludelist` We need an initially-empty register for the excluded list.

```
358 \newtoks\index@excludelist
359 \index@excludelist{}
```

`\ifnot@excluded`
`\expanded@notin` 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`.

```
360 \begingroup
```

²²The `\index` command will expand its argument in the `\output` routine. At this time `\macro@namepart` might have a new value.

²³Warning: the incomplete commentary on `\DoNotIndex` and the macros it calls was written by Dave Love.

First we change `\catcodes` so that `\` is ‘other’ and `|` a temporary for the escape character. This is necessary since our macro names are stored that way in the `\index@excludelist`.

```
361 \catcode'\|=0%
362 \catcode'\|=12
```

Then we define `\ifnot@excluded` to call `\expanded@notin` with two arguments: the first is the string `\` followed by the contents of `\macro@namepart` followed by a `,` and the second is `\the` followed by `\index@excludelist`. To achieve the expansion of `\macro@namepart`, i.e. to pass its contents, we need a suitable number of `\expandafte`rs.

```
363 \gdef\ifnot@excluded{|\expandafter
364     |\expanded@notin|\expandafter{|\expandafter
365     |\macro@namepart,}{|\the|\index@excludelist}}
366 \endgroup
```

The macro `\expanded@notin` now does the dirty work. It first defines a macro `\in@@` with a very special parameter text. If you look closely `\in@@` has three arguments, the first one is delimited by the first argument of `\expanded@notin` (i.e. by the string starting with a `\` and ending with a `,` above), the second is undelimited (which means that it will get the next token after our string, and the third is delimited again and will get the rest up to the token `\in@@`. In other words the token `\in@@` is also used as a delimiter.

```
367 \def\expanded@notin#1#2{%
368   \def\in@@##1#1##2##3\in@@{%
```

Now the replacement text simply compares the second argument (i.e. the undelimited one after our string) to the token `\expanded@notin`. This is an unclosed `\ifx` statement which means that this macro behaves similar to a normal `TeX` conditional.

```
369   \ifx\expanded@notin##2}%
```

After all these preparations we call `\in@@`. First we expand the token after `\in@@` (which is `\the` from the second argument to `\expanded@notin`). As a result we get the contents of the `\index@excludelist` inserted after `\in@@`. After this contents we add once more the string we are looking for, then the token `\expanded@notin` and finally the token `\in@@`.

```
370   \expandafter\in@@#2#1\expanded@notin\in@@}
```

Now what happens when the macro `\in@@` above gets called? The first argument to `\in@@` is delimited by our string. In other words it will get everything from the contents of `\index@excludelist` before this string. If the string is not in `\index@excludelist` then it gets the whole contents, since after it we had inserted the string one more. In this case the next token is `\expanded@notin` which gets assigned to the second argument and the third argument will be empty. If, on the other hand, the string was inside `\index@excludelist` then the second argument will not be the token `\expanded@notin` and the third argument will be all the garbage up to `\in@@`. Therefore testing the second argument, as done in the definition of `\in@@` will tell us whether or not the string is in `\index@includelist` and this was exactly what we wanted. (Deep breath.) You got that?²⁴

²⁴`TeXbook` page 125. The code described above is originally due to Michael Spivak who used a similar method within the `AMS-TeX` macros.

3.8 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 L^AT_EX package files that it is necessary to substitute another character. This is indicated to `makeindex` by means of an ‘index style file’; the character selected for this function is =, and therefore this character too must be specially treated when it is met in a T_EX command. A suitable index style file is provided amongst the supporting files for this style file in `gind.ist` and is generated from this source by processing with `docstrip` to extract the module `gind`. A similar style file `gglo.ist` is supplied for sorting the change information in the glossary file and is extracted as module `gglo`. First of all we add some information to the front of the `.ist` files.

```

371 \package
372 (+gind|gglo)%% This is a MAKEINDEX style file which should be used to
373 (+gind)%% generate the formatted index for use with the doc
374 (+gglo)%% generate the formatted change history for use with the doc
375 (+gind|gglo)%% package. The TeX commands used below are defined in
376 (+gind|gglo)%% doc.sty. The commands for MAKEINDEX like ‘level’
377 (+gind|gglo)%% ‘item_x1’ are described in ‘‘ Makeindex, A General
378 (+gind|gglo)%% Purpose, Formatter-Independent Index Processor’’ by
379 (+gind|gglo)%% Pehong Chen.
380 (+gind|gglo)

```

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

```

381 (*package)
382 \@ifundefined{actualchar}{\def\actualchar{=}}{}
383 \@ifundefined{quotechar}{\def\quotechar{!}}{}
384 \@ifundefined{levelchar}{\def\levelchar{>}}{}
385 \package
386 (+gind|gglo)actual ’=’
387 (+gind|gglo)quote ’!’
388 (+gind|gglo)level ’>’
389 (*package)

```

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

```

390 \@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.

```

391 \@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.

```
392 \def\SpecialIndex#1{\@bsphack\special@index{\expandafter\@gobble
393 \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.

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

`\SpecialMainIndex` 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.

`\SpecialMainIndex` passes the macro name to be indexed on to the macro `\SpecialIndex@`.

```
396 \def\SpecialMainIndex#1{\@bsphack\SpecialIndex@{#1}{\encapchar main}%
397 \@esphack}
```

`\SpecialIndex@` The macro `\SpecialIndex@` does the real work for `\SpecialMainIndex` and `\SpecialUsageIndex`. It takes two arguments: the macro to be indexed (as a control sequence or list of character tokens) and the additional text for the index.

```
398 \begingroup
399 \catcode'\|=0
400 \catcode'\|=12
401 |gdef|@SpecialIndexHelper@#1#2|@nil{%
402 |if |noexpand#1|%
403 |gdef|@gtempa{#2}%
404 |else
405 |begingroup
406 |escapechar|m@ne
407 |expandafter|gdef|expandafter|@gtempa|expandafter{|string#1#2}%
408 |endgroup
409 |fi}
410 |endgroup
411 \def\SpecialIndex@#1#2{%
```

The first thing it does is to convert the macro into a list of characters. Note that a character token list remains (mostly) unchanged.

```
412 \@SpecialIndexHelper@#1\@nil
```

The macro name `_` has to be handled in a special way. The reason is that the space token is skipped when `TEX` is scanning macro parameters, so that the trick used below will not work. So, we check whether the replacement text of `\@tempa` starts with a space token and write the appropriate index entry.

```
413 \def\@tempb{ }%
414 \ifcat \@tempb\@gtempa
415 \special@index{\quotechar\space\actualchar
416 \string\verb\quotechar*\verbatimchar
417 \quotechar\backslash\quotechar\space\verbatimchar#2}%
418 \else
```

Having handled this special case we have to distinguish control sequences consisting of one or more letters and those that consists of exactly one nonletter. As character tokens in the replacement text of the macro `\@gtempa` have all category code 12 (other), this is difficult. For simplicity, we treat all single character control sequences alike, irregardless of whether the character is a letter or not. This has the advantage that it works even for strange settings of the category codes.

We define a utility macro `\@tempb` with two arguments, the second delimited by `\relax`. It will be called later so that the first argument is the first character of the macro name, and the second argument receives the rest of the characters. So we distinguish the two cases above by checking whether the second argument is empty.

```
419 \def\@tempb##1##2\relax{\ifx\relax##2\relax
```

If so, we define the helper macro `\@tempc` in a way that it adds quotechars in critical places.

```
420 \def\@tempc{\special@index{\quotechar##1\actualchar
421 \string\verb\quotechar*\verbatimchar
422 \quotechar\bslash\quotechar##1\verbatimchar#2}}%
```

Otherwise we write the characters as in `\SpecialIndex`.

```
423 \else
424 \def\@tempc{\special@index{##1##2\actualchar
425 \string\verb\quotechar*\verbatimchar
426 \bslash##1##2\verbatimchar#2}}%
427 \fi}%
```

Now pass the list of characters to `tempb` and call `tempc` to do the work.

