The `doc` and `shortvrb` Packages[*]

Frank Mittelbach[†‡§]

Printed January 14, 2024

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

Abstract

Roughly 30 years ago (version 1.0 was dated 1988/05/05) I wrote the first version of the `doc` package, a package to provide code documentation for TeX code. Since then it has been used all over the place to document the LaTeX kernel and most of the packages that are nowadays available. The core code of version 2 (which is the current version) exists since 1998, i.e., for 20 years.

If I would restart from scratch I would do a lot of things differently these days and in fact several other people have tried to come up with better solutions. However, as the saying goes, a bad standard is better than none, `doc` has prevailed and changing it now in incompatible ways is probably not really helpful.

So this is version 3 of the package with some smaller extensions that are upwards compatible but hopefully serve well. Most important modifications are the integration of the `hypdoc` package to enable links within the document (in particular from the index) if so desired. Also integrated are the ideas from the `DoX` package by Didier Verna (although I offer a different interface that imho fits better with the rest of `doc`'s interfaces). Finally I updated a few odds and ends.

---

[*]This file has version number v3.0m dated 2022/11/13.

[†]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).

[§]Version 3 now integrates code from Didier Verna's `DoX` package and some of his documentation was reused (a.k.a. stolen).

Contents

1   Introduction

This is a new version of the `doc` package, written roughly 30 years after the initial release. As the package has been used for so long (and largely unchanged) it is absolutely important to preserve existing interfaces, even if we can agree that they could have been done better.

So this is a light-weight change, basically adding hyperlink support and adding a way to provide generally `doc` elements (not just macros and environments) and try to do this properly (which wasn't the case for environments either in the past). The ideas for this have been stolen from the `DoX` package by Didier Verna even though I didn't keep his interfaces.

Most of the documentation below is from the earlier release which accounts for some inconsistencies in presentation, mea culpa.

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 usually use `article`.

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

`\DocInput`   Finally the ⟨*special input commands*⟩ part should contain one or more
`\IndexInput` `\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 for more details of `\IndexInput`. Multiple `\DocInput`s can be used with a number of included files 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
4 \usepackage[T1]{fontenc}
```

```
 5 \usepackage{xspace}
 6
 7 \OnlyDescription
 8
 9 \EnableCrossrefs
10 %\DisableCrossrefs      % Say \DisableCrossrefs if index is ready
11 \CodelineIndex
12 \RecordChanges          % Gather update information
13 \SetupDoc{reportchangedates}
14 %\OnlyDescription       % comment out for implementation details
15 \setlength\hfuzz{15pt}  % don't show so many
16 \hbadness=7000          % over- and underfull box warnings
17 \begin{document}
18    \DocInput{doc.dtx}
19 \end{document}
20 ⟨/driver⟩
```

2.2 Package options

New in v3

Starting with version 3 the doc package now offers a small number of package options to modify its overall behavior. These are:

hyperref, nohyperref Boolean (default true). Load the hyperref package and make index references to code lines and pages and other items clickable links. nohyperref is the complementary key.

multicol, nomulticol Boolean (default true). Load the multicol package for use in typesetting the index and the list of changes. nomulticol is the complementary key.

debugshow Boolean (default false). Provide various tracing information at the terminal and in the transcript file. In particular show which elements are indexed.

noindex Boolean (default false). If set, all automatic indexing is suppressed. This option can also be used on individual elements as described below.

noprint Boolean (default false). If set, then printing of element names in the margin will be suppressed. This option can also be used on individual elements as described below.

reportchangedates Boolean (default false). If set, then change entries list the date after the version number in the change log.

\SetupDoc   Instead of providing options to the doc package you can call \SetupDoc and provide them there. This allows, for example, to change default values in case doc was already loaded earlier.

2.3 General conventions

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

4

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 characters `^^A` and `^^X` are both defined as a comment character later on.[1] 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` (*env.*)   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*` (*env.*)   Instead of the `macrocode` environment one can also use the `macrocode*` environment which produces the same results except that spaces are printed as ␣ characters.

2.4   Describing the usage of macros and environments

`\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.

As the argument to `\DescribeMacro` is a command name, many people got used to using the (incorrect) short form, i.e., omitting the braces around the argument as in `\DescribeMacro\foo`. This does work as long as the macro name consists only of "letters". However, if the name contains special characters that are normally not of type "letter" (such as @, or in case of expl3 _ and :) this will fail dramatically. `\DescribeMacro` would then receive only a partial command name (up to the first "non-letter") e.g., `\DescribeMacro\foo@bar` would be equivalent to `\DescribeMacro{\foo} @bar` and you can guess that this can resulting in both incorrect output and possibly low-level error messages.

`\DescribeEnv`   An analogous macro `\DescribeEnv` should be used to indicate that a LaTeX environment is explained. It will produce a somewhat different index entry and a slightly different display in the margin. Below I used `\DescribeEnv{verbatim}`.

New in v3   Starting with version 3 the `\Describe...` commands accept an optional ar-

---

[1] In version 2 it was only `^^A`, but many keyboards combine `^` and `A` and automatically turn it into "Ä"; so `^^X` was added as an alternative in version 3.

gument in which you can specify either `noindex` or `noprint` to suppress indexing or printing for that particular instance. Using both would be possible too, but pointless as then the commands wouldn't do anything any more.

2.5 Describing the definition of macros and environments

macro (*env.*) To describe the definition of a (new) macro we use the `macro` environment. It has one argument: the name of the new macro.[2] 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.

New in v3   In fact it is now allowed to specify several macros in the argument, separated by commas. This is a short form for starting several `macro` environments in direct succession. Of course, you should then have also only one matching `\end{macro}`.

\MacrocodeTopsep (*skip*)   There also exist four style parameters: `\MacrocodeTopsep` and `\MacroTopsep`
\MacroTopsep (*skip*) are used to control the vertical spacing above and below the `macrocode` and
\MacroIndent (*dimen*) the `macro` environment, `\MacroIndent` is used to indent the lines of code and
\MacroFont `\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}`.

environment (*env.*)   For documenting the definition of environments one can use the environment `environment` which works like the `macro` environment, except that it expects an ⟨*env-name*⟩ (without a backslash) as its argument and internally provides different index entries suitable for environments. Nowadays you can alternatively specify a comma-separated list of environments.

New in v3   Starting with version 3 these environments accept an optional argument in which you can specify `noindex` or `noprint` or both to suppress indexing or printing for that particular instance. If any such setting is made on the environment level it overwrites whatever default was given when the `doc` element was defined or when the package was loaded.

2.6 Formatting names in the margin

\PrintDescribeMacro As mentioned earlier, some macros and environment print their arguments in
\PrintDescribeEnv the margin. The actual formatting is done by four macros which are user defin-
\PrintMacroName able.[3] They are named `\PrintDescribeMacro` and `\PrintDescribeEnv` (defin-
\PrintEnvName ing how `\DescribeMacro` and `\DescribeEnv` behave) and `\PrintMacroName` and `\PrintEnvName` (called by the `macro` and `environment` environments, respectively).

---

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

[3]You 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 redefine them accept their argument and do nothing with it.

2.7   Providing further documentation items

New in v3

Out of the box the `doc` package offers the above commands and environments to document macros and environments. With version 3 this has now been extended in a generic fashion so that you can easily provide your own items, such as counters, length register, options etc.

`\NewDocElement`   The general syntax for providing a new `doc` element is

$$\texttt{\textbackslash NewDocElement[}\langle \textit{options}\rangle \texttt{]\{}\langle \textit{element-name}\rangle \texttt{\}\{}\langle \textit{env-name}\rangle \texttt{\}}$$

By convention the ⟨*element-name*⟩ has the first letter uppercased as in `Env` or `Macro`.

Such a declaration will define for you

- the command `\Describe`⟨*element-name*⟩ which has the syntax

  $$\texttt{\textbackslash Describe}\langle \textit{element-name}\rangle \texttt{[}\langle \textit{options}\rangle \texttt{]\{}\langle \textit{element}\rangle \texttt{\}}$$

- the environment ⟨*env-name*⟩ which has the syntax

  $$\texttt{\textbackslash begin\{}\langle \textit{env-name}\rangle \texttt{\}[}\langle \textit{options}\rangle \texttt{]\{}\langle \textit{element}\rangle \texttt{\}}$$

- the display command `\PrintDescribe`⟨*element-name*⟩ with the syntax

  $$\texttt{\textbackslash PrintDescribe}\langle \textit{element-name}\rangle \texttt{\{}\langle \textit{element}\rangle \texttt{\}}$$

- and the `\Print`⟨*element-name*⟩`Name` display command for the environment.

If any of the commands or the environment is already defined (which especially with the ⟨*env-name*⟩ is a danger) then you will receive an error telling you so.

`\RenewDocElement`   If you want to modify an existing `doc` element use `\RenewDocElement` instead.

For example, the already provided "`Env`" `doc` element could have been defined simply by making the declaration `\NewDocElement{Env}{environment}` though that's not quite what has been done, as we will see later.

The ⟨*options*⟩ are keyword/value and define further details on how that `doc` element should behave. They are:

`macrolike` Boolean (default `false`). Does this `doc` element starts with a backslash?

`envlike` Boolean. Complementary option to `macrolike`.

`toplevel` Boolean (default `true`). Should all a top-level index entry be made? If set to `false` then either no index entries are produced or only grouped index entries (see `idxgroup` for details).

`notoplevel` Boolean. Complementary option to `toplevel`.

`idxtype` String (default ⟨*env-name*⟩). What to put (in parentheses if non-empty) at the end of a top-level index entry.

`printtype` String (default ⟨*env-name*⟩). What to put (in parentheses if non-empty) after an element name in the margin.

`idxgroup` String (default ⟨*env-name*⟩`s`). Name of the top-level index entry if entries are grouped. They are only grouped if this option is non-empty.

7

noindex Boolean (default `false`). If set this will suppress indexing for elements of this type. This setting overwrite any global setting of `noindex`.

noprint Boolean (default `false`). If set this will suppress printing the element name in the margin. This setting overwrite any global setting of `noprint`.

As usual giving a boolean option without a value sets it to `true`.

2.8 Displaying sample code verbatim

verbatim (*env.*) 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
verbatim* (*env.*) is slightly altered to suppress those characters.[4] The `verbatim∗` environment is
\verb 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` (§2.5).

2.9 Using a special escape character

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

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

2.10 Cross-referencing all macros used

\DisableCrossrefs As already mentioned, every macro name used within a `macrocode` or `macrocode∗`
\EnableCrossrefs 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[5] 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.

---

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

[5]This comment was written about 30 years ago. TeX is still considerably slower but while it took minutes to process a large document (such as the LaTeX kernel documentation) it takes seconds or less these days. Thus `\DisableCrossrefs` isn't really that necessary these days.

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

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.

Production (or not) of the index (via the `\makeindex` commend) is controlled by using or omitting the following declarations in the driver file preamble; if

`\PageIndex`
`\CodelineIndex`
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` and possibly further `\Describe...` commands refer to a page number but those produced by the `macro` environment (or other `doc` element environments) refer to the code lines, which will be numbered

`\theCodelineNo`
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.

`\CodelineNumbered`
When you don't wish to get an index but want your code lines numbered use `\CodelineNumbered` instead of `\CodelineIndex`. This prevents the generation of an unnecessary `.idx` file.

2.11 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`
`\quotechar`
`\encapchar`
The `\actualchar` is used to separate the 'key' and the actual index entry. The `\quotechar` is used before a special index program character to suppress its special meaning. The `\encapchar` separates the indexing information from a letter string 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.

TODO: *describe old* `\SpecialMainIndex` *and* `\SpecialUsageIndex`

`\SpecialMainMacroIndex`
`\SpecialMainEnvIndex`
To produce a main index entry for a macro the `\SpecialMainMacroIndex` macro[9] may be used. It is called 'special' because it has to print its argument

---

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

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

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

verbatim. A similar macro, called `\SpecialMainEnvIndex` is used for indexing the main definition point of an environment.[10]

`\SpecialMacroIndex`
`\SpecialEnvIndex`
To index the usage of a macro or an environment `\SpecialMacroIndex` and `\SpecialEnvIndex` may be used.

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

New in v3
If further code elements are declared with `\NewDocElement{`⟨*name*⟩`}`... then this sets up additional indexing commands, e.g., `\SpecialMain`⟨*name*⟩`Index`.

`\SpecialIndex`
The `macrocode` environment is automatically indexing macros (normally by code line number). You can (with care) also do this manually by `\SpecialIndex`. However, note that if `\CodelineIndex` is used this will generate an entry referring to the last code line which is usually not what you want. It does, however, make some sense if you always refer to pages only, i.e., if you use `\PageIndex`.

`\SpecialShortIndex`
New in v3
For single character macros, e.g., `\{`, doesn't always work correctly. For this reason there is now also a special variant the can produce correct index entries for them.

`\SortIndex`
Additionally a `\SortIndex` command is provided. It takes two arguments—the sort key and the actual index entry.

`\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 `\+`) then that caused a problem because you ended up with an index entry containing `\verb+\++` which will be typeset as '\+' and not as '\+'. In version 3 this is now automatically taken care of (with the help of the `\SpecialShortIndex` command).

New in v3

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

> index entries
> Special macros for ~

Such an entry might be produced with the line:

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

2.12 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 favorite 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 `\MaybeStop` macro, in which case a `\Finale` command should appear at the end of your file.

theindex (*env.*)
IndexColumns (*counter*)
Contrary to standard LaTeX, the index is typeset in three columns by default. This is controlled by the LaTeX counter 'IndexColumns' and can therefore

---

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

be changed with a \setcounter declaration. Additionally one doesn't want to start a new page unnecessarily. Therefore the theindex environment is redefined.

\IndexMin (*dimen*) 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.

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-

\IndexPrologue 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 ...}
```

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

\IndexParms start on the same page. Formatting of the index columns (values for \columnssep etc.) is controlled by the \IndexParms macro. It assigns the following values:

\parindent	$= 0.0$pt	\columnsep	$= 15.0$pt
\parskip	$= 0.0$pt plus 1.0pt	\rightskip	$= 15.0$pt
\mathsurround	$= 0.0$pt	\parfillskip	$= -15.0$pt

\@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 encap-
\code sulated by the \usage macro (which produces *italics*). \code encapsulates page or code line numbers in entries generated by parsing the code inside macrocode environments. As usual these commands are user definable.

