

Grzegorz ‘Natror’ Murzynowski

**The gmdoc Package
i.e., gmdoc.sty and gmdocc.cls**

Contents

a. The <code>gmdoc.sty</code> Package	4	(No) <code>\eof</code>	87
Readme	4	b. The <code>gmdoc</code> Class For <code>gmdoc</code>	
Installation	4	Driver Files	89
Contents of the <code>gmdoc.zip</code> Archive	5	Intro	89
Compiling the Documentation	5	Usage	89
Dependencies	5	The Code	90
Bonus: <code>base</code> Drivers	5	c. <code>gmdocDoc.tex</code>, The Driver File	93
Introduction	6	d. The <code>gmutils</code> Package	94
The User Interface	6	Intro	94
Used Terms	6	Contents of the <code>gmutils.zip</code> Archive	94
Preparing the Source File	7	A couple of abbreviations	94
The Main Input Commands	8	<code>\@ifnextcat</code> , <code>\@ifnextac</code>	96
Package Options	10	<code>\afterfi</code> and <code>Pals</code>	98
The Packages Required	11	Almost an Environment or	
Macros for Marking the Macros	12	Redefinition of <code>\begin</code>	98
Index Ex/Inclusions	14	Improvement of <code>\end</code>	99
The DocStrip Directives	14	From <code>resize</code>	99
The Changes History	14	<code>\firstofone</code> and the Queer	
The Parameters	16	<code>\catcodes</code>	101
The Narration Macros	19	Metasymbols	101
A Queerness of <code>\label</code>	20	Macros for Printing Macros and	
<code>doc</code> -Compatibility	21	Filenames	102
The Code	21	Storing and Restoring the Meanings	
The Package Options	22	of CSs	104
The Dependencies and		Not only preamble!	106
Preliminaries	23	Third Person Pronouns	106
The Core	26	To Save Precious Count Registers	107
Numbering (or Not) of the Lines	34	Improvements to <code>mwcls</code> Sectioning	
Spacing with <code>\everypar</code>	35	Commands	107
Life Among Queer EOLs	36	Compatibilising Standard and	
Adjustment of <code>verbatim</code> and <code>\verb</code>	38	<code>mwcls</code> Sectionings	110
Macros for Marking The Macros	39	<code>enumerate*</code> and <code>itemize*</code>	111
Index Exclude List	55	The Logos	112
Index Parameters	58	Expanding turning stuff all into	
The DocStrip Directives	60	‘other’	113
The Changes History	61	Varia	114
The Checksum	66	e. The <code>gmiflink</code> Package	118
Macros from <code>ltxdoc</code>	67	Introduction, usage	118
<code>\DocInclude</code> and the <code>ltxdoc</code> -Like		Contents of the <code>gmiflink.zip</code> archive	118
Setup	68	The Code	119
<code>\SelfInclude</code>	72	f. The <code>gmverb</code> Package	121
Redefinition of <code>\maketitle</code>	74	Intro, Usage	121
The File’s Date and Version		Contents of the <code>gmverb.zip</code> Archive	122
Information	76	The Code	122
Miscellanea	78	Preliminaries	122
<code>doc</code> -Compatibility	81	The Breakables	123
<code>gmdocing doc.dtx</code>	86		
Polishing, Development and Bugs	87		

Almost-Knuthian <code>\ttverbatim</code> . .	124	Change History	131
The Core: From <code>shortvrb</code>	124	Index	133
<code>doc-</code> And <code>shortvrb</code> -Compatibility .	129		

a. The gmdoc.sty Package¹

April 30, 2007

This is (a documentation of) file `gmdoc.sty`, intended to be used with L^AT_EX 2_ε as a package for documenting (L^A)T_EX files and to be documented with itself.

Written by Natror (Grzegorz Murzynowski),
natror at o2 dot pl

©2006 by Natror (Grzegorz Murzynowski).

This program is subject to the L^AT_EX Project Public License.

See <http://www.ctan.org/tex-archive/help/Catalogue/licenses.lpp1.html> for the details of that license.

LPPL status: "author-maintained".

Many thanks to my T_EX Guru Marcin Woliński for his T_EXnical support.

```
1 <package>
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{gmdoc}
4 [2007/04/29_v0.99e_a_documenting_package_(GM)]
```

Readme

This package is a tool for documenting of (L^A)T_EX packages, classes etc., i.e., the `.sty`, `.cls` files etc. The author just writes the code and adds the commentary preceded with % sign (or another properly declared). No special environments are necessary.

The package tends to be (optionally) compatible with the standard `doc.sty` package, i.e., the `.dtx` files are also compilable with `gmdoc` (they may need very little adjustment, in some rather special cases).

The tools are integrated with `hyperref`'s advantages such as hyperlinking of index entries, contents entries and cross-references.

Installation

Just put the `gmdoc.sty` and `gmdocc.cls` somewhere in the `texmf/tex/latex` branch. Creating a `texmf/tex/latex/gm` directory may be advisable if you consider using other packages written by me.

You should also install `gmverb.sty`, `gmutils.sty` and `gmiflink.sty` (e.g., put them into the same `gm` directory). These packages are available on CTAN as separate `.zip` archives.

Moreover, you should put the `gmglolist` file, a MakeIndex style for the changes' history, into some `texmf/makeindex` (sub)directory.

Then you should refresh your T_EX distribution's files' database most probably.

¹ This file has version number v0.99e dated 2007/04/29.

Contents of the gmdoc.zip Archive

The distribution of the gmdoc package consists of the following six files.

```
gmdoc.sty
gmdocc.cls
gmglo.ist
README
gmdocDoc.tex
gmdocDoc.pdf
```

Compiling the Documentation

The last of the above files (the .pdf, i.e., *this file*) is a documentation compiled from the .sty and .cls files by running L^AT_EX on the gmdocDoc.tex twice, then MakeIndex on the gmdocDoc.idx and gmdocDoc.glo files, and then L^AT_EX on gmdocDoc.tex once more.

MakeIndex shell commands:

```
makeindex -r gmdocDoc
makeindex -r -s gmglo.ist -o gmdocDoc.gls gmdocDoc.glo
```

The -r switch is to forbid MakeIndex to make implicit ranges since the (code line) numbers will be hyperlinks.

Compiling the documentation requires the packages: gmdoc (gmdoc.sty and gmdocc.cls), gmutils.sty, gmverb.sty, gmiflink.sty and also some standard packages: hyperref.sty, color.sty, geometry.sty, multicol.sty, lmodern.sty, fontenc.sty that should be installed on your computer by default.

If you had not installed the mwcls classes (available on CTAN and present in T_EX Live e.g.), the result of your compilation might differ a bit from the .pdf provided in this .zip archive in formatting: If you had not installed mwcls, the standard article.cls class would be used.

Dependencies

The gmdoc bundle depends on some other packages of mine:

```
gmutils.sty,
gmverb.sty,
gmiflink.sty
gmeometric (for the driver of The LATEX 2ε Source)
```

and also on some standard packages:

```
hyperref.sty,
color.sty,
geometry.sty,
multicol.sty,
lmodern.sty,
fontenc.sty
```

that should be installed on your computer by default.

Bonus: base Drivers

As a bonus and example of doc-compatibility there are driver files included (cf. Palestrina, *Missa papae Marcelli* ;-):

source2e_gmdoc.tex
docstrip_gmdoc.tex
doc_gmdoc.tex

oldcomm.sty
(gmsource2e.ist is generated from source2e_gmdoc.tex)

These drivers typeset the respective files from the

.../texmf-dist/source/latex/base

directory of the T_EXLive2005 distribution.

Probably you should redefine the `\BasePath` macro in them so that it points that directory on your computer.

Introduction

There are very sophisticated and effective tools for documenting L^AT_EX macro packages, namely the `doc` package and the `ltxdoc` class. Why did I write another documenting package then?

I like comfort and `doc` is not comfortable enough for me. It requires special marking of the macro code to be properly typeset when documented. I want T_EX to know ‘itself’ where the code begins and ends, without additional marks.

That’s the difference. One more difference, more important for the people for whom the `doc`’s conventions are acceptable, is that `gmdoc` makes use of `hyperref` advantages and makes a hyperlinking index and toc entries and the cross-references, too. (The CSs in the code maybe in the future.)

The rest is striving to level the very high `doc`/`ltxdoc`’s standard, such as (optional) numbering of the codelines and automatic indexing the control sequences e.g.

The `doc` package was and still is a great inspiration for me and I would like this humble package to be considered as a sort of homage to it². If I mention copying some code or narrative but do not state the source explicitly, I mean the `doc` package’s documentation (I have v2.1b dated 2004/02/09).

The User Interface

Used Terms

When I write of a **macro**, I mean a macro in The T_EXbook’s meaning, i.e., a control sequence whose meaning is `\(e/g/x)defined`. By a **macro’s parameter** I mean each of `#⟨digit⟩`s in its definition. When I write about a **macro’s argument**, I mean the value (list of tokens) substituting the corresponding parameter of this macro. (These understandings are according to The T_EXbook, I hope: T_EX is a religion of Book ;-).)

I’ll use a shorthand for ‘control sequence’, **CS**.

When I talk of a **declaration**, I mean a macro that expands to a certain assignment, such as `\itshape` or `\@onlypreamble{⟨CS⟩}`.

Talking of declarations, I’ll use the **OCSR** acronym as a shorthand for ‘observes/ing common T_EX scoping rules’.

By a **command** I mean a certain abstract visible to the end user as a CS but consisting possibly of more than one macro. I’ll talk of a **command’s argument** also in

² As Grieg’s Piano Concerto is a homage to the Schumann’s.

the ‘sense-for-the-end-user’, e.g., I’ll talk of the `\verb command`’s argument although *the macro* `\verb` has no `#⟨digit⟩` in its definition.

The **code** to be typeset verbatim (and with all the bells and whistles) is everything that’s not commented out in the source file and what is not a leading space(s).

The **commentary** or **narrative** is everything after the comment char till the end of a line. The **comment char** is a character the `\catcode` of which is 14 usually i.e., when the file works; if you don’t play with the `\catcodes`, it’s just the `%`. When the file is documented with `gmdoc`, such a char is `re\catcoded` and its rôle is else: it becomes the **code delimiter**.

A line containing any $\mathrm{T\!E\!X}$ code (not commented out) will be called a **codeline**. A line that begins with (some leading spaces and) a code delimiter will be called a **comment line** or **narration line**.

The **user** of this package will also be addressed as **you**.

\heshe Not to favour any particular gender (of the amazingly rich variety, I mean, not of the vulgarly simplified two-element set), in this documentation I use alternating pronouns of third person (`\heshe` etc. commands provided by `gmutils`), so let one be not surprised if ‘he’ sees ‘herself’ altered in the same sentence :-).

Preparing the Source File

When $(\mathrm{L})\mathrm{T\!E\!X}$ with `gmdoc.sty` package loaded typesets the comment lines, the code delimiter is omitted. If the comment continues a codeline, the code delimiter is printed. It’s done so because ending a $\mathrm{T\!E\!X}$ code line with a `%` is just a concatenation with the next line sometimes. Comments longer than one line are typeset continuously with the code delimiters omitted.

^^B The user should just write his splendid code and brilliant commentary. In the latter she may use usual $(\mathrm{L})\mathrm{T\!E\!X}$ commands. The only requirement is, if an argument is divided in two lines, to end such a dividing line with `^^B` sequence that’ll enter the (active) `⟨char2⟩` which shall gobble the line end.

^^A Moreover, if he wants to add a meta-comment i.e., a text that doesn’t appear in the code layer nor in the narrative, she may use the `^^A` sequence that’ll be read by $\mathrm{T\!E\!X}$ as `⟨char1⟩`, which is in `gmdoc` active and defined to gobble the stuff between itself and the next line end.

Note, however, that both `^^A` and `^^B` are usually macros in `gmdoc` and the text being gobbled is their argument so it has to be balanced of braces and, if it occurs inside an `\if...`, it has to have all `\if...`s coupled with `\elses` and `\fis`.

\StraightEOL However, it may be a bit confusing for someone acquainted with the `doc` conventions. If you don’t fancy the `^^B` special sequence, instead you may restore the standard meaning of the line end with the `\StraightEOL` declaration which OCSR. As almost all the control sequences, it may be used also as an environment, i.e., `\begin{StraightEOL} ... \end{StraightEOL}`. However, if for any reason you don’t want to make an environment (a group), there’s a `\StraightEOL`’s counterpart, the `\QueerEOL` declaration that restores again the queer³ `gmdoc`’s meaning of the line end. It OCSR, too. One more point to use `\StraightEOL` is where you wish some code lines to be executed both while loading the file and during the documentation pass (it’s analogous to `doc`’s not embracing some code lines in a **macrocode** environment).

³ In my understanding ‘queer’ and ‘straight’ are not the opposites excluding each other but the counterparts that may cooperate in harmony for people’s good. And, as I try to show with the `\QueerEOL` and `\StraightEOL` declarations, ‘queer’ may be very useful and recommended while ‘straight’ is the standard but not necessarily normative.

As in standard T_EXing, one gets a paragraph by a blank line. Such a line should be %ed of course. A fully blank line is considered a blank *code line* and hence results in a vertical space in the documentation. As in the environments for poetry known to me, subsequent blank lines do not increase such a space.

Then he should prepare a main document file, a **driver** henceforth, to set all the required formattings such as `\documentclass`, paper size etc., and load this package with a standard command i.e., `\usepackage{gmdoc}`, just as `doc`'s documentation says:

“If one is going to document a set of macros with the `[gm]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[⟨options, probably none⟩]{gmdoc}
  ⟨preamble⟩
\begin{document}
  ⟨special input commands⟩
\end{document}
```

The Main Input Commands

`\DocInput` To typeset a source file you may use the `\DocInput` macro that takes the (path and) name of the file *with the extension* as the only argument, e.g., `\DocInput{mybrilliantpackage.sty}`.

(Note that an *installed* package or class file is findable to T_EX even if you don't specify the path.)

`\OldDocInput` If a source file is written with rather `doc` than `gmdoc` in mind, then the `\OldDocInput` command may be more appropriate (e.g., if you break the arguments of commands in the commentary in lines). It also takes the file (path and) name as the argument.

`macrocode` When using `\OldDocInput`, you have to wrap all the code in `macrocode` environments, which is not necessary when you use `\DocInput`. Moreover, with `\OldDocInput` the `macrocode(*)` environments require to be ended with `%\end{macrocode(*)}`.

`\DocInclude` If you wish to document many files in one document, you are provided `\DocInclude` command, analogous to L^AT_EX's `\include` and very likely to `ltxdoc`'s command of the same name. In `gmdoc` it has one mandatory argument that should be the file name *without extension*, just like for `\include`.

The file extensions supported by `\DocInclude` are `.fdd`, `.dtx`, `.cls`, `.sty`, `.tex` and `.fd`. The macro looks for one of those extensions in the order just given. If you need to document files of other extensions, please let me know and most probably we'll make it possible.

`\DocInclude` has also an optional first argument that is intended to be the path of the included file with the levels separated by / (slash) and also ended with a slash. The path given to `\DocInclude` as the first and optional argument will not appear in the headings nor in the footers.

`\maketitle` `\DocInclude` redefines `\maketitle` so that it makes a chapter heading or, in the classes that don't support `\chapter`, a part heading, in both cases with respective toc entries. The default assumption is that all the files have the same author(s) so there's no need to print them in the file heading. If you wish the authors names to be printed, you

`\PrintFilesAuthors` should write `\PrintFilesAuthors` in the preamble or before the relevant `\DocIncludes`.
`\SkipFilesAuthors` If you wish to undeclare printing the authors names, there is `\SkipFilesAuthors` declaration.

Like in `ltxdoc`, the name of an included file appears in the footer of each page with date and version info (if they are provided).

The `\DocIncluded` files are numbered with the letters, the lowercase first, as in `ltxdoc`. Such a filemarker also precedes the index entries, if the (default) codeline index option is in force.

`\includeonly` As with `\include`, you may declare `\includeonly{<filenames separated by commas>}` for the draft versions.

`\SelfInclude` If you wish to include the driver file into your documentation, you may write `\DocInput{<jobname.tex>}`, but a try of `\DocInclude{<jobname>}` would result with input stack overflow caused by infinite `\input{<jobname>.aux}` recursion. But there's `\SelfInclude` at your service that creates and uses `<jobname>.auxx` file instead of the usual `<jobname>.aux`. Its effect is analogous to the `\DocInclude`'s, but *the arguments it takes are totally different*: Since the filename is known, there's no need to state it. The extension is assumed to be `.tex`, but if it's different, you may state it in the first and optional argument. The second argument is mandatory and it's the stuff to be put at begin of file input, this one and no else (with `\AtBeginInputOnce` hook). For the example of usage see line 14 of chapter c.

At the default settings, the `\Doc/SelfIncluded` files constitute chapters if `\chapter` is known and parts otherwise. The `\maketitles` of those files result in the respective headings.

`\ltxLookSetup` If you prefer more `ltxdoc`ish look, in which the files always constitute the parts and those parts have a part's title pages with the file name and the files' `\maketitles` result in (article-like) titles not division headings, then you are provided the `\ltxLookSetup` declaration (allowed only in the preamble). However, even after this declaration the files will be included according to `gmdoc`'s rules not necessarily to the `doc`'s ones (i.e., with minimal marking necessary at the price of active line ends (therefore not allowed between a command and its argument nor inside an argument)).

`\olddocIncludes` On the other hand, if you like the look offered by me but you have the files prepared for `doc` not for `gmdoc`, then you should declare `\olddocIncludes`. Unlike the previous one, this may be used anywhere, because I have the account of including both `doc`-like and `gmdoc`-like files into one document. This declaration just changes the internal input command and doesn't change the sectioning settings.

`\gmdocIncludes` It seems possible that you wish to document the 'old-doc' files first and the 'new-doc' ones after, so the above declaration has its counterpart, `\gmdocIncludes`, that may be used anywhere, too. Before the respective `\DocInclude(s)`, of course.

Both these declarations OCSR.

If you wish to document your files as with `ltxdoc` *and* as with `doc`, you should declare `\ltxLookSetup` in the preamble *and* `\olddocIncludes`.

`\ltxPageLayout` Talking of analogies with `ltxdoc`, if you like only the page layout provided by that class, there is the `\ltxPageLayout` declaration (allowed only in preamble) that only changes the margins and the text width (it's intended to be used with the default paper size). This declaration is contained in the `\ltxLookSetup` declaration.

`\AtBeginInput` If you need to add something at the beginning of the input of file, there's the `\AtBeginInput` declaration that takes one and mandatory argument which is the stuff to be added. This declaration is global. It may be used more than one time and the arguments of each occurrence of it add up and are put at the beginning of input of every subsequent files.

`\AtEndInput` Simili modo, for the end of input, there's the `\AtEndInput` declaration, also one-argument, global and cumulative.

If you need to add something at the beginning of input of only one file, put before the respective input command an `\AtBegInputOnce{<the stuff to be added>}` declaration. It's also global which means that the groups do not limit its scope but it adds its argument only at the first input succeeding it (the argument gets wrapped in a macro that's `\relaxed` at the first use). `\AtBegInputOnces` add up, too.

One more input command is `\IndexInput` (the name and idea of effect comes from `\DocInput`). It takes the same argument as `\DocInput`, the file's (path and) name with extension. (It *has* `\DocInput` inside). It works properly if the input file doesn't contain explicit `<charI>` (`^^A` is OK).

The effect of this command is typesetting of all the input file verbatim, with the code lines numbered and the CSs automatically indexed (`gmdoc.sty` options are in force).

Package Options

As many good packages, this also provides some options:

Due to best \TeX documenting traditions the codelines will be numbered. But if the user doesn't wish that, she may turn it off with the `linesnotnum` option.

However, if he agrees to have the lines numbered, she may wish to reset the counter of lines himself, e.g., when she documents many source files in one document. Then he may wish the line numbers to be reset with every `{section}`'s turn for instance. This is the rôle of the `uresetlinecount` option, which seems to be a bit obsolete however, since the `\DocInclude` command takes care of a proper reset.

Talking of line numbering further, a tradition seems to exist to number only the codelines and not to number the lines of commentary. That's the default behaviour of `gmdoc` but, if someone wants the comment lines to be numbered too, she is provided the `countalllines` option. ⁴⁴¹ Then the narration acquires a bit biblical look ;-), ⁴⁴² as shown in this short example. This option is intended ⁴⁴³ for the draft versions and it is not perfect (as if anything ⁴⁴⁴ in this package was). As you see, the lines ⁴⁴⁵ are typeset continuously with the numbers printed.

By default the `makeidx` package is loaded and initialized and the CSs occurring in the code are automatically (hyper)indexed thanks to the `hyperref` package. If the user doesn't wish to index anything, she should use the `noindex` option.

The index comes two possible ways: with the line numbers (if the lines are numbered) and that's the default, or with the page numbers, if the `pageindex` option is set.

By default, `gmdoc` excludes some 300 CSs from being indexed. They are the most common CSs, \LaTeX internal macros and \TeX primitives. To learn what CSs are excluded actually, see lines [890–990](#).

If you don't want all those exclusions, you may turn them off with the `indexallmacros` option.

If you have ambiguous feelings about whether to let the default exclusions or forbid them, see [p. 14](#) to feed this ambiguity with a couple of declarations.

In `doc` package there's a default behaviour of putting marked macro's or environment's name to a marginpar. In the standard classes it's alright but not all the classes support marginpars. That is the reason why this package enables marginparing when in standard classes, enables or disables it due to the respective option when with Marcin Woliński's classes and in any case provides the options `withmarginpar` and `nomarginpar`. So, in non-standard classes the default behaviour is to disable marginpars. If the marginpars are enabled in `gmdoc`, it will put marked control sequences and environments into marginpars (see [\TextUsage](#) etc.). These options do not affect common using marginpars, which depends on the documentclass.

codespacesblank
\CodeSpacesBlank

My suggestion is to make the spaces in the code visible except the leading ones and that's the default. But if you wish all the code spaces to be blank, I give the option `codespacesblank` reluctantly. Moreover, if you wish the code spaces to be blank only in some areas, then there's `\CodeSpacesBlank` declaration (OCSR).

The Packages Required

`gmdoc` requires (loads if they're not loaded yet) some other packages of mine, namely `gmutils`, `gmverb`, analogous to Frank Mittelbach's `shortvrb`, and `gmiflink` for conditional making of hyperlinks. It also requires `hyperref`, `multicol`, `color` and `makeidx`.

`gmverb` The `gmverb` package redefines the `\verb` command and the `verbatim` environment in such a way that `_`, `{` and `\` are breakable, the first with no 'hyphen' and the other two with the comment char as a hyphen, i.e., `{\subsequent text}` breaks into `{%`
`\subsequent text}` and `\text\mylittlemacro` breaks into `\text%`
`\mylittlemacro`.

`\verbeolOK` As the standard L^AT_EX one, my `\verb` issues an error when a line end occurs in its scope. But, if you'd like to allow line ends in short verbatims, there's the `\verbeolOK` declaration. The plain `\verb` typesets spaces blank and `\verb*` makes them visible, as in the standard version(s).

`\MakeShortVerb` Moreover, `gmverb` provides the `\MakeShortVerb` declaration that takes a one-char control sequence as the only argument and turns the char used into a short verbatim delimiter, e.g., after

`\MakeShortVerb*\|`

(as you see, the declaration has the starred version, which is for visible spaces, and non-starred for blank spaces) to get `\mylittlemacro` you may type `\mylittlemacro|` instead of `\verb+\mylittlemacro+`. Because the char used in the last example is my favourite and is used this way by DEK in The T_EXbook's format, `gmverb` provides a macro `\dekclubs` that expands to the example displayed above.

`\dekclubs`

`\DeleteShortVerb` Be careful because such active chars may interfere with other things, e.g., the `|` with the vertical line marker in `tabulars` and with the `tikz` package. If this happens, you can declare e.g., `\DeleteShortVerb\|` and the previous meaning of the char used shall be restored.

One more difference between `gmverb` and `shortvrb` is that the chars `\active`ated by `\MakeShortVerb`, behave as if they were 'other' in math mode, so you may type e.g., `$k|n$` to get $k|n$ etc.

`gmutils` The `gmutils` package provides a couple of macros similar to some basic (L^A)T_EX ones, rather strictly technical and (I hope) tricky, such as `\afterfi`, `\ifnextcat`, `\addtomacro` etc. It's this package that provides the macros for formatting of names of macros and files, such as `\cs`, `\marg`, `\pk` etc.

`hyperref` The `gmdoc` package uses a lot of hyperlinking possibilities provided by `hyperref` which is therefore probably the most important package required. The recommended situation is that the user loads `hyperref` package with his favourite options *before* loading `gmdoc`.

If she does not, `gmdoc` shall load it with *my* favourite options.

`gmiflink` To avoid an error if a (hyper)referenced label does not exist, `gmdoc` uses the `gmiflink` package. It works e.g., in the index when the codeline numbers have been changed: then they are still typeset, only not as hyperlinks but as a common text.

`multicol` To typeset the index and the change history in balanced columns `gmdoc` uses the `multicol` package that seems to be standard these days.

`color` Also the `multicol` package, required to define the default colour of the hyperlinks, seems to be standard already, and `makeidx`.

Macros for Marking the Macros

The concept (taken from `doc`) is to index virtually all the control sequences occurring in the code. `gmdoc` does that by default and needs no special command. (See below about excluding some macros from being indexed.)

The next concept (also taken from `doc`) is to distinguish some occurrences of some control sequences by putting such a sequence into a marginpar and by special formatting of its index entry. That is what I call marking the macros. `gmdoc` provides also a possibility of analogous marking for the environments' names and other sequences such as `^^A`.

This package provides two kinds of special formatting of the index entries: 'usage', with the reference number italic by default, and 'def' (in `doc` called 'main'), with the reference number roman (upright) and underlined by default. All the reference numbers, also those with no special formatting, are made hyperlinks to the page or the codeline according to the respective indexing option (see p. 10).

The macros and environments to be marked appear either in the code or in the commentary. But all the definitions appear in the code, I suppose. Therefore the 'def' marking macro is provided only for the code case. So we have the `\CodeDefine`, `\CodeUsage` and `\TextUsage` commands.

`\CodeDefine`
`\CodeUsage`
`\TextUsage`

All three take one argument and all three may be starred. The non-starred versions are intended to take a control sequence as the argument and the starred to take whatever (an environment name or a `^^A`-like and also a CS).

`\MakePrivateLetters` You don't have to bother whether `@` is a letter while documenting because even if not, these commands do make it a letter, or more precisely, they execute `\MakePrivateLetters` whatever it does: At the default settings this command makes `*` a letter, too, so a starred version of a command is a proper argument to any of the three '`\...Define/Usage` commands unstarred.

The two `\Code...` commands, if unstarred, mark the next scanned occurrence of their argument in the code. (By 'scanned occurrence' I mean a situation of the CS having been scanned in the code which happens iff its name was preceded by the char declared as `\CodeEscapeChar`). The starred versions of those commands mark just the next codeline and don't make `TEX` look for the scanned occurrence of their argument (which would never happen if the argument is not a CS). Therefore, if you want to mark a definition of an environment `foo`, you should put

```
%\CodeDefine*{foo}
```

right before the code line

```
\newenvironment{foo}{%
```

i.e., not separated by any code line. The starred versions of the `\Code...` commands are also intended to mark implicit definitions of macros, e.g., `\CodeDefine*{@foofalse}` before the line

```
\newif\if@foo.
```

They both are `\outer`.

The `\TextUsage` (one-argument) command is intended to mark usage of a verbatim occurrence of a `TEX` object in the commentary. Unlike the two `\Code...`s, it typesets its argument which means among others that the marginpar appears usually at the same line as the text you wanted to mark. This command also has the starred version primarily intended for the environments names, and secondarily for `^^A`-likes and CSs, too. Currently, the most important difference is that the unstarred version

executes `\MakePrivateLetters` while the starred does both `\MakePrivateLetters` and `\MakePrivateOthers` before reading the argument.

`\Describe` If you consider the marginpars a sort of sub(sub...)section marks, then you may wish to have a command that makes a marginpar of the desired CS (or whatever) at the beginning of its description, which may be fairly far from the first occurrence of its object. Then you have the `\Describe` command which puts its argument in a marginpar and indexes it as a ‘usage’ entry but doesn’t print it in the text. It’s `\outer`.

All four commands just described put their (`\stringed`) argument into a marginpar (if the marginpars are enabled) and create an index entry (if indexing is enabled).

`\CodeMarginize` But what if you want just to make a marginpar with macro’s or environment’s name? Then you have `\CodeMarginize` to declare what to put into a marginpar in the \TeX code (it’s `\outer`) and `\TextMarginize` to do so in the commentary. According to the spirit of this part of the interface, these commands also take one argument and have their starred versions for strings other than control sequences.

`\marginpartt` The marginpars (if enabled) are ‘reverse’ i.e., at the left margin, and their contents is flush right and typeset in a font declared with `\marginpartt`. By default, this declaration is `\let` to `\tt` but it may be advisable to choose a condensed font if there is any. Such a choice is made by `gmdocc.cls` if the Latin Modern fonts are available: in this case `gmdocc.cls` uses Latin Modern Typewriter Light Condensed.

`\gmdmarginpar` If you need to put something in a marginpar without making it typewriter font, there’s the `\gmdmarginpar` macro (that takes one and mandatory argument) that only flushes its contents right.

`\CodeDefIndex` On the other hand, if you don’t want to put a CS (or another verbatim text) in a marginpar but only to index it, then there are `\CodeDefIndex` and `\CodeUsgIndex` to declare special formatting of an entry. The unstarred versions of these commands look for their argument’s scanned occurrence in the code (the argument should be a CS), and the starred ones just take the next code line as the reference point. Both these commands are `\outer`.

`\CodeCommonIndex*` In the code all the control sequences (except the excluded ones, see below) are indexed by default so no explicit command is needed for that. But the environments and other special sequences are not and the two commands described above in their `*ed` versions contain the command for indexing their argument. But what if you wish to index a not scanned stuff as a usual entry? The `\CodeCommonIndex*` comes in rescue, starred for the symmetry with the two previous commands (without `*` it just gobbles it’s argument). It’s `\outer`.

`\TextUsgIndex` Similarly, to index a \TeX object occurring verbatim in the narrative, you have `\TextUsgIndex` and `\TextCommonIndex` commands with their starless versions for a CS argument and the starred for all kinds of the argument.

`macro` Moreover, as in `doc`, the `macro` and `environment` environments are provided. Both take one argument that should be a CS for `macro` and ‘whatever’ for `environment`. Both add the `\MacroTopsep` glue before and after their contents, and put their argument in a marginpar at the first line of their contents (since it’s done with `\strut`, you should not put any blank line (`%ed` or not) between `\begin{macro/environment}` and the first line of the contents). Then `macro` commands the first scanned occurrence of its argument to be indexed as ‘def’ entry and `environment` commands \TeX to index the argument as if it occurred in the next code line (also as ‘def’ entry).

Since it’s possible that you define a CS implicitly i.e., in such a way that it cannot be scanned in the definition (with `\csname...\endcsname` e.g.) and wrapping such a definition (and description) in an `environment` environment would look misguidedly ugly, there’s the `macro*` environment which \TeX nically is just an alias for `environment`.

(To be honest, if you give a `macro` environment a non-CS argument, it will accept it and then it'll work as `environment`.)

Index Ex/Inclusions

`\DoNotIndex` It's understandable⁴ that you don't want some control sequences to be indexed in your documentation. The `doc` package gives a brilliant solution: the `\DoNotIndex` declaration. So do I (although here, \TeX nically it's done another way). It OCSR. This declaration takes one argument consisting of a list of control sequences not to be indexed. The items of this list may be separated with commas, as in `doc`, but it's not obligatory. The whole list should come in curly braces (except when it's one-element), e.g.,

```
\DoNotIndex{\some@macros,\are*_\too\auxiliary\?}
```

(The spaces after the control sequences are ignored.) You may use as many `\DoNotIndexes` as you wish (about half as many as many CSs may be declared, because for each CS excluded from indexing a special CS is declared that stores the ban sentence). Excluding the same CS more than once makes no problem.

I assume you wish most of \LaTeX macros, \TeX primitives etc. to be excluded from your index (as I do). Therefore `gmdoc` excludes some 300 CSs by default. If you don't like it, just set the `indexallmacros` package option.

`\DoIndex` On the third hand, if you like the default exclusions in general but wish to undo just a couple of them, you are given `\DoIndex` declaration (OCSR) that removes a ban on all the CSs given in the argument, e.g.,

```
\DoIndex{\par_\@@par_\endgraf}
```

`\DefaultIndexExclusions` Moreover, you are provided the `\DefaultIndexExclusions` and `\UndoDefaultIndexExclusions` declarations that act according to their names. You may use them in any configuration with the `indexallmacros` option. Both of these declarations OCSR.

The DocStrip Directives

`gmdoc` typesets the DocStrip directives and it does it quite likely as `doc`, i.e., with math sans serif font. It does it automatically whether you use the traditional settings or the new.

Advised by my \TeX Guru, I didn't implement the module nesting recognition (MW told it's not that important.)

So far verbatim mode directive is only half-handled. That is, a line beginning with `%<<END-TAG` will be typeset as a DocStrip directive, but the closing line `%END-TAG` will be not. It doesn't seem to be hard to implement, I only receive some message it's really useful for someone.

The Changes History

The `doc`'s documentation reads:

“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>}{<YYYY/MM/DD 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`

⁴ After reading `doc`'s documentation ;-).

[command] encloses the $\langle date \rangle$ in parentheses and appends the $\langle text \rangle$ to form the printed entry in such a change history [... obsolete remark omitted].

$\backslash\text{RecordChanges}$ To cause the change information to be written out, include $\backslash\text{RecordChanges}$ in the driver[’s preamble or just in the source file (`gmdocc.cls` does it for you)]. To read in and print the sorted change history (in two columns), just put the $\backslash\text{PrintChanges}$ command as the last (commented-out, and thus executed during the documentation pass through the file) command in your package file [or in the driver]. Alternatively, this command may form one of the arguments of the $\backslash\text{StopEventually}$ command, although a change history is probably not required if only the description is being printed. The command assumes that `MakeIndex` or some other program has processed the `.glo` file to generate a sorted `.gls` file. You need a special `MakeIndex` style file; a suitable one is supplied with `doc` [and `gmdoc`], called [... **gm`glo`.ist** for `gmdoc`]. The $\backslash\text{GlossaryMin}$, $\backslash\text{GlossaryPrologue}$ and $\backslash\text{GlossaryParms}$ macros are analagous to the $\backslash\text{Index}...$ versions [see sec. [The Parameters](#) p. 18]. (The L^AT_EX ‘glossary’ mechanism is used for the change entries.)”

In `gmdoc` $\backslash\text{changes}$ is $\backslash\text{outer}$.

As mentioned in the introduction, the glossary, the changes history that is, uses a special `MakeIndex` style, `gmglo.ist`. This style declares another set of the control chars but you don’t have to worry: $\backslash\text{changes}$ takes care of setting them properly. To be precise,

$\backslash\text{MakeGlossaryControls}$ $\backslash\text{changes}$ executes $\backslash\text{MakeGlossaryControls}$ that is defined as

```
\def\actualchar{=} \def\quotechar{!}%
\def\levelchar{>} \edef\encapchar{\twelveclub}
```

Only if you want to add a control character yourself in a changes entry, to quote some char, that is (using level or encapsulation chars is not recommended since $\backslash\text{changes}$ uses them itself), use rather $\backslash\text{quotechar}$.

Before writing an entry to the `.glo` file, $\backslash\text{changes}$ checks if the date (the second mandatory = the third argument) is later than the date stored in the counter **ChangesStartDate**. You may set this counter with a

ChangesStartDate $\backslash\text{ChangesStart}\{\langle version \rangle\}\{\langle year \rangle/\langle month \rangle/\langle day \rangle\}$

declaration.

If the **ChangesStartDate** is set to a date contemporary to T_EX i.e., not earlier than September 1982⁵, then a note shall appear at the beginning of the changes history that informs the reader of omitting the earlier changes entries.

If the date stored in **ChangesStartDate** is earlier than T_EX, no notification of omitting shall be printed. This is intended for a rather tricky usage of the changes start date feature: you may establish two threads of the changes history: the one for the users, dated with four digit year, and the other for yourself only, dated with two or three digit year. If you declare

```
\ChangesStart{\langle version ? \rangle}\{1000/00/00\}
```

or so, the changes entries dated with less-than-four digit year shall be omitted and no notification shall be issued of that.

$\backslash\text{CheckSum}$ While scanning the CSs in the code, `gmdoc` counts them and prints the information about their number on the terminal and in `.log`. Moreover, you may declare $\backslash\text{CheckSum}\{\% \langle number \rangle\}$ before the code and T_EX will inform you whether the number stated by you is correct or not, and what it is. As you guess, it’s not my original idea but I took it from `doc`.

⁵ DEK in *T_EX The Program* mentions that month as of T_EX Version 0 release.

There it is provided as a tool for testing whether the file is corrupted. My T_EX Guru says it's a bit old-fashioned nowadays but I like the idea and use it to document the file's growth. For this purpose `gmdoc` types out lines like