```
428 \expandafter\@tempb\@gtempa\relax
429 \@tempc
430 \fi}
```

Slightly simpler is the main entry for environments

```
431 \def\SpecialMainEnvIndex#1{\@bsphack\special@index{%
432 #1\actualchar
433 {\string\ttfamily\space#1}
434 (environment)%
435 \encapchar main}%
436 \special@index{environments:\levelchar#1\actualchar{%
437 \string\ttfamily\space#1}\encapchar
438 main}\@esphack}
```

The `\SpecialUsageIndex` is similar to `\SpecialMainIndex`, except that it uses the standard `\index` command. `usage` instead of `main`.

```
439 \def\SpecialUsageIndex#1{\@bsphack
440 {\let\special@index\index\SpecialIndex@{#1}{\encapchar usage}}%
441 \@esphack}
```

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

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

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

```
443 \index{#1\actualchar{\protect\ttfamily#1}
444 (environment)\encapchar usage}%
```

The second entry is sorted as a subitem under the key ‘environments’.

```
445 \index{environments:\levelchar#1\actualchar
446         {\protect\ttfamily#1}\encapchar usage}\@esphack}
```

Because both entries correspond 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.

```
447 \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`.

```
448 \def\it@is@a#1{\special@index{\quotechar #1\actualchar
449                               \string\verb\quotechar*\verbatimchar
450                               \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.

```
451 \@ifundefined{LeftBraceIndex}{\def\LeftBraceIndex{%
452   \special@index{\bgroup\actualchar\string\verb\quotechar*\verbatimchar
453   \quotechar\bslash{\verbatimchar\string\iffalse}\string\fi}}{}
454
455 \@ifundefined{RightBraceIndex}{\def\RightBraceIndex{%
456   \special@index{\egroup\actualchar\string\iffalse{\string\fi}\string\verb
457   \quotechar*\verbatimchar\quotechar\bslash\verbatimchar}}{}
458
```

`\PercentIndex` By default we assume a version of `makeindex` without the percent bug is being used.

```
458 \@ifundefined{PercentIndex}
459   {\def\PercentIndex{\it@is@a\percentchar}}{}
460
```

`\OldMakeindex` `\percentchar` Here is one solution for the percent bug in `makeindex`. The macro `\percentchar` denotes a `%12`. Calling this from a package or the driver file sets things up appropriately.

```
460 \def\OldMakeindex{\def\PercentIndex{%
461   \special@index{\quotechar\percentchar\actualchar\string\verb
462   \quotechar*\verbatimchar\quotechar\bslash
463   \percentchar\percentchar\verbatimchar}}
464 {\catcode'\%=12 \gdef\percentchar{}}}
```

3.9 Redefining the index environment

`\ifhave@multicol` By default 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 package file, if

possible. The last page will be reformatted with balanced columns. This requires the `multicols` environment which is described elsewhere. So that `doc` can be run independently of `multicol.sty` we first check for its existence and set the `have@multicol` flag appropriately for use below.

```
465 \newif\ifhave@multicol
```

If we found `multicol.sty` we use it. It would be nice to delay this (and the re-definition of `theindex`) until we knew whether an index was actually required

...

```
466 \IfFileExists{multicol.sty}{\have@multicoltrue
467                               \RequirePackage{multicol}%
468                               }{}
```

`\IndexMin` If `multicol` is in use, 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.

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

`theindex` Now we start the multi-column mechanism, if appropriate. We use the `LATEX` counter `\c@IndexColumns` 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.

```
471 \ifhave@multicol
472   \renewenvironment{theindex}
473     {\begin{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.

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

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

```
475     {\end{multicols}}
```

If we can’t use `multicols` we warn the user and use an environment that’s basically the one from `article.sty`.

```
476 \else
477   \typeout{Can't find multicol.sty -- will use normal index layout if
478           necessary.}
479   \def\theindex{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi
480     \columnseprule \z@ \columnsep 35\p@
481     \twocolumn[\index@prologue]%
482     \IndexParms \let\item\@idxitem \ignorespaces}
483   \def\endtheindex{\if@restonecol\onecolumn\else\clearpage\fi}
484 \fi
```

Here are the necessary `makeindex` declarations. We disable scanning of macro names inside the index with `\scan@allowedfalse\n` to avoid recursion.

```
485 </package>
486 <+gind>preamble
487 <+gind>"\n \begin{theindex} \n \makeatletter\scan@allowedfalse\n"
488 <+gind>postamble
```

```
489 (+gind)"\n\n \end{theindex}\n"
490 (*package)
```

`\IndexPrologue` 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.

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

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

```
492 \@ifundefined{index@prologue}
493   {\def\index@prologue{\section*{Index}%
494     \markboth{Index}{Index}%
495     Numbers written in italic refer to the page
496     where the corresponding entry is described;
497     numbers underlined refer to the
498     \ifcodeline@index
499     code line of the
500     \fi
501     definition; numbers in roman refer to the
502     \ifcodeline@index
503     code lines
504     \else
505     pages
506     \fi
507     where the entry is used.
508   }}{}
```

`\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.

```
509 \@ifundefined{IndexParms}
510   {\def\IndexParms{%
511     \parindent \z@
512     \columnsep 15pt
513     \parskip 0pt plus 1pt
514     \rightskip 15pt
515     \mathsurround \z@
516     \parfillskip=-15pt
517     \small
```

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

```
\subitem
\subsubitem 518 \def\@idxitem{\par\hangindent 30pt}%
```

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

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

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

```
520 \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`.

```
521     \def\indexspace{\par\vspace{10pt plus 2pt minus 3pt}}%
522     }{}
```

`\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.

```
523 \def\efill{\hfill\nopagebreak}%
524 \endpackage
525 \gdef\efill\item_x1  "\\efill \n \subitem "
526 \gdef\efill\item_x2  "\\ "
527 \gdef\efill\item_x2  "\\efill \n \subsubitem "
528 \endpackage
```

`\pfill` The following definitions provide the `\pfill` command; if this is specified in the `\dotfil` index style file to `makeindex` as the delimiter to appear after index items, then the `\dotfill` 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.

```
529 \def\dotfill{\leaders\hbox to.6em{\hss .\hss}\hskip\z@ plus 1fill}%
530 \def\dotfil{\leaders\hbox to.6em{\hss .\hss}\hfil}%
531 \def\pfill{\unskip~\dotfill\penalty500\strut\nobreak
532     \dotfil~\ignorespaces}%
533 \endpackage
534 \gdef\pfill\delim_0  "\\pfill "
535 \gdef\pfill\delim_1  "\\pfill "
536 \gdef\pfill\delim_2  "\\pfill "
537 \endpackage
```

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

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

`\main` The *defining* entry for a macro name is flagged with the string `|main25` 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.

```
539 \@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*.

```
540 \@ifundefined{usage}{\def\usage#1{\textit{#1}}}{}
```

`\PrintIndex` This is the same as `\printindex` in the `makeidx` package.

```
541 \def\PrintIndex{\@input@{\jobname.ind}%
542     \global\let\PrintIndex\empty}
```

`\printindex` Since the above macro was called `\printindex` in older versions of `doc.sty` the following definition was provided up to version 1.9y.

²⁵With the current definition of `\encapchar` substituted for |

```

543 %\def\printindex{\typeout{\string\printindex\space is obsolete!}%
544 %           \typeout{Please use \string\PrintIndex\space
545 %                   if you are a macro implementor^^J
546 %                   or get a newer version of the documented
547 %                   software if you are a user}%
548 %           \PrintIndex}

```

We want headings in the index (and changes list) according to the initial character of the next block of entries and have to instruct `makeindex` appropriately. Unfortunately the specification for this changed sometime between versions 2.4 and 2.11 of `makeindex`. We provide both ways of doing it but unfortunately this will always produce a warning message from `makeindex`. This is for older versions:

```

549 </package>
550 <+gind,gglo>% The next lines will produce some warnings when
551 <+gind,gglo>% running Makeindex as they try to cover two different
552 <+gind,gglo>% versions of the program:
553 <+gind,gglo>lethead_prefix   "{\bfseries\hfil "
554 <+gind,gglo>lethead_suffix   "\hfil}\nopagebreak\n"
555 <+gind>lethead_flag          1
556 <+gglo>lethead_flag          0

```

This works for newer ones:

```

557 <+gind,gglo>heading_prefix   "{\bfseries\hfil "
558 <+gind,gglo>heading_suffix   "\hfil}\nopagebreak\n"
559 <+gind>headings_flag          1
560 <+gglo>headings_flag          0
561 <*package>

```

3.10 Dealing with the change history²⁶

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 second 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 older versions of Chen’s standard `makeindex` program limit 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 `\glossaryentry` commands.²⁷ 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.