2.13 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:

\IndexMin	$= 80.0$pt	\MacroTopsep	$= 7.0$pt plus 2.0pt minus 2.0pt
\marginparwidth	$= 126.0$pt	\MacroIndent	$= 18.63434$pt
\marginparpush	$= 0.0$pt	\MacrocodeTopsep	$= 3.0$pt plus 1.2pt minus 1.0pt
\tolerance	$= 1000$		

2.14 Short input of verbatim text pieces

\MakeShortVerb It is awkward to have to type, say, \verb|...| continually when quoting verbatim
\MakeShortVerb* bits (like macro names) in the text, so an abbreviation mechanism is provided. Pick
\DeleteShortVerb a character $\langle c \rangle$—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`.

2.15   Additional bells and whistles

We provide macros for logos such as WEB, $\mathcal{AMS}$-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
\MaybeStop the last part of your document (which presumably exhibits the code). To make
\StopEventually this work you have to place the command `\MaybeStop` at a suitable point in your
file. This macro[11] has one argument in which you put all information you want to

New in v3   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
\Finale is missing the `\MaybeStop` macro saves its argument in a macro called `\Finale` which can afterwards be used to get things back (usually at the very end). Such a scheme makes changes in two places unnecessary.

     Thus you can use this feature to produce a local guide for the TEX users which describes only the usage of macros (most of them won't be interested in your
\maketitle 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
\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_ε distribution,

---

[11]For about 30 years this macro was called `\StopEventually` which was due to a "false friend" misunderstanding. In the German language the word "eventuell" mean roughly "perhaps" which isn't quite the same as "eventually". But given that this is now used for so long and all over the place we can't drop the old name. So it is still there to allow processing all the existing documentation.

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 versions[13] 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 glossary entry 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 than attempting to format it properly and using up valuable characters in the entry with old `makeindex` versions.

Note that in the history listing, the entry is shown with the page number that corresponds to its place in the source, e.g., general changes put at the very beginning of the file will show up with page number "1", change entries placed elsewhere might have different numbers (not necessarily always very useful unless you are careful).

\RecordChanges   To cause the change information to be written out, include `\RecordChanges` in
\PrintChanges   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 `\MaybeStop` 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 `gglo.ist`.

\GlossaryMin (*dimen*)   The `\GlossaryMin`, `\GlossaryPrologue` and `\GlossaryParms` macros and the
\GlossaryPrologue   counter `GlossaryColumns` are analogous to the `\Index`... versions. (The LaTeX
\GlossaryParms   'glossary' mechanism is used for the change entries.)
GlossaryColumns (*counter*)   From time to time, it is necessary to print a \ without being able to use
\bslash

---

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

[13] Before 2.6.

the \verb command because the \catcode s of the symbols are already firmly established. In this instance we can use the command \bslash presupposing, of course, that the actual font in use at this point contains a 'backslash' as a symbol. Note that this definition of \bslash is expandable; it inserts a $\backslash_{12}$. This means that you have to \protect it if it is used in 'moving arguments'.

\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 recognized and \CheckModules invoke special formatting. This can be turned on and off in the .dtx file or the \Module driver file using \CheckModules and \DontCheckModules. If checking for module \AltMacroFont 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 pro-
(*counter*) duce 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 then \AltMacroFont. The default setting is 0, for this documentation it was set to

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

at the beginning of the file.

3 Examples and basic usage summary

3.1 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⟩
   ...
% \MaybeStop{⟨finale code⟩}
   ...
% \begin{macro}{\fred}
% ⟨commentary on macro fred⟩
%␣␣␣␣\begin{macrocode}
⟨code for macro fred⟩
%␣␣␣␣\end{macrocode}
% \end{macro}
```

```
        ...
% \Finale \PrintIndex \PrintChanges
```

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

3.2   Examples

The default setup includes definitions for the `doc` elements "macro" and "environ-
ment". They correspond to the following declarations:

```
\NewDocElement[macrolike = true ,
               idxtype   = ,
               idxgroup  = ,
               printtype =
              ]{Macro}{macro}


\NewDocElement[macrolike = false ,
               idxtype   = env.   ,
               idxgroup  = environments ,
               printtype = \textit{env.}
              ]{Env}{environment}
```

To showcase the new features of `doc` version 3 to some extend, the current
documentation is done by redefining these declarations and also adding a few
additional declarations on top.

For any internal command we document we use `Macro` and put all of them
under the heading "LATEX commands" (note the use of `\actualchar`):

```
\RenewDocElement[macrolike = true ,
                toplevel  = false,
                idxtype   = ,
                idxgroup  = LaTeX comands\actualchar\LaTeX{} commands ,
                printtype =
               ]{Macro}{macro}
```

We only have package environments so we use `Env` for those and group them
as well:

```
\RenewDocElement[macrolike = false ,
                toplevel  = false,
                idxtype   = env.   ,
                idxgroup  = Package environments,
                printtype = \textit{env.}
               ]{Env}{environment}
```

All the interface commands are also grouped together under the label "Package
commands", we use `InterfaceMacro` for them:

```
\NewDocElement[macrolike = true ,
               toplevel  = false,
               idxtype   = ,
               idxgroup  = Package commands,
               printtype =
              ]{InterfaceMacro}{imacro}
```

15

And since we also have a few obsolete interfaces we add yet another category:

```
\NewDocElement[macrolike = true ,
               toplevel  = false,
               idxtype   = ,
               idxgroup  = Package commands (obsolete),
               printtype =
              ]{ObsoleteInterfaceMacro}{omacro}
```

Another type of category are the package keys:

```
\NewDocElement[macrolike = false ,
               toplevel  = false,
               idxtype   = key  ,
               idxgroup  = Package keys ,
               printtype = \textit{key}
              ]{Key}{key}
```

Finally we have TeX counters (with a backslash in front) and LaTeX counters (no backslash) and the two types of LaTeX length registers:

```
\NewDocElement[macrolike = true ,
               toplevel  = false,
               idxtype   =  counter  ,
               idxgroup  = TeX counters\actualchar \protect\TeX{} counters ,
               printtype = \textit{counter}
              ]{TeXCounter}{tcounter}


\NewDocElement[macrolike = false ,
               toplevel  = false,
               idxtype   =  counter  ,
               idxgroup  = LaTeX counters\actualchar \LaTeX{} counters ,
               printtype = \textit{counter}
              ]{LaTeXCounter}{lcounter}


\NewDocElement[macrolike = true ,
               toplevel  = false,
               idxtype   =  skip  ,
               idxgroup  = LaTeX length\actualchar \LaTeX{} length (skip) ,
               printtype = \textit{skip}
              ]{LaTeXSkip}{lskip}


\NewDocElement[macrolike = true ,
               toplevel  = false,
               idxtype   =  dimen  ,
               idxgroup  = LaTeX length\actualchar \LaTeX{} length (dimen) ,
               printtype = \textit{dimen}
              ]{LaTeXDimen}{ldimen}
```

And we modify the appearance of the index: just 2 columns not 3 and all the code-line entries get prefixed with an "ℓ" (for line) so that they can easily be distinguished from page index entries.

```
\renewcommand\code[1]{\mbox{$\ell$-#1}}
\renewcommand\main[1]{\underline{\mbox{$\ell$-#1}}}
\setcounter{IndexColumns}{2}
```

4 Incompatibilities between version 2 and 3

The basic approach when developing version 3 was to provide a very high level of compatibility with version 2 so that nearly all older documents should work out of the box without the need for any adjustments.

But as with any change there are situations where that change can result in some sort of incompatibility, e.g., if a newly introduce command name was already been defined in the user document then there will be a conflict that is nearly impossible to avoid 100%.

As mentioned earlier, `doc` now supports options on several commands and environments and as a result it is necessary to use braces around the argument for `\DescribeMaro` if the "macro to be described" uses private letters such as `@` or `_` as part of its name. That was always the official syntax but in the past you could get away with leaving out the braces more often than you can now.

The old `doc` documentation also claimed that redefinitions of things like `\PrintDescribeMacro` could be done before loading the package (and not only afterwards) and `doc` would in that case not change those commands. As the setup mechanisms are now much more powerful and general such an approach is not really good. So with `doc` version 3 modifications have to be done after the `doc` package got loaded and the last modification will always win.

I'm temped to drop compatibility with LATEX 2.09 (but so far I have left it in).

In the past it was possible to use macros declared with `\outer` in the argument of `\begin{macro}` or `\DoNotIndex` even though `\outer` is not a concept supported in LATEX. This is no longer possible. More exactly, it is no longer possible to prevent them from being indexed (as `\DoNotIndex` can't be used), but you can pass them to the `macro` environment as follows:

    \begin{macro}[outer]{\foo}

if `\foo` is a macro declared with `\outer`. The technical reason for this change is that in the past various other commands, such as `\{` or `\}` did not work properly in these arguments when they where passed as "strings" and not as single macro tokens. But by switching to macro tokens we can't have `\outer` macros because their feature is to be not allowed in arguments. So what happens above when you use `[outer]` is that the argument is read as a string with four character tokens so that it is not recognized as being `\outer`.

5 Old interfaces no longer really needed

Thirty years is a long time in the life of computer programs, so there are a good number of interfaces within `doc` that are really only of historical interest (or when processing equally old sources. We list them here, but in general we suggest that for new documentation they should not be used.

5.1 makeindex bugs

\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 really still 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.

5.2   File transmission issues

In the early days of the Internet file transmission issues have been a serious problem. There was a famous gateway in Rochester, UK that handled the traffic from the European continent to the UK and that consisted of two IBM machines running with different codepages (that had non-reversible differences). As a result "strange" TeX characters got replaced with something else with the result that the files became unusable.

To guard against this problem (or rather to detect it if something got broken in transfer I added code to doc to check a static character table and also to have a very simple checksum feature (counting backslashes).

These days the \CheckSum is of little value (and a lot of pain for the developer) and character scrambling doesn't happen any more so the \CharacterTable is essentially useless. Thus neither should be used in new developments.

\CharacterTable   To overcome some of the problems of sending files over the networks we devel-
\CheckSum oped 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 signs[14] 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 \MaybeStop (which starts looking for backslashes) and the \Finale command. The latter will inform you either that

---

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

18

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 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.

---

6   Introduction to previous releases

Original 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.

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 documentation was born with the development of the TeX program; it was crystallized in Pascal with the WEB system. The advantages of this method are plain to see (it's easy to make comparisons [2]). Since this development, systems similar to WEB have been developed for other programming languages. But for one of the most complicated programming languages (TeX) the documentation has however been neglected. The TeX world seems to be divided between:—

- a couple of "wizards", who produce many lines of completely unreadable code "off the cuff", and

- many users who are amazed that it works just how they want it to do. Or rather, who despair that certain macros refuse to do what is expected of them.

I do not think that the WEB system is *the* reference work; on the contrary, it is a prototype which suffices for the development of programs within the TeX world. It is sufficient, but not totally sufficient.[15]   As a result

---

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

19

of WEB, new programming perspectives have been demonstrated; unfortunately, though, they haven't been developed further for other programming languages.

The method of documentation of TeX macros which I have introduced here should also only be taken as a first sketch. It is designed explicitly to run under LaTeX alone. Not because I was of the opinion that this was the best starting point, but because from this starting point it was the quickest to develop.[16] As a result of this design decision, I had to move away from the concept of modularization; this was certainly a step backward.

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

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.

Preface to version 1.7 (from around 1992)

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 recognized 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` and that added after version 1.5k of `doc`. 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

---

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

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.);

- 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 `docstrip`ped `.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.

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` user will be grateful for.

7 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.

```
21 ⟨∗package⟩
```

Under LaTeX 2_ε the test to avoid reading `doc` in twice is normally unnecessary. It was kept to only to stay compatible with LaTeX209 styles that `\input doc` directly.