```
% \chchange{v0.98j}{2006/10/19}{4372}
% \chchange{v0.98j}{06/10/19}{4372}
```

and you may place them at the beginning of the source file. Such a line results in setting the check sum to the number contained in the last pair of braces and in making a 'general' changes entry that states the check sum for version *<first brace>* dated *<second brace>* was *<third brace>*.

The Parameters

The `gmdoc` package provides some parameters specific to typesetting the T_EX code:

`\stanzaskip` `\stanzaskip` is a vertical space inserted when a blank (code) line is met. It's equal `0.75\medskipamount` by default (with the *entire* `\medskipamount`'s stretch- and shrink-ability). Subsequent blank code lines do not increase this space.

`\CodeTopsep` At the points where narration begins a new line after the code or an inline comment and where a new code line begins after the narration (that is not an inline comment), a `\CodeTopsep` glue is added. At the beginning and the end of a `macro` or `environment` environment a `\MacroTopsep` glue is added. By default, these two skips are set equal `\stanzaskip`.

`\UniformSkips` `\NonUniformSkips` The `\stanzaskip`'s value is assigned also to the display skips and to `\topsep`. This is done with the `\UniformSkips` declaration executed by default. If you want to change some of those values, you should declare `\NonUniformSkips` in the preamble to discard the default declaration. (To be more precise, by default `\UniformSkips` is executed twice: when loading `gmdoc` and again `\AtBeginDocument` to allow you to change `\stanzaskip` and have the other glues set due to it. `\NonUniformSkips` relaxes the `\UniformSkips`'s occurrence at `\begin{document}`.)

`\stanza` If you want to add a vertical space of `\CodeTopsep` (equal by default `\stanzaskip`), you are provided the `\stanza` command. Similarly, if you want to add a vertical space of the `\MacroTopsep` amount (by default also equal `\stanzaskip`), you are given the `\chunkskip` command. They both act analogously to `\addvspace` i.e., don't add two consecutive glues but put the bigger of them.

`\nostanza` Since `\CodeTopsep` glue is inserted automatically at each transition from the code (or code with an inline comment) to the narration and reverse, it may happen that you want not to add such a glue exceptionally. Then there's the `\nostanza` command.

`\CodeIndent` The T_EX code is indented with the `\CodeIndent` glue and a leading space increases indentation of the line by its (space's) width. The default value of `\CodeIndent` is 1.5 em.

`\TextIndent` There's also a parameter for the indent of the narration, `\TextIndent`, but you should use it only in emergency (otherwise what would be the margins for?). It's 0 sp by default.

By default, typesetting a `\DocInput/Included` file is ended with a codeline containing the text ' □'

`\EOFFMark` `\NoEOF` given by the `\EOFFMark` macro. If you don't like such an ending, you should end the source file with the `\NoEOF` macro in a comment, i.e.,

```
%<some text, why not>\NoEOF
```

This macro redefines `\EOFFMark` and suppresses the End Of File token to close the input properly. It also has the `\endinput` effect so you may put some text you don't want to document after it.

The crucial concept of **gmdoc** is to use the line end character as a verbatim group opener and the comment char, usually the %, as its delimiter. Therefore the ‘knowledge’ what char starts a commentary is for this package crucial and utterly important. The default assumption is that you use % as we all do. So, if you use another character, then you should declare it with `\CodeDelim` typing the desired char preceded by a backslash, e.g., `\CodeDelim\&`. (As just mentioned implicitly, `\CodeDelim\%` is declared by default.)

`\CodeDelim`

This declaration is always global so when- and wherever you change your mind you should express it with a new `\CodeDelim` declaration.

The starred version of `\CodeDelim` changes also the verb ‘hyphen’, the char appearing at the verbatim line breaks that is.

Talking of special chars, the escape char, `\` by default, is also very important for this package as it marks control sequences and allows automatic indexing them for instance. Therefore, if you for any reason choose another than `\` character to be the escape char, you should tell **gmdoc** about it with the `\CodeEscapeChar` declaration. As the previous one, this too takes its argument preceded by a backslash, e.g., `\CodeEscapeChar\!`. (As you may deduct from the above, `\CodeEscapeChar\` is declared by default.)

`\CodeEscapeChar`

The tradition is that in the packages @ char is a letter i.e., of catcode `11`. Frank Mittelbach in **doc** takes into account a possibility that a user wishes some other chars to be letters, too, and therefore he (F.M.) provides the `\MakePrivateLetters` macro. So do I and like in **doc**, this macro makes @ sign a letter. It also makes * a letter in order to cover the starred versions of commands.

`\MakePrivateLetters`

Analogously but for a slightly different purpose, the `\AddtoPrivateOthers` macro is provided here. It adds its argument, which is supposed to be a one-char CS, to the `\doprivateothers` list, whose rôle is to allow some special chars to appear in the marking commands’ arguments (the commands described in section [Macros for Marking the Macros](#)). The default contents of this list is `␣` (the space) and `^` so you may mark the environments names and special sequences like `^^A` safely. This list is also extended with every char that is `\MakeShortVerbed`. (I don’t see a need of removing chars from this list, but if you do, please let me know.)

`\AddtoPrivateOthers`

The line numbers (if enabled) are typeset in the `\LineNumFont` declaration’s scope, which is defined as `{\normalfont\tiny}` by default. Let us also remember, that for each counter there is a `\the<counter>` macro available. The counter for the line numbers is called `codelinenum` so the macro printing it is `\thecodelinenum`. By default we don’t change its L^AT_EX’s definition which is equivalent `\arabic{codelinenum}`.

`\LineNumFont`

`codelinenum`

Three more parameter macros, are `\IndexPrefix`, `\EntryPrefix` and `\HLPrefix`. All three are provided with the account of including multiple files in one document. They are equal (almost) `\@empty` by default. The first may store main level index entry of which all indexed macros and environments would be subentries, e.g., the name of the package. The third may or even should store a text to distinguish equal codeline numbers of distinct source files. It may be the file name too, of course. The second macro is intended for another concept, namely the one from **ltxdoc** class, to distinguish the codeline numbers from different files *in the index* by the file marker. Anyway, if you document just one file per document, there’s no need of redefining those macros, nor when you input multiple files with `\DocInclude`.

`\IndexPrefix`

`\EntryPrefix`

`\HLPrefix`

gmdoc automatically indexes the control sequences occurring in the code. Their index entries may be ‘common’ or distinguished in two (more) ways. The concept is to distinguish the entries indicating the *usage* of the CS and the entries indicating the *definition* of the CS.

`\UsgEntry`

`\DefEntry`

The special formattings of ‘usage’ and ‘def’ index entries are determined by `\UsgEntry` and `\DefEntry` one-parameter macros (the parameter shall be substituted with the ref-

erence number) and by default are defined as `\textit` and `\underline` respectively (as in `doc`).

`\CommonEntryCmd` There's one more parameter macro, `\CommonEntryCmd` that stores the name of the encapsulation for the 'common' index entries (not special) i.e., a word that'll become a CS that will be put before an entry in the `.ind` file. By default it's defined as `{\relax}` and a nontrivial use of it you may see in line 14 of the driver file of this documentation, where it makes all the index entries of the driver's code are formatted as 'usage'.

`IndexColumns` The index comes in a `multicols` environment whose columns number is determined by the `IndexColumns` counter set by default to 3. To save space, the index begins at the same page as the previous text provided there is at least `\IndexMin` of the page height free. By default, `\IndexMin = 133.0pt`.

`\IndexMin`

`\IndexPrologue` The text put at the beginning of the index is declared with a one-argument `\IndexPrologue`. Its default text at current index option you may [admire](#) on page 133. Of course, you may write your own `\IndexPrologue{<brand new index prologue>}`, but if you like the default and want only to add something to it, you are provided `\AtDIPrologue` one-argument declaration that adds the stuff after the default text. For instance, I used it to add a label and hypertarget that is referred to two sentences earlier.

`\AtDIPrologue`

`\IndexLinksBlack` By default the colour of the index entry hyperlinks is set black to let Adobe Reader work faster. If you don't want this, `\let\IndexLinksBlack\relax`. That leaves the index links colour alone and hides the text about black links from the default index prologue.

`\IndexParms`

`\gaddtomacro`

Other index parameters are set with the `\IndexParms` macro defined in line 1035 of the code. If you want to change some of them, you don't have to use `\renewcommand*% \IndexParms` and set all of the parameters: you may `\gaddtomacro\IndexParms{<only the desired changes>}`. (`\gaddtomacro` is an alias for L^AT_EX's `\g@addto@macro` provided by `gmutils`.)

At the default `gmdoc` settings the `.idx` file is prepared for the default settings of `MakeIndex` (no special style). Therefore the index control chars are as usual. But if you need to use other chars as `MakeIndex` controls, know that they are stored in the four macros: `\actualchar`, `\quotechar`, `\levelchar` and `\encapchar` whose meaning you infer from their names. Any redefinition of them *should be done in the preamble* because the first usage of them takes place at `\begin{document}` and on it depends further tests telling T_EX what characters of a scanned CS name it should quote before writing it to the `.idx` file.

`\actualchar`

`\quotechar`

`\levelchar`

`\encapchar`

`\verbatimchar`

Frank Mittelbach in `doc` provides the `\verbatimchar` macro to (re)define the `\verb`'s delimiter for the index entries of the scanned CS names etc. `gmdoc` also uses `\verbatimchar` but defines it as `{&}`. Moreover, a macro that wraps a CS name in `\verb` checks whether the wrapped CS isn't `\&` and if it is, `$` is taken as the delimiter. So there's hardly chance that you'll need to redefine `\verbatimchar`.

So strange delimiters are chosen deliberately to allow any 'other' chars in the environments names.

`\StopEventually`

`\Finale`

`\AlsoImplementation`

`\OnlyDescription`

There's a quadratus of commands taken from `doc`: `\StopEventually`, `\Finale`, `\AlsoImplementation` and `\OnlyDescription` that should be explained simultaneously (in a polyphonic song e.g.).

The `\OnlyDescription` and `\AlsoImplementation` declarations are intended to exclude or include the code part from the documentation. The point between the description and the implementation part should be marked with `\StopEventually{<the stuff to be executed anyway>}` and `\Finale` should be typed at the end of file. Then `\OnlyDescription` defines `\StopEventually` to expand to its argument followed by

`\endinput` and `\AlsoImplementation` defines `\StopEventually` to do nothing but pass its argument to `\Finale`.

The Narration Macros

`\verb` To print the control sequences' names you have the `\verb` macro and its 'shortverb' version whatever you define. But they won't work if you put them in an argument of another macro. For such a situation, or if you just prefer, `gmdoc` (`gmutils`) provides a robust command `\cs`, which takes one obligatory argument, the macro's name without the backslash, e.g., `\cs{mymacro}` produces `\mymacro`. I take account of a need of printing some other text verbatim, too, and therefore `\cs` has the first argument optional, which is the text to be typeset before the mandatory argument. It's the backslash by default, but if you wish to typeset something without the `\`, you may write `\cs[]{%not_a_macro}`. Moreover, for typesetting the environments' names, `gmdoc` (`gmutils`) provides the `\env` macro, that prints its argument verbatim and without a backslash, e.g., `\env{an_environment}` produces `an environment`.

`\pk` To print packages' names sans serif there is a `\pk` one-argument command, and the `\file` `\file` command intended for the filenames.

Because we play a lot with the `\catcodes` here and want to talk about it, there are `\catletter` `\catletter`, `\catother` and `\catactive` macros that print 11, 12 and 13 respectively to concisely mark the most used char categories.

`\catactive` I wish my self-documenting code to be able to be typeset each package separately or several in one document. Therefore I need some 'flexible' sectioning commands and here they are: `\division` and `\subdivision` so far, that by default are `\let` to be `\section` and `\subsection` if such commands are defined in the documentclass. (If not, `\division` and `\subdivision` are `\let` to be `\relax`.)

One more kind of flexibility is to allow using `mwcls` or the standard classes for the same file. There was a trouble with the number and order of the optional arguments of the original `mwcls`'s sectioning commands.

It's resolved in `gmutils` so you are free at this point, and even more free than in the standard classes: if you give a sectioning command just one optional argument, it will be the title to toc and to the running head (that's standard in `scls`⁶). If you give *two* optionals, the first will go to the running head and the other to toc. (In both cases the mandatory argument goes only to the page).

`\SetFileDiv` If you wish the `\DocIncluded` files make other sectionings than the default, you may declare `\SetFileDiv{<sec name without backslash>}`.

`gmlonely` `gmdoc.sty` provides also an environment `gmlonely` to wrap some text you think you may want to skip some day. When that day comes, you write `\skipgmlonely` before the instances of `gmlonely` you want to skip. This declaration has an optional argument which is for a text that'll appear in (stead of) the first `gmlonely`'s instance in every `\DocInput` or `\DocIncluded` file within `\skipgmlonely`'s scope.

An example of use you may see in this documentation: the repeated passages about the installation and compiling the documentation are skipped in further chapters thanks to it.

`gmdoc` provides some \TeX -related logos:

`\AmSTeX` typesets $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{T}\mathcal{E}\mathcal{X}$,
`\BibTeX` $\text{BIB}\mathcal{T}\mathcal{E}\mathcal{X}$,
`\SLiTeX` $\text{SL}\mathcal{T}\mathcal{E}\mathcal{X}$,
`\PlainTeX` $\text{PLAIN}\mathcal{T}\mathcal{E}\mathcal{X}$,

⁶ See `gmutils` for some subtle details.

`\Web` `WEB`,
`\TeXbook` `The TEXbook`,
`\eTeX` `εTEX`,
`\pdfTeX` `pdfεTEX`
`\pdfTeX` `pdfTEX`
`\XeTeX` `XETEX` (the first E will be reversed iff the `graphics` package is loaded) and
`\LaTeXpar` `(LA)TEX`.

(The last logo is defined in `gmutils`.)

`\ds` `DocStrip` not quite a logo, but still convenient.

`copyrnote` The `copyrnote` environment is provided to format the copyright note flush left in `\obeylines`' scope.

`\gmdmarginpar` To put an arbitrary text into a `marginpar` and have it flushed right just like the macros' names, you are provided the `\gmdmarginpar` macro that takes one and mandatory argument which is the contents of the `marginpar`.

`\stanza` To make a vertical space to separate some piece of text you are given two macros:
`\chunkskip` `\stanza` and `\chunkskip`. The first adds `\stanzaskip` while the latter `\MacroTopsep`.
Both of them take care of not cumulating the vspaces.

`quotation` The `quotation` environment is redefined just to enclose its contents in double quotes.

`\GetFileInfo` The `\GetFileInfo{<file name with extension>}` command defines `\filedate`, `\file-`
`\filedate` `version` and `\fileinfo` as the respective pieces of the info (the optional argument)
`\fileversion` provided by `\ProvidesClass/Package/File` declarations. The information of the file
`\fileinfo` you process with `gmdoc` is provided (and therefore gettable) if the file is also loaded (or
the `\Provide...` line occurs in a `\StraightEOL` scope).

If you don't load the file the version of which you'd like to get, you may write
`\FileInfo` `\FileInfo` in the comment layer, before the line containing `\ProvidesPackage/Class`
and the file info will be written to the `.aux` file.

If the input file doesn't contain `\Provides...` in the code layer, there are commands
`\ProvideFileInfo` `\ProvideFileInfo{<file name with extension>}[<info>]` and `\ProvideSelfInfo[<info>]`.
`\ProvideSelfInfo` (`<info>` should consist of: `<year>/<month>/<day>_<version number>_<a short note>`.)

`\filenote` A macro for the standard note is provided, `\filenote`, that expands to "This file has
version number `<version number>` dated `<date>`." To place such a note in the document's
`\thfileinfo` title (or heading, with `\DocInclude` at the default settings), there's `\thfileinfo` macro
that puts `\fileinfo` in `\thanks`.

`\gmdnoindent` Since `\noindent` didn't want to cooperate with my code and narration layers sometimes, I provide `\gmdnoindent` that forces a not indented paragraph if `\noindent` could not.

`\CDPerc` If you declare the code delimiter other than `%` and then want `%` back, you may write
`\CDPerc` instead of `\CodeDelim\%`.

`\CDAnd` If you like `&` as the code delimiter (as I did twice), you may write `\CDAnd` instead of
`\CodeDelim&`.

A Queerness of `\label`

You should be loyally informed that `\label` in `gmdoc` behaves slightly non-standard in the `\DocInput/Included` files: the automatic redefinitions of `\ref` at each code line are *global* (since the code is typeset in groups and the `\refs` will be out of those groups), so a `\reference` in the narrative will point at the last code line not the last section, *unlike* in the standard L^AT_EX.

doc-Compatibility

One of my goals while writing `gmdoc` was to make compilation of doc-like files with `gmdoc` possible. I cannot guarantee the goal has been reached but I *did* compile `doc.dtx` with not a smallest change of that file (actually, there was a tiny little buggie in line 3299 which I fixed remotely with `\AfterMacrocode` tool written specially for that). So, if you wish to compile a doc-like file with my humble package, just try.

The doc commands most important in my opinion are supported by `gmdoc`. Some commands, mostly the obsolete in my opinion, are not supported but give an info on the terminal and in `.log`.

I assume that if one wishes to use doc's interface then he won't use `gmdoc`'s options but just the default. (Some `gmdoc` options may interfere with some doc commands, they may cancel them e.g.)

`\OldDocInput` The main input commands compatible with doc are `\OldDocInput` and `\DocInclude`,
`\DocInclude` the latter however only in the `\olddocIncludes` declaration's scope.
`\olddocIncludes` Within their scope/argument the `macrocode` environments behave as in doc, i.e. they
`macrocode` are a kind of verbatim and require to be ended with `%\end{macrocode(*)}`.

The default behaviour of `macrocode(*)` with the 'new' input commands is different however. Remember at the 'new' fashion the code and narration layers philosophy is in force and that is sustained within `macrocode(*)`. Which means basically that with 'new' settings when you write

```
% \begin{macrocode}
  \alittlemacro % change it to \blaargh
%\end{macrocode}
```

and `\blaargh`'s definition is `{foo}`, you'll get

```
\alittlemacro% change it to foo
```

(Note that 'my' `macrocode` doesn't require the magical `%\end{macrocode}`.)

`oldmc` If you are used to the traditional (doc's) `macrocode` and still wish to use `gmdoc` new
way, you have at least two options: there is the `oldmc` environment analogous to the
traditional (doc's) `macrocode` (it also has the starred version), that's the first option
(I needed the traditional behaviour once in this documentation, find out where & why).
`\VerbMacrocodes` The other is to write `\VerbMacrocodes`. That declaration (OCSR) redefines `macrocode`
and `macrocode*` to behave the traditional way. (It's always executed by `\OldDocInput`
and `\olddocIncludes`.)

For a more detailed discussion of what is doc-compatible and how, see the code section [doc-Compatibility](#).

The Code

The basic idea of this package is to re\catcode ^~M (the line end char) and % (or any other comment char) so that they start and finish typesetting of what's between them as the `TEX` code i.e., verbatim and with the bells and whistles.

The bells and whistles are (optional) numbering of the codelines, and automatic indexing the CSs, possibly with special format for the 'def' and 'usage' entries.

As mentioned in the preface, this package aims at a minimal markup of the working code. A package author writes her splendid code and adds a brilliant comment in `%ed` lines and that's all. Of course, if he wants to make a `\section` or `\emphasise`, she has to type respective CSs.

I see the feature described above to be quite a convenience, however it has some price. See section [Life Among Queer EOLs](#) for details, here I state only that in my opinion the price is not very high.

More detailedly, the idea is to make `^~M` (end of line char) active and to define it to check if the next char i.e., the beginnig of the next line is a `%` and if so to gobble it and just continue usual typesetting or else to start a verbatim scope. In fact, every such a line end starts a verbatim scope which is immediately closed, if the next line begins with (leading spaces and) the code delimiter.

Further details are typographical parameters of verbatim scope and how to restore normal settings after such a scope so that a code line could be commented and still displayed, how to deal with leading spaces, how to allow breaking a moving argument in two lines in the comment layer, how to index and marginpar macros etc.

The Package Options

Maybe someone wants the code lines not to be numbered.

```

5 \newif\if@linesnotnum
linesnotnum 6 \DeclareOption{linesnotnum}{\@linesnotnumtrue}

```

And maybe he or she wishes to declare resetting the line counter along with some sectioning counter him/herself.

```

7 \newif\if@uresetlinecount
uresetlinecount 8 \DeclareOption{uresetlinecount}{\@uresetlinecounttrue}

```

And let the user be given a possibility to count the comment lines.

```

9 \newif\if@countalllines
countalllines 10 \DeclareOption{countalllines}{\@countalllinestrue}

```

Unlike in `doc`, indexing the macros is the default and the default reference is the code line number.

```

11 \newif\if@noindex
noindex 12 \DeclareOption{noindex}{\@noindextrue}

13 \newif\if@pageindex
pageindex 14 \DeclareOption{pageindex}{\@pageindextrue}

```

It would be a great honour to me if someone would like to document L^AT_EX source with this humble package but I don't think it's really probable so let's make an option that'll switch index exclude list properly (see sec. [Index Exclude List](#)).

```

15 \newif\if@indexallmacros
indexallmacros 16 \DeclareOption{indexallmacros}{\@indexallmacrostrue}

```

Some document classes don't support marginpars or disable them by default (as my favourite Marcin Woliński's classes).

```

17 \@ifundefined{if@marginparsused}{\newif\if@marginparsused}{}

```

This switch is copied from `mwbc.cls` for compatibility with it. Thanks to it loading an `mwcls` with `[withmarginpar]` option shall switch marginpars on in this package, too.

To be compatible with the standard classes, let's `\let`:

```

18 \@ifclassloaded{article}{\@marginparsusedtrue}{}
19 \@ifclassloaded{report}{\@marginparsusedtrue}{}
20 \@ifclassloaded{book}{\@marginparsusedtrue}{}

```


And if you don't use `mwcls` nor standard classes, then you have the options:

```
withmarginpar 21 \DeclareOption{withmarginpar}{\@marginparsusedtrue}
nomarginpar   22 \DeclareOption{nomarginpar}{\@marginparsusedfalse}
```

The order of the above conditional switches and options is significant. Thanks to it the options are available also in the standard classes and in `mwcls`.

To make the code spaces blank (they are visible by default except the leading ones).

```
codespacesblank 23 \newif\if@codespacesblank
                24 \DeclareOption{codespacesblank}{\@codespacesblanktrue}
                25 \ProcessOptions
```

The Dependencies and Preliminaries

We require another package of mine that provides some tricky macros analogous to the L^AT_EX standard ones, such as `\newgif` and `\@ifnextcat`.

```
26 \RequirePackage{gmutils}[2007/04/24]
```

A standard package for defining colours,

```
27 \RequirePackage{color}
```

and a colour definition for the hyperlinks not to be too bright

```
28 \definecolor{deepblue}{rgb}{0,0,.85}
```

And the standard package probably most important for `gmdoc`: If the user doesn't load `hyperref` with his favourite options, we do, with *ours*. If she has done it, we change only the links' colour.

```
29 \@ifpackageloaded{hyperref}{\hypersetup{colorlinks=true,
30   linkcolor=deepblue,\urlcolor=blue,\filecolor=blue}}{%
31   \RequirePackage[colorlinks=true,\linkcolor=deepblue,\urlcolor=blue,
32   filecolor=blue,\pdfstartview=FitH,\pdfview=FitBH,
33   pdfpagemode=None]{hyperref}}
```

Now a little addition to `hyperref`, a conditional hyperlinking possibility with the `\gmhypertarget` and `\gmiflink` macros. It *has* to be loaded *after* `hyperref`.

```
34 \RequirePackage{gmiflink}
```

And a slight redefinition of `verbatim`, `\verb(*)` and providing of `\MakeShortVerb(*)`.

```
35 \RequirePackage{gmverb}[2007/04/24]
36 \if@noindex
37   \AtBeginDocument{\gag@index}% for the latter macro see line 706.
38 \else
39   \RequirePackage{makeidx}\makeindex
40 \fi
```

Now, a crucial statement about the code delimiter in the input file. Providing a special declaration for the assignment is intended for documenting the packages that play with %'s `\catcode`. Some macros for such plays are defined [further](#).

The declaration comes in the starred and unstarred version. The unstarred version besides declaring the code delimiter declares the same char as the `verb(atim)` 'hyphen'. The starred version doesn't change the `verb` 'hyphen'. That is intended for the special tricks e.g. for the `oldmc` environment.

If you want to change the verb ‘hyphen’, there is the `\VerbHyphen` declaration provided by `gmverb`.

```
\CodeDelim 41 \def\CodeDelim{\@ifstar\Code@Delim@St\Code@Delim}
42 \def\Code@Delim#1{%
43   {\escapechar\m@ne
\code@delim 44   \@xa\gdef\@xa\code@delim\@xa{\string#1}}}
(\@xa is \expandafter, see gmutils.)
45 \def\Code@Delim@St#1{\Code@Delim{#1}\VerbHyphen{#1}}
```

It is an invariant of `gmdocing` that `\code@delim` stores the current code delimiter (of catcode 12).

The `\code@delim` should be 12 so a space is not allowed as a code delimiter. I don’t think it *really* to be a limitation.

And let’s assume you do as we all do:

```
46 \CodeDelim\%
```

We’ll play with `\everypar`, a bit, and if you use such things as the `{itemize}` environment, an error would occur if we didn’t store the previous value of `\everypar` and didn’t restore it at return to the narration. So let’s assign a `\toks` list to store the original `\everypar`

```
47 \newtoks\gmd@preverypar
48 \newcommand*\settetcodehangi{%
49   \hangindent=\verbatimhangindent_\hangafter=\@ne}% we’ll use it in the inline
comment case. \verbatimhangindent is provided by the gmverb package
and = 3em by default.
50 \@ifdefinable\@@settetcodehangi{\let\@@settetcodehangi=%
\settetcodehangi}
```

We’ll play a bit with `\leftskip`, so let the user have a parameter instead. For normal text (i.e. the comment):

```
\TextIndent 51 \newlength\TextIndent
```

I assume it’s originally equal `\leftskip`, i.e. `\z@`. And for the `TEX` code:

```
52 \newlength\CodeIndent
\CodeIndent 53 \CodeIndent=1,5em\relax
```

And the vertical space to be inserted where there are blank lines in the source code:

```
54 \@ifundefined{stanzaskip}{\newlength\stanzaskip}{}
```

I use `\stanzaskip` in `gmverse` package and derivatives for typesetting poetry. A computer program code *is* poetry.

```
\stanzaskip 55 \stanzaskip=\medskipamount
56 \advance\stanzaskip_\by-.25\medskipamount% to preserve the stretch- and shrink-
ability.
```

A vertical space between the commentary and the code seems to enhance readability so declare

```
57 \newskip\CodeTopsep
58 \newskip\MacroTopsep
```


And let's set them. For the æsthetical minimality⁷ let's unify them and other most important vertical spaces used in `gmdoc`. I think a macro that gathers all these assignments may be handy.

```
\UniformSkips 59 \def\UniformSkips{%
\CodeTopsep    60 \CodeTopsep=\stanzaskip
\MacroTopsep   61 \MacroTopsep=\stanzaskip
               62 \abovedisplayskip=\stanzaskip
               %\abovedisplayshortskip remains untouched as it is 0.0 pt plus 3.0 pt by default.
               63 \belowdisplayskip=\stanzaskip
               64 \belowdisplayshortskip=.5\stanzaskip% due to DEK's idea of making the
               short below display skip half of the normal.
               65 \advance\belowdisplayshortskip_\by\smallskipamount
               66 \advance\belowdisplayshortskip_\by-1\smallskipamount% We advance \below-
               %displayshortskip forth and back to give it the \smallskipamount's
               shrink- and stretchability components.
               67 \topsep=\stanzaskip
               68 \partopsep=\z@
               69 }
```

We make it the default,

```
70 \UniformSkips
```

but we allow you to change the benchmark glue i.e., `\stanzaskip` in the preamble and still have the other glues set due to it: we launch `\UniformSkips` again after the preamble.

```
71 \AtBeginDocument{\UniformSkips}
```

So, if you don't want them at all i.e., you don't want to set other glues due to `\stanzaskip`, you should use the following declaration. That shall discard the unwanted setting already placed in the `\begin{document}` hook.

```
\NonUniformSkips 72 \newcommand*\NonUniformSkips{@\relaxen\UniformSkips}
```

Why do we launch `\UniformSkips` twice then? The first time is to set all the `gmdoc`-specific glues *somehow*, which allows you to set not all of them, and the second time to set them due to a possible change of `\stanzaskip`.

And let's define a macro to insert a space for a chunk of documentation, e.g., to mark the beginning of new macro's explanation and code.

```
\chunkskip 73 \newcommand*\chunkskip{%
              74 \skip0=\MacroTopsep
              75 \if@codeskipput\advance\skip0_\by-\CodeTopsep\fi
              76 \par\addvspace{\skip0}\@codeskipputgtrue}
```

And, for a smaller part of text,

```
\stanza 77 \newcommand*\stanza{%
          78 \skip0=\stanzaskip
          79 \if@codeskipput\advance\skip0_\by-\CodeTopsep\fi
          80 \par\addvspace{\skip0}\@codeskipputgtrue}
```

Since the stanza skips are inserted automatically most often (cf. lines [193](#), [361](#), [202](#), [320](#), [377](#)), sometimes you may need to forbid them.

⁷ The terms 'minimal' and 'minimalist' used in `gmdoc` are among others inspired by the *South Park* cartoon's episode *Mr. Hankey The Christmas (...)* in which 'Philip Glass, a Minimalist New York composer' appears in a 'non-denominational non-offensive Christmas play' ;-). (Philip Glass composed the music to the *Qatsi* trilogy among others)

```
\nostanza 81 \newcommand*\nostanza{%
82   \@codeskipputgtrue\@afternarrgfalse\@aftercodegtrue}% In the ‘code to
narration’ case the first switch is enough but in the counter case ‘narration
to code’ both the second and third are necessary while the first is not.
```

To count the lines where they have begun not before them

```
\@newlinegtrue 83 \newgif\if@newline
\@newlinegfalse \newgif is \newif with global effect i.e., it defines \...gtrue and \...gfalse switch-
\if@newline ers that switch respective Boolean switch globally. See gmutils package for details.
```

To handle the DocStrip directives not *any* %<....

```
\if@dsdir 84 \newgif\if@dsdir
```

This switch will be falsified at the first char of a code line. (We need a switch independent of the one indicating whether the line has or has not been counted because of two reasons: 1. line numbering is optional, 2. counting the line falsifies that switch *before* the first char.)

The Core

Now we define main \inputing command that'll change catcodes. The macros used by it are defined later.

```
\DocInput 85 \newcommand*\DocInput{\bgroup\@makeother\_ \Doc@Input}
\Doc@Input 86 \begingroup\catcode'\^M=\active%
87 \firstofone{\endgroup%
88   \newcommand*\Doc@Input}[1]{\egroup\begingroup%
89     \edef\gmd@inputname{#1}% we'll use it in some notifications.
90     \let\gmd@currentlabel@before=\@currentlabel% we store it 'cause we'll do
\code of \@currentlabel to make proper references to the line numbers
so we want to restore current \@currentlabel after our group.
91     \gmd@setclubpenalty% we wrapped the assignment of \clubpenalty in a macro
because we'll repeat it twice more.
92     \@clubpenalty\clubpenalty_\widowpenalty=3333_\code Most paragraphs of the
code will be one-line most probably and many of the narration, too.
93     \tolerance=1000_\code as in doc.
94     \if@codespacesblank\CodeSpacesBlank\fi% The default is that the code
spaces are visible but here this may be cancelled due to the \codespa-
% cesblank option.
95     \catcode'\^M=\active%
96     \@xa\@makeother\csname\code@delim\endcsname%
97     \gmd@resetlinecount% due to the option uresetlinecount we reset the
linenumber counter or do nothing.
98     \@beginputhook% my first use of it is to redefine \maketitle just at this point
not globally.
99     \everypar=\@xa{\@xa\@codetonarrskip\the\everypar}%
^M 100 \let^M=\gmd@textEOL%
101 \edef\gmd@guardedinput{%
102   \@nx\@input_\code#1\relax% \@nx is \noexpand, see gmutils. \@input is the
true TEX's \input.
103   \gmd@iihook% cf. line 1556
104   \@nx\EOFMark% to pretty finish the input, see line 151.
```

```

105      \@nx\CodeDelim\@xa\@nx\csname\code@delim\endcsname% to ensure the
        code delimiter is the same as at the beginning of input.
106      \@nx^^M\code@delim%
107      }% we add guardians after \inputing a file; somehow an error occurred without
        them.
108      \catcode'\%=9\% for doc-compatibility.
109      \setcounter{Checksum}{0}% we initialize the counter for the number of the
        escape chars (the assignment is \global).
110      \@xa\@xa\@xa^^M\gmd@guardedininput%
111      \par%
112      \@endinghook% It's a hook to let postpone some stuff till the end of input.
        We use it e.g. for the doc-(not)likeliness notifications.
113      \glet\@currentlabel=\gmd@currentlabel@before% we restore value from be-
        fore this group. In a very special case this could cause unexpected be-
        haviour of crossrefs, but anyway we acted globally and so acts hyperref.
114      \endgroup%
115      }% end of \Doc@Input's definition.
116      }% end of \firstofone's argument.

```

So, having the main macro outlined, let's fill in the details.

First, define the queer EOL. We define a macro that ^^M will be let to. \gmd@textEOL will be used also for checking the %^^M case (\@ifnextchar does \ifx).

```

\gmd@textEOL 117 \def\gmd@textEOL{\_ a space just like in normal TeX. We put it first to cooperate
        with ^^M's \expandafter\ignorespaces. It's no problem since a space _10
        doesn't drive TeX out of the vmode.
118      \ifhmode\@afternarrgtrue\@codeskipputgfalse\fi% being in the horizontal
        mode means we've just typeset some narration so we turn the respec-
        tive switches: the one bringing the message 'we are after narration' to
        True (@afternarr) and the 'we have put the code-narration glue' to False
        (@codeskipput). Since we are in a verbatim group and the information
        should be brought outside it, we switch the switches globally (the letter g
        in both).
119      \@newlinegtrue% to \refstep the lines' counter at the proper point.
120      \@dsdirgtrue% to handle the DocStrip directives.
121      \@xa\@trimandstore\the\everypar\@trimandstore% we store the previous value
        of \everypar register to restore it at a proper point. See line 390 for the
        details.
122      \begingroup%
123      \gmd@setclubpenalty% Most paragraphs will be one-line most probably. Since
        some sectioning commands may change \clubpenalty, we set it again here
        and also after this group.
124      \aftergroup\gmd@setclubpenalty%
125      \let\par\@@par% inside the verbatim group we wish \par to be genuine.
126      \ttverbatim% it does \tt and makes specials other or \active-and-breakable.
127      \gmd@DoTeXCodeSpace%
128      \@makeother\|%'cause \ttverbatim doesn't do that.
129      \MakePrivateLetters% see line 451.
130      \@xa\@makeother\code@delim% we are almost sure the code comment char is
        among the chars having been _12ed already. For 'almost' see the \IndexInput
        macro's definition.

```

So, we've opened a verbatim group and want to peek at the next character. If it's %, then we just continue narration, else we process the leading spaces supposed there are any and, if after them is a %, we just continue the commentary as in the previous case or else we typeset the \TeX code.

```

131 \xa@ifnextchar\xa{\code@delim}{%
132 \gmd@continuenarration}{%
133 \gmd@dolspaces% it will launch \gmd@typesettexcode.
134 }% end of \a@ifnextchar's else.
135 }% end of \gmd@textEOL's definition.
136 \def\gmd@setclubpenalty{\clubpenalty=3333\ }

```

For convenient adding things to the begin- and endinput hooks:

```

\AtEndInput 137 \def\AtEndInput{\g@addto@macro\@endinputhook}
\@endinputhook 138 \def\@endinputhook{}

Simili modo

\AtBegInput 139 \def\AtBegInput{\g@addto@macro\@begininputhook}
\@begininputhook 140 \def\@begininputhook{}

```

For the index input hooking now declare a macro, we define it another way at line 1556.

```

141 \emptify\gmd@iihook

```

And let's use it instantly to avoid a disaster while reading in the table of contents.

```

\tableofcontents 142 \AtBegInput{\let\gmd@@toc\tableofcontents
143 \def\tableofcontents{%
144 \a@ifQueerEOL{\StraightEOL\gmd@@toc\QueerEOL}{\gmd@@toc}}

```

As you'll learn from lines 414 and 409, we use those two strange declarations to change and restore the very special meaning of the line end. Without such changes `\tableofcontents` would cause a disaster (it did indeed). And to check the catcode of \sim is the rôle of `\a@ifQueerEOL`:

```

\@ifQueerEOL 145 \long\def\@ifQueerEOL#1#2{%
146 \ifnum\the\catcode'\sim=\active\afterfi{#1}\else\afterfi{#2}\fi}

```

The declaration below is useful if you wish to put sth. just in the nearest input/included file and no else: at the moment of putting the stuff it will erase it from the hook. You may declare several `\AtBegInputOnces`, they add up.