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

²⁷Note that a recent change in L^AT_EX 2.09 changed the command name in the `.glo` file from `\indexentry` to `\glossaryentry`. It is therefore necessary to have a special `makeindex` style file called `gglo.ist` to process this file correctly.

We re-code nearly all chars found in `\sanitize` to letter since the use of special package which make some characters active might upset the `\changes` command when writing its entries to the file. However we have to leave % as comment and `\` as *space* otherwise chaos will happen. And, of course the `\` should be available as escape character.

```
562 \def\changes{\@bsphack\begingroup\@sanitize
563   \catcode'\z@ \catcode'\ 10 \MakePercentIgnore
564   \changes@}
565 \def\changes@#1#2#3{%
566   \protected@edef\@tempa{\noexpand\glossary{#1\levelchar
```

If the macro `\saved@macroname` doesn't contain any macro name (ie is empty) the current changes entry was done at top-level. In this case we precede it by `\generalname`.

```
567           \ifx\saved@macroname\@empty
568             \space
569             \actualchar
570             \generalname
571           \else
572             \saved@indexname
573             \actualchar
574             \string\verb\quotechar*%
575             \verbatimchar\saved@macroname
576             \verbatimchar
577           \fi
578           :\levelchar #3}}%
579   \@tempa\endgroup\@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! It works providing no macro names start with ! or ".)

```
580 \def\saved@macroname{}
```

`\saved@indexname` The macroname being document without a backslash for the index (or the environment name which doesn't have one in the first place).

```
581 \def\saved@indexname{}
```

`\generalname` This macro holds the string placed before changes entries on top-level.

```
582 \def\generalname{General}
```

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

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

`\GlossaryMin` The remaining macros are all analogues of those used for the `theindex` environment.

`\c@GlossaryColumns` 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.

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

`theglossary` The environment `theglossary` is defined in the same manner as the `theindex` environment.
`\endglossary`

```
586 \ifhave@multicol
587 \newenvironment{theglossary}{%
588   \begin{multicols}\c@GlossaryColumns
589   [\glossary@prologue][\GlossaryMin]%
590   \GlossaryParms \let\item\@idxitem \ignorespaces}%
591   {\end{multicols}}
592 \else
593 \newenvironment{theglossary}{%
594   \@restonecoltrue\if@twocolumn\@restonecolfalse\fi
595   \columnseprule \z@ \columnsep 35\p@
596   \twocolumn[\glossary@prologue]%
597   \GlossaryParms \let\item\@idxitem \ignorespaces}
598   {\if@restonecol\onecolumn\else\clearpage\fi}
599 \fi
```

Here are the necessary `makeindex` declarations with scanning disabled as for the index.

```
600 </package>
601 <+ggl>preamble
602 <+ggl>"\n \begin{theglossary} \n
603 <+ggl>  \makeatletter\scan@allowedfalse\n"
604 <+ggl>postamble
605 <+ggl>"\n\n \end{theglossary}\n"
```

This difference from `gind.ist` is necessary if you have an up-to-date L^AT_EX.

```
606 <+ggl>keyword "\glossaryentry"
607 <*package>
```

`\GlossaryPrologue` 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.

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

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

```
611 \ifundefined{glossary@prologue}
612   {\def\glossary@prologue{\section*{\Change History}}%
613     \markboth{\Change History}{\Change History}}%
614   {}}
```

`\GlossaryParms` Unless the user specifies otherwise, we set the change history using the same parameters as for the index except that we make it sort of ragged right as it contains text that often doesn't break nicely in small columns.

```

615 \@ifundefined{GlossaryParms}{\let\GlossaryParms\IndexParms
616 \expandafter\def\expandafter\GlossaryParms\expandafter{\GlossaryParms
617 \rightskip 15pt plus 1fil
618 \parfillskip -15pt plus -1fil\relax}
619 }{}

```

`\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 package 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.

```

620 \def\PrintChanges{\@input@{\jobname.gls}%
621 \global\let\PrintChanges\@empty}

```

3.11 Bells and whistles

`\StopEventually` If `\AlsoImplementation` is in force the whole documentation including the code part will be typeset. This is the default.
`\Finale`
`\AlsoImplementation` 622 `\newcommand\AlsoImplementation{%`
`\OnlyDescription`

To make this happen we have to define `\StopEventually` in a way that its argument is typeset at the very end or more exactly at `\Finale`. For this we save its argument in the macro `\Finale`.

```

623 \long\def\StopEventually##1{\@bsphack\gdef\Finale{##1%

```

But `\Finale` will be called at the very end of a file. This is exactly the point where we want to know if the file is uncorrupted. Therefore we also call `\check@checksum` at this point.

```

624 \check@checksum}%

```

On the other hand: `\StopEventually` is more or less a dividing point between description and code. So we start to look for the check-sum of the documented file by calling `\init@checksum`.

```

625 \init@checksum
626 \@esphack}%
627 }

```

Since `\AlsoImplementation` should be the default we execute it and thus `\StopEventually` gets the desired meaning.

```

628 \AlsoImplementation

```

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.

```

629 \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

```

630 ##1\endinput}\@esphack}

```

If no `\StopEventually` command is given we silently ignore a `\Finale` issued.

```

631 \let\Finale\relax

```

`\meta` The `\meta` macro is a bit tricky. We want to allow line breaks at blanks in the argument but we don't want a break in between. In the past this was done by defining `\meta` in a way that a `_` is active when the argument is scanned. Words are then scanned into `\hboxes`. The active `_` will end the preceding `\hbox` add an ordinary space and open a new `\hbox`. In this way breaks are only possible at spaces. The disadvantage of this method was that `\meta` was neither robust nor could it be `\protected`. The new implementation fixes this problem by defining `\meta` in a radically different way: we prevent hyphenation by defining a `\language` which has no patterns associated with it and use this to typeset the words within the angle brackets.

```
632 \ifx\l@nohyphenation\undefined
633   \newlanguage\l@nohyphenation
634 \fi
635 \DeclareRobustCommand\meta[1]{%
```

Since the old implementation of `\meta` could be used in math we better ensure that this is possible with the new one as well. So we use `\ensuremath` around `\langle` and `\rangle`. However this is not enough: if `\meta@font@select` below expands to `\itshape` it will fail if used in math mode. For this reason we hide the whole thing inside an `\nfss@text` box in that case.

```
636   \ensuremath\langle
637   \ifmmode \expandafter \nfss@text \fi
638   {%
639   \meta@font@select
```

Need to keep track of what we changed just in case the user changes font inside the argument so we store the font explicitly.

```
640   \edef\meta@hyphen@restore
641     {\hyphenchar\the\font\the\hyphenchar\font}%
642   \hyphenchar\font\m@ne
643   \language\l@nohyphenation
644   #1\}%
645   \meta@hyphen@restore
646 } \ensuremath\rangle
647 }
```

`\meta@font@select` Maske font used inside `\meta` customizable.

```
648 \def\meta@font@select{\itshape}
```

`\IndexInput` This next macro may be used to read in a separate file (possibly a package 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.

```
649 \def\IndexInput#1{%
```

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

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

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

```
651   \frenchspacing \@vobeyspaces
```

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

```
652 \input{#1}\endmacrocode
```

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

```
653 \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!

```
654 \def\maketitle{\par
655 \begingroup \def \thefootnote {\fnsymbol {footnote}}}%
656 \setcounter {footnote}\z@
657 \def\@makefnmark{\hbox to\z@{${\m@th^{\@thefnmark}}\hss}}%
658 \long\def\@makefnmark##1{\parindent 1em\noindent
659 \hbox to1.8em{\hss${\m@th^{\@thefnmark}}\##1}%
660 \if@twocolumn \twocolumn [\@maketitle ]%
661 \else \newpage \global \@topnum \z@ \@maketitle \fi
```

For special formatting requirements (such as in TUGboat), we use `pagestyle titlepage` for this; this is later defined to be `plain`, unless already defined, as, for example, by `ltugboat.sty`.