```
22 \@ifundefined{macro@cnt}{}{\endinput}
```

\fileversion As you can see I used macros like `\fileversion` to denote the version number
\filedate and the date. They are defined at the very beginning of the package file (without
\docdate 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 two alternative comment signs. Because all sensible signs are already occupied, we will choose some that can only be entered indirectly:

```
23 \catcode'\^^A=14
24 \catcode'\^^X=14
```

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

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

7.1 Keys supported by doc

In the past this used kvoptions but this will be replaced by using l3keys at some point in the future. Right now this is only a lightweight shift—the code could and should be altered further.

TODO: *cleanup replacement of kvoptions*

Some keys are available as options for use in `\usepackage` some are for the generated item API's:

```
26 \DeclareKeys
27   {
28     noprint         .if    = {doc@noprint},
29     noindex         .if    = {doc@noindex},
30     hyperref        .if    = {doc@hyperref},
31     nohyperref      .ifnot = {doc@hyperref},
32     multicol        .if    = {doc@multicol},
33     nomulticol      .ifnot = {doc@multicol},
34     debugshow       .if    = {doc@debugshow},
35     reportchangedates .if  = {doc@reportchangedates},
36     toplevel        .if    = {doc@toplevel},
37     notoplevel      .ifnot = {doc@toplevel},
38     macrolike       .if    = {doc@macrolike},
```

```
39    envlike           .ifnot = {doc@macrolike},
40    idxtype           .store = \doc@idxtype,
41    idxgroup          .store = \doc@idxgroup,
42    printtype         .store = \doc@printtype,
43    outer             .if    = {doc@outer},
44  }
```

Setting these options to true initially.

```
45 \doc@hyperreftrue
46 \doc@multicoltrue
47 \doc@topleveltrue
```

7.2 Processing the package keys

```
48 \ProcessKeyOptions
```

`\ifscan@allowed`
`\scan@allowedtrue`
`\scan@allowedfalse`

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

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

`\SetupDoc` We need to save the default value for some options because doc elements can locally set them.

TODO: *Use 2e interface for* `\keys_set:nn` *when available*

```
50 \def\SetupDoc#1{%
51   \csname keys_set:nn\endcsname{doc}{#1}%
52     \edef\doc@noprintdefault{\ifdoc@noprint true\else false\fi}%
53   \ifdoc@noindex
```

If we do not index by default then we should also turn off `\scan@allowed`.

```
54     \def\doc@noindexdefault{true}%
55     \scan@allowedfalse
56   \else
57     \def\doc@noindexdefault{false}%
58   \fi
59 }
```

```
60 \SetupDoc{}              % just save the default values
```

7.3 Macros surrounding the 'definition parts'

macrocode (*env.*) 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.

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

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

```
62   \frenchspacing \@vobeyspaces
```

23

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.

```
63     \xmacro@code}
```

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

```
64 \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.

```
65     \topsep \MacrocodeTopsep
```

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

```
66     \@beginparpenalty \predisplaypenalty
```

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

```
67     \if@inlabel\leavevmode\fi
68     \trivlist \parskip \z@ \item[]%
```

The \item command sets the \@labels box but that box is never typeset (as \everypar that normally does this gets redefined later). That is normally not an issue, but produces a problem when typesetting in mixed directions, (e.g., in Japanese), so we explicitly clear it for that use case.

```
69     \global\setbox\@labels\box\voidb@x
```

Additionally, everything should be set in typewriter font. Some people might prefer it somewhat differently; because of this the font choice is macro-driven.[17]

```
70     \macro@font
```

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

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

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

```
73     \blank@linefalse \def\par{\ifblank@line
74                               \leavevmode\fi
75                               \blank@linetrue\@@par
76                               \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.

```
77     \obeylines
78     \let\do\do@noligs \verbatim@nolig@list
79     \let\do\@makeother \dospecials
```

---

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

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`.

```
80      \global\@newlistfalse
81      \global\@minipagefalse
82      \ifcodeline@index
83        \everypar{\global\advance\c@CodelineNo\@ne
84                  \llap{\theCodelineNo\ \hskip\@totalleftmargin}%
85                  \check@module}%
86      \else \everypar{\check@module}%
87      \fi
```

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

```
88      \init@crossref}
```

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

```
89 \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`.

```
90 \def\endmacrocode{%
91                \ifpm@module \endgroup \pm@modulefalse \fi
92                \everypar{}%
93                \global\@inlabelfalse
94                \endtrivlist
```

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

```
95                \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.

```
96 \@ifundefined{MacroFont}{%
97   \if@compatibility
```

Despite the above statement we will call `\small` first if somebody is using a LaTeX2.09 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 . . . :-)

```
98      \def\MacroFont{\small
99                \usefont\encodingdefault
100                      \ttdefault
101                      \mddefault
102                      \shapedefault
103                }%
104    \else
105      \def\MacroFont{\fontencoding\encodingdefault
```

25

```
106                    \fontfamily\ttdefault
107                    \fontseries\mddefault
108                    \fontshape\shapedefault
109                    \small}%
110   \fi
111   }{}
```

\AltMacroFont Although most of the macro code is set in \MacroFont we want to be able to
\macro@font   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.

```
112 \@ifundefined{AltMacroFont}{%
113   \if@compatibility
```

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

```
114     \def\AltMacroFont{\small
115                    \usefont\encodingdefault
116                          \ttdefault
117                          \mddefault
118                          \sldefault
119                    }%
120   \else
121     \def\AltMacroFont{\fontencoding\encodingdefault
122                    \fontfamily\ttdefault
123                    \fontseries\mddefault
124                    \fontshape\sldefault
125                    \small
126                    }%
127   \fi
128   }{}
```

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

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

\check@module This is inserted by \everypar at the start of each macrocode line to check whether
\ifpm@module   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.

```
130 \def\check@module{%
131   \ifcheck@modules
132     \ifpm@module \endgroup \pm@modulefalse \fi
133     \expandafter\futurelet\expandafter\next\expandafter\ch@percent
134   \fi}
135 \newif\ifpm@module
```

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

```
136 \def\DontCheckModules{\check@modulesfalse}
137 \def\CheckModules{\check@modulestrue}
138 \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`.

```
139 \def\ch@percent{%
140   \if \percentchar\next
141     \expandafter\check@angle
142   \fi}
```

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

```
143 \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 LaTeX 2_ε < is active so we have to be a bit careful.

```
144 \begingroup
145 \catcode`\<\active
146 \gdef\ch@angle{\ifx<\next
147     \expandafter\ch@plus@etc
148   \else \percentchar \fi}
```

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

```
149 \gdef\ch@plus@etc<{\futurelet\next\check@plus@etc}
150 \gdef\check@plus@etc{%
151   \if +\next
152     \let\next\pm@module
153   \else\if -\next
154     \let\next\pm@module
155   \else\if *\next
156     \let\next\star@module
157   \else\if /\next
158     \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 >.

```
159   \else\ifx <\next
160     \percentchar
161   \else
162     \let\next\pm@module
163   \fi\fi\fi\fi\fi
164   \next}
165 \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.

```
166 \begingroup
167 \catcode'\~=\active
168 \lccode'\~='\>
169 \lowercase{\gdef\pm@module#1~}{\pm@moduletrue
170    \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.

```
171    \advance\guard@level\@ne
172    \ifnum\guard@level>\c@StandardModuleDepth\AltMacroFont\fi
173 }
```

\star@module If the start or end of a module *block* is indicated, after setting the guard we have
\slash@module 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.

```
174 \lowercase{\gdef\star@module#1~}{%
175   \Module{#1}%
176   \global \advance \guard@level\@ne
177   \ifnum \guard@level>\c@StandardModuleDepth
178     \global\let\macro@font=\AltMacroFont \macro@font
179   \fi}
180 \catcode'\>=\active
181 \gdef\slash@module#1>{%
182   \Module{#1}%
183   \global \advance \guard@level\m@ne
184   \ifnum \guard@level=\c@StandardModuleDepth
185     \global\let\macro@font\MacroFont  \macro@font
186   \fi
187 }
188 \endgroup
```

StandardModuleDepth (*counter*) 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.

```
189 \newcounter{StandardModuleDepth}
```

\guard@level (*counter*) We need a counter to keep track of the guard nesting.

```
190 \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.

```
191 \@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 `oldlfnt` or not. I expect there's a better way. . .

```
192     \def\Module#1{\mod@math@codes$\langle\mathsf{#1}\rangle$}
193   }{}
```

\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.

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

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

```
197 \newskip\MacrocodeTopsep \MacrocodeTopsep = 3pt plus 1.2pt minus 1pt
198 \newdimen\MacroIndent
199 \settowidth\MacroIndent{\rmfamily\scriptsize 00\ }
```

macrocode* (*env.*) Just as in the verbatim environment, there is also a 'star' variant of the macrocode
\endmacrocode* 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)[18] 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*}.

```
200 \@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*.

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

---

[18]Karl 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.

\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.

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

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

```
204 \catcode`\{=12␣\catcode`\}=12
205 \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.

```
206 |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.

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

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

```
209 |endgroup
```

7.4   Macros for the 'documentation parts'

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.

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

212 \setlength\marginparsep{\labelsep}
```

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

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

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

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

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

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

30

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

```
217 \@namedef{verbatim*}{\@beginparpenalty \predisplaypenalty \@verbatim
```

```
218                 \@setupverbvisiblespace
219                 \MacroFont \@vobeyspaces \@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.

```
220 \def\@verbatim{\trivlist \item\relax
221   \if@minipage\else\vskip\parskip\fi
222   \leftskip\@totalleftmargin\rightskip\z@
223   \parindent\z@\parfillskip\@flushglue\parskip\z@
224   \language\l@nohyphenation
225   \@@par
226   \@tempswafalse
```

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

```
227 \def\par{%
228   \if@tempswa
229     \leavevmode \null \@@par\penalty\interlinepenalty
230   \else
231     \@tempswatrue
232     \ifhmode\@@par\penalty\interlinepenalty\fi
233   \fi
```

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

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

The rest is again copied literally from latex.tex (less "").

```
235   \let\do\@makeother \dospecials
236   \obeylines \verbatim@font \@noligs
237   \everypar \expandafter{\the\everypar \unpenalty}%
238 }
```

\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.

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

In the early versions of the package it also redefined \verb because that didn't include the useful test for "newline" in the verbatim text. This is nowadays part of LaTeX so we do not redefine it any longer (the original code is still kept in the file after \endinput to keep the long history intact).

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

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

\MacroTopsep (*skip*) Here is the default value for the `\MacroTopsep` parameter used above.

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

7.5   Formatting the margin

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

7.6   Creating index entries by scanning 'macrocode'

The following macros ensure that index entries are created for each occurrence of a TeX-like command (something starting with '\') providing indexing has been turned on with `\PageIndex` or `\CodelineIndex`. With the default definitions of `\specialMainMacroIndex`, 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 TeX's escape character to introduce command names at places where \ has to belong to some other category. Therefore, we may also need to recognize | as the introducer for a command when setting the text inside the `macrocode` environment. Other users may have the need to make similar reassignments for their macros.

\SpecialEscapechar   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` 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`.

```
244 \begingroup
245 \catcode'\~\active
246 \gdef\SpecialEscapechar#1{%
247     \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.

```
248     \uccode'\~'#1%
```

32

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).

```
249        \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.

```
250        \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.

```
251   \endgroup}
252 \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 $\backslash_{12}$ in case 3b) above. This is done because we have to ensure that '\' has \catcode 13 within the macrocode environment. Otherwise the delimiter for the argument of \xmacro@code would not be found (parameter matching depends on \catcodes).

  Therefore we first re-\catcode some characters.

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

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

```
254 |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.

```
255      |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.

```
256      |catcode|expandafter'|special@escape@char|active
257      |expandafter|let|active@escape@char|scan@macro}
258 |endgroup
```

If there is no special escape character, i.e. if `\SpecialEscapechar` is `\\`, the second last line will overwrite the previous definition of $\backslash_{13}$. In this way all tasks are fulfilled.

For happy documenting we give default values to `\special@escape@char` and `\active@escape@char` with the following line:

```
259 \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.

```
260 \@ifundefined{MakePrivateLetters}
261     {\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 '\'.

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

7.7 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 TeX-like command which is met whilst setting (in verbatim) the contents of `macrocode` environments.

```
263 \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 5.2 for details).

```
264     \special@escape@char
265     \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.

```
266     \ifscan@allowed
```

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

```
267     \let\macro@namepart\@empty
268     \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'.

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

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

```
270     \next}
```

\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).

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

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

```
272 \def\EnableCrossrefs{\@bsphack\scan@allowedtrue
273                    \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.

```
274 \def\macro@switch{\ifcat\noexpand\next a%
275     \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.

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

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

```
277     \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 charac-
ter mustn't be stored with a special category code or exclusion from the index
won't work, so we we use \string to normalize it the same way it is done in
\DoNotIndex, i.e. everything ends up catcode 12 except for the space character.

```
278 \def\short@macro#1{%
279     \edef\macro@namepart{\string#1}%
```

Any indexing is then delegated to \maybe@index@short@macro. Depending on
the actual character seen, this macro has to do different things, which is why we
keep it separate from \maybe@index@macro to avoid the special tests in the more
common case of a multi-letter macro name.

```
280     \maybe@index@short@macro\macro@namepart
```

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).

```
281     \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.

```
282     \scan@allowedtrue }
```

\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`).

```
283 \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'.