```

\gmd@ABIOnce 147 \emptify\gmd@ABIOnce
148 \AtBegInput\gmd@ABIOnce

\AtBegInputOnce 149 \long\def\AtBegInputOnce#1{%
150 \gaddtomacro\gmd@ABIOnce{\g@emptify\gmd@ABIOnce#1}}

```

Many tries of finishing the input cleanly led me to setting the guardians as in line 106 and to

```

\EOFMark 151 \def\EOFMark{\<eof>}

```

Other solutions did print the last code delimiter or would require managing a special case for the macros typesetting \TeX code to suppress the last line's numbering etc.

If you don't like it, see line 1728.

Due to the `codespacesblank` option in the line 94 we launch the macro defined below to change the meaning of a `gmdoc-kernel` macro.

```

152 \begin{obeyspaces}%
153 \gdef\gmd@DoTeXCodeSpace{%
154 \obeyspaces\let_\=\breakabletwelvespace}%
\CodeSpacesBlank 155 \gdef\CodeSpacesBlank{%
156 \let\gmd@DoTeXCodeSpace\gmobeyspaces%
157 \let\gmd@texcodespace=_}% the latter \let is for the \if...s.
\CodeSpacesSmall 158 \gdef\CodeSpacesSmall{%
159 \def\gmd@DoTeXCodeSpace{%
160 \obeyspaces\def_{\,\hskip\z@}}%
161 \def\gmd@texcodespace{\,\hskip\z@}}%
162 \end{obeyspaces}

```

How the continuing of the narration should look like?

```

163 \def\gmd@continuenarration{%
164 \endgroup
165 \gmd@countnarrationline% see below.
166 \@xa\@trimandstore\the\everypar\@trimandstore
167 \everypar=\@xa{\@xa\@codetonarrskip\the\everypar}%
168 \@xa\gmd@checkifEOL\@gobble}

```

Simple, isn't it? (We gobble the 'other' code delimiter. Despite of `\egroup` it's 12 because it was touched by `\futurelet` contained in `\@ifnextchar` in line 131. And in line 245 it's been read as 12. That's why it works in spite of that % is of category 'ignored'.)

Whether we count the narration lines depends on the option `countalllines` which is off by default.

```

169 \if@countalllines
170 \def\gmd@countnarrationline{%
171 \if@newline
172 \grefstepcounter{codelinenum}\@newlinegfalse% the \grefstepcounter
macro, defined in gmverb, is a global version of \refstepcounter, ob-
serving the redefinition made to \refstepcounter by hyperref.
173 \everypar=\@xa{%
174 \@xa\@codetonarrskip\the\gmd@preverypar}% the \hyperlabel@line macro
puts a hypertarget in a \raise i.e., drives TEX into the horizontal
mode so \everypar shall be issued. Therefore we should restore it.
175 \hyperlabel@line
176 {\LineNumFont\thecodelinenum}\,\ignorespaces
177 \fi}%
178 \else
179 \@emptify\gmd@countnarrationline%
180 \fi

```

And typesetting the T_EX code?

```

181 \begingroup\catcode'\^M=\active%
182 \firstofone{\endgroup%
183 \def\gmd@typesettexcode{%
184 \gmd@parfixclosingspace% it's to eat a space closing the paragraph, see below.
It contains \par. A verbatim group has already been opened by \ttverb-
% atim and additional \catcode.

```

```

185 \everypar={\@@settetcodehangl}% At first attempt we thought of giving the
      user a \toks list to insert at the beginning of every code line, but what
      for?
^^M 186 \def^^M{%
187     \@newlinegtrue% to \refstep the counter in proper place.
188     \@dsdirgtrue% to handle the DocStrip directives.
189     \global\gmd@closingspacewd=\z@% we don't wish to eat a closing space after
      a codeline, 'cause there isn't any and a negative rigid \hskip added to
      \parfillskip would produce a blank line.
190     \ifhmode\par\@codeskipputgfalse\else%
191         \if@codeskipput%
192         \else\addvspace{\stanzaskip}\@codeskipputgtrue%
193         \fi% if we've just met a blank (code) line, we insert a \stanzaskip glue.
194     \fi%
195     \prevhmodegfalse% we want to know later that now we are in the vmode.
196     \ifnextchar{\gmd@texcodespace}{%
197         \@dsdirgfalse\gmd@dolspaces}{\gmd@charbychar}%
198     }% end of ^^M's definition.
199     \let\gmd@texcodeEOL=^^M% for further checks inside \gmd@charbychar.
200     \raggedright\leftskip=\CodeIndent%
201     \if@aftercode\gmd@nocodeskip1{iaC}\else\if@afternarr%
202         \if@codeskipput\else\gmd@codeskip1\@codeskipputgtrue%
            \@aftercodegfalse\fi%
203     \else\gmd@nocodeskip1{naN}\fi\fi% if now we are switching from the narra-
      tion into the code, we insert a proper vertical space.
204     \@aftercodegtrue\@afternarrgfalse%
205     \ifdim\gmd@ldspaceswd>\z@% and here the leading spaces.
206         \leavevmode\@dsdirgfalse%
207         \if@newline\grefstepcounter{codelinenum}\@newlinegfalse%
208         \fi%
209         \printlinenumber% if we don't want the lines to be numbered, the respective
            option \lets this CS to \relax.
210         \hyperlabel@line%
211         \mark@envir% index and/or marginize an environment if there is some to be
            done so, see line 658.
212         \hskip\gmd@ldspaceswd%
213         \advance\hangindent_\by\gmd@ldspaceswd%
214         \xdef\settetcodehangl{%
215             \@nx\hangindent=\the\hangindent% and also set the hanging indent set-
                ting for the same line comment case. BTW., this % or rather lack of
                it costed me five hours of debugging and rewriting. Active lineends
                require extreme caution.
216             \@nx\hangafter=1\space}%
217     \else%
218         \glet\settetcodehangl=\@@settetcodehangl%
            % \printlinenumber here produced line numbers for blank lines which
            is what we don't want.
219         \fi% of \ifdim
220         \gmd@ldspaceswd=\z@%
221         \prevhmodegfalse% we have done \par so we are not in the hmode.
222         \@aftercodegtrue% we want to know later that now we are typesetting a code-

```

line.

```

223 \gmd@charbychar% we'll eat the code char by char to scan all the macros and
      thus to deal properly with the case \% in which the % will be scanned and
      won't launch closing of the verbatim group.
224 }%
225 }% end of \gmd@typesettexcode's definitions's group's \firstofone.

```

Now let's deal with the leading spaces once forever. We wish not to typeset `\s` but to add the width of every leading space to the paragraph's indent and to the hanging indent, but only if there'll be any code character not being % in this line (e.g., the end of line). If there'll be only %, we want just to continue the comment or start a new one. (We don't have to worry about whether we should `\par` or not.)

```

226 \newlength\gmd@spacewd% to store the width of a (leading) \s.
227 \newlength\gmd@ldspaceswd% to store total length of gobbled leading spaces.

```

It costed me some time to reach that in my verbatim scope a space isn't `_12` but `_13`, namely `\let` to `\breakabletwelvespace`. So let us `\let` for future:

```

228 \let\gmd@texcodespace=\breakabletwelvespace

```

And now let's try to deal with those spaces.

```

229 \def\gmd@dolspaces{%
230   \ifx\gmd@texcodespace\@let@token
231     \@dsdirfalse
232     \afterfi{\settowidth{\gmd@spacewd}{\twelvespace}}%
233     \gmd@ldspaceswd=\z@
234     \gmd@eatlspace}%
235   \else\afterfi{% about this smart macro and other of its family see gmutils sec. 3.
236     \par\gmd@typesettexcode}%
237   \fi}

```

And now, the iterating inner macro that'll eat the leading spaces.

```

238 \def\gmd@eatlspace#1{%
239   \ifx\gmd@texcodespace#1%
240     \advance\gmd@ldspaceswd_\gmd@spacewd% we don't \advance it \globally
      'cause the current group may be closed iff we meet % and then we'll won't
      indent the line anyway.
241     \afteriffifi\gmd@eatlspace
242   \else
243     \if\code@delim\@nx#1%
244       \gmd@ldspaceswd=\z@
245       \gmd@continuenarration#1%
246     \else_\afterfifi{\gmd@typesettexcode#1}%
247     \fi
248   \fi}%

```

We want to know whether we were in hmode before reading current `\code@delim`. We'll need to switch the switch globally.

```

\prevhmodegtrue 249 \newgif\ifprevhmode
\prevhmodegfalse
\ifprevhmode

```

And the main iterating inner macro which eats every single char of verbatim text to check the end. The case \% should be excluded and it is indeed.

```

\gmd@charbychar 250 \newcommand*\gmd@charbychar[1]{%

```

```

251 \ifhmode\prevhmodegtrue
252 \else\prevhmodegfalse\fi
253 \if\code@delim\@nx#1%
254 \afteriffifi{\gmd@percenthack% to typeset % if a comment continues the code-
    line.
255 \endgroup%
256 \gmd@checkifEOLmixd}% to see if next is ^^M and then do \par.
257 \else% i.e., we've not met the code delimiter
258 \if\code@escape@char\@nx#1%
259 \@dsdirgfalse% yes, just here not before the whole \if because then we
    would discard checking for DocStrip directives doable by the active %
    at the 'old macrocode' setting.
260 \afterfifi{\gmd@counttheline#1\scan@macro}%
261 \else
262 \afterfifi{\gmd@EOLorcharbychar#1}%
263 \fi
264 \fi}

```

One more inner macro because ^^M in T_EX code wants to peek at the next char and possibly launch \gmd@charbychar. We deal with counting the lines thoroughly. Increasing the counter is divided into cases and it's very low level in one case because \refstepcounter and \stepcounter added some stuff that caused blank lines, at least with hyperref package loaded.

```

265 \def\gmd@EOLorcharbychar#1{%
266 \ifx\gmd@texcodeEOL#1%
267 \if@newline
268 \if@countalllines\global\advance\c@codelinenum_\by\@ne
269 \@newlinegfalse\fi
270 \fi
271 \afterfi{#1}% here we print #1.
272 \else% i.e., #1 is not a (very active) line end,
273 \afterfi
274 {\gmd@counttheline#1\gmd@charbychar}% or here we print #1. Here we would
    also possibly mark an environment but there's no need of it because declar-
    ing an environment to be marked requires a bit of commentary and here
    we are after a code ^^M with no commentary.
275 \fi}

276 \def\gmd@counttheline{%
277 \ifvmode
278 \if@newline
279 \grefstepcounter{codelinenum}\@newlinegfalse
280 \hyperlabel@line
281 \fi
282 \printlinenumber
283 \mark@envir
284 \else
285 \if@newline
286 \grefstepcounter{codelinenum}\@newlinegfalse
287 \hyperlabel@line
288 \fi
289 \fi}

```


If before reading current % char we were in horizontal mode, then we wish to print % (or another code delimiter).

```

290 \def\gmd@percenthack{%
291   \ifprevhmode\code@delim\aftergroup\space% We add a space after %, 'cause
        I think it looks better. It's done \aftergroup to make the spaces possible
        after the % not to be typeset.
292   \else\aftergroup\gmd@narrcheckifds@ne% remember that \gmd@precenthack
        is only called when we've the code delimiter and soon we'll close the verbatim
        group and right after \endgroup there waits \gmd@checkifEOLmixd.
293   \fi}
294 \def\gmd@narrcheckifds@ne#1{%
295   \@dsdirgfalse\@ifnextchar<{%
296     \@xa\gmd@docstripdirective\@gobble}{#1}}

```

The macro below is used to look for the %~M case to make a commented blank line make a new paragraph. Long searched and very simple at last.

```

297 \def\gmd@checkifEOL{%
298   \gmd@countnarrationline
299   \everypar=\@xa{\@xa\@codetonarrskip% we add the macro that'll insert a ver-
        tical space if we leave the code and enter the narration.
300     \the\gmd@preverypar}%
301   \@ifnextchar{\gmd@textEOL}{%
302     \@dsdirgfalse\par\ignorespaces}{\gmd@narrcheckifds}}%

```

We check if it's %<, a DocStrip directive that is.

```

303 \def\gmd@narrcheckifds{%
304   \@dsdirgfalse\@ifnextchar<{%
305     \@xa\gmd@docstripdirective\@gobble}{\ignorespaces}}

```

In the 'mixed' line case it should be a bit more complex, though. On the other hand, there's no need to checking for DocStrip directives.

```

306 \def\gmd@checkifEOLmixd{%
307   \gmd@countnarrationline
308   \everypar=\@xa{\@xa\@codetonarrskip\the\gmd@preverypar}%
309   \@afternarrgfalse\@aftercodegtrue
310   \ifhmode\@codeskipputgfalse\fi
311   \@ifnextchar{\gmd@textEOL}{%
312     {\raggedright\gmd@endpe\par}% without \raggedright this \par would be
        justified which is not appropriate for a long codeline that should be broken,
        e.g., 308.
313     \prevhmodegfalse
314     \gmd@endpe\ignorespaces}{%

```

If a codeline ends with % (prevhmode == True) first \gmd@endpe sets the parameters at the T_EX code values and \par closes a paragraph and the latter \gmd@endpe sets the parameters at the narration values. In the other case both \gmd@endpes do the same and \par between them does nothing.

```

315   \def\par{%
316     \ifhmode% (I added this \ifhmode as a result of a heavy debug.)
317       @@par
318       \if@afternarr

```

```

319         \if@aftercode
320         \if@codeskipput\else\gmd@codeskip2\@aftercodegfalse%
            \@codeskipputgtrue\fi
321         \else\gmd@nocodeskip2{naC}%
322         \fi
323         \else\gmd@nocodeskip2{naN}%
324         \fi
325         \prevhmodegfalse\gmd@endpe% when taken out of \ifhmode, this line
            caused some codeline numbers were typeset with \leftskip = 0.
326         \everypar=\@xa{%
327             \@xa\@codetonarrskip\the\gmd@preverypar}%
328         \let\par\@@par%
329         \fi}%
330     \gmd@endpe\ignorespaces}}

```

As we announced, we play with `\leftskip` inside the verbatim group and therefore we wish to restore normal `\leftskip` when back to normal text i.e. the commentary. But, if normal text starts in the same line as the code, then we still wish to indent such a line.

```

331 \def\gmd@endpe{%
332     \ifprevhmode
333         \settcodehangin%ndent
334         \leftskip=\CodeIndent
335     \else
336         \leftskip=\TextIndent
337         \hangindent=\z@
338         \everypar=\@xa{%
339             \@xa\@codetonarrskip\the\gmd@preverypar}%
340     \fi}

```

Numbering (or Not) of the Lines

Maybe you want codelines to be numbered and maybe you want to reset the counter within sections.

```

341 \if@uresetlinecount% with uresetlinecount option...
342     \@relaxen\gmd@resetlinecount% ... we turn resetting the counter by \DocInput
        off...
343     \newcommand*\resetlinecountwith[1]{%
344         \newcounter{codelinenum}[#1]}% ... and provide a new declaration of the
        counter.
345 \else% With the option turned off...
DocInputsCount 346     \newcounter{DocInputsCount}%
codelinenum    347     \newcounter{codelinenum}[DocInputsCount]% ... we declare the \DocInputs'
        number counter andthe codeline counter to be reset with stepping of it.
\gmd@resetlinecount 348     \newcommand*\gmd@resetlinecount{\stepcounter{DocInputsCount}}% ... and
        let the \DocInput increment the \DocInputs number count and thus reset
        the codeline count. It's for unique naming of the hyperref labels.
349 \fi

```

Let's define printing the line number as we did in `gmvb` package.

```

350 \newcommand*\printlinenumber{%

```

```

351 \leavevmode\llap{\rlap{\LineNumFont$\phantom{999}$}\llap{%
      \thecodelinenum}}}%
352 \hskip\leftskip}}
\LineNumFont 353 \def\LineNumFont{\normalfont\tiny}
354 \if@linesnotnum\@relaxen\printlinenumber\fi
355 \newcommand*\hyperlabel@line{%
356 \if@pageindex% It's good to be able to switch it any time not just define it once
      according to the value of the switch set by the option.
357 \else
358 \raisebox{2ex}[1ex][\z@]{\gmhypertarget[clnum.%
359 \HLPrefix\arabic{codelinenum}]}{}}%
360 \fi}

```

Spacing with \everypar

Last but not least, let's define the macro inserting a vertical space between the code and the narration. Its parameter is a relic of a very heavy debug of the automatic vspacing mechanism. Let it remain at least until this package is 2.0 version.

```

361 \newcommand*\gmd@codeskip[1]{\@@par\addvspace\CodeTopsep%
      \@codeskipputgtrue}

```

Sometimes we add the `\CodeTopsep` vertical space in `\everypar`. When this happens, first we remove the `\parindent` empty box, but this doesn't reverse putting `\parskip` to the main vertical list. And if `\parskip` is put, `\addvspace` shall see it not the 'true' last skip. Therefore we need a Boolean switch to keep the knowledge of putting similar vskip before `\parskip`.

@codeskipput

```

\if@codeskipput 362 \newgif\if@codeskipput

```

The below is another relic of the heavy debug of the automatic vspacing. Let's give it the same removal clause as [above](#).

```

363 \newcommand*\gmd@nocodeskip[2]{ }

```

And here is how the two relic macros looked like during the debug. As you see, they are disabled by a false `\if` (look at it closely ;-).

```

364 \if1_1
365 \renewcommand*\gmd@codeskip[1]{%
366 \hbox{\rule{1cm}{3pt}}_#1!!!}
367 \renewcommand*\gmd@nocodeskip[2]{%
368 \hbox{\rule{1cm}{0.5pt}}_#1:_#2_}
369 \fi

```

We'll wish to execute `\gmd@codeskip` wherever a codeline (possibly with an inline comment) is followed by a homogenic comment line or reverse. Let us dedicate a Boolean switch to this then.

```

\aftercodegtrue 370 \newgif\if@aftercode
\aftercodefalse
\if@aftercode

```

This switch will be set true in the moments when we are able to switch from the `TEX` code into the narration and the below one when we are able to switch reversely.

```

\afternarrgtrue 371 \newgif\if@afternarr
\afternarrfalse
\if@afternarr

```

To insert vertical glue between the `TEX` code and the narration we'll be playing with `\everypar`. More precisely, we'll add a macro that the `\parindent` box shall move and the glue shall put.

```

\@codetonarrskip 372 \long\def\@codetonarrskip{%
373   \if@codeskipput\else
374     \if@afternarr\gmd@nocodeskip4{iaN}\else
375     \if@aftercode

```

We are at the beginning of `\everypar`, i.e., \TeX has just entered the hmode and put the `\parindent` box. Let's remove it then.

```

376     {\setbox0=\lastbox}%

```

Now we can put the vertical space and state we are not 'aftercode'.

```

377     \gmd@codeskip4\@codeskipputgtrue
378     \leftskip\TextIndent% this line is a patch against a bug-or-feature that
                           in certain cases the narration \leftskip is left equal the code left-
                           skip. (It happens when there're subsequent code lines after an inline
                           comment not ended with an explicit \par.)
379     \else\gmd@nocodeskip4{naC}%
380     \fi%
381     \fi
382     \fi\@aftercodegfalse}

```

But we play with `\everypar` for other reasons too, and while restoring it, we don't want to add the `\@codetonarrskip` macro infinitely many times. So let us define a macro that'll check if `\everypar` begins with `\@codetonarrskip` and trim it if so. We'll use this macro with proper `\expandaftering` in order to give it the contents of `\everypar`. The work should be done in two steps first of which will be checking whether `\everypar` is nonempty (we can't have two delimited parameters for a macro: if we define a two-parameter macro, the first is undelimited so it has to be nonempty; it costed me some one hour to understand it).

```

\@trimandstore 383 \long\def\@trimandstore#1\@trimandstore{%
384   \def\@trimandstore@hash{#1}%
385   \ifx\@trimandstore@hash\@empty% we check if #1 is nonempty. The \if%
                           % \relax#1\relax trick is not recommended here because using it we
                           couldn't avoid expanding #1 if it'd be expandable.
386   \gmd@preverypar={}%
387   \else
388   \afterfi{\@xa\@trimandstore@ne\the\everypar\@trimandstore}%
389   \fi}

390 \long\def\@trimandstore@ne#1#2\@trimandstore{%
391   \def\trimmed@everypar{#2}%
392   \ifx\@codetonarrskip#1%
393   \gmd@preverypar=\@xa{\trimmed@everypar}%
394   \else
395   \gmd@preverypar=\@xa{\the\everypar}%
396   \fi}

```

We prefer not to repeat `#1` and `#2` within the `\ifs` and we even define an auxiliary macro because `\everypar` may contain some `\ifs` or `\fis`.

Life Among Queer EOLs

When I showed this package to my \TeX Guru he commended it and immediately pointed some disadvantages in the comparison with the `doc` package.

One of them was an expected difficulty of breaking a moving argument (e.g., of a sectioning macro) in two lines. To work it around let's define a line-end eater:

```

397 \catcode'\^^B=\active% note we re\catcode <char2> globally, for the entire docu-
      ment.
398 \bgroup\catcode'\^^M=\active%
399 \firstofone{\egroup%
^^B 400   \def\QueerCharTwo{\long\def^^B##1^^M{\ignorespaces}}
401   \QueerCharTwo
402 \AtBegInput{\@ifQueerEOL{\catcode'\^^B\active}{\QueerCharTwo}% We re-
      peat redefinition of <char2> at begin of the documenting input, because
      doc.dtx suggests that some packages (namely inputenc) may re\catcode such
      unusual characters.

```

As you see the ^^B active char is defined to gobble everything since itself till the end of line and the very end of line. This is intended for harmless continuing a line. The price is affecting the line numbering when `countalllines` option is enabled.

I also liked the doc's idea of comment² i.e., the possibility of marking some text so that it doesn't appear nor in the working version neither in the documentation, got by making ^^A (i.e., <char1>) a comment char.

However, in this package such a trick would work another way: here the line ends are active, a comment char would disable them and that would cause disasters. So let's do it an \active way.

```

403 \catcode'\^^A=\active% note we re\catcode <char1> globally, for the entire docu-
      ment.
404 \bgroup\catcode'\^^M=\active%
405 \firstofone{\egroup%
^^A 406   \def\QueerCharOne{\long\def^^A##1^^M{\ignorespaces^^M}}
407   \QueerCharOne
408 \AtBegInput{\@ifQueerEOL{\catcode'\^^A\active}\QueerCharOne}% see note af-
      ter line 402.

```

As I suggested in the users' guide, \StraightEOL and \QueerEOL are intended to cooperate in harmony for the user's good. They take care not only of redefining the line end but also these little things related to it.

One usefulness of \StraightEOL is allowing linebreaking of the command arguments. Another making possible executing some code lines during the documentation pass.

```

\StraightEOL 409 \def\StraightEOL{%
410   \catcode'\^^M=5\relax
411   \catcode'\^^A=14\relax
412   \catcode'\^^B=14\relax
413   \def\^^M{\_}}
\QueerEOL 414 \def\QueerEOL{%
415   \catcode'\^^M=\active
416   \catcode'\^^A=\active
417   \catcode'\^^B=\active% I only re\catcode <char1> and <char2> hoping no one
      but me is that perverse to make them \active and (re)define. (Let me know
      if I'm wrong at this point.)
418   \let\^^M=\gmd@bslashEOL}

```

To make `^^M` behave more like a ‘normal’ lineend I command it to add a `_10` at first. It works but has one unwelcome feature: if the line has nearly `\textwidth`, this closing space may cause line breaking and setting a blank line. To fix this I `\advance` the `\parfillskip`:

```
419 \def\gmd@parfixclosingspace{%
420     \advance\parfillskip\_by-\gmd@closingspacewd\par}}
```

We’ll put it in a group surrounding `\par` but we need to check if this `\par` is executed after narration or after the code, i.e., whether the closing space was added or not.

```
421 \newskip\gmd@closingspacewd
422 \newcommand*\gmd@setclosingspacewd{%
423     \global\gmd@closingspacewd=\fontdimen2\font%
424     plus\fontdimen3\font\_minus\fontdimen4\font\relax}
```

See also line 189 to see what we do in the codeline case when no closing space is added.

And one more detail:

```
425 \bgroup\catcode'\^^M=\active%
426 \firstofone{\egroup%
427     \def\gmd@bslashEOL{\\_ \@xa\ignorespaces^^M}}
```

The `\QueerEOL` declaration will `\let` it to `\^^M` to make `\^^M` behave properly. If this definition was omitted, `\^^M` would just expand to `_` and thus not gobble the leading % of the next line leave alone typesetting the \TeX code. I type `_` etc. instead of just `^^M` which adds a space itself because I take account of a possibility of redefining the `_` CS by the user, just like in normal \TeX .

We’ll need it for restoring queer definitions for doc-compatibility.

Adjustment of `verbatim` and `\verb`

To make `verbatim(*)` typeset its contents with the \TeX code’s indentation:

```
428 \gaddtomacro\@verbatim{\leftskip=\CodeIndent}
```

And a one more little definition to accomodate `\verb` and pals for the lines commented out.

```
429 \AtBeginInput{\long\def\check@percent#1{%
430     \@xa\ifx\code@delim#1\else\afterfi{#1}\fi}}
```

We also redefine `gmverb`’s `\AddtoPrivateOthers` that has been provided just with `gmdoc`’s need in mind.

```
\AddtoPrivateOthers 431 \def\AddtoPrivateOthers#1{%
432     \@xa\def\@xa\doprivateothers\@xa{%
433         \doprivateothers\do#1}}%
```

We also redefine an internal `\verb`’s macro `\gm@verb@eol` to put a proper line end if a line end char is met in a short verbatim: we have to check if we are in ‘queer’ or ‘straight’ EOLs area.

```
434 \begingroup
435 \obeylines%
436 \AtBeginInput{\def\gm@verb@eol{\obeylines%
437     \def^^M{\verb@egroup\@latex@error{%
438         \@nx\verb\_ended\_by\_end\_of\_line}}%
```

```

439      \@ifQueerEOL{\gmd@textEOL}{\@ehc}}}%
440 \endgroup

```

Macros for Marking The Macros

A great inspiration for this part was the `doc` package again. I take some macros from it, and some tasks I solve a different way, e.g., the `\` (or another escapechar) is not active, because anyway all the chars of code are scanned one by one. And exclusions from indexing are supported not with a list stored as `\toks` register but with separate control sequences for each excluded CS.

The `doc` package shows a very general approach to the indexing issue. It assumes using a special `MakeIndex` style and doesn't use explicit `MakeIndex` controls but provides specific macros to hide them. But here in `gmdoc` we prefer no special style for the index.

```

\actualchar 441 \edef\actualchar{\string\@}
\quotechar  442 \def\quotechar{"}
\encapchar  443 \edef\encapchar{\twelveclub}
\levelchar  444 \def\levelchar{!}

```

However, for the glossary, i.e., the change history, a special style is required, e.g., `gmglo.ist`, and the above macros are redefined by the `\changes` command due to `gmglo.ist` and `gglo.ist` settings.

Moreover, if you insist on using a special `MakeIndex` style, you may redefine the above four macros in the preamble. The `\edefs` that process them further are postponed till `\begin{document}`.

```

\CodeEscapeChar 445 \def\CodeEscapeChar#1{%
446   \begingroup
447   \escapechar\m@ne
\code@escape@char 448   \xdef\code@escape@char{\string#1}%
449   \endgroup}

```

As you see, to make a proper use of this macro you should give it the `\langle char \rangle` CS as an argument. It's an invariant assertion that `\code@escape@char` stores 'other' version of the code layer escape char.

```

450 \CodeEscapeChar\

```

As mentioned in `doc`, someone may have some chars `_{11}`ed.

```

451 \@ifundefined{MakePrivateLetters}{%
\MakePrivateLetters 452   \def\MakePrivateLetters{\makeatletter\catcode'\*=11_\}\{ }

```

A tradition seems to exist to write about e.g., 'command `\section` and command `\section*`' and such an understanding also of 'macro' is noticeable in `doc`. Making the `*` a letter solves the problem of scanning starred commands.

And you may wish some special chars to be `_{12}`.

```

\MakePrivateOthers 453 \def\MakePrivateOthers{\let\do=\@makeother_\doprivateothers}

```

This macro we use to re`\catcode` the space for marking the environments' names and the caret for marking chars such as `^M`, see line 726. So let's define the list:

```

\doprivateothers 454 \def\doprivateothers{\do\_ \do\^}

```

Two chars for the beginning, and also the `\MakeShortVerb` command shall this list enlarge with the char(s) declared. (There's no need to add the backslash to this list since all the relevant commands `\string` their argument whatever it is.)

Now the main macro indexing a macro's name. It would be a verbatim :-) copy of the doc's one if I didn't omit some lines irrelevant with my approach.

```
\scan@macro 455 \def\scan@macro#1{% we are sure to scan at least one token and therefore we define
                this macro as one-parameter.
```

Unlike in doc, here we have the escape char `_` so we may just have it printed during main scan char by char, i.e., in the lines 271 and 274.

So, we step the checksum counter first,

```
456 \step@checksum% (see line 1195 for details),
```

Then, unlike in doc, we do *not* check if the scanning is allowed, because here it's always allowed and required.

Of course, I can imagine horrible perversities, but I don't think they should really be taken into account. Giving the letter a `\catcode` other than `_` surely would be one of those perversities. Therefore I feel safe to take the character a as a benchmark letter.

```
457 \ifcat\_a\_@nx#1%
458 \quote@char#1%
459 \xdef\macro@iname{\maybe@quote#1}% global for symmetry with line 465.
460 \xdef\macro@pname{\string#1}% we'll print entire name of the macro later.
```

We `\string` it here and in the lines 468 and 474 to be sure it is whole `_` for easy testing for special indexentry formats, see line 510 etc.

```
461 \afterfi{\@ifnextcat{a}{\continue@macroscan}{\finish@macroscan}}%
462 \else% #1 is not a letter, so we have just scanned a one-char CS.
```

Another reasonable `\catcodes` assumption seems to be that the digits are `_`. Then we don't have to type `(%)\expandafter\@gobble\string_a`. We do the `\uccode` trick to be sure that the char we write as the macro's name is `_`.

```
463 {\uccode'9=' #1%
464 \uppercase{\xdef\macro@iname{9}}%
465 }%
466 \quote@char#1%
467 \xdef\macro@iname{\maybe@quote\macro@iname}%
468 \xdef\macro@pname{\xiistring#1}%
469 \afterfi\_ \finish@macroscan
470 \fi}
```

This is the end... beautiful friend, the end... of `\scan@macro`'s `\def` the end... The `\xiistring` macro, provided by `gmutils`, is used instead of original `\string` because we wish to get the space `_`.

Now, let's explain some details, i.e., let's define them. We call the following macro having known `#1` to be `_`.

```
471 \def\continue@macroscan#1{%
472 \quote@char#1%
473 \xdef\macro@iname{\macro@iname\_ \maybe@quote#1}%
474 \xdef\macro@pname{\macro@pname\_ \string#1}%
475 \@ifnextcat{a}{\continue@macroscan}{\finish@macroscan}%
476 }
```

The `\@ifnextcat` macro is defined analogously to `\@ifnextchar` but the test done by it is not `\ifx` but `\ifcat`. (Note it wouldn't work for an active char as the 'pattern'.)


```

477 \def\quote@char#1{\uccode'9='#1% at first I took digit 1 for this \uccodeing but
      then #1 meant #<#1> in \uppercase's argument, of course.
478     \uppercase{%
479         \@ifismember_9\of_\indexcontrols\glet\maybe@quote\quotechar%
480         \else\g@emptyify\maybe@quote%
481         \fi}%
482     }}

```

And now let's take care of the MakeIndex control characters. We'll define a list of them to check whether we should quote a char or not. But we'll do it at `\begin{document}` to allow the user to use some special MakeIndex style and in such a case to redefine the four MakeIndex controls' macros. We enrich this list with the backslash because sometimes MakeIndex didn't like it unquoted.

```

\indexcontrols 483 \AtBeginDocument{\xdef\indexcontrols{%
484     \backslashlevelchar\encapchar\actualchar\quotechar}}
\@ifismember 485 \long\def_\@ifismember#1\of#2{%
486     \long\def\in@@##1#1##2\in@@{%
487         \if\xn^^A##2\xn^^A\afterfi\gmd@@iffalse
488         \else\afterfi\gmd@@iftrue
489         \fi}%
490     \xa\in@@#2#1\in@@}%

```

To hide the Booleans from T_EX in some contexts define

```

\gmd@@iftrue 491 \def\gmd@@iftrue{\iftrue}
\gmd@@iffalse 492 \def\gmd@@iffalse{\iffalse}

```

A try to `\let` instead of `\def` resulted with the 'incomplete if' error: T_EX looks at the meanings not the names while counting `\ifs`.

A word of commentary. `doc` in another situation solves the problem of checking similar way, but a little more complexely. On the other hand, `doc`'s solution is more general: here a trouble may arise if the list begins with `^^A`. It seems little possible to me, though. Btw., this macro is used only for catching chars that are MakeIndex's controls so far. How does it work?

`\quote@char` sort of re`\catcodes` its argument through the `\uccode` trick: assigns the argument as the uppercase code of the digit 9 and does further work in the `\uppercase`'s scope so the digit 9 (a benchmark 'other') is substituted by `#1` but the `\catcode` remains so thus the `\@ifismember` macro gets `\quote@char`'s `#1` 'other'ed as the first argument.

`\@ifismember` is defined to take two parameters separated by `\of`. In `\quote@char` the second argument for it is `\indexcontrols` defined as the (expanded) sequence of the MakeIndex controls. `\@ifismember` defines its inner macro `\in@@` to take two parameters separated by the first and the second `\@ifismember`'s parameter, which are here the char investigated by `\quote@char` and the `\indexcontrols` list. The inner macro's parameter string is delimited by the macro itself, why not. `\in@@` is put before a string consisting of `\@ifismember`'s second and first parameters (in such a reversed order) and `\in@@` itself. In such a sequence it looks for something fitting its parameter pattern. `\in@@` is sure to find the parameters delimiter (`\in@@` itself) and the separator, `\@ifismember`'s `#1` i.e., the investigated char, because they are just there. But the investigated char may be found not near the end, where we put it, but among the MakeIndex controls' list. Then the rest of this list and `\@ifismember`'s `#1` put by us become the second argument of `\in@@`. What `\in@@` does with its arguments, is just a check whether the second one is empty. This may happen *iff* the investigated char hasn't been found among the MakeIndex controls' list and then `\in@@` shall expand to `\iffalse`, otherwise it'll expand to `\iftrue`. (The

\after... macros are employed not to (mis)match just got \if... with the test's \fi.) “(Deep breath.) You got that?” If not, try doc’s explanation of \ifnot@excluded, pp.36–37 of the v2.1b dated 2004/02/09 documentation, where a similar construction is attributed to Michael Spivak.

```
493 \newcommand*\finish@macroscan{%
```

We have the macro’s name for indexing in \macro@iname and for print in \macro@pname. So we index it. We do it a bit counter crank way because we wish to use more general indexing macro.

```
494 \if\verbatimchar\macro@pname% it's important that \verbatimchar comes be-
    fore the macro's name: when it was reverse, the \tt CS turned this test
    true and left the \verbatimchar what resulted with '\+tt' typeset. Note
    that this test should turn true iff the scanned macro name shows to be the
    default \verb's delimiter. In such a case we give \verb another delimiter,
    namely $:
495 \def\im@firstpar{[$]}%
496 \else\def\im@firstpar{}\fi
497 \@xa\index@macro\im@firstpar\macro@iname\macro@pname
498 \maybe@marginpar\macro@pname
499 \macro@pname
500 \gmd@charbychar
501 }
```

Now, the macro that checks whether the just scanned macro should be put into a marginpar: it checks the meaning of a very special CS: whose name consists of gmd/2marpar/ and of the examined macro’s name.

```
502 \def\maybe@marginpar#1{%
503 \@ifundefined{gmd/2marpar/#1}{\fi}%
504 \@xa\Text@Marginize\@xa{\bslash#1}% \expandafters
    'cause the \Text@Marginize command applies \string to its argument.
    % \macro@pname, which will be the only possible argument to \maybe-
    % @marginpar, contains the macro's name without the escapechar so we
    added it here.
505 \@xa\g@relaxen\csname\gmd/2marpar/#1\endcsname% we reset the switch.
506 }}
```

The inner macro indexing macro. #1 is the \verb’s delimiter; #2 is assumed to be the macro’s name with MakeIndex-control chars quoted. #3 is a macro storing the 12 macro’s name, usually \macro@pname, built with \stringing every char in lines 460, 468 and 474. #3 is used only to test if the entry should be specially formatted.