```
662 \thispagestyle{titlepage}\@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:

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

`\ps@titlepage` When a number of articles are concatenated into a journal, for example, it is not usual for the title pages of such documents to be formatted differently. Therefore, a class such as `ltugboat` can define this macro in advance. However, if no such definition exists, we use `pagestyle plain` for title pages.

```
666 \ifundefined{ps@titlepage}
667 {\let\ps@titlepage=\ps@plain}{}
```

`\MakeShortVerb` This arranges an abbreviation for `\verb` such that if you say `\MakeShortVerb{\<c>}` subsequently using `<c><text><c>` is equivalent to `\verb<c><text><c>`.²⁸ In addition, the fact that `<c>` is made active is recorded for the benefit of the `verbatim` and `macrocode` environments. Note particularly that the definitions below are global. The first thing we do (it needn't be first) is to record the—presumably new—special character in `\dospecials` and `\@sanitize` using `\add@special`.

Some unwary user might issue `\MakeShortVerb` for a second time, we better protect against this. We assume that this happened if a control sequence `\cc<c>` is bound, the probability that this name is used by another module is low. We will output a warning below, so that a possible error might be noticed by the programmer if he reads the LOG file. (Should have used module internal names, 'though.)

²⁸Warning: the commentary in the rest of this section was written by Dave Love.

`\MakeShortVerb*` This arranges an abbreviation for `\verb*` such that if you say `\MakeShortVerb*{<c>}` subsequently using `<c><text><c>` is equivalent to `\verb*{<c><text><c>}`.

```
668 \package
669 \*package | shortvrb)
670 \def\MakeShortVerb{%
671   \@ifstar
672   {\def\shortvrbdef{\verb*}\@MakeShortVerb}%
673   {\def\shortvrbdef{\verb}\@MakeShortVerb}}

674 \def\@MakeShortVerb#1{%
675   \expandafter\ifx\csname cc\string#1\endcsname\relax

676   \shortvrbinf{Made }{#1}\shortvrbdef
677   \add@special{#1}%
```

Then the character's current catcode is stored in `\cc<c>`.

```
678   \expandafter
679   \xdef\csname cc\string#1\endcsname{\the\catcode'#1}%
```

The character is spliced into the definition using the same trick as used in `\verb` (for instance), having activated `~` in a group.

```
680   \begingroup
681     \catcode'\~\active \lcode'\~'#1%
682     \lowercase{%
```

The character's old meaning is recorded in `\ac<c>` prior to assigning it a new one.

```
683     \global\expandafter\let
684     \csname ac\string#1\endcsname~%
685     \expandafter\gdef\expandafter~\expandafter{\@shortvrbdef~}%
686   \endgroup
```

Finally the character is made active.

```
687   \global\catcode'#1\active
```

If we suspect that `<c>` is already a short reference, we tell the user. Now he or she is responsible if anything goes wrong...

```
688   \else

689   \shortvrbinf\@empty{#1 already}{\@empty\verb(*)}%
690   \fi}
```

`\DeleteShortVerb` Here's the means of undoing a `\MakeShortVerb`, for instance in a region where you need to use the character outside a verbatim environment. It arranges for `\dospecials` and `\@sanitize` to be altered appropriately, restores the saved catcode and, if necessary, the character's meaning (as stored by `\MakeShortVerb`). If the catcode wasn't stored in `\cc<c>` (by `\MakeShortVerb`) the command is silently ignored.

```
691 \def\DeleteShortVerb#1{%
692   \expandafter\ifx\csname cc\string#1\endcsname\relax

693   \shortvrbinf\@empty{#1 not}{\@empty\verb(*)}%
694   \else

695   \shortvrbinf{Deleted }{#1 as}{\@empty\verb(*)}%
696   \rem@special{#1}%
697   \global\catcode'#1\csname cc\string#1\endcsname
```

We must not forget to reset `\cc\c`, otherwise the check in `\MakeShortVerb` for a repeated definition will not work.

```
698 \global \expandafter\let \csname cc\string#1\endcsname \relax
699 \ifnum\catcode'#1=\active
700 \begingroup
701 \catcode'\~\active \lccode'\~'#1%
702 \lowercase{%
703 \global\expandafter\let\expandafter~%
704 \csname ac\string#1\endcsname}%
705 \endgroup \fi \fi}
```

`\@shortvrbinfo` Helper function for info messages.

```
706 \def\@shortvrbinfo#1#2#3{%
707 (shortvrb) \PackageInfo{shortvrb}{%
708 !shortvrb) \PackageInfo{doc}{%
709 #1\expandafter\@gobble\string#2 a short reference
710 for \expandafter\string#3}}
```

`\add@special` This helper macro adds its argument to the `\dospecials` macro which is conventionally used by verbatim macros to alter the catcodes of the currently active characters. We need to add `\do\c` to the expansion of `\dospecials` after removing the character if it was already there to avoid multiple copies building up should `\MakeShortVerb` not be balanced by `\DeleteShortVerb` (in case anything that uses `\dospecials` cares about repetitions).

```
711 \def\add@special#1{%
712 \rem@special{#1}%
713 \expandafter\gdef\expandafter\dospecials\expandafter
714 {\dospecials \do #1}%
```

Similarly we have to add `\@makeother\c` to `\@sanitize` (which is used in things like `\index` to re-catcode all special characters except braces).

```
715 \expandafter\gdef\expandafter\@sanitize\expandafter
716 {\@sanitize \@makeother #1}}
```

`\rem@special` The inverse of `\add@special` is slightly trickier. `\do` is re-defined to expand to nothing if its argument is the character of interest, otherwise to expand simply to the argument. We can then re-define `\dospecials` to be the expansion of itself. The space after `=##1` prevents an expansion to `\relax`!

```
717 \def\rem@special#1{%
718 \def\do##1{%
719 \ifnum'#1='##1 \else \noexpand\do\noexpand##1\fi}%
720 \xdef\dospecials{\dospecials}%
```

Fixing `\@sanitize` is the same except that we need to re-define `\@makeother` which obviously needs to be done in a group.

```
721 \begingroup
722 \def\@makeother##1{%
723 \ifnum'#1='##1 \else \noexpand\@makeother\noexpand##1\fi}%
724 \xdef\@sanitize{\@sanitize}%
725 \endgroup}
726 </package | shortvrb)
727 (*package)
```


`\CharacterTable` The user interface to the character table-checking does some `\catcode`ing and then compares the following table with the stored version. We need to have `@` of type “other” within the table since this is the way it is usually returned when reading in a normal document. To nevertheless have a private letter we use `~` for this purpose. `~` does no harm as a “letter” as it comes last in the table and therefore will not gobble following space.

```
759 \def\CharacterTable{\begingroup \CharTableChanges \character@table}
```

`\character@table` This does the work of comparing the tables and reporting the result. Note that the following code is enclosed in a group with `~` catcoded to letter.

```
760 \begingroup
761 \catcode'\~ =11
762 \gdef\character@table#1{\def\used~table{#1}%
763 \ifx\used~table\default~table
764 \typeout{*****}%
765 \typeout{* Character table correct *}%
766 \typeout{*****}%
767 \else
768 \PackageError{doc}{Character table corrupted}
769 {\the\wrong@table}
770 \show\default~table
771 \show\used~table
772 \fi
773 \endgroup}
```

`\CharTableChanges` When the character table is read in we need to scan it with a fixed set of `\catcodes`. The reference table below was defined by assuming the normal `\catcodes` of \TeX , i.e. `@` is of type other and the only token of type “letter” are the usual letters of the alphabet. If, for some reason, other characters are made “letters” then their `\catcodes` need to be restored before checking the table. Otherwise spaces in the table are gobbled and we get the information that the tables are different, even if they are actually equal. For this reason `\CharTableChanges` can be set up to locally restore the `\catcodes` of such “letters” to “other”.

```
774 \global\let\CharTableChanges@empty
```

`\default~table` Here’s what the table *should* look like (modulo spaces).

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

`\wrong@table` We need a help message in case of problems.
792 `\newhelp\wrong@table{Some of the ASCII characters are corrupted.^^J`
793 `I now \string\show\space you both tables for comparison.}`

3.13 Attaching line numbers to code lines³⁰

The code in this section allows index entries to refer to code line numbers—the number of the first line of macrocode in the macro environment.