```
284     \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'.

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

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

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

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

```
288     \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`.

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

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

```
291 %   \ifnot@excluded
292 %     \edef\@tempa{\noexpand\SpecialIndex{\bslash\macro@namepart}}%
293 %     \@tempa  \fi
294   \maybe@index@macro \macro@namepart
295 }
```

7.8 The index exclude list[20]

The following part of the code is a new implementation using the LaTeX3 programming layer as the constructs and types provided therein are making programming much easier. Over time I will probably replace the rest of that `doc` code too.

```
296 \ExplSyntaxOn
```

`\l__doc_donotindex_seq` Local sequence that holds names (as strings) of commands that should not be indexed. Within a `doc` element environment that element is placed into the sequence so that it isn't unnecessarily index within that part of the code. As the sequence is local it will revert this setting at the end of the environment so that the command is indexed elsewhere (unless it is generally disabled from indexing.

`\g__doc_idxtype_prop` Global property list that holds for all commands that are special `doc` elements

---

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

[20]Info: the incomplete commentary on `\DoNotIndex` and the macros it calls was written by Dave Love.

the type of the element. The key is the command name without backslash and the value is doc element type identifier, e.g., \texttt{Length} for length registers if that type has been set up. doc only indexes commands, that is things starting with an escape character, i.e., a backslash (by default). doc elements that do not start with an escape character, e.g., environments are not identified when parsing code so that aren't indexed automatically inside. Thus for them there is no point in keeping them in that property list.

\doc_dont_index:n
\DoNotIndex

Takes a clist of commands (with backslash) as input and exclude all of them from the index. User facing we make this available as \DoNotIndex.

\ShowIndexingState

Displays the current list of the exclude index list in a fairly low-level form.

\RecordIndexType

This command takes two arguments: a command (with escape char) and its type (i.e., first mandatory argument of a \NewDocElement declaration. If #1 should not be included from the index, then the data is used to record that this command is of this type. The information is then used to generate appropriate index entries. Obviously, index entries generated earlier will be listing the wrong type. Therefore this information is also placed into the .aux file so that it will be available at the beginning of further runs.

This command is internally executed as part of any doc element environment so it only needs to be explicitly given if for some reason a command with a special type has no corresponding environment.

\l__doc_donotindex_seq
\g__doc_idxtype_prop

Declarations.

```
297 \seq_new:N \l__doc_donotindex_seq
298 \prop_new:N \g__doc_idxtype_prop
```

__doc_trace:x

A helper for tracing. . .

```
299 \cs_new:Npn\__doc_trace:x {
300   \legacy_if:nTF{ doc@debugshow }{ \iow_term:x } { \use_none:n }
301 }
```

\doc_dont_index:n
__doc_dont_index:n
__doc_dont_index_aux:n

Parses the argument a clist of commands with \MakePrivateLetters in force (so that special characters are recognized as being part of command names) and puts each command without is backslash as a string into the sequence \l__doc_donotindex_seq.

```
302 \cs_new:Npn \doc_dont_index:n {
303   \group_begin:
304     \MakePrivateLetters
305     \__doc_dont_index:n
306 }
```

```
307 \cs_new:Npn \__doc_dont_index:n #1 {
308   \group_end:
309   \__doc_trace:x{Disable~ indexing~ for~ '\tl_to_str:n{#1}' }
```

Adding the commands to the \l__doc_donotindex_seq sequence is done by mapping the function __doc_dont_index_aux:n on each element in the clist.

```
310   \clist_map_function:nN {#1} \__doc_dont_index_aux:n
311 }
```

We record each command by using its name as a string. This means more tokens in the sequence but it allows to compare names not "action" which is important

as different commands may have the same meaning (e.g., they may not be defined at all),

```
312 \cs_new:Npn \__doc_dont_index_aux:n #1 {
313   \seq_put_right:Nx \l__doc_donotindex_seq {\expandafter\@gobble \string#1}
314 }
```

\DoNotIndex The document-level interface

```
315 \cs_set_eq:NN \DoNotIndex \doc_dont_index:n
```

\ShowIndexingState Some tracing information that may be helpful.

```
316 \def \ShowIndexingState {
317   \__doc_trace:x{Show~ doc~ indexing~ state:}
318   \seq_show:N  \l__doc_donotindex_seq
319   \prop_show:N \g__doc_idxtype_prop
320 }
```

__doc_idxtype_put:Nn **TODO:** *Change name of interface command!*
\RecordIndexType
\RecordIndexTypeAux This is the internal form for \RecordIndexType. The first argument is turned into a string and the rest of the processing is then done by __doc_idxtype_put:nn

```
321 \cs_new:Npn \__doc_idxtype_put:Nn #1#2 {
322   \exp_args:Nx \__doc_idxtype_put:nn { \cs_to_str:N #1 }{#2}
```

We also make an entry in the .aux file so that this declaration becomes immediately available in the next run. However, for this we aren't reusing __doc_idxtype_put:N (a.k.a. \RecordIndexType) since that would result in doubling such lines each time the document is run. Instead we use \RecordIndexTypeAux which is only updating the data structures without writing to the .aux file.

```
323   \protected@write\@auxout{}
324     {\string\RecordIndexTypeAux {\string#1 }{#2} }
325 }
```

When we execute this code from the .aux we better not generate a new line in the .aux. Otherwise those would cumulate over time.

```
326 \cs_new:Npn \RecordIndexTypeAux #1#2 {
327   \exp_args:Nx \__doc_idxtype_put:nn { \cs_to_str:N #1 }{#2}
328 }
```

Similarly, when the .aux is read at the end of the run we should disable that command to avoid unnecessary processing.

```
329 \AtEndDocument{
330   \cs_set_eq:NN \RecordIndexTypeAux \use_none:nn
331 }
```

Finally, we provide the user-level interface

```
332 \cs_set_eq:NN \RecordIndexType \__doc_idxtype_put:Nn
```

__doc_idxtype_put_scan:nn When we want to record an index type for a scanned name we can't turn that into a csname and then call __doc_idxtype_put:Nn because turning it into a csname may change the meaning of the name from "undefined" to \relax. Got bitten by that when processing the kernel sources containing \@undefined within the code: suddenly that wasn't undefined any longer. So here is another version that works only on characters as input. As we don't know whether or not they are proper strings already we first make sure that this is the case.

```
333 \cs_new:Npn \__doc_idxtype_put_scan:nn #1#2 {
334   \exp_args:Nf \__doc_idxtype_put:nn { \tl_to_str:n {#1} }{#2}
```

In this case we also have to append a backslash when writing to the .aux file.

```
335   \protected@write\@auxout{}
336     {\string\RecordIndexTypeAux {\bslash #1 }{#2} }
337 }
```

__doc_idxtype_put_scan:on And here is the one we really need since the characters are stored in some macro.

```
338 \cs_generate_variant:Nn \__doc_idxtype_put_scan:nn {o}
```

\record@index@type@save And here is the interface to the rest of the code:

```
339 \cs_set_eq:NN \record@index@type@save \__doc_idxtype_put_scan:on
```

__doc_idxtype_put:nn This internal command takes two arguments: a command name as string (no backslash) and its type (i.e., first mandatory argument of a \NewDocElement declaration. If #1 is not in \l__doc_donotindex_seq it will add this data to the property list \g__doc_idxtype_prop using #1 as key and #2 as its value. If the key already exist its value will be overwritten. If the command is later marked for exclusion from the index the property list setting remains unchanged but as long as no index is produced for the command it will not be consulted.

Note: the command assumes that #1 is already in string form

```
340 \cs_new:Npn \__doc_idxtype_put:nn #1#2 {
```

No mystery here: if the command is not marked for exclusion from the index add the property. The extra \tl_to_str:n is a safety measure in case the input wasn't already in that form (should only be the case with broken input but . . .)

```
341   \exp_args:NNf
342   \seq_if_in:NnTF \l__doc_donotindex_seq {\tl_to_str:n{#1}}
```

Some tracing info . . .

```
343     {
344       \__doc_trace:x{Not~ recording~ index~ type~ for~ '\bslash #1' }
345     }
346     {
347       \__doc_trace:x{Recording~ index~ type~ for~ '\bslash #1' ~ as~ #2 }
```

Stick the data into the property list:

```
348       \prop_gput:Nnn \g__doc_idxtype_prop {#1}{#2}
349     }
350 }
```

\exp_args:co A helper: construct a function and call it with its first argument expanded once:

```
351 \cs_new:Npn \exp_args:co #1#2
352   { \cs:w #1 \exp_after:wN \cs_end:\exp_after:wN {#2} }
```

\tl_to_str:o Another helper: take some token list variable, expand it and turn it into a string.

```
353 \cs_generate_variant:Nn \tl_to_str:n {o}
```

\maybe@index@macro

This takes a macro name (without backslash) as parsed within a macrocode environment and checks if it should get indexed (i.e., is not on the exclude list) and if so how (i.e., gets it index type property and makes the right choice depending on that.

`\maybe@index@macro`
`\__doc_maybe_index:o` We first make sure that the argument is really a string (so that we have a defined situation) and then pass it on to `\__doc_maybe_index_aux:nN` to do the work. The second argument defines the indexing operation `\SpecialIndex` for multi-letter macros and below `\SpecialShortIndex` for single character macros.

```
354 \cs_new:Npn \__doc_maybe_index:o #1 {
355   \exp_args:Nf \__doc_maybe_index_aux:nN { \tl_to_str:o {#1} }
356                                         \SpecialIndex
357 }
```

And here is what we call it in the older non-expl3 code:

```
358 \cs_set_eq:NN \maybe@index@macro \__doc_maybe_index:o
```

`\maybe@index@short@macro`
`\__doc_maybe_index_short:o` Single character macros are handled similarly but there the indexing is done by `\SpecialShortIndex` and it is simpler because we know that the argument contains a string token not letters.

```
359 \cs_new:Npn \__doc_maybe_index_short:o #1 {
360   \exp_args:No \__doc_maybe_index_aux:nN #1
361                                         \SpecialShortIndex
362 }
363 \cs_set_eq:NN \maybe@index@short@macro \__doc_maybe_index_short:o
```

`\__doc_maybe_index_aux:nN` Take a string (representing a macro without backslash) and make the right choices with respect to indexing.

```
364 \cs_new:Npn \__doc_maybe_index_aux:nN #1#2 {
```

A bit of tracing:

```
365   \__doc_trace:x{Searching~ for~ '\bslash #1'}
```

If the name is on the exclude list do nothing.

```
366   \seq_if_in:NnTF \l__doc_donotindex_seq {#1}

367     {
368       \__doc_trace:x{Not~ indexing~ '\bslash #1' }
369     }
```

Otherwise check if this name has an index type property attached to it.

```
370     {
371       \prop_get:NnNTF \g__doc_idxtype_prop {#1} \l__doc_idxtype_tl
```

If so construct and execute `\Code`⟨*idxtype*⟩`Index`[21] which is done inside `\__doc_maybe_index_aux`

```
372       {
373         \exp_args:Ncno \__doc_maybe_index_aux:Nnn
374             { Code \tl_use:N \l__doc_idxtype_tl Index }
375             {code} {\bslash #1}
376       }
```

Otherwise execute `\SpecialIndex` which is a short form for `\CodeMacroIndex{code}` or execute `\SpecialShortIndex` which deals with some special cases for single letter macros.

```
377       {
378         \__doc_trace:x{Indexing~ '\bslash #1'\space (\string #2)}
379         \exp_args:No #2 {\bslash #1}
380       }
381     }
382 }
```

---

[21] I guess this should really be an internal name not a user-level one.

\SpecialShortIndex **TODO:** *to be documented; also needs cleaning up as it is a mix of old and new right now*

```
383 \cs_new:Npn \SpecialShortIndex #1 {
384     \@SpecialIndexHelper@ #1\@nil
385   \@bsphack
386   \ifdoc@noindex \else
387     \str_case_e:nnF {\@gtempa }
388         {
389             {\cs_to_str:N \^^M } {\def\reserved@a{ \string \space \actualchar }
390                                   \def\reserved@b { \space }
391                                   \let\reserved@c \@empty                          }
```

With the fix for $\backslash_\sqcup$ we now have to look for a real space to handle that command sequence.

```
392             { ~ }                {\def\reserved@a{ \string \space \actualchar }
393                                   \def\reserved@b { \space }
394                                   \let\reserved@c \@empty                          }
395             {\c_left_brace_str} { \def\reserved@a{ \bgroup \actualchar }
396                                   \def\reserved@b { \c_left_brace_str }
397                                   \def\reserved@c { \noexpand\iffalse
398                                                    \c_right_brace_str
399                                                    \noexpand\fi }                 }
400             {\c_right_brace_str} { \def\reserved@a{ \egroup \actualchar
401                                                    \noexpand\iffalse
402                                                     \c_left_brace_str
403                                                    \noexpand\fi }
404                                   \def\reserved@b { \c_right_brace_str }
405                                   \let\reserved@c \@empty                          }
```

The case of \verbatimchar is tricky. We can't stick it into the normal \verb because we would then get something like \verb+\++ which would comes out as "\+" instead of \+. So we use the \verb to only generate the backslash and then use \texttt on the \verbatimchar itself. However, that is not enough if we are unlucky and somebody (like Will :-)) has used something like & with a special catcode for the \verbatimchar. We therefore also apply \string to it when we read it back.