```
\index@macro 507 \newcommand*\index@macro[3][\verbatimchar]{\fi%
508 \@ifundefined{gmd/iexcl/#3}%
509 {\% #3 is not excluded from index
510 \@ifundefined{gmd/defentry/#3}%
511 {\% #3 is not def entry
512 \@ifundefined{gmd/usgentry/#3}%
513 {\% #3 is not usg entry
514 \edef\kind@fentry{\CommonEntryCmd}}}%
515 {\% #3 is usg entry
516 \def\kind@fentry{UsgEntry}}%
517 \un@usgentryze{#3}}%
```

```

518 }%
519 {% #3 is def entry
520 \def\kind@fentry{DefEntry}%
521 \un@defentryze{#3}%
522 }% of gmd/defentry/ test's 'else'
523 \if@pageindex\@pageinclindexfalse\fi% should it be here or there? Defi-
    nitely here because we'll wish to switch the switch with a declaration.
524 \if@pageinclindex
525 \edef\IndexRefCs{gmdindexpagecs{\HLPrefix}{\kind@fentry}{%
    \EntryPrefix}}%
526 \else
527 \edef\IndexRefCs{gmdindexrefcs{\HLPrefix}{\kind@fentry}{%
    \EntryPrefix}}%
528 \fi
529 \edef\@tempa{\IndexPrefix#2\actualchar%
530 \quotechar\bslash\verb*#1\quoted@eschar#2#1% The last macro in this
    line usually means the first two, but in some cases it's redefined to
    be empty (when we use \index@macro to index not a CS).
531 \encapchar\IndexRefCs}%
532 \@xa\special@index\@xa{\@tempa}% We give the indexing macro the argu-
    ment expanded so that hyperref may see the explicit encapchar in order
    not to add its own encapsulation of |hyperpage when the (default)
    hyperindex=true option is in force. (After this setting the \edefs in
    the above may be changed to \defs.)
533 }{}% closing of gmd/iexcl/ test.
534 }}

```

```

\un@defentryze 535 \def\un@defentryze#1{%
536 \@xa@g@relaxen\csname_gmd/defentry/#1\endcsname
537 \g@relaxen\last@defmark}% we care to clear the last definition marker checked
    by \changes.

```

```

\un@usgentryze 538 \def\un@usgentryze#1{%
539 \@xa@g@relaxen\csname_gmd/usgentry/#1\endcsname}
540 \@emptify\EntryPrefix% this macro seems to be obsolete now (v0.98d).

```

For the case of page-indexing a macro in the commentary when codeline index option is on:

```

\@pageinclindextrue 541 \newif\if@pageinclindex
\@pageinclindexfalse 542 \newcommand*\quoted@eschar{\quotechar\bslash}% we'll redefine it when index-
    \if@pageinclindex ing an environment.
    \quoted@eschar

```

Let's initialize \IndexPrefix

```

543 \def\IndexPrefix{}

```

The \IndexPrefix and \HLPrefix ('HyperLabel Prefix') macros are given with account of a possibility of documenting several files in(to) one document. In such case the user may for each file \def\IndexPrefix{\<package name>!} for instance and it will work as main level index entry and \def\HLPrefix{\<package name>} as a prefix in hypertargets in the codelines. They are redefined by \DocInclude e.g.

```

544 \if@linesnotnum\@pageindextrue\fi
545 \AtBeginDocument{%

```

```

546 \if@pageindex
547 \def\gmdindexrefcs#1#2#3#4{\csname#2\endcsname{\hyperpage{#4}}}% in
    the page case we gobble the third argument that is supposed to be the
    entry prefix.
548 \let\gmdindexpagecs=\gmdindexrefcs
549 \else
550 \def\gmdindexrefcs#1#2#3#4{\gmiflink[clnum.#4]{%
551 \csname#2\endcsname{#4}}}%
552 \def\gmdindexpagecs#1#2#3#4{\hyperlink{page.#4}{%
553 \csname#2\endcsname{\gmd@revprefix{#3}#4}}}%
554 \def\gmd@revprefix#1{%
555 \def\@tempa{#1}%
556 \ifx\@tempa\@empty\p.\,\fi}
557 \providecommand*{HLPrefix{}}% it'll be the hypertargets names' prefix in mul-
    ti-docs. Moreover, it showed that if it was empty, hyperref saw du-
    plicates of the hyper destinations, which was perfectly understandable
    (codelinenum.123 made by \refstepcounter and codelinenum.123
    made by \gmhypertarget). But since v0.98 it is not a problem any-
    more 'cause during the automatic \hypertargeting the lines are labeled
    clnum.<number>. When \HLPrefix was defined as dot, MakeIndex re-
    jected the entries as 'illegal page number'.
558 \fi}

```

The definition is postponed till `\begin{document}` because of the `\PageIndex` declaration (added for doc-compatibility), see line 1687.

I design the index to contain hyperlinking numbers whether they are the line numbers or page numbers. In both cases the last parameter is the number, the one before the last is the name of a formatting macro and in `linenum` case the first parameter is a prefix for proper reference in multi-doc.

I take account of three kinds of formatting the numbers: 1. the 'def' entry, 2. a 'usage' entry, 3. a common entry. As in doc, let them be underlined, italic and upright respectively.

```

\DefEntry 559 \def\DefEntry#1{\underline{#1}}
\UsgEntry 560 \def\UsgEntry#1{\textit{#1}}

```

The third option will be just `\relax` by default:

```

\CommonEntryCmd 561 \def\CommonEntryCmd{\relax}

```

In line 514 it's `\edefed` to allow an 'unmöglich' situation that the user wants to have the common index entries specially formatted. I use this to make *all* the index entries of the (`\SelfIncluded`) driver file to be 'usage', see codeline 14 of `gmdocDoc.tex`.

Now let's `\def` the macros declaring a CS to be indexed special way. Each declaration puts the 12ed name of the macro given it as the argument into proper macro to be `\ifxed` in lines 510 and 512 respectively.

But first let's make a version of `\@ifstar` that would work with `*11`. It's analogous to `\@ifstar`.

```

562 \bgroup\catcode'\*=11\%
563 \firstofone{\egroup
\@ifstarl 564 \def\@ifstarl#1{\@ifnextchar\*{\@firstoftwo{#1}}}}

```

Now we are ready to define a couple of commands. The *** versions of them are for marking environments and *implicit* CSs.

```

\CodeDefIndex* 565 \outer\def\CodeDefIndex{\begingroup
\CodeDefIndex 566   \MakePrivateLetters
567   \@ifstarl{\MakePrivateOthers\Code@DefIndexStar}{\Code@DefIndex}}
568 \long\def\Code@DefIndex#1{\endgroup{%
569   \escapechar\m@ne% because we will compare the macro's name with a string
      without the backslash.
570   \global\@defentryze{#1}}}
571 \long\def\Code@DefIndexStar#1{%
572   \endgroup
573   \addto@estoindex{#1}%
574   \@defentryze{#1}}

\gmd@justadot 575 \def\gmd@justadot{.}

\@defentryze 576 \long\def\@defentryze#1{%
577   \@x\let\csname_\gmd/defentry/\string#1\endcsname\gmd@justadot% The LATEX
      \@namedef macro could not be used since it's not 'long'.
\last@defmark 578   \xdef\last@defmark{\string#1}}% we \string the argument 'cause it's a control
      sequence most probably.
\@usgentryze 579 \long\def\@usgentryze#1{%
580   \@x\let\csname_\gmd/usgentry/\string#1\endcsname\gmd@justadot}

      Initialize \envirs@toindex
581 \@emptify\envirs@toindex

      Now we'll do the same for the 'usage' entries:
\CodeUsgIndex* 582 \outer\def\CodeUsgIndex{\begingroup
\CodeUsgIndex 583   \MakePrivateLetters
584   \@ifstarl{\MakePrivateOthers\Code@UsgIndexStar}{\Code@UsgIndex}}

      The * possibility is for marking environments etc.
585 \long\def\Code@UsgIndex#1{\endgroup{%
586   \escapechar\m@ne
587   \global\@usgentryze{#1}}}
588 \long\def\Code@UsgIndexStar#1{%
589   \endgroup
590   \addto@estoindex{#1}%
591   \@usgentryze{#1}}

      For the symmetry, if we want to mark a control sequence or an environment's name
      to be indexed as a 'normal' entry, let's have:
\CodeCommonIndex 592 \outer\def\CodeCommonIndex{\begingroup
593   \MakePrivateLetters
594   \@ifstarl{\MakePrivateOthers\Code@CommonIndexStar}{%
      \Code@CommonIndex}}
595 \long\def\Code@CommonIndex#1{\endgroup}
596 \long\def\Code@CommonIndexStar#1{%
597   \endgroup\addto@estoindex{#1}}

```

And now let's define commands to index the control sequences and environments occurring in the narrative.

```
\text@indexmacro 598 \long\def\text@indexmacro#1{%
599   {\escapechar\m@ne\xdef\macro@pname{\xiistring#1}}%
600   \@xa\quote@mname\macro@pname\relax% we process the CS's name char by char
        and quote MakeIndex controls. \relax is the iterating macro's stopper. The
        scanned CS's quoted name shall be the expansion of \macro@iname.
601   \if\verbatimchar\macro@pname
602     \def\im@firstpar{[$]}%
603   \else\def\im@firstpar{}%
604   \fi
605   {\do@properindex% see line 758.
606     \@xa\index@macro\im@firstpar\macro@iname\macro@pname}}
```

The macro defined below (and the next one) are executed only before a ₁₂ macro's name i.e. a nonempty sequence of ₁₂ character(s). This sequence is delimited (guarded) by \relax.

```
\quote@mname 607 \def\quote@mname{%
608   \def\macro@iname{}%
609   \quote@charbychar}
610 \def\quote@charbychar#1{%
611   \if\relax#1% finish quoting when you meet \relax or:
612   \else
613     \quote@char#1%
614     \xdef\macro@iname{\macro@iname\maybe@quote#1}%
615     \afterfi\quote@charbychar
616   \fi}
```

The next command will take one argument, which in plain version should be a control sequence and in the starred version also a sequence of chars allowed in environment names or made other by \MakePrivateOthers macro, taken in the curly braces.

```
\TextUsgIndex* 617 \def\TextUsgIndex{\begingroup
\TextUsgIndex 618   \MakePrivateLetters
619   \@ifstarl{\MakePrivateOthers\Text@UsgIndexStar}{\Text@UsgIndex}}
620 \long\def\Text@UsgIndex#1{%
621   \endgroup\@usgentryze#1%
622   \text@indexmacro#1}
623 \long\def\Text@UsgIndexStar#1{\endgroup\@usgentryze{#1}%
624   \text@indexenvir{#1}}
\text@indexenvir 625 \long\def\text@indexenvir#1{%
626   \edef\macro@pname{\xiistring#1}%
627   \if\bslash\@xa\@firstofmany\macro@pname\@nil% if \stringed #1 begins
        with a backslash, we will gobble it to make MakeIndex not see it.
628     \edef\@tempa{\@xa@gobble\macro@pname}%
629     \@tempwattrue
630   \else
631     \let\@tempa\macro@pname
632     \@tempwafalse
633   \fi}
```

```

634 \xa\quote@mname\@tempa\relax% we process \stringed #1 char by char and
      quote MakeIndex controls. \relax is the iterating macro's stopper. The
      quoted \stringed #1 shall be the expansion of \macro@iname.
635 {\if@tempswa
636   \def\quoted@eschar{\quotechar\backslash}%
637   \else\@empty\quoted@eschar\fi% we won't print any backslash before an
      environment's name, but we will before a CS's name.
638   \do@properindex% see line 758.
639   \index@macro\macro@iname\macro@pname}}
\TextCommonIndex* 640 \def\TextCommonIndex{\begingroup
\TextCommonIndex 641   \MakePrivateLetters
642   \@ifstarl{\MakePrivateOthers\Text@CommonIndexStar}{%
      \Text@CommonIndex}}
643 \long\def\Text@CommonIndex#1{\endgroup
644   \text@indexmacro#1}
645 \long\def\Text@CommonIndexStar#1{\endgroup
646   \text@indexenvir{#1}}

```

As you see in the lines 521 and 517, the markers of special formatting are reset after first use.

But we wish the CSs not only to be indexed special way but also to be put in marginpars. So:

```

\CodeMarginize* 647 \outer\def\CodeMarginize{\begingroup
\CodeMarginize 648   \MakePrivateLetters
649   \@ifstarl{\MakePrivateOthers\egCode@MarginizeEnvir}{%
      \egCode@MarginizeMacro}}

```

One more expansion level because we wish \Code@MarginizeMacro not to begin with \endgroup because in the subsequent macros it's used *after* ending the re\catcodeing group.

```

650 \long\def\egCode@MarginizeMacro#1{\endgroup
651   \Code@MarginizeMacro#1}
652 \long\def\Code@MarginizeMacro#1{{\escapechar\m@ne
653   \xa\glet\csname_gmd/2marpar/\string#1\endcsname\gmd@justadot
654   }}
655 \long\def\egCode@MarginizeEnvir#1{\endgroup
656   \Code@MarginizeEnvir{#1}}
657 \long\def\Code@MarginizeEnvir#1{\addto@estomarginpar{#1}}

```

And a macro really putting the environment's name in a marginpar shall be triggered at the beginning of the nearest codeline.

Here it is:

```

\mark@envir 658 \def\mark@envir{%
659   \ifx\envirs@tomarginpar\@empty
660   \else
661     \let\do\Text@Marginize
662     \envirs@tomarginpar%
663     \g@emptyify\envirs@tomarginpar%
664   \fi

```

```

665 \ifx\envirs@toindex\@empty
666 \else
667 \def\do##1{% the \envirs@toindex list contains \stringed macros or environ-
        ments' names in braces and each preceded with \do.
668 \if\backslash\@firstofmany##1\@nil% if ##1 begins with a backslash, we will
        gobble it for MakeIndex not see it.
669 \edef\@tempa{\@gobble##1}%
670 \@tempswatrue
671 \else
672 \edef\@tempa{##1}\@tempswafalse
673 \fi
674 \@xa\quote@mname\@tempa\relax% see line 634 & subs. for commentary.
675 {\if@tempswa
676 \def\quoted@eschar{\quotechar\backslash}%
677 \else\@emptify\quoted@eschar\fi
678 \index@macro\macro@iname{##1}}}%
679 \envirs@toindex
680 \g@emptify\envirs@toindex%
681 \fi}

```

One very important thing: initialisation of the list macros:

```

682 \@emptify\envirs@tomarginpar
683 \@emptify\envirs@toindex

```

For convenience we'll make the 'private letters' first not to bother ourselves with `\makeatletter` for instance when we want mark some CS. And `\MakePrivateOthers` for the environment and other string case.

```

\CodeDefine* 684 \outer\def\CodeDefine{\begingroup
\CodeDefine 685 \MakePrivateLetters

```

We do `\MakePrivateLetters` before `\@ifstar1` in order to avoid a situation that `TEX` sees a control sequence with improper name (another CS than we wished) (because `\@ifstar1` establishes the `\catcodes` for the next token):

```

686 \@ifstar1{\MakePrivateOthers\Code@DefEnvir}{\Code@DefMacro}}
\CodeUsage* 687 \outer\def\CodeUsage{\begingroup
\CodeUsage 688 \MakePrivateLetters
689 \@ifstar1{\MakePrivateOthers\Code@UsgEnvir}{\Code@UsgMacro}}

```

And then we launch the macros that close the group and do the work.

```

690 \long\def\Code@DefMacro#1{%
691 \Code@DefIndex#1% we use the internal macro; it'll close the group.
692 \Code@MarginizeMacro#1}
693 \long\def\Code@UsgMacro#1{%
694 \Code@UsgIndex#1% here also the internal macro; it'll close the group
695 \Code@MarginizeMacro#1}

```

The next macro is taken verbatim ;-) from `doc` and the subsequent `\lets`, too.

```

696 \def\codeline@wrindex#1{\if@filesw
697 \immediate\write\@indexfile
698 {\string\indexentry{#1}%
699 {\HLPrefix\number\c@codelinenum}}\fi}

```


We initialize it due to the option (or lack of the option):

```

700 \AtBeginDocument{%
701   \if@pageindex
702     \let\special@index=\index
703   \else
704     \let\special@index=\codeline@wrindex
705   \fi}% postponed till \begin{document} with respect of doc-like declarations.

```

And in case we don't want to index:

```

\gag@index 706 \def\gag@index{\let\index=\@gobble
707   \let\codeline@wrindex=\@gobble}

```

We'll use it in one more place or two. And we'll wish to be able to undo it so let's copy the original meanings:

```

708 \StoreMacros{\index\codeline@wrindex}
\ungag@index 709 \def\ungag@index{\RestoreMacros{\index\@codeline@wrindex}}

```

Our next task is to define macros that'll mark and index an environment or other string in the code. Because of lack of a backslash, no environment's name is scanned so we have to proceed different way. But we wish the user to have symmetric tools, i.e., the 'def' or 'usage' use of an environment should be declared before the line where the environment occurs. Note the slight difference between these and the commands to declare a CS marking: the latter do not require to be used *immediately* before the line containing the CS to be marked. We separate indexing from marginizing to leave a possibility of doing only one of those things.

```

710 \long\def\Code@DefEnvir#1{%
711   \endgroup
712   \addto@estomarginpar{#1}%
713   \addto@estoindex{#1}%
714   \@defentryze{#1}}
715 \long\def\Code@UsgEnvir#1{%
716   \endgroup
717   \addto@estomarginpar{#1}%
718   \addto@estoindex{#1}%
719   \@usgentryze{#1}}
720 \long\def\addto@estomarginpar#1{%
721   \edef\@tempa{\@nx\do{\xiistring#1}}% we \string the argument to allow it
       to be a control sequence.
722   \@xa\addtomacro\@xa\envirs@tomarginpar\@xa{\@tempa}}
723 \long\def\addto@estoindex#1{%
724   \edef\@tempa{\@nx\do{\xiistring#1}}
725   \@xa\addtomacro\@xa\envirs@toindex\@xa{\@tempa}}

```

And now a command to mark a 'usage' occurrence of a CS, environment or another string in the commentary. As the 'code' commands this also has plain and starred version, first for CSs appearing explicitly and the latter for the strings and CSs appearing implicitly.

```

\TextUsage* 726 \def\TextUsage{\begingroup
\TextUsage 727   \MakePrivateLetters
728   \@ifstar1{\MakePrivateOthers\Text@UsgEnvir}{\Text@UsgMacro}}

```

```

729 \long\def\Text@UsgMacro#1{%
730   \endgroup{\tt\xiistring#1}%
731   \Text@Marginize#1%
732   \begingroup\Code@UsgIndex#1% we declare the kind of formatting of the entry.
733   \text@indexmacro#1}

734 \long\def\Text@UsgEnvir#1{%
735   \endgroup{\tt\xiistring#1}%
736   \Text@Marginize{#1}%
737   \@usgentryze{#1}% we declare the ‘usage’ kind of formatting of the entry and
       index the sequence #1.
738   \text@indexenvir{#1}}

```

We don’t provide commands to mark a macro’s or environment’s definition present within the narrative because we think there won’t be any: one defines macros and environments in the code not in the commentary.

```

\TextMarginize* 739 \def\TextMarginize{\begingroup
\TextMarginize 740   \MakePrivateLetters
741   \@ifstarl{\MakePrivateOthers\egText@Marginize}{\egText@Marginize}}
742 \long\def\egText@Marginize#1{\endgroup
743   \Text@Marginize#1}

```

We check whether the margin pars are enabled and proceed respectively in either case.

```

744 \if@marginparsused
745   \reversemarginpar
746   \marginparpush\z@
747   \marginparwidth8pc\relax

```

You may wish to put not only macros and environments to a marginpar.

```

\gmdmarginpar 748 \long\def\gmdmarginpar#1{%
749   \marginpar{\raggedleft\strut
750     \hskip0ptplus100ptminus100pt%
751     #1}}%
752 \else
753   \long\def\gmdmarginpar#1{}%
754 \fi
\Text@Marginize 755 \long\def\Text@Marginize#1{%
756   \gmdmarginpar{\marginpartt\xiistring#1}}

```

Note that the above macro will just gobble its argument if the marginpars are disabled.

It may be advisable to choose a condensed typewriter font for the marginpars, if there is any. (The Latin Modern font family provides a light condensed typewriter font, it’s set in gmdocc class.)

```

757 \let\marginpartt\tt

```

If we count all lines then the index entries for CSs and environments marked in the commentary should have codeline numbers not page numbers and that is `\let` in line 704. On the other hand, if we count only the codelines, then a macro or an environment marked in the commentary should have page number not codeline number. This we declare here, among others we add the letter `p` before the page number.

```

758 \def\do@properindex{%

```

```

759 \if@countalllines\else
760 \@pageinclindextrue
761 \let\special@index=\index
762 \fi}

```

In doc all the ‘working’ T_EX code should be braced in(to) the `macrocode` environments. Here another solutions are taken so to be doc-compatible we only should nearly-ignore `macrocode(*)`s with their Percent and The Four Spaces Preceding ;-). I.e., to ensure the line ends are ‘queer’. And that the DocStrip directives will be typeset as the DocStrip directives. And that the usual code escape char will be restored at `\end{%macrocode}`. And to add the vertical spaces.

If you know doc conventions, note that `gmdoc` *does not* require `\end{macrocode}` to be preceded with any particular number of any char :-).

```

macrocode* 763 \newenvironment*{macrocode*}{%
764 \if@codeskipput\else\par\addvspace\CodeTopsep\@codeskipputgtrue\fi
765 \QueerEOL}%
766 {\par\addvspace\CodeTopsep\CodeEscapeChar\}}

```

Let’s remind that the starred version makes `\` visible, which is the default in `gmdoc` outside `macrocode`.

So we should make the spaces *invisible* for the unstarred version.

```

macrocode 767 \newenvironment*{macrocode}{%
768 \if@codeskipput\else\par\addvspace\CodeTopsep\@codeskipputgtrue\fi
769 \CodeSpacesBlank\QueerEOL}%
770 {\par\addvspace\CodeTopsep\CodeEscapeChar\}}

```

Note that at the end of both the above environments the `\`’s rôle as the code escape char is restored. This is crafted for the `\SpecialEscapechar` macro’s compatibility: this macro influences only the first `macrocode` environment. The situation that the user wants some queer escape char in general and in a particular `macrocode` yet another seems to me “unmöglich, Prinzessin”⁸.

Since the first .dtx I tried to compile after the first published version of `gmdoc` uses a lot of commented out code in `macrocodes`, it seems to me necessary to add a possibility to typeset `macrocodes` as if they were a kind of `verbatim`, that is to leave the code layer and narration layer philosophy.

```

oldmc 771 \let\oldmc\macrocode
oldmc* 772 \let\endoldmc\endmacrocode
773 \n@melet{oldmc*}{macrocode*}
774 \n@melet{endoldmc*}{endmacrocode*}

```

Now we arm `oldmc` and `olmc*` with the the macro looking for `%\end{envir name}`.

```

775 \addtomacro\oldmc{\@oldmacrocode@launch}%
776 \@xa\addtomacro\csname\oldmc*\endcsname{%
777 \@oldmacrocode@launch}
778 \def\@oldmacrocode@launch{%
779 \emptify\gmd@textEOL% to disable it in \gmd@docstripdirective launched
within the code.
780 \glet\stored@code@delim\code@delim
781 \@makeother\^^B\CodeDelim*\^^B%

```

⁸ Richard Strauss after Oscar Wilde, *Salome*.

```

782 \ttverbatim\gmd@DoTeXCodeSpace%
783 \@makeother\|%'cause \ttverbatim doesn't do that.
784 \MakePrivateLetters% see line 451.
785 \docstrips@percent\@makeother\>%

```

sine qua non of the automatic delimiting is replacing possible $*_{12}$ in the environment's name with $*_{11}$. Not to complicate assume $*$ may occur at most once and only at the end. We also assume the environment's name consists only of character tokens whose catcodes (except of $*$) will be the same in the verbatim text.

```

786 \@xa\gmd@currentvxistar\@currentvir*\relax
787 \@oldmacrocode}

788 \bgroup\catcode'*11
789 \firstofone{\egroup
790 \def\gm@xistar{*}}

791 \def\gmd@currentvxistar#1*#2\relax{%
792 \edef\@currentvir{#1\if*#2\gm@xistar\fi}}

```

The trick is that $\#2$ may be either $*_{12}$ or empty. If it's $*$, the test is satisfied and $\backslash\if...\fi$ expands to $\backslash\gm@xistar$. If $\#2$ is empty, the test is also satisfied since $\backslash\gm@xistar$ expands to $*$ but there's nothing to expand to. So, if the environment's name ends with $*_{12}$, it'll be substituted with $*_{11}$ or else nothing will be added. (Note that a $*$ not at the end of env. name would cause a disaster.)

```

793 \bgroup
794 \catcode'[=1\catcode']=2
795 \catcode'\{=\active\@makeother\}
796 \@makeother\^~B
797 \catcode'!=0\catcode'\=\active
798 !catcode'&=14 !catcode'*=11
799 !catcode'!=!active\obeyspaces&
800 !firstofone[!egroup&

\@oldmacrocode 801 !def!@oldmacrocode[&
802 !bgroup!let!=!relax& to avoid writing !noexpand four times.
803 !xdef!oldmc@def[&
804 !def!noexpand!oldmc@end####1!noexpand%!\noexpand\end&
805 !noexpand{!@currentvir}[&
806 #####1^~B!noexpand!end[!@currentvir]!noexpand!gmd@oldmcfinis]]&
807 !egroup& now \oldmc@edef is defined to have one parameter delimited with
      \end{<current env.'s name>}
808 !oldmc@def&
809 !oldmc@end]&
810 ]

811 \def\gmd@oldmcfinis{%
812 \@xa\CodeDelim\stored@code@delim
813 \gmd@mchook}% see line 1625

814 \def\VerbMacrocodes{%
815 \let\macrocode\oldmc
816 \n@melet{macrocode*}{oldmc*}}

```

To handle DocStrip directives in the code (in the old macrocodes case that is).

```

817 \bgroup\catcode'\%\active
818 \firstofone{\egroup
819 \def\docstrips@percent{\catcode'\%\active
820   \let%\gmd@codecheckifds}}

```

The point is, the active % will be expanded when just after it is the \gmd@charbychar cs token and next is some char, the ^^B code delimiter at least. So, if that char is <, we wish to launch DocStrip directive typesetting. (Thanks to \ttverbatim all the < are ‘other’.)

```

821 \def\gmd@codecheckifds#1#2{% note that #1 is just to gobble \gmd@charbychar
      token.
822   \if@dsdir\@dsdirfalse
823     \if\@nx<\@nx#2\afterfifi\gmd@docstripdirective
824     \else\afterfifi{\twelvepercent#1#2}%
825     \fi
826   \else\afterfi{\twelvepercent#1#2}%
827   \fi}

```

macro Almost the same we do with the macro(*) environments, stating only their argument to be processed as the ‘def’ entry. Of course, we should re\catcode it first.

```

828 \newenvironment{macro}{%
829   \@tempskipa=\MacroTopsep
830   \if@codeskipput\advance\@tempskipa_\by-\CodeTopsep\fi
831   \par\addvspace{\@tempskipa}\@codeskipputgtrue
832   \begingroup\MakePrivateLetters\MakePrivateOthers% we make also the ‘pri-
      vate others’ to cover the case of other sequence in the argument. (We’ll
      use the \macro macro also in the environment for describing and defining
      environments.)
833   \gmd@ifonetoken\Hybrid@DefMacro\Hybrid@DefEnvir}%
834   {\par\addvspace\MacroTopsep\@codeskipputgtrue}

```

It came out that the doc’s author(s) give the macro environment also starred versions of commands as argument. It’s OK since (the default version of) \MakePrivateLetters makes * a letter and therefore such a starred version is just one CS. However, in doc.dtx occure macros that mark *implicit* definitions i.e., such that the defined CS is not scanned in the subsequent code.

macro* And for those who want to to use this environment for marking implicit definitions, define the star version:

```

835 \@namedef{macro*}{\let\gmd@ifonetoken\@secondoftwo\macro}
836 \@xa\let\csname_endmacro*\endcsname\endmacro

```

Note that macro and macro* have the same effect for more-than-one-token arguments thanks to \gmd@ifonetoken’s meaning inside unstarred macro (it checks whether the argument is one-token and if it isn’t, \gmd@ifonetoken switches execution to ‘other sequence’ path).

The two environments behave different only with a one-token argument: macro postpones indexing it till the first scanned occurrence while macro* till the first code line met.

Now, let’s complete the details. First define an \if-like macro that turns true when the string given to it consists of just one token (or one {\text}, to tell the whole truth).

```

\gmd@ifsingle 837 \def\gmd@ifsingle#1#2\@nil{%

```

```

838 \def\@tempa{#2}%
839 \ifx\@tempa\@empty}

```

Note it expands to an open `\if...` test (unbalanced with `\fi`) so it has to be used as all the `\ifs`, with optional `\else` and obligatory `\fi`. And cannot be used in the possibly skipped branches of other `\if...`s (then it would result with ‘extra `\fi`/extra `\else`’ errors). But the below usage is safe since both `\gmd@ifsingle` and its `\else` and `\fi` are hidden in a macro (that will not be `\expandafter`).

Note also that giving `\gmd@ifsingle` an `\if...` or so as the first token of the argument will not confuse `TEX` since the first token is just gobbled. The possibility of occurrence of `\if...` or so as a not-first token seems to be negligible.

```

\gmd@ifonetoken 840 \def\gmd@ifonetoken#1#2#3{%
841 \def\@tempb{#3}% We hide #3 from TEX in case it's \if... or so. \@tempa is
      used in \gmd@ifsingle.
842 \gmd@ifsingle#3\@@nil
843 \afterfi{\@xa#1\@tempb}%
844 \else
845 \edef\@tempa{\@xa\string\@tempb}%
846 \afterfi{\@xa#2\@xa{\@tempa}}%
847 \fi}

```

Now, define the mysterious `\Hybrid@DefMacro` and `\Hybrid@DefEnvir` macros. They mark their argument with a certain subtlety: they put it in a `marginpar` at the point where they are and postpone indexing it till the first scanned occurrence or just the first code line met.

```

\Hybrid@DefMacro 848 \long\def\Hybrid@DefMacro#1{%
849 \Code@DefIndex{#1}% this macro closes the group opened by \macro.
850 \Text@MarginizeNext{#1}}

\Hybrid@DefEnvir 851 \long\def\Hybrid@DefEnvir#1{%
852 \Code@DefIndexStar{#1}% this macro also closes the group begun by \macro.
853 \Text@MarginizeNext{#1}}

\Text@MarginizeNext 854 \long\def\Text@MarginizeNext#1{%
855 \gmd@evpaddonce{\Text@Marginize{#1}\ignorespaces}}

```

The following macro adds its argument to `\everypar` using an auxiliary macro to wrap the stuff in. The auxiliary macro has a self-destructer built in so it `\relaxes` itself after first use.

```

\gmd@evpaddonce 856 \long\def\gmd@evpaddonce#1{%
857 \stepnummacro\gmd@oncenum
858 \@xa\long\@xa\edef%
859 \csname_\gmd/evp/NeuroOncer\gmd@oncenum\endcsname{%
860 \@nx\g@relaxen
861 \csname_\gmd/evp/NeuroOncer\gmd@oncenum\endcsname}% Why does it work
      despite it shouldn't? Because when the CS got with \csname...^
      \endcsname is undefined, it's equivalent \relax and therefore unex-
      pandable. That's why it passes \edef and is able to be assigned.
862 \@xa\addtomacro\csname_\gmd/evp/NeuroOncer\gmd@oncenum\endcsname{#1}%
863 \@xa\addto@hook\@xa\everypar\@xa{%
864 \csname_\gmd/evp/NeuroOncer\gmd@oncenum\endcsname}%
865 }

```

866 `\nummacro\gmd@oncenenum%` We store the number uniquifying the auxiliary macro in a macro to save count registers (cf. `gmutils` sec. [To Save Precious Count Registers](#)).

environment Wrapping a description and definition of an environment in a `macro` environment would look inappropriate (‘zgrzytało by’ in Polish) although there’s no `TeX` obstacle to do so. Therefore we define the `environment`, because of aesthetic and psychological reasons.

```
867 \@xa\let\@xa\environment\csname_\macro*\endcsname
868 \@xa\let\@xa\endenvironment\csname_\endmacro*\endcsname
```

Index Exclude List

We want some CSs not to be indexed, e.g., the `LATEX` internals and `TeX` primitives.

`doc` takes `\index@excludelist` to be a `\toks` register to store the list of expelled CSs. Here we’ll deal another way. For each CS to be excluded we’ll make (`\let`, to be precise) a control sequence and then we’ll be checking if it’s undefined (`\ifx`-equivalent `\relax`).⁹

```
\DoNotIndex 869 \def\DoNotIndex{\bgroup\MakePrivateLetters\DoNot@Index}
870 \long\def\DoNot@Index#1{\egroup% we close the group,
871 \let\gmd@iedir\gmd@justadot% we declare the direction of the cluding to be
      excluding. We act this way to be able to reverse the exclusions easily later.
872 \dont@index#1.}
873 \long\def\dont@index#1{%
874 \def\@tempa{\@nx#1}% My TeX Guru’s trick to deal with \fi and such, i.e., to
      hide from TeX when it is processing a test’s branch without expanding.
875 \if\@tempa.% a dot finishes expelling
876 \else
877 \if\@tempa,% The list this macro is put before may contain commas and that’s
      O.K., we just continue the work.
878 \afterfifi\dont@index
879 \else% what is else shall off the Index be expelled.
880 {\escapechar\m@ne
881 \xdef\@tempa{\string#1}}%
882 \@xa\let%
883 \csname_\gmd/iexcl/\@tempa\endcsname=\gmd@iedir% In the default case
      explained e.g. by the macro’s name, the last macro’s meaning is such
      that the test in line 508 will turn false and the subject CS shall not be
      indexed. We \let not \def to spare TeX’s memory.
884 \afterfifi\dont@index
885 \fi
886 \fi}
```

Let’s now give the exclude list copied ~verbatim ;) from `doc.dtx`. I give it in the code layer ’cause I suppose one will document not `LATEX` source but normal packages.