`\codeline@index` Indexing by code line is controlled by the `codeline@index` switch.

`\CodelineNumbered`

```
794 \newif\ifcodeline@index \codeline@indexfalse
795 \let\CodelineNumbered\codeline@indextrue
```

`\codeline@wrindex` The code index entries are written out by `\special@index`. If indexing is by code line this is `\let` to `\codeline@wrindex`; if indexing is by page it is just `\index`. However, if `\nofiles` is given, we omit writing such an index entry at all.

```
796 \def\codeline@wrindex#1{\if@filesw
797     \immediate\write\@indexfile
798     {\string\indexentry{#1}%
799     {\number\c@CodelineNo}}\fi}
```

`\special@index` By default no index entries are written out.

```
800 \let\special@index = \@gobble
```

`\CodelineIndex` This switches on use of the index file with `\makeindex`, sets the switch to indicate code line numbering and defines `\special@index` appropriately.

```
801 \def\CodelineIndex{\makeindex
802     \codeline@indextrue
803     \let\special@index\codeline@wrindex}
```

`\PageIndex` `\PageIndex` is similar.

```
804 \def\PageIndex{\makeindex
805     \codeline@indexfalse
806     \let\special@index\index}
```

`\c@CodelineNo` We need a counter to keep track of the line number.

```
807 \newcount\c@CodelineNo \c@CodelineNo\z@
```

`\theCodelineNo` This provides a hook to control the format of line numbers which may be defined in a class file.

```
808 \@ifundefined{theCodelineNo}
809   {\ifx\selectfont\undefined
810     \def\theCodelineNo{\rmfamily\scriptsize\arabic{CodelineNo}}%
811   \else
812     \def\theCodelineNo{\reset@font\scriptsize\arabic{CodelineNo}}%
813   \fi}
814   {}
```

³⁰Warning: the commentary was written by Dave Love.

3.14 Layout Parameters for documenting package files

`\tolerance` People documenting package files would probably rather have things “sticking out” in overfull `\hboxes` and poorish spacing, because they probably don’t want to spend a lot of time on making all the line breaks perfect!

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

The following `\mathcode` definitions allow the characters ‘\’ and ‘@’ to appear in `\ttfamily` font when invoked in math mode;³¹ particularly for something like `\@abc=1`.

If an *old* version of the `german` package 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`.

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

`\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 class file (like `ltugboat.cls` 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.

```
818 \@ifundefined{DocstyleParms}{-}{-}
```

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

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

`\AmSTeX` Here are a few definitions which can usefully be employed when documenting
`\BibTeX` package files: now we can readily refer to \mathcal{A} -TeX, \mathcal{B} TeX and \mathcal{S} TeX, as well
`\SliTeX` as the usual TeX and LaTeX.

```
820 \@ifundefined{AmSTeX}
821 {\def\AmSTeX{\leavevmode\hbox{${\mathcal A}\kern-.2em\lower.376ex%
822 \hbox{${\mathcal M$}\kern-.2em\mathcal S$-\TeX}}}{-}
823 \@ifundefined{BibTeX}
824 {\def\BibTeX{\rmfamily B\kern-.05em%
825 \textsc{i}\kern-.025em b}\kern-.08em%
826 T\kern-.1667em\lower.7ex\hbox{E}\kern-.125emX}}{-}
827 \@ifundefined{SliTeX}
828 {\def\SliTeX{\rmfamily S\kern-.06emL\kern-.18em\raise.32ex\hbox
829 {\scshape i}\kern-.03em\TeX}}{-}
```

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

```
\Web 830 \@ifundefined{PlainTeX}{\def\PlainTeX{\textsc{Plain}\kern2pt\TeX}}{-}
831 \@ifundefined{Web}{\def\Web{\textsc{Web}}}{-}
```

³¹You 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 `\mathcodes` to work with any family assignments. For an example see the article about the new font selection scheme.

3.15 Changing the `\catcode` of `%`

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

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

`\DocInput` The two macros above are now used to define the `\DocInput` macro which
was introduced in version v1.5l (or so) of the `doc` package. In older versions
`\MakePercentIgnore` was placed at the very end of `doc.sty`.

```
834 \def\DocInput#1{\MakePercentIgnore\input{#1}\MakePercentComment}
```

3.16 GetFileInfo

`\GetFileInfo` Define `\filedate` and friends from `info` in the `\ProvidesPackage` etc. commands.

```
835 \def\GetFileInfo#1{%
836   \def\filename{#1}%
837   \def\@tempb##1 ##2 ##3\relax##4\relax{%
838     \def\filedate{##1}%
839     \def\fileversion{##2}%
840     \def\fileinfo{##3}%
841     \edef\@tempa{\csname ver@#1\endcsname}%
842     \expandafter\@tempb\@tempa\relax? ? \relax\relax}
```

We can now finish the `docstrip` main module.

```
843 </package>
```

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 \TeX book). 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] FRANK MITTELBACH. The `doc`-option. *TUGboat*, Vol. 10(2), pp. 245–273, July 1989.
- [6] FRANK MITTELBACH, DENYS DUCHIER AND JOHANNES BRAAMS. `docstrip.dtx` (to appear). The file is part of the `DOC` package.
- [7] R. E. RASPE (*1737, †1797). Baron Münchhausens narrative of his marvellous travels and campaigns in Russia. Oxford, 1785.
- [8] RAINER SCHÖPF. A New Implementation of \LaTeX 's `verbatim` and `verbatim*` Environments. File `verbatim.doc`, version 1.4i.

Index

Numbers written in *italic* refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in *roman* refer to the code lines where the entry is used.