```
406             {\verbatimchar}  { \def\reserved@a{ \quotechar\verbatimchar
407                                                    \actualchar }
408                                   \let\reserved@b \@empty
409                                   \def\reserved@c
410                                       { \string\texttt{\string\string\verbatimchar} } }
411         }
412         { \def\reserved@a {\quotechar \@gtempa \actualchar }
413           \def\reserved@b {\quotechar \@gtempa  }
414           \let\reserved@c \@empty                                }
415     \special@index {
416     \reserved@a
417     \string\verb
418     \quotechar *\verbatimchar \quotechar \bslash
419     \reserved@b
420     \verbatimchar
421     \reserved@c
422     \encapchar code}
423   \fi
```

41

```
424   \@esphack
425 }
```

`\__doc_maybe_index_aux:Nnn` Execute the function passed on as first argument taking argument 2 and 3 as input.

```
426 \cs_new:Npn \__doc_maybe_index_aux:Nnn #1#2#3 {
```

We have to be a little careful: as that function name is constructed it may not actually exist (as constructions generate \relax in TeX in that case). In that case we raise an error, otherwise we execute (with a little bit of tracing info):

```
427       \cs_if_exist:NTF #1
428         {
429           \__doc_trace:x{Indexing~ '#3'\space as~
430                         \tl_use:N \l__doc_idxtype_tl }
431         #1{#2}{#3}
432         }
433         {
434           \PackageError{doc}{Doc~ element~
435             '\tl_use:N \l__doc_idxtype_tl'~ unknown}%
436
437           {When~ using~ '\string\RecordIndexType'~ the~ type~ must~
438            be~ known~\MessageBreak
439            to~ the~ system,~ i.e.,~ declared~ via~
440            '\string\NewDocElement'\MessageBreak
441            before~ it~ can~ be~ used~ in~ indexing.}
442         }
443 }
```

Back to old style coding ...

```
444 \ExplSyntaxOff
```

7.9   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 favorite such program.

To assist the reader in locating items in the index, all such entries are sorted alphabetically *ignoring* the initial '\'; this is achieved by issuing an \index command which contains the 'actual' operator for makeindex. The default value for the latter operator is '@', but the latter character is so popular in LaTeX 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 TeX 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.

```
445 ⟨/package⟩
446 ⟨+gind | gglo⟩%% This is a MAKEINDEX style file which should be used to
```

```
447 ⟨+gind⟩%% generate the formatted index for use with the doc
448 ⟨+gglo⟩%% generate the formatted change history for use with the doc
449 ⟨+gind | gglo⟩%% package. The TeX commands used below are defined in
450 ⟨+gind | gglo⟩%% doc.sty.  The commands for MAKEINDEX like 'level'
451 ⟨+gind | gglo⟩%% 'item_x1' are described in '' Makeindex, A General
452 ⟨+gind | gglo⟩%% Purpose, Formatter-Independent Index Processor'' by
453 ⟨+gind | gglo⟩%% Pehong Chen.
454 ⟨+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
\levelchar file.

```
455 ⟨∗package⟩
456 \@ifundefined{actualchar}{\def\actualchar{=}}{}
457 \@ifundefined{quotechar}{\def\quotechar{!}}{}
458 \@ifundefined{levelchar}{\def\levelchar{>}}{}
459 ⟨/package⟩
460 ⟨+gind | gglo⟩actual '='
461 ⟨+gind | gglo⟩quote '!'
462 ⟨+gind | gglo⟩level '>'
463 ⟨∗package⟩
```

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

```
464 \@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.

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

\@SpecialIndexHelper@ **TODO:** *doc or drop*

```
466 \begingroup
467  \catcode'\|=0
468  \catcode'\\=12
469 |gdef|@SpecialIndexHelper@#1#2|@nil{%
470   |if |noexpand#1\%
471     |gdef|@gtempa{#2}%
472   |else
473     |begingroup
474       |escapechar|m@ne
475       |expandafter|gdef|expandafter|@gtempa|expandafter{|string#1#2}%
476     |endgroup
477   |fi}
478 |endgroup
```

\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.

```
479 \def\SortIndex#1#2{%
```

```
480    \ifdoc@noindex\else
481      \index{#1\actualchar#2}%
482    \fi
483 }
```

\LeftBraceIndex These two macros fix the problems with makeindex. Note the 'hack' with
\RightBraceIndex \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.

```
484 \@ifundefined{LeftBraceIndex}{\def\LeftBraceIndex{%
485    \special@index{\bgroup\actualchar
486                   \string\verb% % to fool emacs highlighting
487                   \quotechar*\verbatimchar
488                   \quotechar\bslash{\verbatimchar\string\iffalse}\string\fi}}}{}
489
490 \@ifundefined{RightBraceIndex}{\def\RightBraceIndex{%
491 \special@index{\egroup\actualchar\string\iffalse{\string\fi
492          \string\verb% % to fool emacs highlighting
493          \quotechar*\verbatimchar\quotechar\bslash}\verbatimchar}}}{}
```

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

```
494 \@ifundefined{PercentIndex}
495    {\def\PercentIndex{\it@is@a\percentchar}}{}
```

\OldMakeindex Here is one solution for the percent bug in makeindex. The macro \percentchar
\percentchar denotes a %$_{12}$. Calling this from a package or the driver file sets things up appro-
priately.

```
496 \def\OldMakeindex{\def\PercentIndex{%
497    \special@index{\quotechar\percentchar\actualchar
498          \string\verb% % to fool emacs highlighting
499          \quotechar*\verbatimchar\quotechar\bslash
500          \percentchar\percentchar\verbatimchar}}}
501 {\catcode`\%=12 \gdef\percentchar{%}}
```

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

```
502 \def\it@is@a#1{\special@index{\quotechar #1\actualchar
503                   \string\verb% % to fool emacs highlighting
504                   \quotechar*\verbatimchar
505                   \quotechar\bslash\quotechar#1\verbatimchar}}
```

7.10   Redefining the index environment

\IndexMin (*dimen*) If multicol is in use, when the index is started we compute the remaining space on
IndexColumns (*counter*) 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.

44

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

theindex (*env.*) 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.

```
508 \ifdoc@multicol
```

```
509     \RequirePackage{multicol}
```

```
510     \renewenvironment{theindex}
511         {\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.

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

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

```
513         {\end{multicols}}
```

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

```
514 \else
515     \def\theindex{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi
516         \columnseprule \z@  \columnsep 35\p@
517         \twocolumn[\index@prologue]%
518         \IndexParms \let\item\@idxitem \ignorespaces}
519     \def\endtheindex{\if@restonecol\onecolumn\else\clearpage\fi}
520 \fi
```

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

```
521 ⟨/package⟩
522 ⟨+gind⟩preamble
523 ⟨+gind⟩"\n \\begin{theindex} \n \\makeatletter\\scan@allowedfalse\n"
524 ⟨+gind⟩postamble
525 ⟨+gind⟩"\n\n \\end{theindex}\n"
526 ⟨*package⟩
```

\IndexPrologue The \IndexPrologue macro is used to place a short message into the document
\index@prologue 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.

```
527 \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.

```
528 \@ifundefined{index@prologue}
529         {\def\index@prologue{\section*{Index}%
530                 \markboth{Index}{Index}%
531                 Numbers written in italic refer to the page
532                 where the corresponding entry is described;
533                 numbers underlined refer to the
```

```
534             \ifcodeline@index
535               code line of the
536             \fi
537             definition; numbers in roman refer to the
538             \ifcodeline@index
539               code lines
540             \else
541               pages
542             \fi
543             where the entry is used.
544             }}{}
```

\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.

```
545 \@ifundefined{IndexParms}
546    {\def\IndexParms{%
547        \parindent \z@
548        \columnsep 15pt
549        \parskip 0pt plus 1pt
550        \rightskip 15pt
551        \mathsurround \z@
552        \parfillskip=-15pt
553         \small
```

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

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

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

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

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

```
556        \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.

```
557        \def\indexspace{\par\vspace{10pt plus 2pt minus 3pt}}%
558        }}{}
```

\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.

```
559 \def\efill{\hfill\nopagebreak}%
560 ⟨/package⟩
561 ⟨+gind | gglo⟩item_x1    "\\efill \n \\subitem "
562 ⟨+gglo⟩item_x2    "\\ "
563 ⟨+gind⟩item_x2    "\\efill \n \\subsubitem "
564 ⟨*package⟩
```

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

```
565 \def\pfill{\unskip~%
566         \leaders\hbox to.6em{\hss .\hss}\hfill
567         \penalty500\strut\nobreak
568         \leaders\hbox to.6em{\hss .\hss}\hfil
569         ~\ignorespaces}%
570 ⟨/package⟩
571 ⟨+gind | gglo⟩delim_0   "\\pfill "
572 ⟨+gind | gglo⟩delim_1   "\\pfill "
573 ⟨+gind | gglo⟩delim_2   "\\pfill "
574 ⟨∗package⟩
```

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

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

\main The *defining* entry for a macro name is flagged with the string `|main`[22] 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.

```
576 \@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*.

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

\code The `\code` macro is used to indicate index entries to code lines that aren't main entries. By default we do nothing special with them the usage of a macro.

```
578 \@ifundefined{code}{\def\code#1{#1}}{}
```

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

```
579 \def\PrintIndex{\@input@{\jobname.ind}%
580                  \global\let\PrintIndex\@empty}
```

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:

```
581 ⟨/package⟩
582 ⟨+gind, gglo⟩% The next lines will produce some warnings when
583 ⟨+gind, gglo⟩% running Makeindex as they try to cover two different
```

---

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

```
584 ⟨+gind, gglo⟩% versions of the program:
585 ⟨+gind, gglo⟩lethead_prefix   "{\\bfseries\\hfil "
586 ⟨+gind, gglo⟩lethead_suffix   "\\hfil}\\nopagebreak\n"
587 ⟨+gind⟩lethead_flag        1
588 ⟨+gglo⟩lethead_flag        0
```

This works for newer ones:

```
589 ⟨+gind, gglo⟩heading_prefix   "{\\bfseries\\hfil "
590 ⟨+gind, gglo⟩heading_suffix   "\\hfil}\\nopagebreak\n"
591 ⟨+gind⟩headings_flag        1
592 ⟨+gglo⟩headings_flag        0
593 ⟨∗package⟩
```

7.11   Dealing with the change history[23]

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 there-
fore uses the normal \glossaryentry commands.[24] 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.

   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.

```
594 \def\changes{\@bsphack\begingroup\@sanitize
595     \catcode'\\\z@ \catcode'\ 10 \MakePercentIgnore
596     \changes@}
597 \def\changes@#1#2#3{%
598     \protected@edef\@tempa{\noexpand\glossary{#1%
```

If asked for we also show the date of in the change log (after the version).

```
599                 \ifdoc@reportchangedates
600                  \space -- #2\fi
601                  \levelchar
```

---

[23]The whole section was proposed by Brian HAMILTON KELLY. He also documented and
debugged the macros as well as many other parts of this package.
    [24]Note that a recent change in LATEX 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.

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.

```
602                    \ifx\saved@macroname\@empty
```

Putting a ! at the beginning of the entry hopefully moves this entry to the very beginning during sorting.

```
603                        \quotechar!%
604                        \actualchar
605                        \generalname
606                    \else

607                        \saved@indexname
608                        \actualchar
609                        \string\verb% % to fool emacs highlighting
610                        \quotechar*%
611                        \verbatimchar\saved@macroname
612                        \verbatimchar
613                    \fi
614                    :\levelchar #3}}%
615  \@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 ".)

```
616 \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).

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

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

```
618 \def\generalname{General}
```

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

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

\GlossaryMin (*dimen*) The remaining macros are all analogues of those used for the theindex environ-
GlossaryColumns (*counter*) ment. 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.

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

49

theglossary (*env.*) The environment `theglossary` is defined in the same manner as the `theindex` environment.

```
622 \ifdoc@multicol
623   \newenvironment{theglossary}{%
624     \begin{multicols}\c@GlossaryColumns
625                      [\glossary@prologue][\GlossaryMin]%
626     \GlossaryParms \let\item\@idxitem \ignorespaces}%
627   {\end{multicols}}
628 \else
629   \newenvironment{theglossary}{%
630       \@restonecoltrue\if@twocolumn\@restonecolfalse\fi
631       \columnseprule \z@  \columnsep 35\p@
632       \twocolumn[\glossary@prologue]%
633       \GlossaryParms \let\item\@idxitem \ignorespaces}
634     {\if@restonecol\onecolumn\else\clearpage\fi}
635 \fi
```

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

```
636 ⟨/package⟩
637 ⟨+gglo⟩preamble
638 ⟨+gglo⟩"\n \\begin{theglossary} \n
639 ⟨+gglo⟩    \\makeatletter\\scan@allowedfalse\n"
640 ⟨+gglo⟩postamble
641 ⟨+gglo⟩"\n\n \\end{theglossary}\n"
```

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

```
642 ⟨+gglo⟩keyword "\\glossaryentry"
643 ⟨*package⟩
```

\GlossaryPrologue The `\GlossaryPrologue` macro is used to place a short message above the glossary
\glossary@prologue 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.

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

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

```
647 \@ifundefined{glossary@prologue}
648     {\def\glossary@prologue{\section*{{Change History}}%
649             \markboth{{Change History}}{{Change History}}%
650             }}{}
```

\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.