```
887 \DoNotIndex{\DoNotIndex}% the index entries of these two CSs would be rejected
      by MakeIndex anyway.
```

```
888 \begin{MakePrivateLetters}% Yes, \DoNotIndex does \MakePrivateLetters on
      its own but No, it won’t have any effect if it’s given in another macro’s \def.
```

```
\DefaultIndexExclusions 889 \gdef\DefaultIndexExclusions{%
```

⁹ This idea comes from Marcin Woliński.

890 \DoNotIndex{\@ \@par \@beginparpenalty \@empty}%
 891 \DoNotIndex{\@flushglue \@gobble \@input}%
 892 \DoNotIndex{\@makefnmark \@makeother \@maketitle}%
 893 \DoNotIndex{\@namedef \@ne \@spaces \@tempa}%
 894 \DoNotIndex{\@tempb \@tempswafalse \@tempswattrue}%
 895 \DoNotIndex{\@thanks \@thefnmark \@topnum}%
 896 \DoNotIndex{\@@ \@elt \@forloop \@fortmp \@gtempa \@totalleftmargin}%
 897 \DoNotIndex{\ " \@ifundefined \@nil \@verbatim \@vobeyspaces}%
 898 \DoNotIndex{\| \~ _ \active \advance \aftergroup \begingroup \bgroup}%
 899 \DoNotIndex{\mathcal \csname \def \documentstyle \dospecials \edef}%
 900 \DoNotIndex{\egroup}%
 901 \DoNotIndex{\else \endcsname \endgroup \endinput \endtrivlist}%
 902 \DoNotIndex{\expandafter \fi \fnsymbol \futurelet \gdef \global}%
 903 \DoNotIndex{\hbox \hss \if \if@inlabel \if@tempswa \if@twocolumn}%
 904 \DoNotIndex{\ifcase}%
 905 \DoNotIndex{\ifcat \iffalse \ifx \ignorespaces \index \input \item}%
 906 \DoNotIndex{\jobname \kern \leavevmode \leftskip \let \llap \lower}%
 907 \DoNotIndex{\m@ne \next \newpage \nobreak \noexpand
 \nonfrenchspacing}%
 908 \DoNotIndex{\obeylines \or \protect \raggedleft \rightskip \rm \sc}%
 909 \DoNotIndex{\setbox \setcounter \small \space \string \strut}%
 910 \DoNotIndex{\strutbox}%
 911 \DoNotIndex{\thefootnote \thispagestyle \topmargin \trivlist \tt}%
 912 \DoNotIndex{\twocolumn \typeout \vss \vtop \xdef \z@}%
 913 \DoNotIndex{\, \@bsphack \@esphack \@noligs \@vobeyspaces
 \@xverbatim}%
 914 \DoNotIndex{\ ' \catcode \end \escapechar \frenchspacing \glossary}%
 915 \DoNotIndex{\hangindent \hfil \hfill \hskip \hspace \ht \it \langle}%
 916 \DoNotIndex{\leaders \long \makelabel \marginpar \markboth
 \mathcode}%
 917 \DoNotIndex{\mathsurround \mbox}% % \newcount \newdimen \newskip
 918 \DoNotIndex{\nopagebreak}%
 919 \DoNotIndex{\parfillskip \parindent \parskip \penalty \raise
 \rangle}%
 920 \DoNotIndex{\section \setlength \TeX \topsep \underline \unskip}%
 921 \DoNotIndex{\vskip \vspace \widetilde \ \ \ \ @date \@defpar}%
 922 \DoNotIndex{\ []}% see line 887.
 923 \DoNotIndex{\count@ \ifnum \loop \today \uppercase \uccode}%
 924 \DoNotIndex{\baselineskip \begin \tw@}%
 925 \DoNotIndex{\a \b \c \d \e \f \g \h \i \j \k \l \m \n \o \p \q}%
 926 \DoNotIndex{\r \s \t \u \v \w \x \y \z \A \B \C \D \E \F \G \H}%
 927 \DoNotIndex{\I \J \K \L \M \N \O \P \Q \R \S \T \U \V \W \X \Y \Z}%
 928 \DoNotIndex{\1 \2 \3 \4 \5 \6 \7 \8 \9 \0}%
 929 \DoNotIndex{\! \# \\$ \% \& \ ' \ (\) \ . \ : \ ; \ < \ = \ > \ ? \ _}% \+ seems to be so
 rarely used that it may be advisable to index it.
 930 \DoNotIndex{\discretionary \immediate \makeatletter \makeatother}%
 931 \DoNotIndex{\meaning \newenvironment \par \relax \renewenvironment}%
 932 \DoNotIndex{\repeat \scriptsize \selectfont \the \undefined}%
 933 \DoNotIndex{\arabic \do \makeindex \null \number \show \write \@ehc}%
 934 \DoNotIndex{\@author \@ehc \@ifstar \@sanitize \@title}%
 935 \DoNotIndex{\if@minipage \if@restonecol \ifeof \ifmmode}%


```

936 \DoNotIndex{\lccode% %\newtoks
937 \onecolumn\openin\p@\SelfDocumenting}%
938 \DoNotIndex{\settowidth\@resetonecoltrue\@resetonecolfalse\bf}%
939 \DoNotIndex{\clearpage\closein\lowercase\@inlabelfalse}%
940 \DoNotIndex{\selectfont\mathcode\newmathalphabet\rmdefault}%
941 \DoNotIndex{\bfdefault}%

```

From the above list I removed some `\new...` declarations 'cause I think it may be useful to see gathered the special `\new...`s of each kind. For the same reason I would not recommend excluding from the index such declarations as `\AtBeginDocument`, `\AtEndDocument`, `\AtEndOfPackage`, `\DeclareOption`, `\DeclareRobustCommand` etc. But the common definitions, such as `\new/providecommand` and `\(e/g/x)defs`, as the most common, in my opinion excluded should be.

And some my exclusions:

```

942 \DoNotIndex{\@@input\@auxout\@currentlabel\@dblarg}%
943 \DoNotIndex{\@ifdefinable\@ifnextchar\@ifpackageloaded}%
944 \DoNotIndex{\@indexfile\@let@token\@sptoken\^}% the latter comes from
    CSs like \^M, see sec. 662.
945 \DoNotIndex{\addto@hook\addvspace}%
946 \DoNotIndex{\CurrentOption}%
947 \DoNotIndex{\emph\empty\firstofone}%
948 \DoNotIndex{\font\fontdimen\hangindent\hangafter}%
949 \DoNotIndex{\hyperpage\hyperlink\hypertarget}%
950 \DoNotIndex{\ifdim\ifhmode\iftrue\ifvmode\medskipamount}%
951 \DoNotIndex{\message}%
952 \DoNotIndex{\NeedsTeXFormat\newcommand\newif}%
953 \DoNotIndex{\newlabel}%
954 \DoNotIndex{\of}%
955 \DoNotIndex{\phantom\ProcessOptions\protected@edef}%
956 \DoNotIndex{\protected@xdef\protected@write}%
957 \DoNotIndex{\ProvidesPackage\providecommand}%
958 \DoNotIndex{\raggedright}%
959 \DoNotIndex{\raisebox\refstepcounter\ref\rlap}%
960 \DoNotIndex{\reserved@a\reserved@b\reserved@c\reserved@d}%
961 \DoNotIndex{\stepcounter\subsection\textit\textsf\thepage\tiny}%
962 \DoNotIndex{\copyright\footnote\label\LaTeX}%
963 \DoNotIndex{\@eha\@endparenv\if@endpe\@endpefalse\@endpetrue}%
964 \DoNotIndex{\@evenfoot\@oddfoot\@firstoftwo\@secondoftwo}%
965 \DoNotIndex{\@for\@gobbletwo\@idxitem\@ifclassloaded}%
966 \DoNotIndex{\@ignorefalse\@ignoretrue\if@ignore}%
967 \DoNotIndex{\@input@\@input}%
968 \DoNotIndex{\@latex@error\@mainaux\@nameuse}%
969 \DoNotIndex{\@nomath\@oddfoot}% %\@onlypreamble should be indexed IMO.
970 \DoNotIndex{\@outerparskip\@partaux\@partlist\@plus}%
971 \DoNotIndex{\@sverb\@sxverbatim}%
972 \DoNotIndex{\@tempcnta\@tempcntb\@tempskipa\@tempskipb}%
    I think the layout parameters even the kernel, should not be excluded:
    % \@topsep \@topsepadd \abovedisplayskip \clubpenalty etc.
973 \DoNotIndex{\@writeckpt}%
974 \DoNotIndex{\bfseries\chapter\part\section\subsection}%
975 \DoNotIndex{\subsubsection}%

```

```

976 \DoNotIndex{\char\check@mathfonts\closeout}%
977 \DoNotIndex{\fontsize\footnotemark\footnotetext\footnotesize}%
978 \DoNotIndex{\g@addto@macro\hfilneg\Huge\huge}%
979 \DoNotIndex{\hyphenchar\if@partsw\IfFileExists\in@@}%
980 \DoNotIndex{\include\includeonly\indexspace}%
981 \DoNotIndex{\itshape\language\LARGE\Large\large}%
982 \DoNotIndex{\lastbox\lastskip\m@th\makeglossary}%
983 \DoNotIndex{\maketitle\math@fontsfalse\math@fontstrue\mathsf}%
984 \DoNotIndex{\MessageBreak\noindent\normalfont\normalsize}%
985 \DoNotIndex{\on@line\openout\outer}%
986 \DoNotIndex{\parbox\part\rmfamily\rule\sbox}%
987 \DoNotIndex{\sf@size\sffamily\skip}%
988 \DoNotIndex{\textsc\textup\toks@\ttfamily\vbox}%
%%\DoNotIndex{\begin*} maybe in the future, if the idea gets popular...
989 \DoNotIndex{\hspace*\newcommand*\newenvironment*
\providecommand*}%
990 \DoNotIndex{\renewenvironment*\section*\chapter*}%
991 }% of \DefaultIndexExclusions.

```

I put all the expellings into a macro because I want them to be optional.

```
992 \end{MakePrivateLetters}
```

And we execute it due to the (lack of) counter-corresponding option:

```

993 \if@indexallmacros\else
994 \DefaultIndexExclusions
995 \fi

```

If we expelled so many CSs, someone may like it in general but he/she may need one or two expelled to be indexed back. So

```

\DoIndex 996 \def\DoIndex{\bgroup\MakePrivateLetters\Do@Index}
997 \long\def\Do@Index#1{\egroup\@relaxen\gmd\iedir\dont@index#1.}% note we
only redefine an auxiliary CS and launch also \dont@index inner macro.

```

And if a user wants here make default exclusions and there do not make them, he may use the \DefaultIndexExclusions declaration herself. This declaration OCSR, but anyway let's provide the counterpart. It OCSR, too.

```

\UndoDefaultIndexExclusions 998 \def\UndoDefaultIndexExclusions{%
999 \StoreMacro\DoNotIndex
1000 \let\DoNotIndex\DoIndex
1001 \DefaultIndexExclusions
1002 \RestoreMacro\DoNotIndex}

```

Index Parameters

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

```

\IndexPrologue 1003 \long\def\IndexPrologue#1{\@bsphack\def\index@prologue{#1}\@esphack}
\indexdiv 1004 \def\indexdiv{\@ifundefined{chapter}{\section*}{\chapter*}}
\index@prologue 1005 \@ifundefined{index@prologue}{\def\index@prologue{\indexdiv{Index}}%

```

```

1006 \markboth{Index}{Index}%
1007 Numbers written in italic refer to the \if@pageindex pages \else
1008 code \lines \fi where the
1009 corresponding entry is described; numbers underlined refer to the
1010 \if@pageindex \else code line of the \fi definition; numbers in
1011 roman refer to the \if@pageindex pages \else code \lines \fi where
1012 the entry is used.
1013 \if@pageindex \else
1014 \ifx\HLPrefix@empty
1015 The numbers preceded with 'p.' are page numbers.
1016 \else The numbers with no prefix are page numbers.
1017 \fi \fi
1018 \ifx\IndexLinksBlack\relax \else
1019 All the numbers are hyperlinks.
1020 \fi
1021 \gmd@dip@hook% this hook is intended to let a user add something without re-
        defining the entire prologue, see below.
1022 }}{}}

```

During the preparation of this package for publishing I needed only to add something at the end of the default index prologue. So

```

1023 \@emptify\gmd@dip@hook
\AtDIPrologue 1024 \long\def\AtDIPrologue#1{\g@addto@macro\gmd@dip@hook{#1}}

```

The Author(s) of doc assume multicol is known not to everybody. My assumption is the other so

```

1025 \RequirePackage{multicol}

```

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

```

\IndexMin 1026 \newdimen\IndexMin \IndexMin=\133pt\relax% originally it was set 80 pt, but
        with my default prologue there's at least 4.7 cm needed to place the prologue
        and some index entries on the same page.
IndexColumns 1027 \newcount\c@IndexColumns \c@IndexColumns=\3
theindex 1028 \renewenvironment{theindex}
1029 {\begin{multicols}\c@IndexColumns[\index@prologue][\IndexMin]%
1030 \IndexLinksBlack
1031 \IndexParms\let\item\@idxitem\ignorespaces}%
1032 {\end{multicols}}
\IndexLinksBlack 1033 \def\IndexLinksBlack{\hypersetup{linkcolor=black}}% To make Adobe Reader
        work faster.

1034 \@ifundefined{IndexParms}
\IndexParms 1035 {\def\IndexParms{%
1036 \parindent\z@
1037 \columnsep\15pt
1038 \parskip\0pt\plus\1pt
1039 \rightskip\15pt
1040 \mathsurround\z@

```

```

1041 \parfillskip=-15pt\plus1fil\% doc defines this parameter rigid but
      that's because of the stretchable space (more precisely, a \dotfill)
      between the item and the entries. But in gmdoc we define no such
      special delimiters, so we add an ifinite stretch.
1042 \small
1043 \def\@idxitem{\par\hangindent130pt}%
1044 \def\subitem{\@idxitem\hspace*{15pt}}%
1045 \def\subsubitem{\@idxitem\hspace*{25pt}}%
1046 \def\indexspace{\par\vspace{10pt\plus2pt\minus3pt}}%
1047 \ifx\EntryPrefix\@empty\else\raggedright\fi% long (actually, a quite
      short but nonempty entry prefix) made space stretches so terribly large
      in the justified paragraphs that we should make \raggedright rather.
1048 \ifnum\c@IndexColumns>\tw@\raggedright\fi% the numbers in narrow col-
      umns look better when they are \raggedright in my opinion.
1049 }}{}

```

```

\PrintIndex 1050 \def\PrintIndex{% we ensure the standard meaning of the line end character not to
      cause a disaster.

```

```

1051 \ifQueerEOL{\StraightEOL\printindex\QueerEOL}{\printindex}}

```

Remember that if you want to change not all the parameters, you don't have to redefine the entire `\IndexParms` macro but you may use a very nice \LaTeX command `\g@addto@macro` (it has `\global` effect, also with an apeless name (`\gaddtomacro`) provided by `gmutils`. (It adds its second argument at the end of definition of its first argument provided the first argument is a no-argument macro.) Moreover, `gmutils` provides also `\addtomacro` that has the same effect except it's not `\global`.

The DocStrip Directives

```

1052 {\@makeother\<\@makeother\>
1053 \glet\sgtleftxii=<
\gmd@docstripdirective 1054 \gdef\gmd@docstripdirective{%
1055 \begingroup\let\do=\@makeother
1056 \do*\do/\do+\do-\do\,\do\&\do\|\do\!\do\(\do\)\do\>\do\<%
1057 \@ifnextchar{<}{%
1058 \let\do=\@makeother\dospecials
1059 \gmd@docstripverb}
1060 {\gmd@docstripinner}}%

1061 \if11%
1062 \gdef\Debug@dstroon{\ifnum\c@codelinenum>1178\ifdtraceon\fi
1063 \ifnum\c@codelinenum>1184\ifdtraceoff\emptify\Debug@dstroon\fi}%
1064 \else
1065 \global\emptify\Debug@dstroon
1066 \fi

1067 \gdef\gmd@docstripinner#1>{%
1068 \endgroup
1069 \def\gmd@modulehashone{%
1070 \Module{#1}\space
1071 \@afternarrgfalse\@aftercodegtrue\@codeskipputgfalse}%
1072 \gmd@textEOL\gmd@modulehashone\Debug@dstroon}

```

A word of explanation: first of all, we close the group for changed `\catcodes`; the directive's text has its `\catcodes` fixed. Then we put the directive's text wrapped with the formatting macro into one macro in order to give just one token the `gmdoc`'s `TEX` code scanner. Then launch this big `TEX` code scanning machinery by calling `\gmd@textEOL` which is an alias for the 'narrative' meaning of the line end. This macro opens the verbatim group and launches the char-by-char scanner. That is this scanner because of what we encapsulated the directive's text with the formatting into one macro: to let it pass the scanner. That's why in the 'old' macrocodes case the active `%` closes the group before launching `\gmd@docstripdirective`.

The 'verbatim' directive macro works very similarly.

```

1073 \catcode'\^^M=\active%
1074 \gdef\gmd@docstripverb<#1^^M{%
1075   \endgroup%
1076   \def\gmd@modulehashone{%
1077     \ModuleVerb{#1}\@afternarrgfalse\@aftercodegtrue%
1078     \@codeskipputgfalse}%
1079   \gmd@docstripshook%
1080   \gmd@textEOL\gmd@modulehashone^^M}%
1081 }

  (~Verbatim ;- ) from doc:)

\Module 1082 \providecommand*\Module[1]{\{\mod@math@codes$\langle\mathsf{#1}^
        \rangle$\}}

\ModuleVerb 1083 \providecommand*\ModuleVerb[1]{\{\mod@math@codes$\langle\langle^
        \mathsf{#1}$\}}

1084 \def\mod@math@codes{\mathcode'\|= "226A_\mathcode'\&="2026_\}

```

The Changes History

The contents of this section was copied ~verbatim from the `doc`'s documentation, with only smallest necessary changes. Then my additions were added :-)).

“To provide a change history log, the `\changes` command has been introduced. This takes [one optional and] three [mandatory] arguments, respectively, [the macro that'll become the entry's second level,] the version number of the file, the date of the change, and some detail regarding what change has been made [i.e., the description of the change]. 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. [... I ommit an obsolete remark about then-older `MakeIndex`'s versions.]

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

We put the definition inside a macro that will be executed by (the first use of) `\RecordChanges`. And we provide the default definition of `\changes` as a macro just

gobbling its arguments. We do this to provide no changes' writing out if `\RecordChanges` is not used.

```

\gmd@DefineChanges 1085 \def\gmd@DefineChanges{%
\changes 1086 \outer\long\def\changes{\@bsphack\begingroup\@sanitize
1087 \catcode'\z@\catcode'\_10\MakePercentIgnore
1088 \MakePrivateLetters\_StraightEOL
1089 \MakeGlossaryControls
1090 \changes@}}

1091 \newcommand\changes[4][\PackageWarningNoLine{gmdoc}{%
1092 ^^JThe\_bslash\_changes\_command\_used\_on\_line
1093 ^^Jwith\_no\_string\RecordChanges\space\_declared.
1094 ^^JI\_shall\_not\_warn\_you\_again\_about\_it}%
1095 \renewcommand\changes[4][{}]}

1096 \def\MakeGlossaryControls{%
1097 \def\actualchar{=}\def\quotechar{!}%
1098 \def\levelchar{>}\edef\encapchar{\twelveclub}}% for the glossary the 'ac-
tual', the 'quote' and the 'level' chars are respectively =, ! and >, the
'encap' char remains untouched. I decided to preserve the doc's settings for
the compatibility.

\changes@ 1099 \newcommand\changes@[4][\generalname]{%
1100 \if@RecentChange{#3}% if the date is later than the one stored in \c@Changes-
% StartDate,
1101 \@tempswafalse
1102 \ifx\generalname#1% then we check whether a CS-entry is given in the optional
first argument or is it unchanged.
1103 \ifx\last@defmark\relax\else% if no particular CS is specified in #1, we
check whether \last@defmark contains something and if so, we put it
into \@tempb scratch macro.
1104 \@tempswatrue
1105 \edef\@tempb{% it's a bug fix: while typesetting traditional .dtxes, \last@defmark
came out with \ at the beginning (which resulted with \\<name> in
the change log) but while typesetting the 'new' way, it occurred with-
out the bslash. So we gobble the bslash if it's present and two lines
below we handle the exception of \last@defmark = {} (what would
happen if a definition of \\ was marked in new way gmdocing).
1106 \if\bslash\last@defmark\else\last@defmark\fi}%
1107 \ifx\last@defmark\bslash\let\@tempb\last@defmark\fi%
1108 \fi
1109 \else% the first argument isx not \generalname i.e., a particular CS is specified
by it (if some day one wishes to \changes \generalname, he should type
\changes[generalname]...)
1110 \@tempswatrue
1111 {\escapechar\m@ne
1112 \xdef\@tempb{\string#1}}
1113 \fi
1114 \protected@edef\@tempa{\@nx\glossary{%
1115 \if\relax\GeneralName\relax\else
1116 \GeneralName% it's for the \DocInclude case to precede every \changes
of the same file with the file name, cf. line 1244.

```

```

1117      \fi
1118      #2\levelchar%
1119      \if@tempswa% If the macro \last@defmark doesn't contain any CS name
                  (i.e., is empty) nor #1 specifies a CS, the current changes entry was
                  done at top-level. In this case we precede it by \generalname.
1120      \@tempb
1121      \actualchar\slash\_verb*%
1122      \if\verbatimchar\@tempb$\else\verbatimchar\fi
1123      \quotechar\slash\_@tempb
1124      \if\verbatimchar\@tempb$\else\verbatimchar\fi
1125      \else
1126      \space\actualchar\generalname
1127      \fi
1128      :\levelchar#4\encapchar\_hyperpage}}}%
1129  \@tempa
1130  \fi\endgroup\@esphack}

```

Let's initialize \last@defmark and \GeneralName.

```

1131  \@relaxen\last@defmark
1132  \@emptify\GeneralName

```

Let's explain \if@RecentChange. We wish to check whether the change's date is later than date declared (if any limit date *was* declared). First of all, let's establish a counter to store the declared date. The untouched counters are equal 0 so if no date is declared there'll be no problem. The date will have the $\langle YYYYMMDD \rangle$ shape both to be easily compared and readable.

```

\c@ChangesStartDate 1133  \newcount\c@ChangesStartDate
\if@RecentChange 1134  \def\if@RecentChange#1{%
1135      \gmd@setChDate#1\@nil\@tempcnta
1136      \ifnum\@tempcnta>\c@ChangesStartDate}
1137  \def\gmd@setChDate#1/#2/#3\@nil#4{% the last parameter will be a \count reg-
                  ister.
1138      #4=#1\relax
1139      \multiply#4\_by\@M
1140      \count8=#2\relax% I know it's a bit messy not to check whether the #4 \count
                  is \count8 but I know this macro will only be used with \count0\_(\@tem-
                  % pcnta) and some higher (not a scratch) one.
1141      \multiply\count8\_by100\_%
1142      \advance#4\_by\count8\_by\count8=\z@
1143      \advance#4\_by#3\relax}

```

Having the test defined, let's define the command setting the date counter. #1 is to be the version and #2 the date $\{\langle year \rangle / \langle month \rangle / \langle day \rangle\}$.

```

\ChangesStart 1144  \def\ChangesStart#1#2{%
1145      \gmd@setChDate#2\@nil\c@ChangesStartDate
1146      \typeout{\JPackage\gmdoc\_info:\JChanges\_start\_date\_#1\_memorized
1147      as\_string<\the\c@ChangesStartDate\string>\_on\_online.\_J}
1148      \advance\c@ChangesStartDate\_m@ne% we shall show the changes at the specified
                  day and later.
1149      \ifnum\c@ChangesStartDate>19820900\_%10 see below.

```

¹⁰ DEK writes in *T_EX, The Program* of September 1982 as the date of T_EX Version 0.

```

1150 \edef\@tempa{%
1151 \@nx\g@addto@macro\@nx\glossary@prologue{%
1152 The\changes
1153 \if\relax\GeneralName\relax\else\of\GeneralName\space\fi
1154 earlier\than
1155 #1\if\relax#1\relax\#2\else(#2)\fi\space\are\not\shown.}}%
1156 \@tempa
1157 \fi}

```

(Explanation to line 1149.) My TeX Guru has remarked that the change history tool should be used for documenting the changes that may be significant for the users not only for the author and talking of what may be significant to the user, no changes should be hidden since the first published version. However, the changes' start date may be used to provide hiding the author's 'personal' notes: she should only date the 'public' changes with the four digit year and the 'personal' ones with two digit year and set `\ChangesStart{}\{1000/0/0}` or so.

In line 1149 I establish a test value that corresponds to a date earlier than any TeX stuff and is not too small (early) to ensure that hiding the two digit year changes shall not be mentioned in the changes prologue.

“The entries [of a given version number] are sorted for convenience by the name of [the macro explicitly specified as the first argument or] the most recently introduced macroname (i.e., that in the most recent `\begin{macro}` command [or `\CodeDefine`]). We therefore provide `[\last@defmark]` 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 ".)

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

```

\generalname 1158 \def\generalname{General}

```

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

I add to it also the `\writeing` definition of the `\changes` macro to ensure no changes are written out without `\RecordChanges`.

```

\RecordChanges 1159 \def\RecordChanges{\makeglossary\gmd@DefineChanges
1160 \relaxen\RecordChanges}

```

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

```

1161 \newdimen\GlossaryMin \GlossaryMin = 80pt
1162 \newcount\c@GlossaryColumns \c@GlossaryColumns = 2

```

“The environment `theglossary` is defined in the same manner as the `theindex` environment.”

```

theglossary 1163 \newenvironment{theglossary}{%
1164 \begin{multicols}\c@GlossaryColumns
1165 [\glossary@prologue][\GlossaryMin]%
1166 \GlossaryParms\let\item\idxitem\ignorespaces}%
1167 {\end{multicols}}

```


Here is the MakeIndex style definition:

```

1168 </package>
1169 <+gmglo> preamble
1170 <+gmglo> "\n_\n\begin{theglossary}_\n
1171 <+gmglo> \\\makeatletter\n"
1172 <+gmglo> postamble
1173 <+gmglo> "\n\n_\n\end{theglossary}_\n"
1174 <+gmglo> keyword_\n\glossaryentry"
1175 <+gmglo> actual_'='
1176 <+gmglo> quote_'!'
1177 <+gmglo> level_'>'
1178 <*package>

```

The MakeIndex shell command for the glossary should look as follows:

```
makeindex -r -s gmglo.ist -o <myfile>.gls <myfile>.glo
```

where `-r` commands MakeIndex not to make implicit page ranges, `-s` commands MakeIndex to use the style stated next not the default settings and the `-o` option with the subsequent filename defines the name of the output.

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

```

\GlossaryPrologue 1179 \long\def\GlossaryPrologue#1{\@bsphack
1180 \def\glossary@prologue{#1}%
1181 \@esphack}

```

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

```

1182 \@ifundefined{glossary@prologue}
\glossary@prologue 1183 {\def\glossary@prologue{\indexdiv{{Change_History}}}%
1184 \markboth{{Change_History}}{{Change_History}}%
1185 }}{}

```

“Unless the user specifies otherwise, we set the change history using the same parameters as for the index.”

```

1186 \AtBeginDocument{%
\GlossaryParms 1187 \@ifundefined{GlossaryParms}{\let\GlossaryParms\IndexParms}{}}

```

“To read in and print the sorted change history, just put the `\PrintChanges` command as the last (commented-out, and thus executed during the documentation pass through the file) command in your package file. Alternatively, this command may form one of the arguments of the `\StopEventually` command, although a change history is probably not required if only the description is being printed. The command assumes that MakeIndex or some other program has processed the `.glo` file to generate a sorted `.gls` file.”

```

\PrintChanges 1188 \def\PrintChanges{% to avoid a disaster among queer EOLs:
1189 \@ifQueerEOL
1190 {\StraightEOL\@input@{\jobname.gls}\QueerEOL}%
1191 {\@input@{\jobname.gls}}%
1192 \g@emptyify\PrintChanges}

```

The Checksum

`doc` provides a checksum mechanism that counts the backslashes in the scanned code. Let's do almost the same.

At the beginning of the source file you may put the `\Checksum` macro with a number (in one of `TEX`'s formats) as its argument and `TEX` with `gmdoc` shall count the number of the *escape chars* in the source file and tell you in the `.log` file (and on the terminal) whether you have typed the right number. If you don't type `\Checksum`, `TEX` anyway will tell you how much it is.

```
\check@sum 1193 \newcount\check@sum
\Checksum 1194 \def\Checksum#1{\@bsphack\global\check@sum#1\relax\@esphack}
Checksum 1195 \newcounter{Checksum}
1196 \newcommand*\step@checksum{\stepcounter{Checksum}}
```

And we'll use it in the line 456 (`\stepcounter` is `\global`). See also the `\chscchange` declaration, l. 1231.

However, the check sum mechanism in `gmdoc` behaves slightly different than in `doc` which is nicely visible while `gmdoc`ing `doc`: `doc` states its check sum to be 2171 and our count counts 2126. The mystery lies in the fact that `doc`'s `Checksum` mechanism counts the code's backslashes no matter what they mean and the `gmdoc`'s the escape chars so, among others, `\\` at the default settings increases `doc`'s `Checksum` by 2 while the `gmdoc`'s by 1. (There are 38 occurrences of `\\` in `doc.dtx` macrocodes, I counted myself.)¹¹

"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."

In `gmdoc` we have the `\AtEndInput` hook.

```
1197 \AtEndInput{\check@checksum}
```

Based on the lines 723–741 of `doc.dtx`.

```
\check@checksum 1198 \def\check@checksum{\relax
1199 \ifnum\check@sum=\z@
1200 \typeout{*****}%
1201 \typeout{*The input file \gmd@inputname\space has no Checksum
1202 stated.}%
1203 \typeout{*The current checksum is \the\c@Checksum.}%
1204 \gmd@chscchangeline% a check sum changes history entry, see below.
1205 \typeout{* (package gmdoc info.)}
1206 \typeout{*****}%
1207 \else
1208 \ifnum\check@sum=\c@Checksum
1209 \typeout{*****}%
1210 \typeout{*The input file \gmd@inputname: Checksum passed.}%
1211 \gmd@chscchangeline
1212 \typeout{* (package gmdoc info.)}
1213 \typeout{*****}%
1214 \else
1215 \typeout{*****!!!!!!}%
1216 \typeout{*The input file \gmd@inputname:}
1217 \typeout{*The CheckSum stated: \the\check@sum\space<>my
```

¹¹ My opinion is that nowadays a check sum is not necessary for checking the completeness of a file but I like it as a marker of file development and this more than that is its rôle in `gmdoc`.

```

1218         count:\the\c@Checksum.}
1219         \gmd@chschangeline
1220         \typeout{*\the\c@Checksum.}
1221         \typeout{*****!*****!*****!*****!*****!*****!}%
1222     \fi
1223 \fi
1224 \global\check@sum\z@}

```

As I mentioned above, I use the check sum mechanism to mark the file growth. Therefore I provide a macro that produces a line on the terminal to be put somewhere at the beginning of the source file's commentary for instance.

```

1225 \def\gmd@chschangeline{%
\chschange 1226 \typeout{\twelvepercent\space\string\chschange{%
1227         \fileversion}{\the\year/\the\month/\the\day}{\the\c@Checksum}}
1228 \typeout{\twelvepercent\space\string\chschange{\fileversion}{%
1229         \@xa\@gobbletwo\the\year/\the\month/\the\day}{% with two digit year
                in case you use \ChangesStart.
1230         \the\c@Checksum}}}

```

And here the meaning of such a line is defined:

```

1231 \newcommand*\chschange[3]{%
1232     \csname\changes\endcsname{#1}{#2}{Checksum_#3}% \csname... 'cause \cha-
        % nges is \outer.
1233     \Checksum{#3}}

```

It will make a 'General' entry in the change history unless used in some `\CodeDefine`'s scope or inside a macro environment. It's intended to be put somewhere at the beginning of the documented file.

Macros from ltxdoc

I'm not sure whether this package still remains 'minimal' but I liked the macros provided by ltxdoc.cls so much...

The next page setup declaration is intended to be used with the article's default Letter paper size. But since

```

\ltxPageLayout 1234 \newcommand*\ltxPageLayout{%
        "Increase the text width slightly so that width the standard fonts 72 columns of code
        may appear in a macrocode environment."
1235     \setlength{\textwidth}{355pt}%
        "Increase the marginpar width slightly, for long command names. And increase the
        left margin by a similar amount."
        To make these settings independent from the defaults (changed e.g. in gmdocc.cls) we
        replace the original \addtolengths with \setlengths.
1236     \setlength\marginparwidth{95pt}%
1237     \setlength\oddsidemargin{82pt}%
1238     \setlength\evensidemargin{82pt}}

```

\DocInclude and the ltxdoc-Like Setup

Let's provide a command for including multiple files into one document. In the `ltxdoc` class such a command is defined to include files as parts. But we prefer to include them as chapters in the classes that provide `\chapter`. We'll redefine `\maketitle` so that it make a chapter or a part heading *unlike* in `ltxdoc` where the file parts have their titlepages with only the filename and article-like titles made by `\maketitle`.

But we will also provide a possibility of typesetting multiple files exactly like with the `ltxdoc` class.

```
\DocInclude      So, define the \DocInclude command, that acts
                  "more or less exactly the same as \include, but uses \DocInput on a dtx [or .fdd]
                  file, not \input on a tex file."
                  Our version will accept also .sty, .cls, and .tex files.

1239 \newcommand*\DocInclude{\bgroup\@makeother\_ \Doc@Include}% First, we make
      % _ 'other' in order to allow it in the filenames.

\Doc@Include 1240 \newcommand*{\Doc@Include}[2] []{% originally it took just one argument. Here we
                  make it take two, first of which is intended to be the path (with the closing /).
                  This is intended not to print the path in the page footers only the filename.

1241 \egroup% having the arguments read, we close the group opened by the previous
                  macro for _12.

\HLPrefix 1242 \gdef\HLPrefix{\filesep}%
\EntryPrefix 1243 \gdef\EntryPrefix{\filesep}% we define two rather kernel parameters to expand
                  to the file marker. The first will bring the information to one of the default
                  % \IndexPrologue's \ifs. Therefore the definition is global. The latter is
                  such for symmetry.

\GeneralName 1244 \def\GeneralName{#2\actualchar\pk{#2}\_}% for the changes'history main level
                  entry.

1245 \relax
1246 \clearpage
1247 \docincludeaux
1248 \def\currentfile{gmdoc-IncludeFileNotFound.000}%
1249 \let\fullcurrentfile\currentfile
1250 \IfFileExists{#1#2.fdd}{\edef\currentfile{#2.fdd}}{% it's not .fdd,
1251 \IfFileExists{#1#2.dtx}{\edef\currentfile{#2.dtx}}{% it's not .dtx ei-
                  ther,
1252 \IfFileExists{#1#2.sty}{\edef\currentfile{#2.sty}}{% it's not .sty,
1253 \IfFileExists{#1#2.cls}{\edef\currentfile{#2.cls}}{% it's not .cls,
1254 \IfFileExists{#1#2.tex}{\edef\currentfile{#2.tex}}{% it's not .tex,
1255 \IfFileExists{#1#2.fd}{\edef\currentfile{#2.fd}}{% so it must
                  be .fd or error.
1256 \PackageError{gmdoc}{\string\DocInclude\space\_file
1257 #1#2.fdd/dtx/sty/cls/tex/fd\_not\_found.}}}}}}}%
1258 \edef\fullcurrentfile{#1\currentfile}%
1259 \ifnum\@auxout=\@partaux
1260 \latexerr{\string\DocInclude\space\_cannot\_be\_nested}\@eha
1261 \else\_ \docinclude{#1}#2\_ \fi% Why is #2 delimited with _ not braced as we
                  are used to, one may ask.

\@docinclude 1262 \def\@docinclude#1#2\_ {% To match the macro's parameter string, is an answer. But
                  why is \@docinclude defined so? Originally, in ltxdoc it takes one argument
```

and it's delimited with a space probably in resemblance to the true `\input`
`% (\input in LATEX).`

```
1263 \clearpage
1264 \if@filesw\gmd@writemauxinpau{#2.aux}\fi% this strange macro with a long
      name is another thing to allow _ in the filenames (see line 1291).
1265 \@tempswattrue
1266 \if@partsw\@tempswafalse\edef\@tempb{#2}%
1267 \for_\@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswattrue\fi}%
1268 \fi
1269 \if@tempswa\let\@auxout\@partaux
1270 \if@filesw
1271 \immediate\openout\@partaux#2.aux\relax% Yes, only #2. It's to cre-
      ate and process the partial .aux files always in the main document's
      (driver's) directory.
1272 \immediate\write\@partaux{\relax}%
1273 \fi
```

“We need to save (and later restore) various index-related commands which might be changed by the included file.”

```
1274 \StoringAndRelaxingDo\gmd@doIndexRelated
1275 \if@ltxDocInclude\part{\currentfile}% In the ltxdoc-like setup we make
      a part title page with only the filename and the file's \maketitle will
      typeset an article-like title.
1276 \else\let\maketitle=\InclMaketitle
1277 \fi% In the default setup we redefine \maketitle to typeset a common chapter
      or part heading.
1278 \if@ltxDocInclude\xdef@filekey\fi
1279 \GetFileInfo{\currentfile}% it's my (GM) addition with the account of using
      file info in the included files' title/heading etc.
1280 \incl@DocInput{\fullcurrentfile}% originally just \currentfile.
1281 \if@ltxDocInclude\else\xdef@filekey\fi% in the default case we add new
      file to the file key after the input because in this case it's the files own
      \maketitle what launches the sectioning command that increases the
      counter.
```

And here is the moment to restore the index-related commands.

```
1282 \RestoringDo\gmd@doIndexRelated
1283 \clearpage
1284 \gmd@writeckpt{#1#2}%
1285 \if@filesw\immediate\closeout\@partaux\fi
1286 \else\@nameuse{cp@#1#2}%
1287 \fi
1288 \let\@auxout\@mainaux}% end of \@docinclude.
```

(Two is a sufficient number of iterations to define a macro for.)

```
\xdef@filekey 1289 \def\xdef@filekey{{\@relaxen\ttfamily}% This assignment is very trickly crafted:
      it makes all \ttfamilys present in the \filekey's expansion unexpandable
      not only the one added in this step.
1290 \xdef\filekey{\filekey,\thefilediv={\ttfamily\currentfile}}}}
```

To allow `_` in the filenames we must assure `_` will be `_12` while reading the filename.
 Therefore define

```
1291 \def\gmd@writemauxinpaux#1{% this name comes from ‘write outto main .aux to
      input partial .aux’.
```

We wrap `\@input{<partial .aux>}` in a `_12` hacked scope. This hack is especially recommended here since the `.aux` file may contain a non-`\global` stuff that should not be localized by a group that we would have to establish if we didn’t use the hack. (Hope you understand it. If not, notify me and for now I’ll only give a hint: “Look at it with the \TeX ’s eyes”. More uses of this hack are to be seen in `gmutils` where they are a bit more explained.)

```
1292 \immediate\write\@mainaux{%
1293   \bgroup\string\@makeother\string\_%
1294   \string\firstofone{\egroup
1295   \string\@input{#1}}}}
```

We also slightly modify a \LaTeX kernel macro `\@writeckpt` to allow `_` in the file name.

```
1296 \def\gmd@writeckpt#1{%
1297   \immediate\write\@partaux{%
1298     \string\bgroup\string\@makeother\string\_%
1299     \string\firstofone{\charlb\string\egroup}
1300   \@writeckpt{#1}%
1301   \immediate\write\@partaux{\@charrb}}
```

```
\gmd@doIndexRelated 1302 \def\gmd@doIndexRelated{%
1303   \do\tableofcontents_\do\makeindex_\do\EnableCrossrefs
1304   \do\PrintIndex_\do\printindex_\do\RecordChanges_\do\PrintChanges
1305   \do\theglossary_\do\endtheglossary}
1306 \@emptify\filesep
```

The `ltxdoc` class establishes a special number format for multiple file documentation numbering needed to document the \LaTeX sources. I like it too, so

```
\aalph 1307 \def\aalph#1{\@aalph{\csname_c@#1\endcsname}}
\@aalph 1308 \def\@aalph#1{%
1309   \ifcase#1\or_a\or_b\or_c\or_d\or_e\or_f\or_g\or_h\or_i\or
1310     j\or_k\or_l\or_m\or_n\or_o\or_p\or_q\or_r\or_s\or
1311     t\or_u\or_v\or_w\or_x\or_y\or_z\or_A\or_B\or_C\or
1312     D\or_E\or_F\or_G\or_H\or_I\or_J\or_K\or_L\or_M\or
1313     N\or_O\or_P\or_Q\or_R\or_S\or_T\or_U\or_V\or_W\or
1314     X\or_Y\or_Z\else\@ctrerr\fi}
```

A macro that initialises things for `\DocInclude`.