Symbols	A	
<code>\#</code> 780	<code>\active@escape@char</code>	<code>\check@angle</code> . . . 98, <u>100</u>
<code>\\$</code> 781 <u>281</u> , 294, 307	<code>\check@checksum</code> 624, <u>734</u>
<code>\%</code> . . . 167, 201, 402,	<code>\actualchar</code>	<code>\check@module</code> 42, 43, <u>87</u>
464, 781, 832, 833 8, <u>381</u> , 393, 415,	<code>\check@modulesfalse</code> 93
<code>\&</code> . . . 151, 317, 319, 781	420, 424, 432,	<code>\check@modulestrue</code> .
<code>*</code> 8, <u>538</u> , 783	436, 443, 445, 94, 95
<code>\-</code> 152, 224, 784	447, 448, 452,	<code>\check@percent</code> 197, <u>201</u>
<code>\@MakeShortVerb</code> . . .	456, 461, 569, 573	<code>\check@plus@etc</code> . . . <u>106</u>
. 672, 673, 674	<code>\add@special</code> . . 677, <u>711</u>	<code>\check@sum</code>
<code>\@SpecialIndexHelper@</code>	<code>\addto@hook</code> . . . 356, <u>357</u>	. . . 735, 736, 742,
. 412	<code>\AlsoImplementation</code>	748, 754, 755, 757
<code>\@idxitem</code> . . . 9, 474, 11, <u>622</u>	<code>\CheckModules</code> . . . 13, <u>93</u>
482, 518, 590, 597	<code>\AltMacroFont</code>	<code>\Checksum</code> 12, <u>757</u>
<code>\@indexfile</code> 797	. . . 13, <u>69</u> , 129, 135	<code>\close@crossref</code> 52, <u>299</u>
<code>\@input@</code> 541, 620	<code>\AmSTeX</code> <u>820</u>	<code>\codeline@index</code> . . . <u>794</u>
<code>\@latex@error</code> . 215, 221	<code>\AtBeginDocument</code> 18, 86	<code>\codeline@indexfalse</code>
<code>\@makefnstext</code> 658	 794, 805
<code>\@minipagefalse</code> . . . 38	B	<code>\codeline@indextrue</code>
<code>\@newlistfalse</code> 37	<code>\BibTeX</code> <u>820</u> 795, 802
<code>\@restonecolfalse</code> .	<code>\blank@linefalse</code> 30, <u>46</u>	<code>\codeline@windex</code> .
. 479, 594	<code>\blank@linetrue</code> . 32, <u>46</u> 796, 803
<code>\@restonecoltrue</code> . .	<code>\bslash</code> 13, <u>184</u> ,	<code>\CodelineIndex</code> 5, 7, <u>801</u>
. 479, 594	291, 330, 349,	<code>\CodelineNumbered</code> 7, <u>794</u>
<code>\@shortvrbdef</code>	417, 422, 426,	<code>\columnsep</code>
. . . 672, 673, 676, 685	450, 453, 457, 462	. . . 9, 480, 512, 595
<code>\@shortvrbinf</code> 676,	<code>\bslash@cnt</code> 733, 739,	<code>\columnseprule</code> 480, 595
689, 693, 695, <u>706</u>	742, 748, <u>755</u> , 758	D
<code>\@sverb</code> 207, <u>218</u>	C	<code>\DeclareRobustCommand</code>
<code>\@sxverbatim</code> 189	<code>\c@CodelineNo</code> . . 40, 635
<code>\@tempc</code> . . . 420, 424, 429	261, 268, 799, <u>807</u>	<code>\default~table</code>
<code>\@verbatim</code> <u>190</u>	<code>\c@GlossaryColumns</code> 763, 770, <u>775</u>
<code>\{</code> 166, 789 584, 588	<code>\DeleteShortVerb</code> . .
<code>\}</code> 166, 790	<code>\c@IndexColumns</code> <u>469</u> , 473	. . . 10, <u>691</u> , 730, 731
<code>\^</code> 17, 18, 788	<code>\c@StandardModuleDepth</code>	<code>\DeleteShortverb</code> . . 728
<code>^^A</code> 5, <u>15</u>	. . . 129, 134, 141, <u>146</u>	<code>\Describe@Env</code> <u>172</u>
<code>_</code> 788	<code>\ch@angle</code> 100, <u>101</u>	<code>\Describe@Macro</code> . . . <u>172</u>
<code>\ </code> . . . 151, 165, 184,	<code>\ch@percent</code> 90, <u>96</u>	<code>\DescribeEnv</code> 6, <u>172</u>
290, 361, 399, 789	<code>\ch@plus@etc</code> . . 104, <u>106</u>	<code>\DescribeMacro</code> . . 6, <u>172</u>
<code>\~</code> . . . 124, 125, 219,	<code>\changes</code> 11, <u>562</u>	<code>\DisableCrossrefs</code> .
228, 282, 285,	<code>\changes@</code> 564, 565 4, 7, 309
681, 701, 761, 790	<code>\char</code> 229	<code>\do@noligs</code>
<code>_</code> 41, 161,	<code>\character@table</code> 35, 199, 205, <u>224</u>
167, 274, 276, 759, <u>760</u>	<code>\do@not@index</code> . 353, <u>354</u>
278, 280, 538, 563	<code>\CharacterTable</code> 12, <u>759</u>	<code>\docdate</code> 14
	<code>\CharTableChanges</code> .	<code>\DocInput</code> . . . 4, 12, <u>834</u>
 759, <u>774</u>	<code>\DocstyleParms</code> . 10, <u>818</u>
		<code>\documentclass</code> 2

<code>\meta@font@select</code> .	<code>\produce@index</code> 323 , 327	<code>\SpecialIndex</code> 8 , 349 , 392
..... 639 , 648	<code>\protected@edef</code> ... 566	<code>\SpecialIndex@</code>
<code>\meta@hyphen@restore</code>	<code>\ps@plain</code> 667 396 , 398 , 440
..... 640 , 645	<code>\ps@titlepage</code> .. 11 , 666	<code>\SpecialMainEnvIndex</code>
<code>\mod@math@codes</code> 149 , 151	 8 , 266 , 396
<code>\Module</code> 13 ,	Q	<code>\SpecialMainIndex</code> .
127 , 132 , 139 , 148	<code>\quotechar</code> 8 , 381 , 394 , 8 , 263 , 396
<code>\more@macroname</code> 341 , 342	415 , 416 , 417 ,	<code>\SpecialUsageIndex</code> .
N	420 , 421 , 422 , 8 , 176 , 396
<code>\newcommand</code> 622	425 , 448 , 449 ,	<code>\StandardModuleDepth</code> 13
<code>\newcounter</code> 146	450 , 452 , 453 ,	<code>\star@module</code> .. 113 , 131
<code>\newhelp</code> 792	457 , 461 , 462 , 574	<code>\step@checksum</code> 302 , 758
<code>\newif</code> 46 ,	R	<code>\StopEventually</code> 11 , 622
92 , 95 , 308 , 465 , 794	<code>\RecordChanges</code> 6 , 12 , 583	<code>\subitem</code> 518
<code>\newlanguage</code> 633	<code>\rem@special</code>	<code>\subsubitem</code> 518
<code>\nfss@text</code> 637 696 , 712 , 717	<code>\sxmacro@code</code> . 162 , 169
<code>\noindent</code> 658	<code>\RequirePackage</code> ... 467	T
O	<code>\reset@font</code> 812	<code>\textit</code> 540
<code>\OldMakeindex</code> . 8 , 9 , 460	<code>\reversemarginpar</code> . 182	<code>\textsc</code> ... 825 , 830 , 831
<code>\OnlyDescription</code> ..	<code>\RightBraceIndex</code> ..	<code>\theCodelineNo</code> 7 , 41 , 808
..... 7 , 11 , 622 336 , 451	<code>theglossary</code> (environ-
P	<code>\rightskip</code> 9	ment) 586
<code>\PackageError</code> . 747 , 768	<code>\rmfamily</code>	<code>\theindex</code> 479
<code>\PackageInfo</code> .. 707 , 708	. 161 , 810 , 824 , 828	<code>theindex</code> (environ-
<code>\PageIndex</code> 7 , 804	S	ment) 9 , 471
<code>\parfillskip</code> 9	<code>\saved@indexname</code> ..	<code>\tolerance</code> 10 , 815
<code>\parindent</code> 9	. 240 , 242 , 572 , 581	<code>\ttdefault</code> 57 , 63 , 73 , 79
<code>\parskip</code> 9	<code>\saved@macroname</code> 238 ,	<code>\ttfamily</code> 206 ,
<code>\percentchar</code> 97 , 105 ,	240 , 567 , 575 , 580	433 , 437 , 443 , 446
117 , 337 , 459 , 460	<code>\scan@allowedfalse</code> .	U
<code>\PercentIndex</code> 308 , 309 , 324 , 329	<code>\updefault</code> 59 , 65
..... 337 , 458 , 460	<code>\scan@allowedtrue</code> .	<code>\usage</code> 10 , 540
<code>\pfill</code> 529 308 , 310 , 325	<code>\used~table</code> 762 , 763 , 771
<code>\PlainTeX</code> 830	<code>\scan@macro</code> ... 294 , 300	<code>\usefont</code> 56 , 72
<code>\pm@module</code>	<code>\scshape</code> 829	
. 109 , 111 , 119 , 123	<code>\sfdefault</code> 157	V
<code>\pm@modulefalse</code> . 48 , 89	<code>\short@macro</code> .. 314 , 316	<code>\verb</code> 6 , 204
<code>\pm@moduletrue</code> ... 126	<code>\slash@module</code> . 115 , 131	<code>\verb@balance@group</code>
<code>\predisplaypenalty</code> .	<code>\sldefault</code> 75 , 81 208 , 220 , 222
..... 24 , 186 , 188	<code>\SliTeX</code> 820	<code>\verb@egroup</code>
<code>\PrintChanges</code> .. 12 , 620	<code>\SortIndex</code> 8 , 447 208 , 220 , 223
<code>\PrintDescribeEnv</code> .	<code>\special@escape@char</code>	<code>\verb@eol@error</code> 206 , 208
..... 6 , 180 , 273 281 ,	<code>\verbatim</code> 186
<code>\PrintDescribeMacro</code>	293 , 301 , 328 , 329	<code>verbatim</code> (environ-
..... 6 , 175 , 273	<code>\special@index</code> 392 ,	ment) 6 , 186
<code>\PrintEnvName</code> 6 , 257 , 273	415 , 420 , 424 ,	<code>verbatim*</code> (environ-
<code>\PrintIndex</code>	431 , 436 , 440 ,	ment) 6 , 186
.. 9 , 541 , 544 , 548	448 , 452 , 456 ,	<code>\verbatim@nolig@list</code>
<code>\printindex</code> 543	461 , 800 , 803 , 806	. 35 , 199 , 205 , 224
<code>\PrintMacroName</code> ...	<code>\SpecialEnvIndex</code> ..	<code>\verbatimchar</code> 8 , 391 ,
..... 6 , 255 , 273 8 , 181 , 442	394 , 416 , 417 ,
	<code>\SpecialEscapechar</code> .	421 , 422 , 425 ,
	.. 7 , 281 , 296 , 299	