```
651 \@ifundefined{GlossaryParms}{\let\GlossaryParms\IndexParms
652   \expandafter\def\expandafter\GlossaryParms\expandafter{\GlossaryParms
653     \rightskip 15pt plus 1fil
654     \parfillskip -15pt plus -1fil\relax}
655 }{}
```

50

\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 `\MaybeStop` 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.

```
656 \def\PrintChanges{\@input@{\jobname.gls}%
657                     \global\let\PrintChanges\@empty}
```

7.12 Bells and whistles

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

To make this happen we have to define `\MaybeStop` 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`.

```
659     \long\def\MaybeStop##1{\@bsphack\gdef\Finale{##1%
```

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

```
660                     \check@checksum}%
```

On the other hand: `\MaybeStop` 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`.

```
661                 \init@checksum
662                 \@esphack}%
663         }
```

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

```
664 \AlsoImplementation
```

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

```
665 \def\OnlyDescription{\@bsphack\long\def\MaybeStop##1{%
```

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

```
666             ##1\endinput}\@esphack}
```

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

```
667 \let\Finale\relax
```

\StopEventually The old wrong name for `\MaybeStop`. We need to use `\def` (i.e., expansion) as `\MaybeStop` gets redefined once in a while.

```
668 \def\StopEventually{\MaybeStop}
```

\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.

```
669 \ifx\l@nohyphenation\undefined
670   \newlanguage\l@nohyphenation
671 \fi
```

```
672 \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.

```
673     \ensuremath\langle
674     \ifmmode \expandafter \nfss@text \fi
675     {%
676       \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.

```
677       \edef\meta@hyphen@restore
678         {\hyphenchar\the\font\the\hyphenchar\font}%
679       \hyphenchar\font\m@ne
680       \language\l@nohyphenation
681       #1\/%
682       \meta@hyphen@restore
683     }\ensuremath\rangle
684 }
```

\meta@font@select Make font used inside \meta customizable.

```
685 \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.

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

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

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

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

```
688     \frenchspacing \@vobeyspaces
```

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

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

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

```
690        \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!

```
691 \def\maketitle{\par
692        \begingroup \def \thefootnote {\fnsymbol {footnote}}%
693        \setcounter {footnote}\z@
694        \def\@makefnmark{\hbox to\z@{$\m@th^{\@thefnmark}$\hss}}%
695        \long\def\@makefntext##1{\parindent 1em\noindent
696             \hbox to1.8em{\hss$\m@th^{\@thefnmark}$}##1}%
697        \if@twocolumn \twocolumn [\@maketitle ]%
698        \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`.

```
699        \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:

```
700        \setcounter {footnote}\z@
701        \gdef\@date{\today}\gdef\@thanks{}%
702        \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.

```
703 \@ifundefined{ps@titlepage}
704        {\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⟩.[25] 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.)

---

[25]Warning: 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⟩`.

```
705 ⟨/package⟩
706 ⟨∗package | shortvrb⟩
707 \def\MakeShortVerb{%
708   \@ifstar
709     {\def\@shortvrbdef{\verb*}\@MakeShortVerb}%
710     {\def\@shortvrbdef{\verb}\@MakeShortVerb}}
```

\@MakeShortVerb

```
711 \def\@MakeShortVerb#1{%
712   \expandafter\ifx\csname cc\string#1\endcsname\relax

713     \@shortvrbinfo{Made }{#1}\@shortvrbdef
714     \add@special{#1}%
```

Then the character's current catcode is stored in `\cc\⟨c⟩`.

```
715     \expandafter
716     \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.

```
717     \begingroup
718       \catcode`\˜\active  \lccode`\˜`#1%
719       \lowercase{%
```

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

```
720       \global\expandafter\let
721         \csname ac\string#1\endcsname˜%
722       \expandafter\gdef\expandafter˜\expandafter{\@shortvrbdef˜}}%
723     \endgroup
```

Finally the character is made active.

```
724     \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 . . .

```
725   \else

726     \@shortvrbinfo\@empty{#1 already}%
727                         {\@empty\verb% % to fool emacs highlighting
728                          (*)}%
729   \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 cat-
code 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.

```
730 \def\DeleteShortVerb#1{%
731   \expandafter\ifx\csname cc\string#1\endcsname\relax
```

54

```
732    \@shortvrbinfo\@empty{#1 not}%
733                              {\@empty\verb% % to fool emacs highlighting
734                               (*)}%
735    \else

736    \@shortvrbinfo{Deleted }{#1 as}%
737                              {\@empty\verb% % to fool emacs
738                                            % highlighting
739                               (*)}%
740    \rem@special{#1}%
741    \global\catcode`#1\csname cc\string#1\endcsname
```

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

```
742    \global \expandafter\let \csname cc\string#1\endcsname \relax
743    \ifnum\catcode`#1=\active
744      \begingroup
745        \catcode`\~\active   \lccode`\~`#1%
746        \lowercase{%
747          \global\expandafter\let\expandafter~%
748          \csname ac\string#1\endcsname}%
749      \endgroup \fi \fi}
```

\@shortvrbinfo Helper function for info messages.

```
750 \def\@shortvrbinfo#1#2#3{%
751 ⟨shortvrb⟩   \PackageInfo{shortvrb}{%
752 ⟨!shortvrb⟩   \PackageInfo{doc}{%
753      #1\expandafter\@gobble\string#2 a short reference
754                                  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\`$\langle c \rangle$ 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).

```
755 \def\add@special#1{%
756   \rem@special{#1}%
757   \expandafter\gdef\expandafter\dospecials\expandafter
758     {\dospecials \do #1}%
```

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

```
759    \expandafter\gdef\expandafter\@sanitize\expandafter
760      {\@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`!

```
761 \def\rem@special#1{%
762   \def\do##1{%
763     \ifnum`#1=`##1 \else \noexpand\do\noexpand##1\fi}%
764   \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.

```
765    \begingroup
766      \def\@makeother##1{%
767        \ifnum`#1=`##1 \else \noexpand\@makeother\noexpand##1\fi}%
768      \xdef\@sanitize{\@sanitize}%
769    \endgroup}
770 ⟨/package | shortvrb⟩
771 ⟨∗package⟩
```

7.13 Providing a checksum and character table[26]

\init@checksum   The checksum mechanism works by counting backslashes in the macrocode. This initializes the count (when called from `\MaybeStop`).

```
772 \def\init@checksum{\relax
773     \global\bslash@cnt\z@}
```

\check@checksum   This reports the sum compared with the value (`\bslash@cnt`) the file advertises. It's called from `\Finale` (if that hasn't been re-defined).

```
774 \def\check@checksum{\relax
775     \ifnum\check@sum>\m@ne
```

We do nothing if the checksum in the file is negative (or not given as it is initialized with -1).

```
776        \ifnum\check@sum=\z@
777          \typeout{********************************}%
778          \typeout{* This macro file has no checksum!}%
779          \typeout{* The checksum should be \the\bslash@cnt!}%
780          \typeout{********************************}%
781        \else
782          \ifnum\check@sum=\bslash@cnt
783            \typeout{******************}%
784            \typeout{* Checksum passed *}%
785            \typeout{******************}%
786          \else
787            \PackageError{doc}{Checksum not passed
788                      (\the\check@sum<>\the\bslash@cnt)}%
789            {The file currently documented seems to be wrong.^^J%
790             Try to get a correct version.}%
791          \fi
792        \fi
793     \fi
794     \global\check@sum\m@ne}
```

\check@sum (*counter*)   We need to define counters, `\bslash@cnt` for the number of backslashes counted
\bslash@cnt (*counter*)   and `\check@sum` for the value advertised by the file if any. A negative value means there is no checksum checking which is the default.

```
795 \newcount\check@sum              \check@sum  = \m@ne
796 \newcount\bslash@cnt             \bslash@cnt = \z@
```

---

[26]Warning: the commentary in this section was written by Dave Love.

56

\CheckSum This is the interface to setting \check@sum.

```
797 \def\CheckSum#1{\@bsphack\global\check@sum#1\relax\@esphack}
```

\step@checksum This advances the count when a backslash is encountered in the macrocode.

```
798 \def\step@checksum{\global\advance\bslash@cnt\@ne}
```

\CharacterTable The user interface to the character table-checking does some \catcodeing 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.

```
799 \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.

```
800 \begingroup
801   \catcode`\~=11
802   \gdef\character@table#1{\def\used~table{#1}%
803     \ifx\used~table\default~table
804         \typeout{**************************}%
805         \typeout{* Character table correct *}%
806         \typeout{**************************}%
807     \else
808       \PackageError{doc}{Character table corrupted}
809                         {\the\wrong@table}
810       \show\default~table
811       \show\used~table
812     \fi
813     \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".

```
814   \global\let\CharTableChanges\@empty
```

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

```
815   \makeatother
816   \gdef\default~table
817     {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
818      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
819      Digits         \0\1\2\3\4\5\6\7\8\9
820      Exclamation    \!     Double quote   \"     Hash (number) \#
821      Dollar         \$     Percent        \%     Ampersand     \&
822      Acute accent   \'     Left paren     \(     Right paren   \)
```

57

```
823        Asterisk       \*      Plus          \+      Comma           \,
824        Minus          \-      Point         \.      Solidus         \/
825        Colon          \:      Semicolon     \;      Less than       \<
826        Equals         \=      Greater than  \>      Question mark \?
827        Commercial at \@       Left bracket  \[      Backslash       \\
828        Right bracket \]       Circumflex    \^      Underscore      \_
829        Grave accent   \`      Left brace    \{      Vertical bar    \|
830        Right brace    \}      Tilde         \~}
831 \endgroup
```

\wrong@table   We need a help message in case of problems.

```
832    \newhelp\wrong@table{Some of the ASCII characters are corrupted.^^J
833            I now \string\show\space you both tables for comparison.}
```

7.14   Attaching line numbers to code lines[27]

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

```
834 \newif\ifcodeline@index \codeline@indexfalse
835 \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.

```
836 \def\codeline@wrindex#1{\if@filesw
837     \begingroup
838         \set@display@protect
839         \immediate\write\@indexfile
840             {\string\indexentry{#1}%
841              {\number\c@CodelineNo}}%
842     \endgroup
843    \fi}
```

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

```
844 \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.

```
845 \def\CodelineIndex{\makeindex
846                    \codeline@indextrue
847                    \let\special@index\codeline@wrindex}
```

\PageIndex   \PageIndex is similar.

```
848 \def\PageIndex{\makeindex
849                \codeline@indexfalse
850                \let\special@index\index}
```

---

[27]Warning: the commentary was written by Dave Love.

58

CodelineNo (*counter*) We need a counter to keep track of the line number.

```
851 \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.

```
852 \@ifundefined{theCodelineNo}
853   {\ifx\selectfont\undefined
854     \def\theCodelineNo{\rmfamily\scriptsize\arabic{CodelineNo}}%
855   \else
856     \def\theCodelineNo{\reset@font\scriptsize\arabic{CodelineNo}}%
857   \fi}
858   {}
```

7.15 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!

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

The following \mathcode definitions allow the characters '\' and '@' to appear in \ttfamily font when invoked in math mode;[28] 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.

```
860 { \catcode'\"=12
861   \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.

```
862 \@ifundefined{DocstyleParms}{}{\DocstyleParms}
```

Clear out \DocstyleParms after use (or non-use).

```
863   \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 $\mathcal{A}\mathcal{M}\mathcal{S}$-TeX, BibTeX and SliTeX, as well
\SliTeX as the usual TeX and LaTeX.

```
864 \@ifundefined{AmSTeX}
865   {\def\AmSTeX{\leavevmode\hbox{$\mathcal A\kern-.2em\lower.376ex%
866         \hbox{$\mathcal M$}\kern-.2em\mathcal S$-\TeX}}}{}
```

---

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

```
867 \@ifundefined{BibTeX}
868   {\def\BibTeX{{\rmfamily B\kern-.05em%
869     \textsc{i\kern-.025em b}\kern-.08em%
870     T\kern-.1667em\lower.7ex\hbox{E}\kern-.125emX}}}{}
871 \@ifundefined{SliTeX}
872   {\def\SliTeX{{\rmfamily S\kern-.06emL\kern-.18em\raise.32ex\hbox
873                   {\scshape i}\kern -.03em\TeX}}}{}
```

\PlainTeX There's even a PLAIN TEX and a WEB.
\Web
```
874 \@ifundefined{PlainTeX}{\def\PlainTeX{\textsc{Plain}\kern2pt\TeX}}{}
875 \@ifundefined{Web}{\def\Web{\textsc{Web}}}{}
```

7.16   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.
```
876 \def\MakePercentIgnore{\catcode`\%9\relax}
877 \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.
```
878 \def\DocInput#1{\MakePercentIgnore\input{#1}\MakePercentComment}
```

7.17   GetFileInfo

\GetFileInfo Define \filedate and friends from info in the \ProvidesPackage etc. commands.
```
879 \def\GetFileInfo#1{%
880   \def\filename{#1}%
881   \def\@tempb##1 ##2 ##3\relax##4\relax{%
882     \def\filedate{##1}%
883     \def\fileversion{##2}%
884     \def\fileinfo{##3}}%
885   \edef\@tempa{\csname ver@#1\endcsname}%
886   \expandafter\@tempb\@tempa\relax? ? \relax\relax}
```

8   Integrating hypdoc

If the option hyperref is selected (which is the default), then we load the hypdoc
package. We do that as late as possible so that we don't generate option clashes if
it is also loaded in the preamble. That package currently changes more commands
than it should (not knowing about their new definitions defined below) so we have
to save and restore a few.

Midterm all this code in hypdoc should be directly included in doc. For now,
while they are separate we have to do this juggling.
```
887 \AddToHook{begindocument/before}[doc/hyperref]{%
888   \ifdoc@hyperref
```
Annoying to code around issue #22
```
889     \expandafter\let\expandafter\doc@eoph@@k\csname doc.sty-h@@k\endcsname
```

We require the package without any option so if it was already loaded there is no
option clash.