```
\docincludeaux 1315 \def\docincludeaux{%
```

We set the things for including the files only once.

```
1316 \global\@relaxen\docincludeaux
```

By default, we will include multiple files into one document as chapters in the classes that provide `\chapter` and as parts elsewhere.

```
1317 \ifx\filediv\relax
1318   \ifx\filedivname\relax% (nor \filediv neither \filedivname is defined by
      the user)
1319   \@ifundefined{chapter}{%
1320     \SetFileDiv{part}}}%
```

```

1321     {\SetFileDiv{chapter}}%
1322     \else% (\filedivname is defined by the user, \filediv is not)
1323     \SetFileDiv{\filedivname}% why not? Inside is \edef so it'll work.
1324     \fi
1325 \else% (\filediv is defined by the user
1326     \ifx\filedivname\relax% and \filedivname is not)
1327     \PackageError{gmdoc}{You've redefined \string\filediv\space
1328     without redefining \string\filedivname.}{Please redefine the
1329     two macros accordingly. You may use \string\SetFileDiv{name
1330     without\bslash}.}%
1331     \fi
1332 \fi
1333 \@addtoreset{codelinenum}{\filedivname}% remember it has a \global effect
1334     in fact. For each file we'll reset codelinenum.
1335 \def\thefilediv{\aalph{\filedivname}}% The files will be numbered with let-
1336     ters, lowercase first.
1337 \let\filekey=\@gobble
1338 \g@addto@macro\index@prologue{%
1339     \gdef\@oddfoot{\parbox{\textwidth}{\strut\footnotesize
1340     \raggedright{\bfseries\filekey}}% The footer for the
1341     pages of index.
1342     \glet\@evenfoot\@oddfoot}% anyway, it's intended to be onside.
1343 \g@addto@macro\glossary@prologue{%
1344     \gdef\@oddfoot{\strut\ChangeHistory\hfill\thepage}% The footer for
1345     the changes history.
1346     \glet\@evenfoot\@oddfoot}%
1347 \gdef\@oddfoot{% The footer of the file pages will be its name and, if there is
1348     a file info, also the date and version.
1349     \xa\ifx\curname\currentfile\endcurname\relax
1350     File\thefilediv:\ttfamily\currentfile}%
1351     \else
1352     \GetFileInfo{\currentfile}%
1353     File\thefilediv:\ttfamily\filename}%
1354     Date:\filedate\%
1355     Version\fileversion
1356     \fi
1357     \hfill\thepage}%
1358 \glet\@evenfoot\@oddfoot% see line 1341.
1359 \xa\def\curname\filedivname\endcurname{File}% we redefine the name
1360     of the proper division to 'File'.
1361 \ifx\filediv\section
1362     \let\division=\subsection
1363     \let\subdivision=\subsubsection% If \filediv is higher than \section we
1364     don't change the two divisions (they are \section and \subsection by
1365     default). \section seems to me the lowest reasonable sectioning com-
1366     mand for the file. If \filediv is lower you should rather rethink the level
1367     of a file in your documentation not redefine the two divisions.
1368 \fi}% end of \docincludeaux.

```

The `\filediv` and `\filedivname` macros should always be set together. Therefore provide a macro that takes care of both at once. Its `#1` should be a sectioning name without the backslash.

```
\SetFileDiv 1361 \def\SetFileDiv#1{%
1362     \edef\filedivname{#1}%
1363     \@xa\let\@xa\filediv\csname#1\endcsname}
```

\SelfInclude

I needed to include the driver file into a documentation so I wrote a macro in case I'll need it again or you'll need it. We define it immediately i.e., without the `\catcodes` trick because it uses `\jobname` inside so the filename will be all ₁₂ anyway.

```
\SelfInclude 1364 \newcommand*{\SelfInclude}[2][.tex]{% As you guess, the optional #1 is the job-
name's extension. The second parameter is not for the filename (note it's
known: as \jobname!), but for the stuff to be put at begin input.
1365     \AtBegInputOnce{#2}%
1366     \gdef\HLPrefix{\filesep}%
1367     \gdef\EntryPrefix{\filesep}% we define two rather kernel parameters etc. as
in \DocInclude.
1368     \relax
1369     \clearpage
1370     \docincludeaux
1371     \edef\currentfile{\jobname#1}%
1372     \let\fullcurrentfile\currentfile
1373     \def\GeneralName{\jobname\actualchar\pk{\jobname}_}% for the changes his-
tory main level entry.
1374     \ifnum\@auxout=\@partaux
1375         \@latexerr{\string\DocInclude\space cannot be nested}\@eha
1376     \else
1377         \if@filesw
1378             \gmd@writemauxinpau{\jobname.auxx}% this queer macro allows _ in the
file names. In this particular case \string\jobname would do, but
anyway we provide a more general solution. Note the .auxx extension
used instead of .aux. This is done to avoid an infinite recurrence of
% \inputs.
1379             \fi
1380             \@tempswatrue
1381             \if@partsw_\@tempswafalse\edef\@tempb{\jobname}%
1382             \@for
1383             \@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswatrue\fi}%
1384             \fi
1385             \if@tempswa_\let\@auxout\@partaux
1386             \if@filesw
1387                 \immediate\openout\@partaux_\jobname.auxx\relax
1388                 \immediate\write\@partaux{\relax}
1389             \fi
1390             \StoringAndRelaxingDo% provided by gmutils
1391             \gmd@doIndexRelated
1392             \if\ltxDocInclude\part{\currentfile}%
```

“We need to save (and later restore)...”


```

1393     \else\let\maketitle=\InclMaketitle
1394     \fi
1395     \if@ltxDocInclude\xdef@filekey\fi
1396     \GetFileInfo{\currentfile}% it's my (GM) addition with the account of
        using file info in the included files' title etc.
1397     \incl@DocInput{\fullcurrentfile}% originally just \currentfile, no dif-
        ference in \SelfInclude.
1398     \if@ltxDocInclude\else\xdef@filekey\fi% in the default case we add new
        file to the file key after the input because in this case it's files own
        \maketitle what launches the sectioning command that increases the
        counter.

```

And here is the moment to restore the index-related commands.

```

1399     \RestoringDo
1400     \gmd@doIndexRelated
1401     \clearpage% among others, causes the \writes to be executed which is crucial
        for proper toc-ing e.g.
1402     \gmd@writeckpt{\jobname.x}% note the .x in the checkpoint used to distin-
        guish this instance (input) of the driver file from its main instance.
1403     \if@filesw_\immediate\closeout\@partaux_\fi
1404     \else\@nameuse{cp@\jobname.x}% note .x: it's used for the same reason as
        above.
1405     \fi
1406     \let\@auxout\@mainaux
1407     \fi}% end of \SelfInclude.

```

The ltxdoc class makes some preparations for inputting multiple files. We are not sure if the user wishes to use ltxdoc-like way of documenting (maybe he will prefer what I offer, gmdocc.cls e.g.), so we put those preparations into a declaration.

```

1408 \newif\if@ltxDocInclude
\ltxLookSetup 1409 \newcommand*\ltxLookSetup{%
1410     \SetFileDiv{part}%
1411     \ltxPageLayout
1412     \@ltxDocIncludetrue
1413 }
1414 \@onlypreamble\ltxLookSetup

```

The default is that we \DocInclude the files due to the original gmdoc input settings.

```

1415 \let\incl@DocInput=\DocInput
1416 \@emptyify\currentfile% for the pages outside the \DocInclude's scope. In force
        for all includes.

```

If you want to \Doc/SelfInclude doc-like:

```

\olddocIncludes 1417 \newcommand*\olddocIncludes{%
1418     \let\incl@DocInput=\OldDocInput}

```

And, if you have set the previous and want to set it back:

```

\gmdocIncludes 1419 \newcommand*\gmdocIncludes{%
1420     \let\incl@DocInput=\DocInput
1421     \AtBeginInput{\QueerEOL}}% to move back the \StraightEOL declaration put at
        begin input by \olddocIncludes.

```

Redefinition of \maketitle

\maketitle A not-so-slight alteration of the \maketitle command in order it allow multiple titles in one document seems to me very clever. So let's copy again (ltxdoc.dtx the lines 643–656):

“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!”

But here in gmdoc we'll do it locally for (each) input not to change the main title settings if there are any.

```

1422 \AtBegInput{%
\maketitle 1423   \providecommand*\maketitle{\par
1424     \begingroup\def\thefootnote{\fnsymbol{footnote}}%
1425     \setcounter{footnote}\z@
1426     \def\@makefnmark{\hbox{to\z@{\m@th^{\@thefnmark}}\hss}}%
1427     \long\def\@makefntext##1{\parindent\lempar\noindent
1428       \hbox{to1.8em{\hss\m@th^{\@thefnmark}}\hss}}%
1429     \if@twocolumn\@twocolumn[\@maketitle]%
1430     \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.”

```

1431   \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:”

```

1432   \setcounter{footnote}\z@
1433   \gdef\@date{\today}\g@emptify\@thanks%
1434   \g@emptify\@author\g@emptify\@title%
1435   }%

```

“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.”

```

1436   \@ifundefined{ps@titlepage}{\let\ps@titlepage=\ps@plain}{}%

```

And let's provide \@maketitle just in case: an error occurred without it at T_EXing with mwbk.cls because this class with the default options does not define \@maketitle. The below definitions are taken from report.cls and mwrep.cls.

```

\@maketitle 1437   \providecommand*\@maketitle{%
1438     \newpage\null\vskip2em\relax%
1439     \begin{center}%
1440       \titlesetup
1441       \let\footnote\thanks
1442       {\LARGE\@title\par}%
1443       \vskip1.5em%
1444       {\large\lineskip.5em%
1445         \begin{tabular}[t]{c}%
1446           \strut\@author
1447           \end{tabular}\par}%

```

```

1448      \vskip1em%
1449      {\large\@date}%
1450      \end{center}%
1451      \par\vskip1.5em\relax}%

```

We'd better restore the primary meanings of the macros making a title. (L^AT_EX 2_ε source, File F: ltsect.dtx Date: 1996/12/20 Version v1.0z, lines 3.5.7.9–12.14–17.)

```

\title 1452      \providecommand*\title[1]{\gdef\@title{#1}}
\author 1453      \providecommand*\author[1]{\gdef\@author{#1}}
\date    1454      \providecommand*\date[1]{\gdef\@date{#1}}
\thanks  1455      \providecommand*\thanks[1]{\footnotemark
1456              \protected@xdef\@thanks{\@thanks
1457              \protect\footnotetext[\the\c@footnote]{#1}}%
1458      }%
1459      \providecommand*\and{% \vskip1em\begin{tabular}
1460              \end{tabular}%
1461              \hskip1em\@plus.17fil%
1462              \begin{tabular}[t]{c}}% \vskip1em\end{tabular} And finally, let's initialize \tit-
              % lesetup if it is not yet.
\titlesetup 1463      \providecommand*\titlesetup{}%
1464      }% end of \AtBegInput.

```

The ltxdoc class redefines the `\maketitle` command to allow multiple titles in one document. We'll do the same and something more: our `\Doc/SelfInclude` will turn the file's `\maketitle` into a part or chapter heading. But, if the `\ltxLookSetup` declaration is in force, `\Doc/SelfInclude` will make for an included file a part's title page and an article-like title.

Let's initialize the file division macros.

```

1465      \@relaxen\filediv
1466      \@relaxen\filedivname

```

If we don't include files the ltxdoc-like way, we wish to redefine `\maketitle` so that it typesets a division's heading.

Now, we redefine `\maketitle` and its relatives.

```

\InclMaketitle 1467      \def\InclMaketitle{%
1468              {\def\and{,}% we make \and just a comma.
1469              {\let\thanks=\@gobble% for the toc version of the heading we discard \thanks.
1470              \protected@xdef\incl@titletotoc{\@title\if@fshda\protect\space
1471              (\@author)\fi}% we add the author iff the 'files have different authors'
              % (@fshda)
1472              }%
\tasks 1473      \def\thanks##1{\footnotemark
1474              \protected@xdef\@thanks{\@thanks% to keep the previous \thanks if there
              were any.
1475              \protect\footnotetext[\the\c@footnote]{##1}}}% for some mysteri-
              ous reasons so defined \thanks do typeset the footnote mark and
              text but they don't hyperlink it properly. A hyperref bug?
1476              \@emptyify\@thanks
1477              \protected@xdef\incl@filedivtitle{%
1478              [{\incl@titletotoc}]% braces to allow [ and ] in the title to toc.
1479              {\protect\@title

```

```

1480      {\smallerr% this macro is provided by the gmutils package after the relsize
           package.
1481      \if@fshda\|[0.15em]\protect\@author
1482      \if\relax\@date\relax\else,\fi
1483      \else
1484      \if\relax\@date\relax\else\|[0.15em]\fi
1485      \fi

```

The default is that all the included files have the same author(s). In this case we won't print the author(s) in the headings. Otherwise we wish to print them. The information which case are we in is brought by the `\if@fshda` switch defined in line 1494.

If we wish to print the author's name (`\if@fshda`), then we'll print the date after the author, separated with a comma. If we don't print the author, there still may be a date to be printed. In such a case we break the line, too, and print the date with no comma.

```

1486      \protect\@date}}}% end of \incl@filedivtitle's brace (2nd or 3rd ar-
           gument).
1487      }% end of \incl@filedivtitle's \protected@xdef.

```

We `\protect` all the title components to avoid expanding `\footnotemark` hidden in `\thanks` during `\protected@xdef` (and to let it be executed during the typesetting, of course).

```

1488      }% end of the comma-\and's group.
1489      \@xa\filediv\incl@filedivtitle
1490      \@thanks
1491      \g@relaxen\@author_\g@relaxen\@title_\g@relaxen\@date
1492      \g@emptyify\@thanks
1493      }% end of \InclMaketitle.

```

What I make the default, is an assumption that all the multi-documented files have the same author(s). And with the account of the other possibility I provide the below switch and declaration.

```

1494 \newif\if@fshda
           (its name comes from files have different authors).
\PrintFilesAuthors 1495 \newcommand*\PrintFilesAuthors{\@fshdatrue}
           And the counterpart, if you change your mind:
1496 \newcommand*\SkipFilesAuthors{\@fshdafalse}

```

The File's Date and Version Information

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

```

\GetFileInfo 1497 \def\GetFileInfo#1{%
1498   \def\filename{#1}%
1499   \def\@tempb##1_\##2_\##3\relax##4\relax{%
1500     \def\filedate{##1}%
1501     \def\fileversion{##2}%
1502     \def\fileinfo{##3}}%
1503   \edef\@tempa{\csname_\ver@#1\endcsname}%
1504   \@xa\@tempb\@tempa\relax?_\relax\relax}

```

Since we may documentally input files that we don't load, as `doc` e.g., let's define a declaration to be put (in the comment layer) before the line(s) containing `\Provides...`

The `\FileInfo` command takes the stuff till the closing `]` and subsequent line end, extracts from it the info and writes it to the `.aux` and rescans the stuff. ϵ TeX provides a special primitive for that action but we remain strictly TeXnical and do it with writing to a file and inputting that file.

```

\FileInfo 1505 \newcommand*\FileInfo{%
1506     \bgroup
1507     \let\do\@makeother
1508     \do\_\do\{\do\}\do\^~M\do\\%
1509     \gmd@fileinfo}

1510 \bgroup
1511 \catcode'\z@
1512 \catcode'\@ne
1513 \catcode'\tw@
1514 \let\do\@makeother
1515 \do\_\do\{\do\}\do\^~M\do\\%
1516 !firstofone<!egroup%
1517 !def!gmd@fileinfo#1Provides#2{#3}#4[#5]#6
1518 <%
1519 !egroup%
1520 !gmd@writeFI<#2><#3><#5>%
1521 !gmd@docrescan<#1Provides#2{#3}#4[#5]#6
1522 >%
1523 >%
1524 >

\gmd@writeFI 1525 \def\gmd@writeFI#1#2#3{%
1526     \immediate\write\@auxout{%
1527         \global\@nx\@namedef{%
1528             ver@#2.\if_P\@firstofmany#1\@nil_Psty\else_cls\fi}{#3}}

\gmd@docrescan 1529 \def\gmd@docrescan#1{%
1530     \newwrite\gmd@docrescanfile
1531     \immediate\openout\gmd@docrescanfile=\jobname.docrescan\relax
1532     {\newlinechar='\^~M%
1533         \immediate\write\gmd@docrescanfile{%
1534             \twelvepercent#1\twelvepercent\@nx\NoEOF}%
1535     }%
1536     \immediate\closeout\gmd@docrescanfile
1537     \@@input\jobname.docrescan
1538 }

```

And, for the case the input file doesn't contain `\Provides...`, a macro for explicit providing the file info. It's written in analogy to `\ProvidesFile`, source 2 ϵ , file L v1.1g, l. 102.

```

\ProvideFileInfo 1539 \def\ProvideFileInfo#1{%
1540     \begingroup
1541     \catcode'\_10_\catcode\endlinechar_10_%
1542     \@makeother\/\@makeother\&%
1543     \kernel@ifnextchar[{\gmd@providefii{#1}}{\gmd@providefii{#1}[]}%
1544     }

1545 \def\gmd@providefii#1[#2]{%
    (we don't write the file info to .log)

```

```

1546 \xa\xdef\csname_ver@#1\endcsname{#2}%
1547 \endgroup}

```

And a self-reference abbreviation (intended for providing file info for the driver):

```

\ProvideSelfInfo 1548 \def\ProvideSelfInfo{\ProvideFileInfo{\jobname.tex}}

```

A neat conventional statement used in doc's documentation e.g., to be put in \thanks to the title or in a footnote:

```

\filenote 1549 \newcommand*\filenote{This file has version number \fileversion{}
dated \filedate{}}.}

```

And exactly as \thanks:

```

\thfileinfo 1550 \newcommand*\thfileinfo{\thanks\filenote}

```

Miscellanea

The main inputting macro, \DocInput has been provided. But there's another one in doc and it looks very reasonably: \IndexInput. Let's make analogous one here:

```

1551 \bgroup\obeylines
1552 \firstofone{\egroup
\IndexInput 1553 \def\IndexInput#1{%
1554 \StoreMacro\code@delim%
1555 \CodeDelim*\^^Z%
1556 \def\gmd@iihook{% this hook is \edefed!
1557 \@nx^^M%
1558 \code@delim\relax\@nx\let\@nx\EOFMark\relax}%
1559 \DocInput{#1}\RestoreMacro\code@delim}}

```

How does it work? We assume in the input file is no explicit *<char1>*. This char is chosen as the code delimiter and will be put at the end of input. So, entire file contents will be scanned char by char as the code.

The below environment I designed to be able to skip some repeating texts while documenting several packages of mine into one document. At the default settings it's just a \StraightEOL group and in the \skipgmlonely declaration's scope it gobbles its contents.

```

gmlonely 1560 \newenvironment{gmlonely}{\StraightEOL}{}
\skipgmlonely 1561 \newcommand\skipgmlonely[1][1]{%
1562 \def\@tempa{%
1563 \def\gmd@skipgmltext{\g@empty\gmd@skipgmltext#1}}%
1564 \@tempa
1565 \xa\AtBegInput\xa{\@tempa}%
1566 \renewenvironment{gmlonely}{%
1567 \StraightEOL
1568 \@fileswfalse% to forbid writing to .toc, .idx etc.
1569 \setbox0=\vbox\bgroup}{\egroup\gmd@skipgmltext}}

```

Sometimes in the commentary of this package, so maybe also others, I need to say some char is of category 12 ('other sign'). This I'll mark just as ₁₂ got by \catother.

```

1570 \bgroup\catcode'\_ =8_ we ensure the standard \catcode of _
1571 \firstofone{\egroup
\catother 1572 \newcommand*\catother{$\}_{12}$}%

```

Similarly, if we need to say some char is of category 13 (‘active’), we’ll write `_13`, got by `\catactive`

```
\catactive 1573 \newcommand*\catactive{$\_13$}%
              and a letter, _11
```

```
\catletter 1574 \newcommand*\catletter{$\_11$}% .
1575 }
```

For the copyright note first I used just `verse` but it requires marking the line ends with `\\` and indents its contents while I prefer the copyright note to be flushed left. So

```
copyrnote 1576 \newenvironment*{copyrnote}{%
1577 \StraightEOL\everypar{\hangindent3em\relax\hangafter1\_}%
1578 \par\addvspace\medskipamount\parindent\z@\obeylines}{%
1579 \@codeskipputgfalse\stanza}
```

I renew the quotation environment to make the fact of quoting visible

```
quotation 1580 \renewenvironment{quotation}{\par‘\ignorespaces}{\unskip’\par}
```

For some mysterious reasons `\noindent` doesn’t work with the first (narrative) paragraph after the code so let’s work it around:

```
\gmdnoindent 1581 \newcommand*\gmdnoindent{\leavevmode\hskip-\parindent}
```

When a verbatim text occurs in an inline comment, it’s advisable to precede it with `%` if it begins a not first line of such a comment not to mistake it for a part of code. For this purpose provide

```
\nlpercent 1582 \newcommand*\nlpercent{%
1583 \@ifstar{\def\@tempa{\tt\twelvepercent}}{%
1584 \emptify\@tempb\nlpercent}%
1585 {\emptify\@tempa
1586 \def\@tempb{\gmboxedspace}\nlpercent}}
1587 \newcommand*\gmboxedspace{\hbox{\normalfont{_\_}}}
1588 \newcommand*\nlpercent[1][\_]{%
1589 \unskip
1590 \discretionary{\@tempa}{\tt\twelvepercent\gmboxedspace}{\@tempb}%
1591 \penalty10000\hskip0sp\relax}
```

As you see, `\nlpercent` inserts a discretionary that breaks to `%` at the beginning of the lower line. Without the break it’s a space (alas at its natural width i.e., not flexible) or, with the starred version, nothing. The starred version puts `%` also at the end of the upper line.

TODO: make the space flexible (most probably it requires using sth. else than `\discretionary`).

An optional hyphen for CSs in the inline comment:

```
1592 \@ifundefined{+}{\typeout{^^Jgmdoc.sty:_\_redefining_\_bslash+.\_}}
\+ 1593 \def\+{\discre{\normalfont-}}{\tt\twelvepercent\gmboxedspace}{\_}
```

And finally, some logos:

“Here are a few definitions which can usefully be employed when documenting package files: now we can readily refer to $\mathcal{M}\mathcal{S}$ -TEX, BibTEX and SUTEX, as well as the usual TEX and LATEX. There’s even a PLAIN TEX and a WEB.”

```
1594 \@ifundefined{AmSTeX}
```

```
\AmSTeX 1595 {\def\AmSTeX{\leavevmode\hbox{$\mathcal_A\kern-.2em\lower.376ex%
1596 \hbox{$\mathcal_M$}\kern-.2em\mathcal_S$-\TeX}}}\}
```

```
\BibTeX 1597 \DeclareLogo\BibTeX{\rmfamily_B\kern-.05em%
1598 \textsc{i\kern-.025em_b}\kern-.08em%
1599 \TeX}}
```

```
\SliTeX 1600 \DeclareLogo\SliTeX{\rmfamily_S\kern-.06emL\kern-.18em\raise.32ex^
\hbox
1601 {\scshape_i}\kern-.03em\TeX}}
```

```
\PlainTeX 1602 \DeclareLogo\PlainTeX{\textsc{Plain}\kern2pt\TeX}
```

```
\Web 1603 \DeclareLogo\Web{\textsc{Web}}
```

There's also the (L^A)T_EX logo got with the `\LaTeXpar` macro provided by `gmutils`. And here The T_EXbook's logo:

```
\TeXbook 1604 \DeclareLogo[The_TeX_book]\TeXbook{The_TeX_book}
```

```
\eTeX 1605 \DeclareLogo\eTeX{$\varepsilon\TeX}$% The  $\varepsilon$ -TEX manual uses ' $\varepsilon$ -TEX' but that
would look strange in pdf $\varepsilon$ TEX.
```

```
\pdfeTeX 1606 \DeclareLogo\pdfeTeX{pdf\eTeX}
```

```
\pdfTeX 1607 \DeclareLogo\pdfTeX{pdf\TeX}
```

```
\XeTeX 1608 \DeclareLogo\XeTeX{X\kern-.125em\relax
1609 \@ifundefined{reflectbox}{%
1610 \lower.5ex\hbox{E}\kern-.1667em\relax}{%
1611 \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
1612 \TeX}% As you see, if TEX doesn't recognize \reflectbox (graphics isn't loaded),
the first E will not be reversed. This version of the command is intended
for non-XETEX usage. With XETEX, the reversed E you get as the Unicode
Latin Letter Reversed E.
```

```
\ds 1613 \@ifundefined{ds}{\def\ds{DocStrip}}{\}
```

Finally, a couple of macros for documenting files playing with %'s catcode(s). Instead of % I used &. They may be at the end because they're used in the commented thread i.e. after package's `\usepackage`.

```
\CDAnd 1614 \newcommand*\CDAnd{\CodeDelim\&}
```

```
\CDPerc 1615 \newcommand*\CDPerc{\CodeDelim\%}
```

And for documenting in general:

A general sectioning command because I foresee a possibility of typesetting the same file once as independent document and another time as a part of bigger whole.

```
1616 \@ifdefinable\division{}% just to test (this LATEX check issues an error if the first
argument is already defined).
```

```
1617 \@ifundefined{section}{%
```

```
\division 1618 \@relaxen\division}{%
```

```
\division 1619 \let\division=\section}
```

```
1620 \@ifdefinable\subdivision{}% just to test (see above).
```

```
1621 \@ifundefined{subsection}{%
```

```
\subdivision 1622 \@relaxen\subdivision}{%
```

```
\subdivision 1623 \let\subdivision=\subsection}
```

The `\lets` are inside `\@ifundefineds` because I'm not sure whether you will typeset a documentation with usual classes. Maybe too much care.

To kill a tiny little bug in doc.dtx (in line 3299 \@tempb and \@tempc are written plain not verbatim):

```

1624 \newcounter{gmd@mc}
1625 \def\gmd@mchook{\stepcounter{gmd@mc}%
1626   \gmd@mcdiag
1627   \csname\gmd@mchook\the\c@gmd@mc\endcsname}
\AfterMacrocode 1628 \long\def\AfterMacrocode#1#2{\@namedef{gmd@mchook#1}{#2}}

```

What have I done? I declare a new counter and employ it to count the `macrocode(*)`s (and `oldmc(*)`s too, in fact) and attach a hook to (after) the end of every such environment. That lets us to put some stuff pretty far inside the compiled file (for the buggie in doc.dtx, to redefine \@tempb/c).

One more detail to explain and define: the `\gmd@mcdiag` macro may be defined to type out a diagnostic message (the `macrocode(*)`'s number, code line number and input line number).

```

\gmd@mcdiag 1629 \@emptyify\gmd@mcdiag
\mcdiagOn 1630 \def\mcdiagOn{\def\gmd@mcdiag{%
1631   \typeout{^^J\bslash\end{macrocode(*)}\_No.\the\c@gmd@mc
1632   \space\on@line,\_c\ln.\the\c@codelinenum.}}}
\mcdiagOff 1633 \def\mcdiagOff{\@emptyify\gmd@mcdiag}

```

doc-Compatibility

My T_EX Guru recommended me to write hyperlinking for doc. The suggestion came out when writing of gmdoc was at such a stage that I thought it to be much easier to write a couple of \lets to make gmdoc able to typeset sources written for doc than to write a new package that adds hyperlinking to doc. So...

The doc package makes % an ignored char. Here the % delimits the code and therefore has to be 'other'. But only the first one after the code. The others we may re\catcode to be ignored and we do it indeed in line 108.

At the very beginning of a doc-prepared file we meet a nice command \CharacterTable. My T_EX Guru says it's a bit old fashioned these days so let's just make it notify the user:

```

\CharacterTable 1634 \def\CharacterTable{\begingroup
1635   \@makeother\{\@makeother\}%
1636   \Character@Table}
1637 \begingroup
1638 \catcode'\<=1\_catcode'\>=2\_
1639 \@makeother\{\@makeother\}%
1640 \firstofone<\endgroup
1641   \def\Character@Table#1{#2}<\endgroup
1642   \message<^^J^^J\_gmdoc.sty\_package:^^J
1643   ====\_The\_input\_file\_contains\_the\_\_bslash\_CharacterTable.^^J
1644   ====\_If\_you\_really\_need\_to\_check\_the\_correctness\_of\_the\_chars,^^J
1645   ====\_please\_notify\_the\_author\_of\_gmdoc.sty\_at\_the\_email\_
        address^^J

```

```
1646      ===_given_in_the_legal_notice_in_gmdoc.sty.^~J^~J>>>
```

Similarly as `doc`, `gmdoc` provides `macrocode`, `macro` and `environment` environments. Unlike in `doc`, `\end{macrocode}` *does not* require to be preceded with any particular number of spaces. Unlike in `doc`, it *is not* a kind of `verbatim`, however, which means the code and narration layers remains in force inside it which means that any text after the first % in a line will be processed as narration (and its control sequences will be executed). For a discussion of a possible workaround see line 1724.

Let us now look over other original `doc`'s control sequences and let's 'domesticate' them if they are not yet.

kici kici
`\DescribeMacro` The `\DescribeMacro` and `\DescribeEnv` commands seem to correspond with my
`\DescribeEnv` `\TextUsage` macro in its plain and starred version respectively except they don't typeset their arguments in the text i.e., they do two things of the three. So let's `\def` them to do these two things in this package, too:

```
\DescribeMacro 1647 \outer\def\DescribeMacro{%
1648     \begingroup\MakePrivateLetters
1649     \gmd@ifonetoken\Describe@Macro\Describe@Env}
```

Note that if the argument to `\DescribeMacro` is not a (possibly starred) control sequence, then as an environment's name shall it be processed *except* the `\MakePrivateOthers` re`\catcode`ing shall not be done to it.

```
\DescribeEnv 1650 \outer\def\DescribeEnv{%
1651     \begingroup\MakePrivateOthers\Describe@Env}
```

Actually, I've used the `\Describe...` commands myself a few times, so let's `\def` a common command with a starred version:

```
\Describe 1652 \outer\def\Describe{%
1653     \begingroup\MakePrivateLetters
1654     \@ifstarl{\MakePrivateOthers\Describe@Env}{\Describe@Macro}}
```

The below two definitions are adjusted ~s of `\Text@UsgMacro` and `\Text@UsgEnvir`.

```
1655 \long\def\Describe@Macro#1{%
1656     \endgroup
1657     \strut\Text@Marginize#1%
1658     \@usgentryze#1% we declare kind of formatting the entry
1659     \text@indexmacro#1\ignorespaces}

1660 \def\Describe@Env#1{%
1661     \endgroup
1662     \strut\Text@Marginize{#1}%
1663     \@usgentryze{#1}% we declare the 'usage' kind of formatting the entry and index
        the sequence #1.
1664     \text@indexenvir{#1}\ignorespaces}
```

Note that here the environments' names are typeset in `\tt` font just like the macros', *unlike* in `doc`.

My understanding of 'minimality' includes avoiding too much freedom as causing chaos not beauty. That's the philosophical and aesthetic reason why I don't provide
`\MacroFont` `\MacroFont`. In my opinion there's a noble tradition of typesetting the `TEX` code in `\tt` font nad this tradition sustained should be. If one wants to change the tradition, let her redefine `\tt`, in `TEX` it's no problem. I suppose `\MacroFont` is not used explicitly, and that it's (re)defined at most, but just in case let's `\let`:

```

1665 \let\MacroFont\tt

\CodeIndent      We have provided \CodeIndent in line 52. And it corresponds with doc's \MacroIn-
\MacroIndent      dent so

\MacroIndent 1666 \let\MacroIndent\CodeIndent

And similarly the other skips:

\MacrocodeTopsep 1667 \let\MacrocodeTopsep\CodeTopsep

\MacroTopsep      Note that \MacroTopsep is defined in gmdoc and has the same rôle as in doc.

1668 \let\SpecialEscapechar\CodeEscapeChar

\theCodelineNo    \theCodelineNo is not used in gmdoc. Instead of it there is \LineNumFont decla-
\LineNumFont      ration and a possibility to redefine \thecodelinenumber as for all the counters. Here the
\LineNumFont      \LineNumFont is used two different ways, to set the benchmark width for a linenumber
among others, so it's not appropriate to put two things into one macro. Thus let's give
the user a notice if he defined this macro:

Because of possible localness of the definitions it seems to be better to add a check
at the end of each \DocInput or \IndexInput.

1669 \AtEndInput{\@ifundefined{theCodelineNo}{\PackageInfo{gmdoc}{The
1670      \string\theCodelineNo\space\macro\has\no\effect\here,\please\use
1671      \string\LineNumFont\space\for\setting\the\font\and/or
1672      \string\thecodelinenumber\space\to\set\the\number\format.}}}

I hope this lack will not cause big trouble.

For further notifications let's define a shorthand:

\noeffect@info 1673 \def\noeffect@info#1{\@ifundefined{#1}{\PackageInfo{gmdoc}{^^J%
1674      The\backslash#1\macro\is\not\supported\by\this\package^^J
1675      and\therefore\has\no\effect\but\this\notification.^^J
1676      If\you\think\it\should\have,\please\contact\the\maintainer^^J
1677      indicated\in\the\package's\legal\note.^^J}}}

The four macros formatting the macro and environment names, namely

\PrintDescribeMacro \PrintDescribeMacro, \PrintMacroName, \PrintDescribeEnv and \PrintEnvName are
\PrintMacroName      not supported by gmdoc. They seem to me to be too internal to take care of them. Note
\PrintDescribeEnv     that in the name of (aesthetical) minimality and (my) convenience I deprive you of easy
\PrintEnvName          knobs to set strange formats for verbatim bits: I think they are not advisable.

Let us just notify the user.

1678 \AtEndInput{%
1679     \noeffect@info{PrintDescribeMacro}%
1680     \noeffect@info{PrintMacroName}%
1681     \noeffect@info{PrintDescribeEnv}%
1682     \noeffect@info{PrintEnvName}%

\CodeLineNumbered The \CodeLineNumbered declaration of doc seems to be equivalent to our noindex
option with the linesnotnum option set off so let's define it such a way.

1683 \def\CodeLineNumbered{\AtBeginDocument{\gag@index}}
1684 \@onlypreamble\CodeLineNumbered

Note that if the linesnotnum option is in force, this declaration shall not revert its
effect.

I assume that if one wishes to use doc's interface then she'll not use gmdoc's options
but just the default.

```

The `\CodelineIndex` and `\PageIndex` declarations correspond with the `gmdoc`'s default and the `pageindex` option respectively. Therefore let's `\let`

```
\CodelineIndex 1685 \let\CodelineIndex\@pageindexfalse
1686 \@onlypreamble\CodelineIndex
```

```
\PageIndex 1687 \let\PageIndex\@pageindextrue
1688 \@onlypreamble\PageIndex
```

The next two declarations I find useful and smart:

```
\DisableCrossrefs 1689 \def\DisableCrossrefs{\@bsphack\gag@index\@esphack}
\EnableCrossrefs 1690 \def\EnableCrossrefs{\@bsphack\ungag@index
1691 \def\DisableCrossrefs{\@bsphack\@esphack}\@esphack}
```

The latter definition is made due to the footnote 6 on p.8 of the Frank Mittelbach's `doc`'s documentation and both of them are copies of lines 302–304 of it modulo `\(un)gag@index`.

The subsequent few lines I copy almost verbatim ;-) from the lines 611–620.

```
\AlsoImplementation 1692 \newcommand*\AlsoImplementation{\@bsphack%
1693 \long\def\StopEventually##1{\gdef\Finale{##1}}% we define \Finale just to
expand to the argument of \StopEventually not to add anything to the
end input hook because \Finale should only be executed if entire document
is typeset.
```

`\@init@checksum` is obsolete in `gmdoc` at this point: the `Checksum` counter is reset just at the beginning of (each of virtually numerous) input(s).

```
1694 \@esphack}
1695 \AlsoImplementation
```

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

```
\OnlyDescription 1696 \def\OnlyDescription{\@bsphack\long\def\StopEventually##1{%
```

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

```
1697 ##1\endinput}\@esphack}
```

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

```
1698 \@relaxen\Finale
```

`\meta` The `\meta` macro is so beautifully crafted in `doc` that I couldn't resist copying it into `gmutils`. It's also available in Knuthian (The `TEXbook` format's) disguise `\<the argument>`.

The checksum mechanism is provided and developed for a slightly different purpose.

Most of `doc`'s indexing commands have already been ‘almost defined’ in `gmdoc`:

```
\SpecialMainIndex 1699 \let\SpecialMainIndex=\CodeDefIndex
\SpecialMainEnvIndex 1700 \def\SpecialMainEnvIndex{\curname_\CodeDefIndex\endcurname*}% we don't type
\CodeDefIndex explicitly here 'cause it's \outer, remember?
\SpecialIndex 1701 \let\SpecialIndex=\CodeCommonIndex
\SpecialUsageIndex 1702 \let\SpecialUsageIndex=\TextUsgIndex
```

```

\SpecialEnvIndex 1703 \def\SpecialEnvIndex{\csname\_TextUsgIndex\endcsname*}
\SortIndex 1704 \def\SortIndex#1#2{\index{#1\actualchar#2}}

    “All these macros are usually used by other macros; you will need them only in an
    emergency.”
    Therefore I made the assumption(s) that ‘Main’ indexing macros are used in my
    ‘Code’ context and the ‘Usage’ ones in my ‘Text’ context.

