Any rollback request before 2016-02-15 we try to fullfil with the 2016 version:

The doc and shortvrb Packages*

Frank Mittelbach†‡§

Printed June 11, 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 v3.0p dated 2024/04/26.

†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` `\IndexInput`

Finally the ⟨*special input commands*⟩ part should contain one or more `\DocInput{`⟨*file name*⟩`}` and/or `\IndexInput{`⟨*file name*⟩`}` commands. The `\DocInput` command is used for files prepared for the `doc` package whereas `\IndexInput` can be used for all kinds of macro files. See page 13 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'.

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}
```

[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.

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

[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.

\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} (if you change it later you might see strange typesetting effects if you are unlucky).

\MacroFont does not alter the font of \verb or \verb* because it is often used to make the font size of the code displays smaller, which would look odd if used within a paragraph. If you decide to use a different font family and want to use the same family with \verb you need to alter the font setup for \ttfamily in addition to \MacroFont.

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
\PrintDescribeEnv
\PrintMacroName
\PrintEnvName

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

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

\NewDocElement[⟨*options*⟩]{⟨*element-name*⟩}{⟨*env-name*⟩}

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

  \Describe⟨*element-name*⟩[⟨*options*⟩]{⟨*element*⟩}

[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.

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

  `\begin{`⟨*env-name*⟩`}[`⟨*options*⟩`]{`⟨*element*⟩`}`

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

  `\PrintDescribe`⟨*element-name*⟩`{`⟨*element*⟩`}`

- 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.

`\ProvideDocElement` This declaration does nothing when the doc element is already declared, otherwise it works like `\NewDocElement`. It can be useful if you have many documentation files that you may want to process individually as well as together.

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.

`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.*)
`verbatim*` (*env.*)
`\verb`

It is often a good idea to include examples of the usage of new macros in the text. Because of the `%` sign in the first column of every row, the `verbatim` environment is slightly altered to suppress those characters.[4] The `verbatim*` environment is changed in the same way. The `\verb` command is re-implemented to give an error report if a newline appears in its argument. The `verbatim` and `verbatim*` environments set text in the style defined by `\MacroFont` (§2.5).

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`
`\EnableCrossrefs`

As already mentioned, every macro name used within a `macrocode` or `macrocode*` environment will produce an index entry. In this way one can easily find out where a specific macro is used. Since TEX is considerably slower[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. You might use several `\DoNotIndex` commands: their lists will be concatenated. In this article I used `\DoNotIndex` for all macros which are already defined in LATEX.

All three above declarations are local to the current group.

`\PageIndex`

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 neither is used, no index is produced. Using `\PageIndex` makes all index en-

[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.

\CodelineIndex tries 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 automatically.[7] The style of this numbering can be controlled by defining the macro \theCodelineNo. Its default definition is to use scriptsize arabic numerals; a user-supplied definition won't be overwritten.

\theCodelineNo

\CodelineNumbered When you don't wish to get an index but want your code lines numbered use \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 The \actualchar is used to separate the 'key' and the actual index entry. The

\quotechar \quotechar is used before a special index program character to suppress its special

\encapchar meaning. The \encapchar separates the indexing information from a letter string which makeindex uses as a TEX command to format the page number associated with a special entry. It is used in this package to apply the \main and the \usage

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

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

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

\SpecialMainMacroIndex To produce a main index entry for a macro the \SpecialMainMacroIndex

\SpecialMainEnvIndex macro[9] may be used. It is called 'special' because it has to print its argument verbatim. A similar macro, called \SpecialMainEnvIndex is used for indexing the main definition point of an environment.[10]

\SpecialMacroIndex To index the usage of a macro or an environment \SpecialMacroIndex and

\SpecialEnvIndex \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

[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.

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

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` 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.

New in v3

`\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.*) Contrary to standard LaTeX, the index is typeset in three columns by de-
`IndexColumns` (*counter*) fault. This is controlled by the LaTeX counter 'IndexColumns' and can therefore be changed with a `\setcounter` declaration. Additionally one doesn't want to start a new page unnecessarily. Therefore the `theindex` environment is redefined.

`\IndexMin` (*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 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:

\IndexParms

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