```
890  \RequirePackage{hypdoc}
891  \expandafter\let\csname doc.sty-h@@k\endcsname\doc@eoph@@k
```

After hypdoc got loaded we need to undefine those macros again so that later on
Macro and Env doc items appear to be undefined.

```
892  \let\PrintDescribeMacro \@@PrintDescribeMacro
893  \let\PrintDescribeEnv \@@PrintDescribeEnv
894  \let\PrintMacroName \@@PrintMacroName
895  \let\PrintEnvName \@@PrintEnvName
896  \let\SpecialUsageIndex \@@SpecialUsageIndex
897  \let\SpecialEnvIndex \@@SpecialEnvIndex
898  \let\SortIndex \@@SortIndex
899  \let\DescribeMacro \@@DescribeMacro
900  \let\DescribeEnv \@@DescribeEnv
```

The package adds new definitions for \special@index into \CodelineIndex and
\PageIndex but since we are loading it very late we are already past them (in the
preamble). So we test the final state and do it here, if necessary.

```
901  \ifx\special@index\@gobble  % do we write index entries at all?
902  \else
903    \ifcodeline@index
904      \let\special@index\HD@codeline@wrindex
905    \else
906      \let\special@index\HD@page@wrindex
907    \fi
908  \fi
```

The amsmath documentation uses \env in headings and with hyperref enabled this
causes trouble in bookmarks.

> **TODO:** *fix elsewhere eventually*

```
909  \AddToHook{class/amsdtx/after}{%
910    \pdfstringdefDisableCommands{\let\env\@empty }}%
```

That package also adds extra code into \index entries but it doesn't know about
all the stuff that doc does (now). So we need to provide us with two helpers that
handle the \encapchar case in some entries.

```
911  \def\doc@providetarget{\HD@target}%
912  \def\doc@handleencap#1{\encapchar hdclindex{\the\c@HD@hypercount}{#1}}%
```

If that package is not loaded these helpers do little to nothing.

```
913  \else
914    \let\doc@providetarget\@empty
915    \def\doc@handleencap#1{\encapchar #1}%
```

We define the next commands just in case the user changed the option hyperref
from true to false without removing the auxiliary files.

```
916    \def\hdclindex#1#2{\ifx\@nil#2\@nil\else\csname #2\expandafter\endcsname\fi}%
917    \def\hdpindex   #1{\ifx\@nil#1\@nil\else\csname #1\expandafter\endcsname\fi}%
918  \fi
919  }
```

9 Integrating the `DoX` package code

The code in this section is largely taken over from the `DoX` package by Didier with only minor modifications (so far). This means it is a bit back and forth and both the code and the documentation need further updates.

9.1 `DoX` environments

`\@doc@env`
`\@doc@env@`

TODO: *original doc – fix*

$\{\langle are\text{-}we\text{-}macrolike\rangle\}\{\langle item\rangle\}\{\langle indextype\rangle\}\{\langle name\rangle\}$

In `doc.sty`, the `macro` and `environment` environments go through the `\m@cro@` macro which implements specific parts by testing a boolean condition as its first argument. This mechanism is not extensible, so I have to hack away a more generic version that would work for any new `dox` item, only which looks pretty much like the original one (with the addition of options management).

First step is to see if we have a comma-separated list of names in `#3` and if so we call the macro doing the work individually for each

```
920 \ExplSyntaxOn
```

```
921 \long\def\@doc@env#1#2#3{
```

The `\endgroup` here closes the scanning of names (using special catcodes.

```
922   \endgroup
923   \clist_map_inline:nn {#3} { \@doc@env@{#1}{#2}{##1} }
924 }
925
926 \ExplSyntaxOff
```

And here is the payload for each name from the given list:

```
927 \long\def\@doc@env@#1#2#3{%
928   \topsep\MacroTopsep
929   \trivlist
930     \edef\saved@macroname{\string#3}%
```

Since version 2.1g, `doc` creates a `\saved@indexname` command which in used by `\changes`. We now support that as well. The expansion of this command depends on whether the documented item is macrolike or not, which we don't know here (it's only know by `\NewDocElement`). That's why we need one specific command generating `\saved@indexname` the right way for every single item. These commands are named `\@Save`$\langle item\rangle$`IndexName`; they are technically part of the generated API, only not meant for public use.

TODO: *above docu is no longer right (but code needs further changes anyway*

`#1` is either `TT` (for true = macrolike) or `TF`. If true then we drop the first char from `\saved@macroname` and store the result in `\saved@indexname` and use the latter for sorting in the index.

```
931     \if #1%
932       \edef\saved@indexname{\expandafter\@gobble\saved@macroname}%
933 %
```

If the `doc` element described is macrolike but not a normal "macro" then its type should be recorded and this is the places where this happens. For macros (which should make up the bulk of these items we don't do this and for anything else that looks from an indexing perspective like a macro we don't do that either to

keep the list of exceptions small. That would be the case if the indexing command \Code⟨*doc-element*⟩Index is equivalent to \CodeMacroIndex.

```
934        \expandafter\ifx
935                   \csname Code#2Index\endcsname
936                   \CodeMacroIndex
937        \else
938          \record@index@type@save
939             {\saved@indexname}{#2}%
940          \fi
941      \else
942        \let\saved@indexname\saved@macroname
943      \fi
944 %
945      \def\makelabel##1{\llap{##1}}%
946      \if@inlabel
947        \let\@tempa\@empty
948        \count@\macro@cnt
949        \loop\ifnum\count@>\z@
950          \edef\@tempa{\@tempa\hbox{\strut}}\advance\count@\m@ne
951        \repeat
952        \edef\makelabel##1{\llap{\vtop to\baselineskip{\@tempa\hbox{##1}\vss}}}%
953        \advance\macro@cnt\@ne
954      \else
955        \macro@cnt\@ne
956      \fi
957      \ifdoc@noprint
958        \item
959      \else
960        \edef\@tempa{%
961          \noexpand\item[%
```

The second notable modification to the original macro involves dynamically constructing the name of the print macro:

```
962          \noexpand\doc@providetarget
963          \noexpand\strut
964          \noexpand\@nameuse{Print#2Name}{\saved@macroname}]}%
965        \@tempa
966      \fi
967      \ifdoc@noindex\else
968        \global\advance\c@CodelineNo\@ne
```

and the third one involves dynamically constructing the name of the index macro:

```
969          \csname SpecialMain#2Index\expandafter\endcsname
970            \expandafter{\saved@macroname}\nobreak
971          \global\advance\c@CodelineNo\m@ne
972      \fi
```

Suppress further \index entries when we are within a macrolike environment. (There is no point doing that for non-macrolike environments are index entries are only generated for items starting with a backslash anyway.

> **TODO:** *fix*

```
973      \if#1\expandafter\DoNotIndex \expandafter {\saved@macroname}\fi
974      \ignorespaces}
```

\doc@env {⟨*true-value*⟩}{⟨*item*⟩}[⟨*options*⟩]

Handle optional arguments and call `\@doc@env`. Because environments can be nested, we can't rely on grouping for getting options default values. Hence, we need to reset the options at every call.

TODO: *Use 2e interface for* `\keys_set:nn` *when available*

```
975 \def\doc@env#1#2[#3]{%
976    \@nameuse{doc@noprint\doc@noprintdefault}%
977    \@nameuse{doc@noindex\doc@noindexdefault}%
978    \csname keys_set:nn\endcsname{doc}{#3}%
979    \begingroup
980      \ifdoc@outer
981        \catcode'\\12
982      \fi
983      \MakePrivateLetters
984      \@doc@env{#1}{#2}%
985 }
```

9.2 doc descriptions

\@doc@describe {⟨*item*⟩}{⟨*name*⟩}

```
986 \def\@doc@describe#1#2{%
987    \ifdoc@noprint\else
988      \marginpar{\raggedleft
```

The hyperref target has to be in horizontal mode (which is the case if it is after the `\strut`).

```
989              \strut
990              \doc@providetarget
991              \@nameuse{PrintDescribe#1}{#2}}%
992    \fi
993    \ifdoc@noindex\else
994      \@nameuse{Special#1Index}{#2}%
995    \fi
996    \@esphack
997    \endgroup
998    \ignorespaces}
```

\doc@describe {⟨*item*⟩}[⟨*options*⟩]

Handle optional arguments and call `\@doc@describe`.

TODO: *Use 2e interface for* `\keys_set:nn` *when available*

```
999 \def\doc@describe#1[#2]{%
1000   \leavevmode\@bsphack
1001   \csname keys_set:nn\endcsname{doc}{#2}%
1002   \@doc@describe{#1}}
```

9.3 API construction

\@temptokenb A scratch register (which may have been defined elsewhere)

```
1003 \@ifundefined{temptokenb}{\newtoks\@temptokenb}{}
```

\doc@createspecialmainindex {⟨*item*⟩}{⟨*idxtype*⟩}{⟨*idxcat*⟩}

createspecialmainmacrolikeindex $\{\langle item\rangle\}\{\langle idxtype\rangle\}\{\langle idxcat\rangle\}$

TODO: *original doc – fix*

The "macrolike" version does something similar to doc's `\SpecialIndex@` macro, but simplified. Let's just hope nobody will ever define $\backslash_\sqcup$ or nonletter macros as macrolike `doc` elements...

```
1004 \def\doc@createspecialindexes#1#2#3{%

1005   \@temptokena{\space (#2)}%
1006   \@temptokenb{#3:}%

1007   \@nameedef{SpecialMain#1Index}##1{%
1008     \noexpand\@bsphack
1009   \ifdoc@toplevel
1010     \noexpand\special@index{##1\noexpand\actualchar
1011     {\string\ttfamily\space##1}%
1012     \ifx\@nil#2\@nil\else \the\@temptokena \fi
1013     \noexpand\encapchar main}%
1014   \fi
1015   \ifx\@nil#3\@nil\else
1016     \noexpand\special@index{\the\@temptokenb\noexpand\levelchar
1017        ##1\noexpand\actualchar{\string\ttfamily\space##1}%
1018        \noexpand\encapchar main}%
1019   \fi
1020     \noexpand\@esphack}%

1021   \@nameedef{Special#1Index}##1{%
1022     \noexpand\@bsphack
1023   \ifdoc@toplevel
1024     \noexpand\doc@providetarget
1025     \noexpand\index{##1\noexpand\actualchar{\string\ttfamily\space##1}%
1026     \ifx\@nil#2\@nil\else \the\@temptokena \fi
1027          \noexpand\doc@handleencap{usage}}%
1028   \fi
1029   \ifx\@nil#3\@nil\else
1030     \noexpand\index{\the\@temptokenb\noexpand\levelchar
1031        ##1\noexpand\actualchar{\string\ttfamily\space##1}%
1032          \noexpand\doc@handleencap{usage}}%
1033   \fi
1034     \noexpand\@esphack}}

1035 \def\doc@createspecialmacrolikeindexes#1#2#3{%

1036   \@temptokena{\space (#2)}%
1037   \@temptokenb{#3:}%

1038   \@nameedef{Code#1Index}##1##2{%
1039     \noexpand\@SpecialIndexHelper@##2\noexpand\@nil
1040     \noexpand\@bsphack
1041   \noexpand\ifdoc@noindex\noexpand\else
1042     \ifdoc@toplevel
1043       \noexpand\special@index{\noexpand\@gtempa\noexpand\actualchar
1044 \string\verb% % to fool emacs highlighting
1045 \noexpand\quotechar*\noexpand\verbatimchar
1046 \noexpand\bslash\noexpand\@gtempa\noexpand\verbatimchar
1047 \ifx\@nil#2\@nil\else \the\@temptokena \fi
1048 \noexpand\encapchar ##1}%
```

```
1049      \fi
1050      \ifx\@nil#3\@nil\else
1051        \noexpand\special@index{\the\@temptokenb\noexpand\levelchar
1052  \noexpand\@gtempa\noexpand\actualchar
1053  \string\verb% % to fool emacs highlighting
1054  \noexpand\quotechar*\noexpand\verbatimchar
1055  \noexpand\bslash\noexpand\@gtempa\noexpand\verbatimchar
1056  \noexpand\encapchar ##1}%
1057      \fi
1058    \noexpand\fi
1059      \noexpand\@esphack}%