426, 449, 450,	W	X
452, 453, 457,	\Web	830
462, 463, 575, 576	\wrong@table ..	769, 792
	\xmacro@code . . .	21, 164

Change History

BHK	\endtheindex: Incorporated new multicols env.	39
\GlossaryMin: Added to support \changes.	\meta: Macro added.	46
\GlossaryParms: Added to support \changes.	theindex: Incorporated new multicols env.	39
\GlossaryPrologue: Added to support \changes.	v1.5a	
\PrintChanges: Added to support \changes.	General: Now input multicols.sty instead of multicols.sty	38
\RecordChanges: Renames former \PrintChanges command.	theindex: Call multicols first	39
\c@GlossaryColumns: Added to support \changes.	v1.5b	
\changes: Changed definition of \protect.	\macro@cnt: vbox to vtop changed in makelabel (test)	25
Documented \changes command.	v1.5c	
\endglossary: Added to support \changes.	\produce@index: Corrected bad bug by placing the scan@allowedfalse macro into short@macro	31
\glossary@prologue: Added to support \changes.	\short@macro: Corrected bad bug by putting the scan@allowedfalse macro before printing the argument.	30
\saved@macroname: Provided for sorting outside macro environment	v1.5d	
theglossary: Added to support \changes.	\Describe@Env: \marginparwidth setting added.	22
v1.0p	v1.5e	
General: Use new error commands	\macro@cnt: ht strutbox changed to baselineskip (test)	25
v1.4?	v1.5f	
General: changes to the index env. use DEK's algorithm and implement a twocols env.	\macro@cnt: MacroTopsep parameter added.	25
v1.4r	General: Thanks to Brian who documented the \changes macro feature.	1
General: twocols env. placed into separate file	v1.5g	
v1.4s	\PlainTeX: space between plain and TeX changed.	53
\produce@index: Added noexpand to all \if tests to avoid garbage produced by new active chars	General: MacroTopsep now called MacrocodeTopsep and new MacroTopsep added	1
Used \string for the same reason.	v1.5h	
v1.4t	General: All lines shortened to ¡72 characters	1
\c@IndexColumns: Counter added.	v1.5i	
	\check@percent: Definition changed to 'long'	23

Macro <code>\next</code> used to guard against macro with arguments	23	v1.5o	<code>\changes</code> : New sorting.	43
General: Avoid reading the file twice.	14	v1.5p	<code>theglossary</code> : Now call <code>\multicols</code> first.	44
v1.5j		v1.5q	<code>\CharacterTable</code> : Made character table more readable.	51
<code>\AmSTeX</code> : Macro <code>AmsTeX</code> renamed to <code>AmSTeX</code>	53	General: ‘...Listing macros renamed to ‘...Input. Suggested by R. Wonneberger	1	
<code>\Describe@Env</code> : <code>\ignorespaces</code> added as a temporary fix	22	v1.5r	<code>\endmacrocode</code> : Support for code line no. (Undoc)	16
<code>\Describe@Macro</code> : <code>\ignorespaces</code> added as a temporary fix	22	<code>macrocode</code> : Support for code line no. (Undoc)	15	
<code>\maketitle</code> : <code>thispagestyle plain</code> removed	47	v1.5s	<code>\LeftBraceIndex</code> : Support for code line no. (Undoc)	38
General: Corrections by Ron Whitney added	1	<code>\MacroIndent</code> : Support for code line no. (Undoc)	21	
v1.5k		<code>\PercentIndex</code> : Support for code line no. (Undoc)	38	
<code>\Checksum</code> : Macro added to support checksum.	50	<code>\RightBraceIndex</code> : Support for code line no. (Undoc)	38	
<code>\Finale</code> : Support for checksum.	45	<code>\SpecialIndex</code> : Support for code line no. (Undoc)	36	
<code>\PrintIndex</code> : <code>\printindex</code> changed to <code>\PrintIndex</code>	41	<code>\SpecialUsageIndex</code> : Support for code line no. (Undoc)	36	
<code>\StopEventually</code> : Support for checksum.	45	<code>\codeline@index</code> : Support for code line no. (Undoc)	52	
<code>\bslash@cnt</code> : Macro added to support checksum.	50	<code>\it@is@a</code> : Support for code line no. (Undoc)	38	
<code>\check@checksum</code> : Macro added to support checksum.	50	<code>\macro@cnt</code> : Support for code line no. (Undoc)	26	
<code>\check@sum</code> : Macro added to support checksum.	50	v1.5t	<code>\CharacterTable</code> : Make <code>~</code> letter in <code>chartable</code> macros.	51
<code>\endenvironment</code> : Fix for save stack problem.	27	<code>\IndexInput</code> : Call <code>\endmacrocode</code> instead of <code>\endtrivlist</code> .	47	
<code>\init@checksum</code> : Macro added to support checksum.	50	<code>\macro@code</code> : Call <code>\leavevmode</code> to get <code>\everypar</code> on blank lines.	16	
<code>\macro@cnt</code> : Fix for save stack problem.	24, 25	Common code added.	16	
<code>\maketitle</code> : Added		<code>\produce@index</code> : Added <code>\relax</code> as a possible token to allow extensions.	31	
<code>\ps@titlepage</code>	47	<code>macrocode</code> : Common code moved to <code>\macro@code</code> .	15	
<code>\ps@titlepage</code> : Added				
<code>\ps@titlepage</code>	47	v1.5u	<code>\CharacterTable</code> : Made <code>@</code> other in default table.	51
<code>\scan@macro</code> : Support for checksum added.	29	<code>\CodelineIndex</code> : Added <code>\PageIndex</code> and <code>\CodelineIndex (Undoc)</code>	52	
<code>\step@checksum</code> : Macro added to support checksum.	50			
v1.5l				
<code>\c@CodelineNo</code> : Counter added to support code line numbers	52			
<code>\macro@code</code> : Code line numbers supported.	16			
v1.5m				
<code>\CharacterTable</code> : Macro added to check character translation problems.	51			
<code>\changes</code> : <code>\actualchar</code> in second level removed.	43			

\DocstyleParms: \DocStyleParms now empty	53	\DeleteShortverb: Added (from newdoc).	50
\check@percent: equal sign added.	23	\MacroFont: Added \reset@font for NFSS.	17
v1.5v		\MakeShortVerb: Added (from newdoc but now alters \dospecials, \@sanitize). . .	47
\Describe@Env:		\Module: Added.	20
\MakePrivateLetters added.	22	\PrintIndex: Documentation moved to interface section. . .	41
Macro added.	22	\add@special: Added for short verb facility.	49
\Describe@Macro: Macro added.	22	\bslash: Moved \bslash documentation to ‘user interface’ part	22
\changes: ‘Re-code a lot of chars.	43	\ch@angle: Added.	18
\m@cro@: \macro@ renamed to \m@cro@ since AmSTeX defines another macro of the same name.	24	\ch@percent: Added.	18
v1.5w		\check@angle: Added.	18
\macro@code: Skip of \@totalleftmargin added. . .	16	\check@plus@etc: Added.	18
\meta: Breaks at space allowed. . .	46	\codeline@index: Documented code line no. support.	52
General: Counter codelineno renamed to CodelineNo	1	\do@not@index: Replaced with newdoc version.	33
v1.5x		\ifhave@multicol: Added to support avoiding multicol.sty .	38
\MacroFont: \math@fontsfalse added for NFSS.	17	\ifnot@excluded: Replaced with newdoc version.	33
v1.5y		\ifpm@module: Added.	18
\MacroIndent: Default changed. . .	21	\macro@cnt: Catcode backslash to other (from newdoc).	24
\c@CodelineNo: Default changed.	52	Removed redundant code checking for \par.	26
v1.5z		\macro@font: Added to support distinction of modules.	17
\Finale: Define \Finale globally.	45	\pm@module: Added.	19
v1.6a		\rem@special: Added for short verb facility.	49
\meta: Extra space bug corrected.	46	\saved@macroname: Changed string used for better sorting.	43
v1.6b		\short@macro: Ensure character stored in \macro@namepart as ‘letter’ so index exclusion works.	30
\MacroIndent: \rm moved before \scriptsize to avoid unnecessary fontwarning.	21	\slash@module: Added.	19
\c@CodelineNo: \rm moved before \scriptsize to avoid unnecessary fontwarning.	52	\theCodelineNo: Existing definition not overwritten. . . .	52
v1.6c		Use \reset@font for NFSS. . . .	52
\changes: Again new sorting. . . .	43	\verb: Added math mode check (from verbatim.sty)	23
v1.6e		Now warns about newlines (from newdoc with ‘@noligs added). . .	23
theglossary: Turned into env definition.	44	\wrong@table: Moved to where the catcodes are right so it works.	52
theindex: Turned into env definition.	39		
v1.7a			
\@sverb: Added for \verb change. Now same as in verbatim.sty. . .	24		
\@verbatim: Removed redundant \tt.	23		
\CheckModules: Added.	18		
\DeleteShortVerb: Added (from newdoc but now alters \dospecials, \@sanitize). . .	48		
Check for previous matched \MakeShortVerb to avoid error.	48		