\verbatimchar    Frank Mittelbach in doc provides the \verbatimchar macro to (re)define the
\verb(*)’s delimiter for the index entries. The gmdoc package uses the same macro and
its default definition is {&}. When you use doc you may have to redefine \verbatimchar
if you use (and index) the \+ control sequence. gmdoc does a check for the analogous
situation (i.e., for processing \&) and if it occurs it takes $ as the \verb*’s delimiter. So
strange delimiters are chosen deliberately to allow any ‘other’ chars in the environments’
names. If this would cause problems, please notify me and we’ll think of adjustments.

\verbatimchar 1705 \def\verbatimchar{&}

    One more a very neat macro provided by doc. I copy it verbatim and put into gmutils,
    too. (\DeclareRobustCommand doesn’t issue an error if its argument has been defined,
    it only informs about redefining.)

\* 1706 \DeclareRobustCommand*{\leavevmode\lower.8ex\hbox{$\,\widetilde{\_}\^
\,\,$}}

\IndexPrologue    \IndexPrologue is defined in line 1003. And other doc index commands too.

\main 1707 \@ifundefined{main}{}{\let\DefEntry=\main}
\usage 1708 \@ifundefined{usage}{}{\let\UsgEntry=\usage}

    About how the DocStrip directives are supported by gmdoc, see section The Doc-
    Strip.... This support is not that sophisticated as in doc, among others, it doesn’t count
    the modules’ nesting. Therefore if we dont want an error while gmdocumenting doc-
    prepared files, better let’s define doc’s counter for the modules’ depths.

StandardModuleDepth 1709 \newcounter{StandardModuleDepth}

    For now let’s just mark the macro for further development

\DocstyleParms 1710 \noeffect@info{DocstyleParms}

    For possible further development or to notify the user once and forever:

\DontCheckModules 1711 \@emptyify\DontCheckModules\_noeffect@info{DontCheckModules}
\CheckModules 1712 \@emptyify\CheckModules\_noeffect@info{CheckModules}

\Module    The \Module macro is provided exactly as in doc.

\AltMacroFont 1713 \@emptyify\AltMacroFont\_noeffect@info{AltMacroFont}

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

\MakePercentIgnore 1714 \def\MakePercentIgnore{\catcode'\%9\relax}
\MakePercentComment 1715 \def\MakePercentComment{\catcode'\%14\relax}

```

gmdocing doc.dtx

The author(s) of `doc` suggest(s):

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

Therefore I hope that after `doc.dtx` has been gmdoc-ed, one can say gmdoc is doc-compatible “at most—if not at all”.

T_EXing the original `doc` with my humble¹² package was a challenge and a milestone experience in my T_EX life.

One of minor errors was caused by my understanding of a ‘shortverb’ char: due to `gmverb`, in the math mode an active ‘shortverb’ char expands to itself’s ‘other’ version thanks to `\string` (It’s done with | in mind). `doc`’s concept is different, there a ‘shortverb’ char should in the math mode work as shortverb. So let it be as they wish: `gmverb` provides `\OldMakeShortVerb` and the oldstyle input commands change the inner macros so that also `\MakeShortVerb` works as in `doc` (cf. line 1718).

We also redefine the `macro` environment to make it mark the first code line as the point of defining of its argument, because `doc.dtx` uses this environment also for implicit definitions.

```
\OldDocInput 1716 \def\OldDocInput{%
1717   \AtBegInputOnce{\StraightEOL
1718     \let\@MakeShortVerb=\old@MakeShortVerb
1719     \let\gmd@@macro\macro
1720     \def\macro{\let\gmd@ifonetoken\@secondoftwo\gmd@@macro}% (Of course,
naïve \exp...\let\exp...\macro\cs...\macro*\endcs... caused an
infinite loop since in the definition of macro* the \macro macro occurs.)
\VerbMacrocodes 1721   \VerbMacrocodes}%
1722   \bgroup\@makeother\__% it's to allow _ in the filenames. The next macro will close
the group.
1723   \Doc@Input}
```

We don’t switch the `@codeskipput` switch neither we check it because in ‘old’ world there’s nothing to switch this switch in the narration layer.

I had a hot and wild T_EX all the night nad what a bliss when the ‘Succesfully formatted 67 page(s)’ message appeared.

My package needed fixing some bugs and adding some compatibility adjustments (listed in the previous section) and the original `doc.dtx` source file needed a few adjustments too because some crucial differences came out. I’d like to write a word about them now.

The first but not least is that the author(s) of `doc` give the CS marking commands non-macro arguments sometimes, e.g., `\DescribeMacro{StandardModuleDepth}`. Therefore we should launch the *starred* versions of corresponding gmdoc commands. This means the `doc`-like commands will not look for the CS’s occurrence in the code but will mark the first codeline met.

Another crucial difference is that in gmdoc the narrative and the code layers are separated with only the code delimiter and therefore may be much more mixed than in `doc`. among others, the `macro` environment is *not* a typical *verbatim* like: the texts commented out within `macrocode` are considered a normal commentary i.e., not verbatim. Therefore some macros ‘commented out’ to be shown verbatim as an example source must

¹² What a *false* modesty! ;-)

have been ‘additionally’ verbatimized for `gmdoc` with the shortverb chars e.g. You may also change the code delimiter for a while, e.g., the line

```
1724 %\AVerySpecialMacro%\delete the first%\when...
```

was got with

```
\CodeDelim*\.
```

```
% \AVerySpecialMacro % delete the first % when.\unskip|..\CDPerc
```

One more difference is that my shortverb chars expand to their ₁₂ versions in the math mode while in doc remain shortverb, so I added a declaration `\OldMakeShortVerb` etc.

Moreover, it’s \TeX ing doc what inspired adding the `\StraightEOL` and `\QueerEOL` declarations.

Polishing, Development and Bugs

- `\MakePrivateLetters` theoretically may interfere with `\active` eating some chars to allow linebreaks. But making a space or an opening brace a letter seems so perverse that we may feel safe not to take account of such a possibility.

- When `countalllines` option is enabled, the comment lines that don’t produce any printed output result with a (blank) line too because there’s put a hypertext at the beginning of them. But for now let’s assume this option is for draft versions so hasn’t be perfect.

- Marcin Woliński suggests to add the `marginpar` clauses for the AMS classes as we did for the standard ones in the lines 18–20. Most probably I can do it on request when I only know the classes’ names and their ‘marginpar status’.

- When the `countalllines` option is in force, some `\list` environments shall raise the ‘missing `\item`’ error if you don’t put the first `\item` in the same line as `\begin{environment}` because the (comment-) line number is printed.

- I’m prone to make the control sequences hyperlinks to the(ir) ‘definition’ occurrences. It doesn’t seem to be a big work compared with what has been done so far.

- Is `\RecordChanges` really necessary these days? Shouldn’t be the `\makeglossary` command rather executed by default?¹³

- Do you use `\listoftables` and/or `\listoffigures` in your documentations? If so, I should ‘EOL-straighten’ them like `\tableofcontents`, I suppose (cf. line 143).

- Some lines of non-printing stuff such as `\CodeDefine...` and `\changes` connecting the narration with the code resulted with unexpected large vertical space. Adding a fully blank line between the printed narration text and not printed stuff helped.

- My \TeX Guru remarked that `\jobname` expands to the main file name without extension iff that extension is `.tex`. Should I replace `\jobname` with `\jobnamewoe` then? (The latter always expands to the file name without extension.)

- About the DocStrip [verbatim mode directive](#) see above.

(No) `\eof`

If the user doesn’t wish the documentation to be ended by `\eof`, he should end the file with a comment ending with `\NoEOF` macro defined below¹⁴:

¹³ It’s understandable that ten years earlier writing things out to the files remarkably decelerated \TeX , but nowadays it does not in most cases. That’s why `\makeindex` is launched by default in `gmdoc`.

¹⁴ Thanks to Bernd Raichle at Bacho \TeX 2006 Pearl Session where he presented `\inputing` a file inside `\edef`.

```

1725 \bgroup\catcode'\^^M\active_\firstofone{\egroup%
1726 \def\@NoEOF#1^^M{%
1727 \@relaxen\EOFMark\@xa\@nx\endinput^^M}}
\NoEOF 1728 \def\NoEOF{\QueerEOL\@NoEOF}

As you probably see, \NoEOF has also the \endinput effect.

1729 \endinput
1730 \endpackage

```


b. The gmdocc Class For gmdoc Driver Files¹

Written by Natror (Grzegorz Murzynowski),
natror at o2 dot pl

©2006 by Natror (Grzegorz Murzynowski).

This program is subject to the L^AT_EX Project Public License.

See <http://www.ctan.org/tex-archive/help/Catalogue/licenses.lppl.html> for the details of that license.

LPPL status: "author-maintained".

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesClass{gmdocc}
3 [2007/03/02_v0.75_a_class_for_gmdoc_driver_files_(GM)]
```

Intro

This file is a part of **gmdoc** bundle and provides a document class for the driver files documenting (L^A)T_EX packages &a. with my **gmdoc.sty** package. It's not necessary, of course: most probably you may use another document class you like.

By default this class loads **mwart** class with **a4paper** (default) option and **lmodern** package with **QX** fontencoding. It loads also my **gmdoc** documenting package which loads some auxiliary packages of mine and the standard ones.

If the **mwart** class is not found, the standard **article** class is loaded instead. Similarly, if the **lmodern** is not found, the standard Computer Modern font family is used in the default font encoding.

Usage

For the ideas and details of **gmdoc**ing of the (L^A)T_EX files see the **gmdoc.sty** file's documentation (chapter [a](#)). The rôle of the **gmdocc** document class is rather auxiliary and exemplary. Most probably, you may use your favourite document class with the settings you wish. This class I wrote to meet my needs of fine formatting, such as not numbered sections and sans serif demi bold headings.

However, with the users other than myself in mind, I added some conditional clauses that make this class works also if an **mwcls** class or the **lmodern** package are unknown.

Of rather many options supported by **gmdoc.sty**, this class chooses my favourite, i.e., the default. An exception is made for the **noindex** option, which is provided by this class and passed to **gmdoc.sty**. This is intended for the case you don't want to make an index.

Simili modo, the **nochanges** option is provided to turn creating the change history off.

¹ This file has version number v0.75 dated 2007/03/02.

Both of the above options turn the *writing out to the files* off. They don't turn off `\PrintIndex` nor `\PrintChanges`. (Those two commands are no-ops by themselves if there's no `.ind` (n) or `.gls` file respectively.)

`outeroff` One more option is `outeroff`. It's intended for compiling the documentation of macros defined with the `\outer` prefix. It `\relaxes` this prefix so the '`\outer`' macros' names can appear in the arguments of other macros, which is necessary to pretty mark and index them.

I decided not to make discarding `\outer` the default because it seems that L^AT_EX writers don't use it in general and `gmdoc.sty` *does* make some use of it.

`debug` This class provides also the `debug` option. It turns the `\if@debug` Boolean switch True and loads the `trace` package that was a great help to me while debugging `gmdoc.sty`.

The default base document class loaded by `gmdocc.cls` is Marcin Woliński's `mwart`. If you have not installed it on your computer, the standard `article` will be used.

Moreover, if you like MW's classes (as I do) and need `\chapter` (for multiple files' input e.g.), you may declare another `mwcls` with the option homonimic with the class's name: `mwrep` for `mwrep` and `mwbk` for `mwbk`. For the symmetry there's also `mwart` option (equivalent to the default setting).

`mwrep` The existence test is done for any MW class option as it is in the default case.
`mwbk`
`mwart`

`\EOFMark` The `\EOFMark` is in this class typesets like this (of course, you can redefine it as you wish):

□

The Code

```
4 \def\gmcc@baseclass{mwart}% the default is Marcin Woliński's class (mwcls) anal-
   ogous to article.
```

Since you can choose the standard `article` class, we'd better provide a Boolean switch to keep the score of what was chosen. It's to avoid unused options if `article` is chosen.

```
5 \newif\ifgmcc@mwcls
6 \gmcc@mwclstrue
mwart 7 \DeclareOption{mwart}{\def\gmcc@baseclass{mwart}}% The mwart class may
   also be declared explicitly.
mwrep 8 \DeclareOption{mwrep}{\def\gmcc@baseclass{mwrep}}% If you need chapters,
   this option chooses an MW's class that corresponds to report,
mwbk 9 \DeclareOption{mwbk}{\def\gmcc@baseclass{mwbk}}% and this MW's class corre-
   sponds to book.
article 10 \DeclareOption{article}{\gmcc@mwclsfalse}% you can also choose article.
outeroff 11 \DeclareOption{outeroff}{\let\outer\relax}% This option allows \outer-prefixed
   macros to be gmdoc-processed with all the bells and whistles.

\if@debug 12 \newif\if@debug
debug 13 \DeclareOption{debug}{\@debugtrue}% This option causes trace to be loaded and
   the Boolean switch of this option may be used to hide some things needed only
   while debugging.

noindex 14 \DeclareOption{noindex}{%
15 \PassOptionsToPackage{\CurrentOption}{gmdoc}}% This option turns the writ-
   ing outto .idx file off.
```

```

16 \newif\if@gmccnochanges
nochanges 17 \DeclareOption{nochanges}{\@gmccnochangestrue}% This option turns the writ-
ing outto .glo file off.
gmeometric 18 \def\gmTheGeometry{geometry}
19 \DeclareOption{gmeometric}{%
20 \IfFileExists{gmeometric.sty}{%
21 \def\gmTheGeometry{gmeometric}}{}}% It's an experimental option that causes
the \geometry macro provided by geometry package is not restricted to
the preamble. This option causes the gmeometric package is loaded (if
available) that works the limitation around.
22 \ProcessOptions
23 \ifgmcc@mwcls
24 \IfFileExists{\gmcc@baseclass.cls}{\gmcc@mwclsfalse}% As announced,
we do the ontological test to any mwcls.
25 \fi
26 \ifgmcc@mwcls
27 \LoadClass[fleqn,oneside,noindentfirst,11pt,withmarginpar,
28 sfheadings]{\gmcc@baseclass}%
29 \else
30 \LoadClass[fleqn,11pt]{article}% Otherwise the standard article is loaded.
31 \fi
32 \AtBeginDocument{\mathindent=\CodeIndent}

The fleqn option makes displayed formuals be flushed left and \mathindent is their
indentation. Therefore we ensure it is always equal \CodeIndent just like \leftskip in
verbatim. Thanks to that and the \edverbs declaration below you may display single
verbatim lines with \[...]:

\[\verb|\stuff|\].

33 \IfFileExists{lmodern.sty}{% We also examine the ontological status of this pack-
age
34 \RequirePackage{lmodern}% and if it shows to be satisfactory (the package shows
to be), we load it and set the proper font encoding.
35 \RequirePackage[QX]{fontenc}%
36 }{}
37 \RequirePackage[margin=2.7cm,left=4cm,
38 right=2.2cm]{\gmTheGeometry}% Now we set the page layout.
39 \def\gmdoccMargins{%
40 \geometry{margin=2.7cm,left=4cm,right=2.2cm}}
41 \if@debug% For debugging we load also the trace package that was very helpful to
me.
42 \RequirePackage{trace}%
43 \errorcontextlines=100% And we set an error info parameter.
44 \fi

\ifdtraceon 45 \newcommand*\ifdtraceon{\if@debug\afterfi\traceon\fi}
\ifdtraceoff 46 \newcommand*\ifdtraceoff{\if@debug\traceoff\fi}

We load the core package:

47 \RequirePackage{gmdoc}

```

```

48 \@ifpackageloaded{lmodern}{% The Latin Modern font family provides a light con-
    densed typewriter font that seems to be the most suitable for the marginpar
    CS marking.
49 \def\marginpartt{\normalfont\fontseries{lc}\ttfamily}}%
50 \raggedbottom
51 \setcounter{secnumdepth}{0}% We wish only the parts and chapters to be num-
    bered.
52 \renewcommand*\thesection{\arabic{section}}% isn't it redundant at the above
    setting?
53 \@ifnotmw{}{%
54 \ifclassloaded{mwart}{% We set the indentation of Contents:
55 \SetTOCIndents{{}\quad\quad\quad\quad\quad\quad}{% for
    mwart
56 \SetTOCIndents{{}\bf9.\enspace}\quad\quad\quad\quad\quad\quad}{%
    and for the two other mwcls.
57 \pagestyle{outer}}% We set the page numbers to be printed in the outer and
    bottom corner of the page.
58 \def\titlesetup{\bfseries\sffamily}% We set the title(s) to be boldface and sans
    serif.
59 \if@gmccnochanges\let\RecordChanges\relax\fi% If the nochanges option is on,
    we discard writing outto the .glo file.
60 \RecordChanges% We turn the writing the \changes outto the .glo file if not the
    above.
61 \dekclubs% We declare the club sign | to be a shorthand for \verb*.
62 \edverbs% to redefine \[ so that it puts a shortverb in a \hbox.
63 \smartunder% and we declare the _ char to behave as usual in the math mode and
    outside math to be just an underscore.
64 \exhyphenpenalty\hyphenpenalty% 'cause mwcls set it =10000 due to Polish cus-
    toms.
65 \VerbT1% to ensure T1 font encoding in verbatims (in code).
66 \RequirePackage{amssymb}
67 \def\EOFMark{\rightline{\ensuremath{\square}}}
68 \endinput

```

c. gmdocDoc.tex, The Driver File

```
1 \documentclass[outeroff,mwrep]{gmdocc}
2 \twocoltoc
3 \title{The\pk{gmdoc}\Package\i.e.,\pk{gmdoc.sty}and
4   \pk{gmdocc.cls}}
5 \author{Grzegorz'Natror'Murzynowski}
6 \date{EuroBacho\TeX\2007}
7 \begin{document}
8 \maketitle
9 \setcounter{page}{2}% hyperref cries if it sees two pages numbered 1.
10 \tableofcontents
11 \DoIndex\maketitle
12 \DocInclude{gmdoc}
13 \DocInclude{gmdocc}
14 \SelfInclude{\def\CommonEntryCmd{UsgEntry}%
15   \filediv[\file{gmdocDoc.tex},\The\Driver
16   File]{\file{gmdocDoc.tex},\gmhypertarget{The\Driver}\File}%
17   \label{Driver}}
```

For your convenience I decided to add the documentations of the three auxiliary packages:

```
18 \skipgmlonely[\stanza\The\remarks\about\installation\and\compiling
19   of\the\documentation\are\analogous\to\those\in\the\chapter
20   \pk{gmdoc.sty}\and\therefore\omitted.\stanza]
21 \DocInclude{gmutils}
22 \DocInclude{gmiflink}
23 \DocInclude{gmverb}
24 \typeout{%
25   Produce\change\log\with^^J%
26   makeindex-r-s_gmglo.ist-o\jobnamewoe.gls\jobnamewoe.glo^^J
27   (gmglo.ist\should\be\put\into\some\texmf\makeindex\directory.)^^J}
28 \PrintChanges
29 \typeout{%
30   Produce\index\with^^J%
31   makeindex-r\jobnamewoe^^J}
32 \PrintIndex
33 \end{document}
```

MakeIndex shell commands:

```
34 makeindex-r_gmdocDoc
35 makeindex-r-s_gmglo.ist-o_gmdocDoc.gls_gmdocDoc.glo
(gmglo.ist should be put into some texmf/makeindex directory.)
```

And “That’s all, folks” ;-).

d. The gmutils Package¹

Written by Grzegorz Murzynowski,
natror at o2 dot pl

©2005, 2006, 2007 by Grzegorz Murzynowski.

This program is subject to the L^AT_EX Project Public License.

See <http://www.ctan.org/tex-archive/help/Catalogue/licenses.lppl.html> for the details of that license.

LPPL status: "author-maintained".

Many thanks to my T_EX Guru Marcin Woliński for his T_EXnical support.

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{gmutils}
3 [2007/04/28_v0.80_some_rather_TeXnical_macros,_some_of_them_
   tricky_(GM)]
```

Intro

The `gmutils.sty` package provides some macros that are analogous to the standard L^AT_EX ones but extend their functionality, such as `\@ifnextcat`, `\addtomacro` or `\begin(*)`. The others are just conveniences I like to use in all my TeX works, such as `\afterfi`, `\pk` or `\cs`.

I wouldn't say they are only for the package writers but I assume some nonzero (L^A)T_EX-awareness of the user.

For details just read the code part.

The remarks about installation and compiling of the documentation are analogous to those in the chapter `gmdoc.sty` and therefore omitted.

Contents of the gmutils.zip Archive

The distribution of the `gmutils` package consists of the following four files.

```
gmutils.sty
README
gmutilsDoc.tex
gmutilsDoc.pdf
```

A couple of abbreviations

```
\@xa 4 \let\@xa\expandafter
\@nx 5 \let\@nx\noexpand
```

The `\newgif` declaration's effect is used even in the L^AT_EX 2_ε source by redefining some particular user defined ifs (UD-ifs henceforth) step by step. The goal is to make the

¹ This file has version number v0.80 dated 2007/04/28.

UD-if's assignment global. I needed it at least twice during `gmdoc` writing so I make it a macro. It's an almost verbatim copy of L^AT_EX's `\newif` modulo the letter *g* and the `\global` prefix. (File `d: ltdefns.dtx` Date: 2004/02/20 Version v1.3g, lines 139–150)

```
\newif 6 \def\newgif#1{%
       7   {\escapechar\m@ne
       8     \global\let#1\iffalse
       9     \@gif#1\iftrue
      10     \@gif#1\iffalse
      11   }}
```

‘Almost’ is also in the detail that in this case, which deals with `\global` assignments, we don’t have to bother with storing and restoring the value of `\escapechar`: we can do all the work inside a group.

```
12 \def\@gif#1#2{%
13   \@xa\gdef\csname\@xa@gobbletwo\string#1%
14   g% the letter g for ‘\global’.
15   \@xa@gobbletwo\string#2\endcsname
16   {\global\let#1#2}}
```

After `\newgif\ifffoo` you may type `{\foogtrue}` and the `\ifffoo` switch becomes globally equal `\iftrue`. Simili modo `\foogfalse`. Note the letter *g* added to underline globalness of the assignment.

If for any reason, no matter how queer ;-) may it be, you need *both* global and local switchers of your `\if...`, declare it both with `\newif` and `\newgif`.

Note that it's just a shorthand. `\global\if<switch>true/false` *does* work as expected.

There's a trouble with `\refstepcounter`: defining `\@currentlabel` is local. So let's `\def` a `\global` version of `\refstepcounter`.

Warning. I use it because of very special reasons in `gmdoc` and in general it is probably not a good idea to make `\refstepcounter` global since it is contrary to the original L^AT_EX approach.

```
\grefstepcounter 17 \newcommand*\grefstepcounter[1]{%
                  18   {\let\protected@edef=\protected@xdef\refstepcounter{#1}}}
```

Naïve first try `\globaldefs=\tw@` raised an error `unknown_\command_\reserved@e`. The matter was to globalize `\protected@edef` of `\@currentlabel`.

Thanks to using the true `\refstepcounter` inside, it observes the change made to `\refstepcounter` by `hyperref`.

Another shorthand. It may decrease a number of `\expandafters` e.g.

```
\glet 19 \def\glet{\global\let}
```

L^AT_EX provides a very useful `\g@addto@macro` macro that adds its second argument to the current definition of its first argument (works iff the first argument is a no argument macro). But I needed it some times in a document, where `@` is not a letter. So:

```
\gaddtomacro 20 \let\gaddtomacro=\g@addto@macro
```

The redefining of the first argument of the above macro(s) is `\global`. What if we want it local? Here we are:

```
\addto@macro 21 \long\def\addto@macro#1#2{%
                22   \toks@\@xa{#1#2}%
```

```

23 \edef#1{\the\toks@}%
24 }% (\toks@ is a scratch register, namely \toks0.)

```

And for use in the very document,

```

\addtomacro 25 \let\addtomacro=\addto@macro
\@emptyify 26 \newcommand*\@emptyify[1]{\let#1=\@empty}
\emptyify 27 \@ifdefinable\emptyify{\let\emptyify\@emptyify}

```

Note the two following commands are in fact one-argument.

```

\g@emptyify 28 \newcommand*\g@emptyify{\global\@emptyify}
\gemptyify 29 \@ifdefinable\gemptyify{\let\gemptyify\g@emptyify}
\@relaxen 30 \newcommand*\@relaxen[1]{\let#1=\relax}
\relaxen 31 \@ifdefinable\relaxen{\let\relaxen\@relaxen}

```

Note the two following commands are in fact one-argument.

```

\g@relaxen 32 \newcommand*\g@relaxen{\global\@relaxen}
\grelaxen 33 \@ifdefinable\grelaxen{\let\grelaxen\g@relaxen}

```

For the heavy debugs I was doing while preparing `gmdoc`, as a last resort I used `\showlists`. But this command alone was usually too little: usually it needed setting `\showboxdepth` and `\showboxbreadth` to some positive values. So,

```

\gmshowlists 34 \def\gmshowlists{\showboxdepth=1000\showboxbreadth=1000\showlists}
\nameshow 35 \newcommand*\nameshow[1]{\@xa\show\csname#1\endcsname}

```

Standard `\string` command returns a string of ‘other’ chars except for the space, for which it returns `_10`. In `gmdoc` I needed the spaces in macros’ and environments’ names to be always `_12`, so I define

```

\Xiistring 36 \def\Xiistring#1{%
37 \if\@nx#1\twelvespace
38 \twelvespace
39 \else
40 \string#1%
41 \fi}

```

`\@ifnextcat`, `\@ifnextac`

As you guess, we `\def \@ifnextcat` à la `\@ifnextchar`, see L^AT_EX 2_ε source dated 2003/12/01, file `d`, lines 253–271. The difference is in the kind of test used: while `\@ifnextchar` does `\ifx`, `\@ifnextcat` does `\ifcat` which means it looks not at the meaning of a token(s) but at their `\catcode`(s). As you (should) remember from The T_EXbook, the former test doesn’t expand macros while the latter does. But in `\@ifnextcat` the peeked token is protected against expanding by `\noexpand`. Note that the first parameter is not protected and therefore it shall be expanded if it’s a macro. Because an assignment is involved, you can’t test whether the next token is an active char.

```

\@ifnextcat 42 \long\def\@ifnextcat#1#2#3{%
43 \def\reserved@d{#1}%
44 \def\reserved@a{#2}%
45 \def\reserved@b{#3}%
46 \futurelet\@let@token\@ifncat}

```



```

47 \def\@ifncat{%
48   \ifx\@let@token\@sptoken
49     \let\reserved@c\@xifncat
50   \else
51     \ifcat\reserved@d\@nx\@let@token
52       \let\reserved@c\reserved@a
53     \else
54       \let\reserved@c\reserved@b
55     \fi
56   \fi
57   \reserved@c}

58 {\def\:{\let\@sptoken=_}\:_% this makes \@sptoken a space token.
59 \def\:{\@xifncat}\@xa\gdef\:{\futurelet\@let@token\@ifncat}}

```

Note the trick to get a macro with no parameter and requiring a space after it. We do it inside a group not to spoil the general meaning of \: (which we extend later).

But how to peek at the next token to check whether it's an active char? First, we look with \@ifnextcat whether there stands a group opener. We do that to avoid taking a whole {...} as the argument of the next macro, that doesn't use \futurelet but takes the next token as an argument, tests it and puts back intact.

```

\@ifnextac 60 \long\def\@ifnextac#1#2{%
61   \@ifnextcat\bgroup{#2}{\gm@ifnac{#1}{#2}}}

62 \long\def\gm@ifnac#1#2#3{%
63   \ifcat\@nx~\@nx#3\afterfi{#1#3}\else\afterfi{#2#3}\fi}

```

Yes, it won't work for an active char \let to {₁, but it *will* work for an active char \let to a char of catcode ≠ 1. (Is there anybody on Earth who'd make an active char working as \bgroup?)

Now, define a test that checks whether the next token is a genuine space, ₁₀ that is. First define a CS let such a space. The assignment needs a little trick (The T_EXbook appendix D) since \let's syntax includes one optional space after =.

```

64 \let\@tempa\*%
65 \def\*{%
66   \let\*\@tempa
67   \let\gm@letspace=_}%
68 \*_%

\@ifnextspace 69 \def\@ifnextspace#1#2{%
70   \let\@reserveda\*%
71   \def\*{%
72     \let\*\@reserveda
73     \ifx\@let@token\gm@letspace\afterfi{#1}%
74     \else\afterfi{#2}%
75     \fi}%
76   \futurelet\@let@token\*}

```

First use of this macro is for an active - that expands to --- if followed by a space.

\afterfi and Pals

It happens from time to time that you have some sequence of macros in an `\if...` and you would like to expand `\fi` before expanding them (e.g., when the macros should take some tokens next to `\fi...` as their arguments. If you know how many macros are there, you may type a couple of `\expandafters` and not to care how terrible it looks. But if you don't know how many tokens will there be, you seem to be in a real trouble. There's the Knuthian trick with `\next`. And here another, revealed to me by my T_EX Guru.

I think the situations when the Knuthian (the former) trick is not available are rather seldom, but they are imaginable at least: the `\next` trick involves an assignment so it won't work e.g. in `\edef`. But in general it's only a matter of taste which one to use.

One warning: those macros peel the braces off, i.e.,

```
\if...\afterfi{\@makeother\^^M}\fi
```

causes a leakage of `^^M12`. To avoid pollution write

```
\if...\afterfi{\bgroup\@makeother\^^M\egroup}\fi.
```

```
\afterfi 77 \long\def\afterfi#1#2\fi{\fi#1}
```

And two more of that family:

```
\afterfifi 78 \long\def\afterfifi#1#2\fi#3\fi{\fi\fi#1}
```

```
\afteriffifi 79 \long\def\afteriffifi#1#2\if#3\fi#4\fi{\fi#1}
```

Notice the refined elegance of those macros, that cover both 'then' and 'else' cases thanks to #2 that is discarded.

```
80 \long\def\afterififfiffifi#1#2\fi#3\fi#4\fi{\fi#1}
```

```
81 \long\def\afteriffiffifi#1#2\fi#3\fi#4\fi{\fi\fi#1}
```

```
82 \long\def\afterfififfifi#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}
```

Almost an Environment or Redefinition of \begin

We'll extend the functionality of `\begin`: the non-starred instances shall act as usual and we'll add the starred version. The difference of the latter will be that it won't check whether the 'environment' has been defined so any name will be allowed.

This is intended to structure the source with named groups that don't have to be especially defined and probably don't take any particular action except the scoping.

(If the `\begin*`'s argument is a (defined) environment's name, `\begin*` will act just like `\begin`.)

Original L^AT_EX's `\begin`:

```
\def\begin#1{%
  \@ifundefined{#1}%
    {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
    {\def\reserved@a{\def\@currenvir{#1}%
      \edef\@currenvline{\on@line}%
      \csname #1\endcsname}}%
  \@ignorefalse
  \begingroup\@endpefalse\reserved@a}
```

```
\@begnamedgroup 83 \@ifdefinable\@begnamedgroup{\relax}
```

```
84 \def\@begnamedgroup#1{%
```

```

85 \ignorefalse% not to ignore blanks after group
86 \begingroup\endpfalse
87 \def\@currentvir{#1}%
88 \edef\@currentvline{\on@line}%
89 \csname_#1\endcsname}% if the argument is a command's name (an environment's
    e.g.), this command will now be executed. (If the corresponding control
    sequence hasn't been known to TEX, this line will act as \relax.)

```

For back compatibility with my earlier works

```

\namegroup 90 \let\namegroup\@begnamedgroup
    And for the ending
\enamegroup 91 \def\enamegroup#1{\end{#1}}
    And we make it the starred version of \begin.
\old@begin 92 \let\old@begin\begin
    \begin 93 \def\begin{\@ifstar{\@begnamedgroup}{\old@begin}}
\begin*

```

Improvement of \end

It's very clever and useful that `\end` checks whether its argument is `ifx`-equivalent `@currentvir`. However, it works not quite as I would expect: Since the idea of environment is to open a group and launch the cs named in the `\begin`'s argument. That last thing is done with `\csname... \endcsname` so the char catcodes are equivalent. Thus should be also in the `\end`'s test and therefore we ensure the compared texts are both expanded and made all 'other'.

```

94 \def\@checkend#1{%
95   \edef\reserved@a{\@xa\string\csname#1\endcsname}%
96   \edef\exii@currentvir{\@xa\string\csname\@currentvir\endcsname}%
97   \ifx\reserved@a\exii@currentvir\else\@badend{#1}\fi}

```

Thanks to it you may write `\begin{macrocode*}` with `*12` and end it with `\end{. macrocode*}` with `*11` (that was the problem that led me to this solution). The error messages looked really funny:

```
!_LaTeX_Error:_\begin{macrocode*}_on_input_line_1844_ended_by_\end{macrocode*}.
```

Of course, you might write also `\end{macrocode\star}` where `\star` is defined as 'other' star or letter star.

From relsize

As file `relsize.sty`, v3.1 dated July 4, 2003 states, L^AT_EX 2_ε version of these macros was written by Donald Arseneau asnd@triumf.ca and Matt Swift swift@bu.edu after the L^AT_EX 2.09 `smaller.sty` style file written by Bernie Cosell cosell@WILMA.BBN.COM.

I take only the basic, non-math mode commands with the assumption that there are the predefined font sizes.

```

\relsize    You declare the font size with \relsize{<n>} where <n> gives the number of steps
            ("mag-step" = factor of 1.2) to change the size by. E.g., n = 3 changes from \normalsize
\smaller    to \LARGE size. Negative n selects smaller fonts. \smaller == \relsize{-1};
\larger     \larger == \relsize{1}. \smallerr(my addition) == \relsize{-2}; \largerr
\smallerr   guess yourself.
\largerr

```

(Since `\DeclareRobustCommand` doesn't issue an error if its argument has been defined and it only informs about redefining, loading `relsize` remains allowed.)

```

\relsize 98 \DeclareRobustCommand*\relsize[1]{%
99   \ifmmode_\@nomath\relsize\else
100     \begingroup
101     \@tempcnta_\% assign number representing current font size
102     \ifx\@currsz\@normalsize_\4\else_\_\_\% funny order is to have most ...
103     \ifx\@currsz\small_\3\else_\_\_\_\_\_\_\_\% ...likely sizes checked first
104     \ifx\@currsz\footnotesize_\2\else
105     \ifx\@currsz\large_\5\else
106     \ifx\@currsz\Large_\6\else
107     \ifx\@currsz\LARGE_\7\else
108     \ifx\@currsz\scriptsize_\1\else
109     \ifx\@currsz\tiny_\0\else
110     \ifx\@currsz\huge_\8\else
111     \ifx\@currsz\Huge_\9\else
112       4\rs@unknown@warning_\% unknown state: \normalsize as start-
        ing point
113     \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
    Change the number by the given increment:
114     \advance\@tempcnta#1\relax
    watch out for size underflow:
115     \ifnum\@tempcnta<\z@\rs@size@warning{small}{\string\tiny}.
        \@tempcnta\z@\fi
116     \@xa\endgroup
117     \ifcase\@tempcnta_\% set new size based on altered number
118     \tiny_\or_\scriptsize_\or_\footnotesize_\or_\small_\or_.
        \normalsize_\or
119     \large_\or_\Large_\or_\LARGE_\or_\huge_\or_\Huge_\else
120     \rs@size@warning{large}{\string\Huge}\Huge
121 \fi\fi}% end of \relsize.

\rs@size@warning 122 \providecommand*\rs@size@warning[2]{\PackageWarning{gmutils_\
        (relsize)}{\%
123   Size_\requested_\is_\too_\#1.\MessageBreak_\Using_\#2_\instead}}

\rs@unknown@warning 124 \providecommand*\rs@unknown@warning{\PackageWarning{gmutils_\
        (relsize)}{\Current_\font_\size
125   is_\unknown!_\(Why?!?)\MessageBreak_\Assuming_\string\normalsize}}

    And a handful of shorthands:

\larger 126 \DeclareRobustCommand*\larger[1][\@one]{\relsize{+#1}}
\smaller 127 \DeclareRobustCommand*\smaller[1][\@one]{\relsize{-#1}}
\textlarger 128 \DeclareRobustCommand*\textlarger[2][\@one]{\relsize{+#1}#2}
\textsmaller 129 \DeclareRobustCommand*\textsmaller[2][\@one]{\relsize{-#1}#2}
\largerr 130 \DeclareRobustCommand*\largerr{\relsize{+2}}
\smallerr 131 \DeclareRobustCommand*\smallerr{\relsize{-2}}

```

`\firstofone` and the Queer `\catcodes`

Remember that once a macro's argument has been read, its `\catcodes` are assigned forever and ever. That's what is `\firstofone` for. It allows you to change the `\catcodes` locally for a definition *outside* the changed `\catcodes`' group. Just see the below usage of this macro 'with T_EX's eyes', as my T_EX Guru taught me.