1060    \@nameedef{SpecialMain#1Index}##1{%
1061      \expandafter\noexpand\csname Code#1Index\endcsname
1062          {main}{##1}}%

1063    \@nameedef{Special#1Index}##1{%
1064      \noexpand\@SpecialIndexHelper@##1\noexpand\@nil
1065      \noexpand\@bsphack
1066    \noexpand\ifdoc@noindex\noexpand\else
1067      \ifdoc@toplevel
1068        \noexpand\doc@providetarget
1069        \noexpand\index{\noexpand\@gtempa\noexpand\actualchar
1070  \string\verb% % to fool emacs highlighting
1071  \noexpand\quotechar*\noexpand\verbatimchar
1072  \noexpand\bslash\noexpand\@gtempa\noexpand\verbatimchar
1073  \ifx\@nil#2\@nil\else \the\@temptokena \fi
1074  \noexpand\doc@handleencap{usage}}%
1075      \fi
1076      \ifx\@nil#3\@nil\else
1077        \noexpand\index{\the\@temptokenb\noexpand\levelchar
1078  \noexpand\@gtempa\noexpand\actualchar
1079  \string\verb% % to fool emacs highlighting
1080  \noexpand\quotechar*\noexpand\verbatimchar
1081  \noexpand\bslash\noexpand\@gtempa\noexpand\verbatimchar
1082  \noexpand\doc@handleencap{usage}}%
1083      \fi
1084    \noexpand\fi
1085      \noexpand\@esphack}}
```

\doc@createdescribe {⟨*item*⟩}

```
1086 \def\doc@createdescribe#1{%
1087   \@namedef{Describe#1}{%
```

Because of the optional argument we have to set \MakePrivateLetters already before parsing that (fingers crossed). Otherwise incorrect but quite common usage, such as \DescribeMacro\foo@bar will break because the scan for the optional argument will tokenize the following input (i.e., \foo in that case) before the @ sign becomes a letter. As a result DescribeMacro would receive only \foo as its argument.

```
1088      \begingroup
1089        \MakePrivateLetters
1090        \@ifnextchar[%]
1091        {\doc@describe{#1}}{\doc@describe{#1}[]}}}
```

\doc@createenv   {⟨*item*⟩}{⟨*envname*⟩}

```
1092 \def\doc@createenv#1#2#3{%
1093   \@namedef{#3}{%
1094     \@ifnextchar[%
1095     {\doc@env{#1}{#2}}{\doc@env{#1}{#2}[]}}%
```

Instead of \letting the end of the environment to \endtrivlist we use one level of expansion. This way any possible change in that environment (if that ever happens) is properly reflected.

```
1096   \@namedef{end#3}{\endtrivlist}%
1097 %   \expandafter\let\csname end#3\endcsname\endtrivlist
1098 }
```

\@nameedef

```
1099 \def\@nameedef#1{\expandafter\edef\csname #1\endcsname}
```

9.4   API creation

The whole user interface is created in one macro call.

```
 defaults:

  idxtype   = #3
  idxgroup  = #3s
  printtype =
```

\doc@declareerror

```
1100 \def\doc@declareerror#1#2{%
1101   \PackageError{doc}{Doc element '#1/#2' already defined?\@gobble}%
1102     {There is already a definition for
1103       '\string\Print#1Name',\MessageBreak
1104       '\string\PrintDescribe#1'
1105       or the environment '#2'.\MessageBreak
1106       Maybe you are overwriting something by mistake!\MessageBreak
1107       Otherwise use '\string\RenewDocElement' instead.}%
1108 }
```

\doc@notdeclarederror

```
1109 \def\doc@notdeclarederror#1#2{%
1110   \PackageError{doc}{Doc element '#1/#2' unknown}%
1111     {I expected an existing definition for
1112       '\string\Print#1Name',\MessageBreak
1113       '\string\PrintDescribe#1' and
1114       the environment '#2' but\MessageBreak
1115       not all of them are defined.\MessageBreak
1116       Maybe you wanted to use
1117       '\string\NewDocElement'?}%
1118 }
```

\NewDocElement   [⟨*options*⟩]{⟨*name*⟩}{⟨*envname*⟩}

```
1119 \newcommand\NewDocElement[3][]{%
```

```
1120    \@ifundefined{Print#2Name}%
1121       {\@ifundefined{PrintDescribe#2}%
1122          {\@ifundefined{#3}%
1123             {\@ifundefined{end#3}%
1124                {\@NewDocElement{#1}}%
1125                \doc@declareerror
1126             }\doc@declareerror
1127          }\doc@declareerror
1128       }\doc@declareerror
1129    {#2}{#3}%
1130 }
```

\RenewDocElement [⟨options⟩]{⟨name⟩}{⟨envname⟩}

```
1131 \newcommand\RenewDocElement[3][]{%
1132    \@ifundefined{Print#2Name}\doc@notdeclarederror
1133       {\@ifundefined{PrintDescribe#2}\doc@notdeclarederror
1134          {\@ifundefined{#3}\doc@notdeclarederror
1135             {\@ifundefined{end#3}\doc@notdeclarederror
1136                {\@NewDocElement{#1}}%
1137             }%
1138          }%
1139       }%
1140    {#2}{#3}%
1141 }
```

\@NewDocElement {⟨options⟩}{⟨name⟩}{⟨envname⟩}

```
1142 \def\@NewDocElement#1#2#3{%
1143    \doc@macrolikefalse
1144    \doc@topleveltrue
```

> **TODO:** *Use 2e interface for* `\keys_set:nn` *when available*

```
1145    \def\doc@idxtype{#3}%
1146    \def\doc@idxgroup{#3s}%
1147    \let\doc@printtype\@empty
1148    \csname keys_set:nn\endcsname{doc}{#1}%
```

\Print...Name {⟨name⟩}

> **TODO:** *extremely messy this with so many* `\expandafters` *... should reimplement in expl3*

```
1149    \ifx\doc@printtype\@empty
1150       \@temptokena{}%
1151    \else
1152       \@temptokena\expandafter{\expandafter
1153          \textnormal\expandafter{\expandafter
1154          \space\expandafter
1155          (\doc@printtype)}}%
1156    \fi
1157    \@nameedef{Print#2Name}##1{%
1158       {\noexpand\MacroFont
1159       \ifdoc@macrolike
1160          \noexpand\string
```

```
1161        \fi
1162        ##1%
1163        \the\@temptokena
1164      }}%
```

\PrintDescribe... {⟨name⟩}

```
1165   \expandafter\let\csname PrintDescribe#2\expandafter\endcsname
1166              \csname Print#2Name\endcsname
```

\SpecialMain...Index {⟨name⟩}

\Special...Index {⟨name⟩}

```
1167   \edef\doc@expr{%
1168     \ifdoc@macrolike
1169       \noexpand\doc@createspecialmacrolikeindexes
1170     \else
1171       \noexpand\doc@createspecialindexes
1172     \fi
1173     {#2}%
1174    }%
1175   \expandafter\expandafter\expandafter
1176   \doc@expr
1177   \expandafter\expandafter\expandafter
1178     {\expandafter\doc@idxtype\expandafter}\expandafter
1179     {\doc@idxgroup}%
```

\Describe... [⟨options⟩]{⟨name⟩}

```
1180     \doc@createdescribe{#2}%
```

\metaDocElement (env.) **TODO:** *can't have formatting in argument – fix*
[⟨options⟩]{⟨name⟩}

```
1181     \ifdoc@macrolike
1182       \doc@createenv{TT}{#2}{#3}%
1183     \else
1184       \doc@createenv{TF}{#2}{#3}%
1185     \fi
1186 }
```

9.5   Setting up the default `doc` elements

9.5.1   Macro facilities

Macros get only a single index entry (no index group, no index type) and they do not get any label either when printing in the margin.

```
1187 \NewDocElement[macrolike = true ,
1188            idxtype  = ,
1189            idxgroup = ,
1190            printtype =
1191            ]{Macro}{macro}
```

SpecialMainIndex In doc v2 we had \SpecialMainIndex and \SpecialMainEnvIndex but now with additional doc elements we always add the element name after "Main" so this would be \SpecialMainMacroIndex. We use \def not \let so any redefinition of \SpecialMainMacroIndex will be transparent.

```
1192 \def\SpecialMainIndex{\SpecialMainMacroIndex}
```

SpecialUsageIndex doc v2 also had \SpecialUsageIndex which is now called \SpecialMacroIndex generating the "usage" index entry for a macro. Again we provide that as an alias via \def.

In fact the documentation of doc v2 claimed that one can use this for both macros and environments but that was never true as for environments the result was that the first character was dropped in sorting of the index. The correct way is to use \SpecialEnvIndex for this.

```
1193 \def\SpecialUsageIndex{\SpecialMacroIndex}
```

\SpecialIndex

```
1194 \def\SpecialIndex    {\CodeMacroIndex{code}}
```

9.5.2 Environment facilities

Providing documentation support for environments. Here we differ from doc V2 by marking the environments with "(*env.*)" when printing the name in the margin.

```
1195 \NewDocElement[macrolike = false ,
1196                idxtype   = env.   ,
1197                idxgroup  = environments ,
1198                printtype = \textit{env.}
1199                ]{Env}{environment}
```

To be able to restore the definition after hypdoc is loaded we better save them here. We only load the package at the end of the preamble, but the user might do this earlier and then chaos is ensured.

```
1200    \let\@@PrintDescribeMacro \PrintDescribeMacro
1201    \let\@@PrintDescribeEnv \PrintDescribeEnv
1202    \let\@@PrintMacroName \PrintMacroName
1203    \let\@@PrintEnvName \PrintEnvName
1204    \let\@@SpecialUsageIndex \SpecialUsageIndex
1205    \let\@@SpecialEnvIndex \SpecialEnvIndex
1206    \let\@@SortIndex \SortIndex
1207    \let\@@DescribeMacro \DescribeMacro
1208    \let\@@DescribeEnv \DescribeEnv
```

10 Misc additions

\cs

```
1209 \DeclareRobustCommand\cs[1]{\texttt{\bslash #1}}
```

amsdtx has its own definition for \cs but that now gets overwritten because the class loads doc afterwards. So for now we reinstall it here.

TODO: *fix elsewhere*

```
1210 \AddToHook{class/amsdtx/after}{%
1211    \DeclareRobustCommand\cs[1]{%
```

```
1212    \@boxorbreak{%
1213        \ntt
1214        \addbslash#1\@empty
1215        \@xp\@xp\@xp\@indexcs\@xp\@nobslash\string#1\@nil
1216    }%
1217  }%
1218  \def\cn{\cs}%
1219 }
```

We can now finish the `docstrip` main module.

```
1220 ⟨/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 TeXbook). Addison-Wesley, Vol. A, 1986.

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

[5] Frank Mittelbach. The `doc`-option. *TUGboat*, Vol. 10(2), *pp.* 245–273, July 1989.

[6] Frank Mittelbach, Denys Duchier and Johannes Braams. `docstrip.dtx`. The file is part of core LaTeX.

[7] R. E. Raspe (*1737, †1797). Baron Münchhausens narrative of his marvelous 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.

72

75

Change History

BHK – 1989/04/26
   \changes: Changed definition of
     \protect. 48
    Documented \changes
     command. 48
   \glossary@prologue: Added to
     support \changes. 50
   GlossaryColumns: Added to
     support \changes. 49
   \GlossaryMin: Added to support
     \changes. 49
   \GlossaryParms: Added to
     support \changes. 50
   \GlossaryPrologue: Added to
     support \changes. 50
   \PrintChanges: Added to support
     \changes. 51
   \RecordChanges: Renames former
     \PrintChanges command. . . . 49
   \saved@macroname: Provided for
     sorting outside macro
     environment 49
   theglossary: Added to support
     \changes. 50

v1.0p – 1994/05/21
   General: Use new error commands 1
v1.4? – 1989/04/16
   General: changes to the index env. 44
v1.4? – 1989/04/19
   General: use DEK's algorithm and
     implement a twocols env. 44
v1.4r – 1989/04/22
   General: twocols env. placed into
     separate file 44

v1.4t – 1989/04/24
   \endtheindex: Incorporated new
     multicols env. 45
   IndexColumns: Counter added. . . 44
   \meta: Macro added. 52
   theindex: Incorporated new
     multicols env. 45
v1.5a – 1989/04/26
   General: Now input multicol.sty
     instead of multcols.sty 44
   theindex: Call multicols first . . 45
v1.5c – 1989/04/27
   \short@macro: Corrected bad bug
     by putting the
     scan@allowedfalse macro before
     printing the argument. 35
v1.5d – 1989/04/28
   General: \marginparwidth setting
     added. 30
v1.5f – 1989/04/29
   General: Thanks to Brian who
     documented the \changes
     macro feature. 1
v1.5g – 1989/05/07
   General: MacroTopsep now called
     MacrocodeTopsep and new
     MacroTopsep added 1
   \PlainTeX: space between plain
     and TeX changed. 60
v1.5h – 1989/05/17
   General: All lines shortened to <72
     characters 1
v1.5i – 1989/06/07
   General: Avoid reading the file
     twice. 22

78