General: Upgrade for LaTeX2e . . .	1	v1.9r	
v1.9b			<code>\maketitle</code> : Added new definitions of <code>\@makefnmark</code> and <code>\@makefnmark</code>
<code>\macro@code</code> : Forcing any label from macro env.	15		47
v1.9d		v1.9s	
General: Protected changes entry. .	1		<code>\SpecialUsageIndex</code> : Added missing percent and changed to <code>\ttfamily</code>
v1.9e			37
<code>\SpecialEnvIndex</code> : The index needs protecting	37	v1.9t	
<code>\SpecialUsageIndex</code> : use <code>\ttfamily</code> with <code>\string</code>	37		<code>\MacroFont</code> : Removed <code>\math@fontsfalse</code> (different math setup /pr1622
v1.9e.2			17
<code>\DeleteShortVerb</code> : -js: Reset 'cc'(c) in in <code>\DeleteShortVerb</code>	49		<code>\macro@font</code> : Removed <code>\math@fontsfalse</code> (different math setup /pr1622
<code>\MakeShortVerb</code> : -js: Check if (c) is already an abbreviation for <code>\verb</code>	47		17
v1.9f		v1.9u	General: Use <code>\GetFileInfo</code>
<code>\SpecialUsageIndex</code> : should have used <code>\noexpand</code> , sigh	37		1
v1.9g			<code>\changes</code> : Use <code>\protected@edef</code> Use value of <code>\saved@macroname</code> to find out about change entries at outer level
<code>\SpecialEnvIndex</code> : should have used <code>\noexpand</code> , sigh	37		43
v1.9h			<code>\generalname</code> : Macro added
<code>\PrintChanges</code> : Use <code>\@input@</code> instead of <code>\@input</code>	45		43
<code>\PrintIndex</code> : Use <code>\@input@</code> instead of <code>\@input</code>	41		<code>\macro@cnt</code> : Removed brace group which killed <code>\DoNotIndex</code>
v1.9i			26
<code>\SpecialEnvIndex</code> : should have used <code>\protect</code>	37		<code>\saved@macroname</code> : Now empty by default
<code>\SpecialUsageIndex</code> : should have used <code>\protect</code>	37	v1.9v	43
v1.9j			<code>\@shortvrbinfo</code> : (DPC) Macro added
<code>\SpecialUsageIndex</code> : Back to string:-)	37		49
v1.9k			<code>\DeleteShortVerb</code> : (DPC) Use <code>\@shortvrbinfo</code>
<code>\ch@angle</code> : Have < active	18		48
<code>\endenvironment</code> : Don't checkfor nesting	27		<code>\MakeShortVerb</code> : (DPC) Use <code>\@shortvrbinfo</code>
<code>\macro@cnt</code> : Don't omit extra group	24		48
Fix probably no longer necessary	24	v1.9w	
Remove <code>\macro@level</code>	25		<code>\AlsoImplementation</code> : Macro added
v1.9m			45
<code>\ifhave@multicol</code> : Use <code>\RequirePackage</code> to load multicol	39		<code>\PrintChanges</code> : Turn the cmd into a noop after use.
v1.9n			45
<code>\OnlyDescription</code> : Ignore <code>\Finale</code> if no <code>\StopEventually</code> is given	45		<code>\PrintIndex</code> : Turn the cmd into a noop after use.
v1.9o			41
<code>\GetFileInfo</code> : Macro added	54		<code>\index@prologue</code> : Text changed .
		v1.9x	40
			<code>\index@prologue</code> : Text depends on code lines used
		v1.9y	40
			<code>\MacroFont</code> : Support compat mode
			17
			<code>\macro@font</code> : Support compat mode
		v1.9z	17
			<code>\GetFileInfo</code> : Missing percent latex/2404
			54
			<code>\printindex</code> : Commented out . .
			41

v2.0a	<code>\macro@font</code> : Support changing <code>\MacroFont</code> in preamble	17	v2.0l	<code>\meta</code> : Fixing changes for (pr/3170)	46
v2.0b	General: Init docs private comment char at begin of document again (pr2581)	15	v2.0m	<code>\meta</code> : More fixing changes for (pr/3170)	46
v2.0c	<code>\SpecialIndex@</code> : Macro added.	36	v2.0n	<code>\check@plus@etc</code> : Partly support docstrip’s “verbatim” directive (pr/3331)	19
	<code>\SpecialUsageIndex</code> : Correctly handle single character control sequences like <code>\<</code>	36	v2.1a	<code>\@shortvrbinfo</code> : (HjG) Third argument added on behalf of <code>\MakeShortVerb*</code>	49
v2.0d	<code>\SpecialUsageIndex</code> : Correctly handle second index entry by using <code>\special@index</code> not <code>\index</code> (PR/2928).	37		<code>\DeleteShortVerb</code> : (HjG) Notify user if it’s not a short verb character	48
v2.0e	<code>\SpecialUsageIndex</code> : Use <code>\string</code> , not <code>\protect</code> in argument to <code>\special@index</code>	37		<code>\MakeShortVerb*</code> : (HjG) Added * form	48
	<code>\short@macro</code> : Correctly use the case-changing trick.	30	v2.1b	<code>\SpecialEnvIndex</code> : environment names incorrectly sorted in index (pr/3615)	38
v2.0f	<code>\SpecialIndex@</code> : Temp fix to allow strange code in arg 1 (PR 2968)	36	v2.1c	<code>\SpecialUsageIndex</code> : environment names incorrectly sorted in index (pr/3615)	37
v2.0g	<code>\Describe@Env</code> : Parse backslash as letter in argument to <code>\DescribeMacro</code>	22	v2.1d	General: Corrected description of <code>\changes</code> macro.	11
	<code>\SpecialIndex@</code> : Correct so-called temp fix. I’m not going to explain this.	36	v2.1e	<code>\mod@math@codes</code> : (pr/4096)	20
v2.0h	<code>\Describe@Env</code> : Correct errors introduced in v2.0g.	22	v2.1f	<code>\bslash@cnt</code> : Suppress <code>\Checksum</code> check if no checksum is specified in the file	50
v2.0i	<code>\meta</code> : New implementation (pr/3170)	46		<code>\check@checksum</code> : Suppress <code>\Checksum</code> check if no checksum is specified in the file	50
v2.0j	<code>\index@prologue</code> : Less obscure wording? (CAR pr/3202)	40	v2.1g	<code>\GlossaryParms</code> : Use ragged setting by default	44
v2.0k	<code>\meta@font@select</code> : Macro added (pr/3170)	46		<code>\changes</code> : Use <code>\saved@indexname</code>	43
				<code>\macro@cnt</code> : Use <code>\saved@indexname</code>	25
				<code>\saved@indexname</code> : Use <code>\saved@indexname</code>	43