\@idxitem Additionally it defines `\@idxitem` (which will be used when an `\item` command is encountered) and selects `\small` size. If you want to change any of these values you have to define them all.

\main \usage \code The page numbers for main index entries are encapsulated by the `\main` macro (underlining its argument) and the numbers denoting the description are encapsulated by the `\usage` macro (which produces *italics*). `\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`	= 80.0pt	`\MacroTopsep`	= 7.0pt plus 2.0pt minus 2.0pt
`\marginparwidth`	= 126.0pt	`\MacroIndent`	= 10.66406pt
`\marginparpush`	= 0.0pt	`\MacrocodeTopsep`	= 3.0pt plus 1.2pt minus 1.0pt
`\tolerance`	= 1000		

2.14 Short input of verbatim text pieces

\MakeShortVerb \MakeShortVerb* \DeleteShortVerb It is awkward to have to type, say, `\verb|...|` continually when quoting verbatim bits (like macro names) in the text, so an abbreviation mechanism is provided. Pick a character ⟨*c*⟩—one which normally has catcode 'other' unless you have very good reason not to—which you don't envisage using in the text, or not using often. (I like `"`, but you may prefer `|` if you have `"` active to do umlauts, for instance.) Then if you say `\MakeShortVerb{\`⟨*c*⟩`}` you can subsequently use ⟨*c*⟩⟨*text*⟩⟨*c*⟩ as the equivalent of `\verb`⟨*c*⟩⟨*text*⟩⟨*c*⟩; analogously, the `*`-form `\MakeShortVerb*{\`⟨*c*⟩`}` gives you the equivalent of `\verb*`⟨*c*⟩⟨*text*⟩⟨*c*⟩. Use `\DeleteShortVerb{\`⟨*c*⟩`}` if you subsequently want ⟨*c*⟩ to revert to its previous meaning—you can always turn it on again after the unusual section. The 'short verb' commands make global changes. The abbreviated `\verb` may not appear in the argument of another command just like `\verb`. However the 'short verb' character may be used freely in

the `verbatim` and `macrocode` environments without ill effect. `\DeleteShortVerb` is silently ignored if its argument does not currently represent a short verb character. Both commands type a message to tell you the meaning of the character is being changed.

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

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

2.15 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` `\MaybeStop` `\StopEventually` New in v3

You can use the `\OnlyDescription` declaration in the driver file to suppress the last part of your document (which presumably exhibits the code). To make this work you have to place the command `\MaybeStop` at a suitable point in your file. This macro[11] has one argument in which you put all information you want to see printed if your document ends at this point (for example a bibliography which is normally printed at the very end). When the `\OnlyDescription` declaration is missing the `\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.

`\Finale`

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

`\maketitle` `\ps@titlepage`

`\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, 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

[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.

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` `\PrintChanges` To cause the change information to be written out, include `\RecordChanges` in the driver file. To read in and print the sorted change history (in two columns), just put the `\PrintChanges` command as the last (commented-out, and thus executed during the documentation pass through the file) command in your package file. Alternatively, this command may form one of the arguments of the `\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*) `\GlossaryPrologue` `\GlossaryParms` `GlossaryColumns` (*counter*) The `\GlossaryMin`, `\GlossaryPrologue` and `\GlossaryParms` macros and the counter `GlossaryColumns` are analogous to the `\Index`... versions. (The LaTeX 'glossary' mechanism is used for the change entries.)

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

`\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` `\CheckModules` `\Module` `\AltMacroFont` The 'module' directives of the docstrip system [6] are normally recognized and

[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.

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

`StandardModuleDepth` (*counter*)

Sometimes (as in this file) the whole code is surrounded by modules to produce several files from a single source. In this case it is clearly not appropriate to format all code lines in a special `\AltMacroFont`. For this reason a counter `StandardModuleDepth` is provided which defines the level of module nesting which is still supposed to be formatted in `\MacroFont` rather 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 "environment". 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 commands\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}
```

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`
`\CheckSum`

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

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

at the beginning of the file then character translation failures will be detected, provided of course, that the used `doc` package has a correct default table. The percent 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 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

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

```
%    \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 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

[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.

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 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 `\%`.

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

- 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 makeindex 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. 18);
- Because the accompanying makeindex style files support the inconsistent attribute specifications of older and newer versions makeindex 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.

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.

T