```
\firstofone 132 \long\def\firstofone#1{#1}

            And this one is defined, I know, but it's not \long with the standard definition.

\gobble     133 \long\def\gobble#1{}
\gobbletwo  134 \let\gobbletwo\@gobbletwo

            135 \bgroup\catcode'\_ =8_\%
            136 \firstofone{\egroup
\subs       137   \let\subs=_}

            138 \bgroup\@makeother\_%
            139 \firstofone{\egroup
\twelveunder 140   \def\twelveunder{_{}}

            Now, let's define such a smart _ (underscore) which will be usual _8 in the math mode
            and _12 ('other') outside math.

            141 \bgroup\catcode'\_ =\active
            142 \firstofone{\egroup
\smartunder  143   \newcommand*\smartunder{%
            144     \catcode'\_ =\active
            145     \def_{\ifmmode\subs\else\_fi}}}% We define it as \_ not just as \twelveunder
            because some font encodings don't have _ at the \char'\_ position.

            146 \begingroup\catcode'\!=0
            147 \@makeother\!
            148 !firstofone{!endgroup%
\twelvebackslash 149   !newcommand*!twelvebackslash{\}}

\bslash     150 \@ifundefined{bslash}{\let\bslash=\twelvebackslash}{\}

            151 \begingroup \@makeother\%
            152 \firstofone{\endgroup
\twelvepercent 153   \def\twelvepercent{%%}}

            154 \begingroup_\@makeother\&%
            155 \firstofone{\endgroup%
\twelveand    156   \def\twelveand{&}}

            157 \begingroup\@makeother\_%
            158 \firstofone{\endgroup%
\twelvespace  159   \def\twelvespace{_{}}}
```

Metasymbols

I fancy also another Knuthian trick for typesetting (*metasymbols*) in The T_EXbook. So I repeat it here. The inner `\meta` macro is copied verbatim from doc's v2.1b documentation dated 2004/02/09 because it's so beautifully crafted I couldn't resist. I only don't make it `\long`.

“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.”

```
160 \ifx\l@nohyphenation\undefined
161   \newlanguage\l@nohyphenation
162 \fi
\meta 163 \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.”

```
164   \ensuremath\langle
165   \ifmmode\@xa\@nfss@text\fi
166   {%
167     \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.

```
168     \edef\meta@hyphen@restore{%
169       \hyphenchar\the\font\the\hyphenchar\font}%
170     \hyphenchar\font\m@ne
171     \language\l@nohyphenation
172     #1\/%
173     \meta@hyphen@restore
174   }\ensuremath\rangle
175 }
```

But I define `\meta@font@select` as the brutal and explicit `\it` instead of the original `\itshape` to make it usable e.g. in the `gmdoc`’s `\cs` macro’s argument.

```
176 \def\meta@font@select{\it}
```

The below `\meta`’s drag² is a version of The `TEXbook`’s one.

```
\<...> 177 \def\<#1>{\meta{#1}}
```

Macros for Printing Macros and Filenames

First let’s define three auxiliary macros analogous to `\dywiz` from `polski.sty`: a shorthands for `\discretionary` that’ll stick to the word not spoiling its hyphenability and that’ll won’t allow a linebreak just before nor just after themselves. The `\discretionary` `TEX` primitive has three arguments: #1 ‘before break’, #2 ‘after break’, #3 ‘without break’, remember?

```
\discre 178 \def\discre#1#2#3{\kern0sp\discretionary{#1}{#2}{#3}\penalty10000~
          \hskip0sp\relax}
\discret 179 \def\discret#1{\kern0sp\discretionary{#1}{#1}{#1}\penalty10000~
          \hskip0sp\relax}
```

² Think of the drags that transform a very nice but rather standard ‘auntie’ (‘Tante’ in Deutsch) into a most adorable Queen ;-).

A tiny little macro that acts like \- outside the math mode and has its original meaning inside math.

```
180 \def\:{\ifmmode\afterfi{\mskip\medmuskip}\else\afterfi{\discret{}}\fi}
\vs 181 \newcommand*{\vs}{\discre{\textvisiblespace}{\textvisiblespace}}
```

Then we define a macro that makes the spaces visible even if used in an argument (i.e., in a situation where re\catcodeing has no effect).

```
\printspaces 182 \def\printspaces#1{\let~=\vs_\let\_=\vs_\gm@pswords#1\_@@nil}}
183 \def\gm@pswords#1\_#2@@nil{%
184   \if\relax#1\relax\else#1\fi
185   \if\relax#2\relax\else\vs\penalty\hyphenpenalty\gm@pswords#2\_@@nil\fi}%
      note that in the recursive call of \gm@pswords the argument string is not
      extended with a guardian space: it has been already by \printspaces.
```

```
\sfname 186 \DeclareRobustCommand*\sfname[1]{\textsf{\printspaces{#1}}}
\file 187 \let\file\sfname% it allows the spaces in the filenames (and prints them as _).
```

The below macro I use to format the packages' names.

```
\pk 188 \DeclareRobustCommand*{\pk}[1]{\textsf{\textup{#1}}}
```

Some (if not all) of the below macros are copied from doc and/or ltxdoc.

A macro for printing control sequences in arguments of a macro. Robust to avoid writing an explicit \ into a file. It calls \ttfamily not \tt to be usable in headings which are boldface sometimes.

```
\cs 189 \DeclareRobustCommand*{\cs}[2][\bslash]{%
190   \def~{-{\discretionary{\rmfamily-}}{~}{~}}%
191   \def\{\{\char'\}\}\def\{\{\char'\}\}\ttfamily\_#1#2}}
```

```
\env 192 \DeclareRobustCommand*{\env}[1]{\cs[] {#1}}
```

And one for encouraging linebreaks e.g., before long verbatim words.

```
\possfil 193 \newcommand*\possfil{\hfil\penalty1000\hfilneg}
```

The five macros below are taken from the ltxdoc.dtx.

“\cmd{\foo} Prints \foo verbatim. It may be used inside moving arguments. \cs{^foo} also prints \foo, for those who prefer that syntax. (This second form may even be used when \foo is \outer).”

```
\cmd 194 \def\cmd#1{\cs{\@xa\cmd@to@cs\string#1}}
195 \def\cmd@to@cs#1#2{\char\number'#2\relax}
      \marg{text} prints {\text}, ‘mandatory argument’.
```

```
\marg 196 \def\marg#1{\ttfamily\char'\{\}\meta{#1}\ttfamily\char'\}}
      \oarg{text} prints [{text}], ‘optional argument’. Also \oarg[text] does that.
```

```
\oarg 197 \def\oarg{\@ifnextchar[\@oargsq\@oarg}
198 \def\@oarg#1{\ttfamily[]\meta{#1}\ttfamily}}
199 \def\@oargsq[#1]{\@oarg{#1}}
      \parg{te,xt} prints (\text), ‘picture mode argument’.
```

```
\parg 200 \def\parg{\@ifnextchar(\@pargp\@parg}
201 \def\@parg#1{\ttfamily()\meta{#1}\ttfamily}}
202 \def\@pargp(#1){\@parg{#1}}
```

But we can have all three in one command.

```

203 \AtBeginDocument{%
\arg 204 \let\math@arg\arg
205 \def\arg{\ifmmode\math@arg\else\afterfi{%
206 \ifnextchar[%
207 \oargsq{\ifnextchar(%
208 \pargp\marg}}\fi}%
209 }

```

Storing and Restoring the Meanings of CSs

A command to store the current meaning of a CS in another macro to temporarily redefine the CS and be able to set its original meaning back (when grouping is not recommended):

```

\StoreMacro 210 \def\StoreMacro{%
211 \bgroup\makeatletter\ifstar\egStore@MacroSt\egStore@Macro}
212 \long\def\egStore@Macro#1{\egroup\Store@Macro{#1}}
213 \long\def\egStore@MacroSt#1{\egroup\Store@MacroSt{#1}}
214 \long\def\Store@Macro#1{%
215 \xa\let\csname_/gml/store/string#1\endcsname#1}
216 \long\def\Store@MacroSt#1{%
217 \edef\gmu@smtempa{%
218 \nx\let\x\csname_/gml/store/string#1\endcsname\x\x\csname#1^
\endcsname}
219 \gmu@smtempa}

```

We make the `\StoreMacro` command a three-step to allow usage of the most inner macro also in the next command.

The starred version, `\StoreMacro*` works with csnames (without the backslash). It's first used to store the meanings of robust commands, when you may need to store not only `\foo`, but also `\csname_foo\endcsname`.

The next command iterates over a list of CSs and stores each of them. The CS may be separated with commas but they don't have to.

```

\StoreMacros 220 \long\def\StoreMacros{\bgroup\makeatletter\Store@Macros}
221 \long\def\Store@Macros#1{\egroup
222 \let\gml@storeCS\Store@Macro
223 \gml@storemacros#1.}

```

And the inner iterating macro:

```

224 \long\def\gml@storemacros#1{%
225 \def\@tempa{\nx#1}% My TEX Guru's trick to deal with \fi and such, i.e., to
hide #1 from TEX when it is processing a test's branch without expanding.
226 \if\@tempa.% a dot finishes storing.
227 \else
228 \if\@tempa,% The list this macro is put before may contain commas and that's
O.K., we just continue the work.
229 \afterfifi\gml@storemacros
230 \else% what is else this shall be stored.
231 \gml@storeCS{#1}% we use a particular CS to may \let it both to the storing
macro as above and to the restoring one as below.
232 \afterfifi\gml@storemacros

```



```

233     \fi
234     \fi}

```

And for the restoring

```

\RestoreMacro 235 \def\RestoreMacro{%
236     \bgroup\makeatletter\@ifstar\egRestore@MacroSt\egRestore@Macro}
237 \long\def\egRestore@Macro#1{\egroup\Restore@Macro{#1}}
238 \long\def\egRestore@MacroSt#1{\egroup\Restore@MacroSt{#1}}
239 \long\def\Restore@Macro#1{%
240     \@xa\let\@xa#1\csname_/gml/store/string#1\endcsname}
241 \long\def\Restore@MacroSt#1{%
242     \edef\gmu@smtempa{%
243         \@nx\let\@xa\@nx\csname#1\endcsname\@xa\@nx\csname/gml/store#1^
            \endcsname}
244     \gmu@smtempa}

```

```

\RestoreMacros 245 \long\def\RestoreMacros{\bgroup\makeatletter\Restore@Macros}
246 \long\def\Restore@Macros#1{\egroup
247     \let\gml@storeCS\Restore@Macro% we direct the core CS towards restoring and
        call the same iterating macro as in line 223.
248     \gml@storemacros#1.}

```

As you see, the `\RestoreMacros` command uses the same iterating macro inside, it only changes the meaning of the core macro.

And to restore *and* use immediately:

```

249 \def\StoredMacro{\bgroup\makeatletter\Stored@Macro}
250 \long\def\Stored@Macro#1{\egroup\Restore@Macro#1#1}

```

It happened (see the definition of `\@docinclude` in `gmdoc.sty`) that I needed to `\relax` a bunch of macros and restore them after some time. Because the macros were rather numerous and I wanted the code more readable, I wanted to `\do` them. After a proper defining of `\do` of course. So here is this proper definition of `\do`, provided as a macro (a declaration).

```

\StoringAndRelaxingDo 251 \long\def\StoringAndRelaxingDo{%
252     \def\do##1{\@xa\let\csname_/gml/store/string##1\endcsname##1%
253         \let##1\relax}}

```

And here is the counter-definition for restore.

```

\RestoringDo 254 \long\def\RestoringDo{%
255     \def\do##1{%
256         \@xa\let\@xa##1\csname_/gml/store/string##1\endcsname}}

```

And to store a cs as explicitly named cs, i.e. to `\let` one csname another (`\n@melet` not `\@namelet` because the latter is defined in Till Tantau's `beamer` class another way):

```

257 \def\n@melet#1#2{%
258     \edef\@tempa{%
259         \let\@xa\@nx\csname#1\endcsname
260         \@xa\@nx\csname#2\endcsname}%
261     \@tempa}

```

Not only preamble!

Let's remove some commands from the list to erase at begin document! Primarily that list was intended to save memory not to forbid anything. Nowadays, when memory is cheap, the list of only-preamble commands should be rethought IMO.

```
\not@onlypreamble 262 \newcommand\not@onlypreamble[1]{\%  
263   \def\do##1{\ifx##1\else\@nx\do\@nx##1\fi}%  
264   \xdef\@preamblecmds{\@preamblecmds}}  
  
265 \not@onlypreamble\@preamblecmds  
266 \not@onlypreamble\@ifpackageloaded  
267 \not@onlypreamble\@ifclassloaded  
268 \not@onlypreamble\@ifl@aded  
269 \not@onlypreamble\@pkgextension
```

And let's make the message of only preamble command's forbidden use informative a bit:

```
270 \def\gm@notprerr{\_can\_be\_used\_only\_in\_preamble\_(\on@line)}  
271 \AtBeginDocument{%  
272   \def\do#1{\@nx\do\@nx#1}%  
273   \edef\@preamblecmds{%  
274     \def\@nx\do##1{%  
275       \def##1{!\_@nx\string##1\_@nx\gm@notprerr}}%  
276     \@preamblecmds}}
```

Third Person Pronouns

Is a reader of my documentations 'she' or 'he' and does it make a difference?

Not to favour any gender in the personal pronouns, define commands that'll print alternately masculine and feminine pronoun of third person. By 'any' I mean not only typically masculine and typically feminine but the entire amazingly rich variety of people's genders, *including* those who do not describe themselves as 'man' or 'woman'.

One may say two pronouns is far too little to cover this variety but I could point Ursula's K. LeGuin's *The Left Hand Of Darkness* as another acceptable answer. In that moody and moderate SF novel the androgynous persons are usually referred to as 'mister', 'sir' or 'he': the meaning of reference is extended. Such an extension also my automatic pronouns do suggest. It's *not* political correctness, it's just respect to people's diversity.

```
277 \newcounter{gm@PronounGender}  
  
\gm@atppron 278 \newcommand*\gm@atppron[2]{%  
279   \stepcounter{gm@PronounGender}% remember \stepcounter is global.  
280   \ifodd\arabic{gm@PronounGender}#1\else#2\fi}  
  
\heshe 281 \newcommand*\heshe{\gm@atppron{he}{she}}  
\hisher 282 \newcommand*\hisher{\gm@atppron{his}{her}}  
\himher 283 \newcommand*\himher{\gm@atppron{him}{her}}  
\hishers 284 \newcommand*\hishers{\gm@atppron{his}{hers}}  
  
\HeShe 285 \newcommand*\HeShe{\gm@atppron{He}{She}}  
\HisHer 286 \newcommand*\HisHer{\gm@atppron{His}{Her}}  
\HimHer 287 \newcommand*\HimHer{\gm@atppron{Him}{Her}}  
\HisHers 288 \newcommand*\HisHers{\gm@atppron{His}{Hers}}
```

To Save Precious Count Registers

It's a contribution to T_EX's ecology ;-). You can use as many CSs as you wish and you may use only 256 count registers (although in eT_EX there are 2¹⁶ count registers, which makes the following a bit obsolete).

```
289 \newcommand*\nummacro[1]{\gdef#1{0}}
290 \newcommand*\stepnummacro[1]{%
291   \@tempcnta=#1\relax
292   \advance\@tempcnta by 1\relax
293   \xdef#1{\the\@tempcnta}}% Because of some mysterious reasons explicit \count0
    interfered with page numbering when used in \gmd@evpaddonce in gmdoc.
294 \newcommand*\addtonummacro[2]{%
295   \count0=#1\relax
296   \advance\count0 by #2\relax
297   \xdef#1{\the\count0}}
```

Need an explanation? The `\nummacro` declaration defines its argument (that should be a CS) as `{0}` which is analogous to `\newcount` declaration but doesn't use up any count register.

Then you may use this numeric macro as something between T_EX's count CS and L^AT_EX's counter. The macros `\stepnummacro` and `\addtonummacro` are analogous to L^AT_EX's `\stepcounter` and `\addtocounter` respectively: `\stepnummacro` advances the number stored in its argument by 1 and `\addtonummacro` advances it by the second argument. As the L^AT_EX's analogoi, they have the global effect (the effect of global warming ;-)).

So far I've used only `\nummacro` and `\stepnummacro`. Notify me if you use them and whether you need sth. more, `\multiplynummacro` e.g.

Improvements to mwcls Sectioning Commands

That is, 'Expe-ri-mente'³ mit MW sectioning & `\refstepcounter` to improve mwcls's cooperation with hyperref. They shouldn't make any harm if another class (non-mwcls) is loaded.

We `\refstep` sectioning counters even if the sectionings are not numbered, because otherwise

1. pdfT_EX cried of multiply defined `\labels`,
2. e.g. in a table of contents the hyperlink `<rozdzia\l\Kwiaty\polskie>` linked not to the chapter's heading but to the last-before-it change of `\ref`.

```
298 \AtBeginDocument{% because we don't know when exactly hyperref is loaded and
    maybe after this package.
299   \@ifpackageloaded{hyperref}{\newcounter{NoNumSecs}%
300     \setcounter{NoNumSecs}{617}% to make \refing to an unnumbered section
        visible (and funny?).
301     \def\gm@hyperrefstepcounter{\refstepcounter{NoNumSecs}}%
302     \DeclareRobustCommand*\gm@targetheading[1]{%
303       \hypertarget{#1}{#1}}}% end of then
304   {\def\gm@hyperrefstepcounter{}}%
```

³ A. Berg, *Wozzeck*.

```

305 \def\gm@targetheading#1{#1}}% end of else
306 }% of \AtBeginDocument

```

Auxiliary macros for the kernel sectioning macro:

```

307 \def\gm@dontnumbersectionsoutofmainmatter{%
308 \if@mainmatter\else_\HeadingNumberedfalse_\fi}
309 \def\gm@clearpagesduetoopenright{%
310 \if@openright\cleardoublepage\else_\clearpage\fi}

```

To avoid \defining of \mw@sectionxx if it's undefined, we redefine \def to gobble the definition and restore the original meaning of itself.

Why shouldn't we change the ontological status of \mw@sectionxx (not define if undefined)? Because some macros (in gmdocc e.g.) check it to learn whether they are in an mwcls or not.

But let's make a shorthand for this test since we'll use it three times in this package and maybe also somewhere else.

```

\@ifnotmw 311 \long\def\@ifnotmw#1#2{\@ifundefined{mw@sectionxx}{#1}{#2}}
312 \let\gmu@def\def
313 \@ifnotmw{%
314 \StoreMacro\gmu@def_\def\gmu@def#14#2{\RestoreMacro\gmu@def}}{}

```

I know it may be of bad taste (to write such a way *here*) but I feel so lonely and am in an alien state of mind after 3 hour sleep last night and, worst of all, listening to sir Edward Elgar's flamboyant Symphonies d'Art Nouveau.

A *decent* person would just wrap the following definition in \@ifundefined's Else. But look, the definition is so long and I feel so lonely etc. So, I define \def (for some people there's nothing sacred) to be a macro with two parameters, first of which is delimited by digit 4 (the last token of \mw@sectionxx's parameter string) and the latter is undelimited which means it'll be the body of the definition. Such defined \def does nothing else but restores its primitive meaning by the way sending its arguments to the Gobbled Tokens' Paradise. Luckily, \RestoreMacro contains \let not \def.

The kernel of MW's sectioning commands:

```

315 \gmu@def\mw@sectionxx#1#2[#3]#4{%
316 \edef\mw@HeadingLevel{\csname_\#1@level\endcsname
317 \space}% space delimits level number!
318 \ifHeadingNumbered
319 \ifnum_\mw@HeadingLevel>\c@secnumdepth_\HeadingNumberedfalse_\fi

```

line below is in ifundefined to make it work in classes other than mwbk

```

320 \@ifundefined{if@mainmatter}{\{^
\gm@dontnumbersectionsoutofmainmatter}
321 \fi
% \ifHeadingNumbered
% \refstepcounter{#1}%
% \protected@edef\HeadingNumber{\csname the#1\endcsname\relax}%
% \else
% \let\HeadingNumber\@empty
% \fi
322 \def\HeadingRHeadText{#2}%
323 \def\HeadingTOCText{#3}%

```

```

324 \def\HeadingText{#4}%
325 \def\mw@HeadingType{#1}%
326 \if\mw@HeadingBreakBefore
327   \if@specialpage\else\thispagestyle{closing}\fi
328   \@ifundefined{if@openright}{\gm@clearpagesduetoopenright}%
329   \if\mw@HeadingBreakAfter
330     \thispagestyle{blank}\else
331     \thispagestyle{opening}\fi
332     \global\@topnum\z@
333 \fi% of \if\mw@HeadingBreakBefore

```

placement of \refstep suggested by me (GM)

```

334 \ifHeadingNumbered
335   \refstepcounter{#1}%
336   \protected@edef\HeadingNumber{\csname\the#1\endcsname\relax}%
337 \else
338   \let\HeadingNumber\@empty
339   \gm@hyperrefstepcounter
340 \fi% of \ifHeadingNumbered
341 \if\mw@HeadingRunIn
342   \mw@runinheading
343 \else
344   \if\mw@HeadingWholeWidth
345     \if@twocolumn
346       \if\mw@HeadingBreakAfter
347         \onecolumn
348         \mw@normalheading
349         \pagebreak\relax
350       \if@twoside
351         \null
352         \thispagestyle{blank}%
353         \newpage
354       \fi% of \if@twoside
355     \twocolumn
356   \else
357     \@topnewpage[\mw@normalheading]%
358   \fi% of \if\mw@HeadingBreakAfter
359 \else
360   \mw@normalheading
361   \if\mw@HeadingBreakAfter\pagebreak\relax\fi
362 \fi% of \if@twocolumn
363 \else
364   \mw@normalheading
365   \if\mw@HeadingBreakAfter\pagebreak\relax\fi
366 \fi% of \if\mw@HeadingWholeWidth
367 \fi% of \if\mw@HeadingRunIn
368 }

```

(End of Experimente with MW sectioning.)

Compatibilising Standard and mwcls Sectionings

If you use Marcin Woliński’s document classes (`mwcls`), you might have met their little queerness: the sectioning commands take two optional arguments instead of standard one. It’s reasonable since one may wish one text to be put into the running head, another to the toc and yet else to the page. But the order of optionalities causes an incompatibility with the standard classes: MW section’s first optional argument goes to the running head not to toc and if you’ve got a source file written with the standard classes in mind and use the first (and only) optional argument, the effect with `mwcls` would be different if not error.

Therefore I counter-assign the commands and arguments to reverse the order of optional arguments for sectioning commands when `mwcls` are in use and reverse, to make `mwcls`-like sectioning optionals usable in the standard classes.

With the following in force, you may both in the standard classes and in `mwcls` give a sectioning command one or two optional arguments (and mandatory the last, of course). If you give just one optional, it goes to the running head and to toc as in `scls` (which is unlike in `mwcls`). If you give two optionals, the first goes to the running head and the other to toc (like in `mwcls` and unlike in `scls`).

(In both cases the mandatory last argument goes only to the page.)

What more is unlike in `scls`, it’s that even with them the starred versions of sectioning commands allow optionals (but they still send them to the Gobbled Tokens’ Paradise).

(In `mwcls`, the only difference between starred and non-starred sec commands is (not) numbering the titles, both versions make a contents line and a mark and that’s not changed with my redefinitions.)

```
369 \ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those
      written with two optionals.
370 \def\gm@secini{\gm@la}%
\gm@secxx 371 \def\gm@secxx#1#2[#3]#4{%
372 \ifx\gm@secstar\@empty
373 \n@melet{\gm@true@#1mark}{#1mark}% a little trick to allow a special version
      of the heading just to the running head.
374 \@namedef{#1mark}##1{% we redefine \<sec>mark to gobble its argument and
      to launch the stored true marking command on the appropriate argu-
      ment.
375 \csname\gm@true@#1mark\endcsname{#2}%
376 \n@melet{#1mark}{\gm@true@#1mark}% after we’ve done what we wanted
      we restore original \#1mark.
377 }%
378 \def\gm@secstar{[#3]}% if \gm@secstar is empty, which means the section-
      ing command was written starless, we pass the ‘true’ sectioning com-
      mand #3 as the optional argument. Otherwise the sectioning command
      was written with star so the ‘true’ s.c. takes no optional.
379 \fi
380 \@xa\@xa\csname\gm@secini#1\endcsname
381 \gm@secstar{#4}}%
382 }{% we are in mwcls and want to reverse MW’s optionals order i.e., if there’s just one
      optional, it should go both to toc and to running head.
383 \def\gm@secini{\gm@mw}%
384 \let\gm@secmarkh\@gobble% in mwcls there’s no need to make tricks for special
      version to running headings.
```

```

\gm@secxx 385 \def\gm@secxx#1#2[#3]#4{%
386 \xa\xa\csname\gm@secini#1\endcsname
387 \gm@secstar[#2][#3]{#4}}%
388 }

389 \def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
390 \def\gm@secx#1[#2]{%
391 \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}% if there's only
one optional, we double it not the mandatory argument.

392 \def\gm@straightensec#1{% the parameter is for the command's name.
393 \@ifundefined{#1}{}{% we don't change the ontological status of the command
because someone may test it.
394 \n@melet{\gm@secini#1}{#1}%
395 \@namedef{#1}{%
396 \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
397 \def\gm@secstar{}\gm@sec{#1}}}%
398 }%

399 \let\do\gm@straightensec
400 \do{part}\do{chapter}\do{section}\do{subsection}\do{subsubsection}
401 \@ifnotmw{\do{paragraph}}% this 'straightening' of \paragraph with the stan-
dard article caused the 'TeX capacity exceeded' error. Anyway, who on Earth
wants paragraph titles in toc or running head?

```

enumerate* and itemize*

We wish the starred version of `enumerate` to be just numbered paragraphs. But `hyperref` redefines `\item` so we should do it a smart way, to set the L^AT_EX's `list` parameters that is.

(Marcin Woliński in `mwcls` defines those environments slightly different: his item labels are indented, mine are not; his subsequent paragraphs of an item are not indented, mine are.)

```

enumerate* 402 \@namedef{enumerate*}{%
403 \ifnum\@enumdepth>\thr@@
404 \toodeep
405 \else
406 \advance\@enumdepth\@ne
407 \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
408 \xa\list\csname_label\@enumctr\endcsname{%
409 \partopsep\topsep\topsep\z@\leftmargin\z@
410 \itemindent\@parindent\%
411 \labelwidth\@parindent
412 \advance\labelwidth-\labelsep
413 \listparindent\@parindent
414 \usecounter\@enumctr
415 \def\makelabel##1{##1\hfil}}%
416 \fi}
417 \@namedef{endenumerate*}{\endlist}

itemize* 418 \@namedef{itemize*}{%
419 \ifnum\@itemdepth>\thr@@

```

```

420     \@toodeep
421   \else
422     \advance\@itemdepth\@ne
423     \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
424     \@xa\list\csname\@itemitem\endcsname{%
425       \partopsep\topsep_\topsep\z@_\leftmargin\z@
426       \itemindent\@parindent
427       \labelwidth\@parindent
428       \advance\labelwidth-\labelsep
429       \listparindent\@parindent
430       \def\makelabel##1{##1\hfil_}}%
431   \fi}
432 \@namedef{enditemize*}{\endlist}

```

The Logos

We'll modify The L^AT_EX logo now to make it fit better to various fonts.

```

433 \let\oldLaTeX\LaTeX
434 \let\oldLaTeXe\LaTeXe
435 \def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
436 \newcommand*\DeclareLogo[3][\relax]{%

```

#1 is for non-L^AT_EX spelling and will be used in the PD1 encoding (to make pdf bookmarks);

#2 is the command, its name will be the PD1 spelling by default,

#3 is the definition for all the font encodings except PD1.

```

437   \ifx\relax#1\def\@tempa{\@xa@gobble\string#2}%
438   \else
439     \def\@tempa{#1}%
440   \fi
441   \edef\@tempa{%
442     \@nx\DeclareTextCommand\@nx#2{PD1}{\@tempa}}
443   \@tempa
444   \DeclareTextCommandDefault#2{#3}}
445 \DeclareLogo\LaTeX{%
446   {%
447     L%
448     \setbox\z@\hbox{\check@mathfonts
449       \fontsize\sf@size\z@
450       \math@fontsfalse\selectfont
451       A}%
452     \kern-.57\wd\z@
453     \sbox\tw@_T%
454     \vbox_\to\ht\tw@{\copy\z@_\vss}%
455     \kern-.2\wd\z@}% originally -, 15 em for T.
456   {%
457     \ifdim\fontdimen1\font=\z@
458     \else
459       \count\z@=\fontdimen5\font

```



```

460      \multiply\count\z@_by_64\relax
461      \divide\count\z@_by_p@
462      \count\tw@=\fontdimen1\font
463      \multiply\count\tw@_by\count\z@
464      \divide\count\tw@_by_64\relax
465      \divide\count\tw@_by\tw@
466      \kern-\the\count\tw@_sp\relax
467      \fi}%
468      \TeX}

469 \DeclareLogo\LaTeXe{\mbox{\m@th\if
470      b\expandafter\@car\f@series\@nil\boldmath\fi
471      \LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}

‘(L)TEX’ in my opinion better describes what I work with/in than just ‘LATEX’.

```

```

\LaTeXpar 472 \DeclareLogo[(La)TeX]{\LaTeXpar}{%
473      {%
474      \setbox\z@\hbox{({}% )
475      \copy\z@
476      \kern-.2\wd\z@_L%
477      \setbox\z@\hbox{\check@mathfonts
478      \fontsize\sf@size\z@
479      \math@fontsfalse\selectfont
480      A}%
481      \kern-.57\wd\z@
482      \sbox\tw@_T%
483      \vbox_1to\ht\tw@{\box\z@%
484      \vss}%
485      }%
486      \kern-.07em% originally –, 15 em for T.
487      {% (
488      \sbox\z@)%
489      \kern-.2\wd\z@\copy\z@
490      \kern-.2\wd\z@}\TeX
491      }

```

Expanding turning stuff all into ‘other’

While typesetting a unicode file contents with `inputenc` package I got a trouble with some Unicode sequences that expanded to unexpandable CSs: they couldn’t be used within `\csname... \endcsname`. My T_EXGuru advised to use `\meanig` to make all the name ‘other’. So—here we are.

Don’t use them in `\edefs`, they would expand not quite.

The next macro turns its #2 all into ‘other’ chars and assigns them to #1 which has to be a CS or an active char.

```

\def@other 492 \long\def\def@other#1#2{%
493      \long\def\gm@def@other@tempa{#2}%
494      \all@other#1{#2}}

```

The next macro is intended to be put in `\edefs` with a macro argument. The meaning of the macro will be made all ‘other’ and the words ‘(long) macro:-i’ gobbled.

```
\all@other 495 \def\all@other#1{\@xa\gm@gobmacro\meaning#1}
```

The `\gm@gobmacro` macro above is applied to gobble the `\meaning`'s beginnig, `long_`
`macro:->` all 'other' that is.

```
496 \edef\@tempa{%
\gm@gobmacro 497 \def\@nx\gm@gobmacro##1\@xa@gobble\string\macro:->{}}
498 \@tempa
```

In the next two macros' names, 'unex' stands both for not expanding the argument(s)
and for disastrously partial unexpandability of the macros themselves.

```
\unex@namedef 499 \long\def\unex@namedef#1#2{%
500 \edef\@other\gmu@tempa{#1}%
501 \@xa\long\@xa\def\csname\gmu@tempa\endcsname{#2}}

\unex@nameuse 502 \long\def\unex@nameuse#1{%
503 \edef\@other\gmu@tempa{#1}%
504 \csname\gmu@tempa\endcsname}
```

Varia

A very neat macro provided by doc. I copy it `~verbatim`.

```
\* 505 \DeclareRobustCommand*\*{\leavevmode\lower.8ex\hbox{$\,\widetilde{\_\_}^
\,\$}}
```

The standard `\obeyspaces` declaration just changes the space's `\catcode` to 13 ('active'). Usually it is fairly enough because no one 'normal' redefines the active space. But we are *not* normal and we do *not* do usual things and therefore we want a declaration that not only will `\activate` the space but also will (re)define it as the `_` primitive. So define `\gmobeyspaces` that obeys this requirement.

(This definition is repeated in `gmverb`.)

```
506 \begin{catcode}\_ \active
\gmobeyspaces 507 \gdef\gmobeyspaces{\catcode\_ \active\let\_ \_}
508 \end{catcode}
```

While typesetting poetry, I was surprised that sth. didn't work. The reason was that original `\obeylines` does `\let` not `\def`, so I give the latter possibility.

```
509 \bgroup\catcode'\^M\active% the comment signs here are crucial.
510 \firstofone{\egroup%
\defobeylines 511 \def\defobeylines{\catcode'\^M=13\_ \def\^M{\par}}}%}
```

Another thing I dislike in L^AT_EX yet is doing special things for `\...skip`'s, 'cause I like the Knuthian simplicity. So I sort of restore Knuthian meanings:

```
\deksmallskip 512 \def\deksmallskip{\vskip\smallskipamount}
\undeksmallskip 513 \def\undeksmallskip{\vskip-\smallskipamount}
\dekmedskip 514 \def\dekmedskip{\vskip\medskipamount}
\dekbigskip 515 \def\dekbigskip{\vskip\bigskipamount}

\hfillneg 516 \def\hfillneg{\hskip\_opt\_plus\_1fill\relax}
```

In some `\if(cat?)` test I needed to look only at the first token of a tokens' string (first letter of a word usually) and to drop the rest of it. So I define a macro that expands to the first token (or `{\text}`) of its argument.

```
\@firstofmany 517 \long\def\@firstofmany#1#2\@nil{#1}
```

A mark for the **TODO!**s:

```
\TODO 518 \newcommand*\TODO}[1][]{\%
519 \sffamily\bfseries\huge\TODO!\if\relax#1\relax\else\space\fi#1}}
```

I like twocolumn tables of contents. First I tried to provide them by writing `\begin{multicols}{2}` and `\end{multicols}` outto the .toc file but it worked wrong in some cases. So I redefine the internal L^AT_EX macro instead.

```
\twocoltoc 520 \newcommand*\twocoltoc{%
521 \RequirePackage{multicol}%
\starttoc 522 \def\starttoc##1{%
523 \begin{multicols}{2}\makeatletter\@input{\jobname_##1}%
524 \if@files\@xa\newwrite\csname tf##1\endcsname
525 \immediate\openout\csname tf##1\endcsname\jobname_##1^
\relax
526 \fi
527 \@nobreakfalse\end{multicols}}
528 \@onlypreamble\twocoltoc
```

The macro given below is taken from the multicol package (where its name is `\enough@room`). I put it in this package since I needed it in two totally different works.

```
\enoughpage 529 \newcommand\enoughpage[1]{%
530 \par
531 \dimen0=\pagegoal
532 \advance\dimen0 by-\pagetotal
533 \ifdim\dimen0<#1\relax\newpage\fi}
```

Two shorthands for debugging:

```
\tOnLine 534 \newcommand*\tOnLine{\typeout{\on@line}}
\OnAtLine 535 \let\OnAtLine\on@line
```

An equality sign properly spaced:

```
\equals 536 \newcommand*\equals{${}={}$}
```

And for the L^AT_EX's pseudo-code statements:

```
\eequals 537 \newcommand*\eequals{${}=={}}$}
```

The job name without extension.

```
538 \def\gm@jobn#1.#2\@nil{#1}
\jobnamewoe 539 \def\jobnamewoe{\@xa\gm@jobn\jobname.\@nil}% We add the dot to be sure
there is one although I'm not sure whether you can TEX a file that has no
extension.
```

While typesetting a UTF-8 ls-R result I found a difficulty that follows: UTF-8 encoding is handled by the inputenc package. It's O.K. so far. The UTF-8 sequences are managed using active chars. That's O.K. so far. While writing such sequences to a file, the active chars expand. You feel the blues? When the result of expansion is read again, it sometimes is again an active char, but now it doesn't star a correct UTF-8 sequence.

Because of that I wanted to 'freeze' the active chars so that they would be `\writen` to a file unexpanded. A very brutal operation is done: we look at all 256 chars' catcodes and if we find an active one, we `\let` it `\relax`. As the macro does lots and lots of assignments, it shouldn't be used in `\edefs`.

```

\freeze@actives 540 \def\freeze@actives{%
541   \count\z@\z@
542   \@whilenum\count\z@<\@ccclvi\do{%
543     \ifnum\catcode\count\z@=\active
544       \uccode'\~= \count\z@
545       \uppercase{\let~\relax}%
546     \fi
547     \advance\count\z@\@ne}}

```

A macro that typesets all 256 chars of given font. It makes use of \@whilenum.

```

\ShowFont 548 \newcommand*\ShowFont[1][6]{%
549   \begin{multicols}{#1}[The\currentfont(the\fontencoding\encoding):]
550   \parindent\z@
551   \count\z@\m@ne
552   \@whilenum\count\z@<\@ccclv\do{
553     \advance\count\z@\@ne
554     \the\count\z@:\char\count\z@\par}
555   \end{multicols}}

```

A couple of macros for typesetting liturgic texts such as psalmody of Liturgia Horarum. I wrap them into a declaration since they'll be needed not every time.

```

\liturgiques 556 \newcommand*\liturgiques[1][red]{% Requires the color package.
557   \gmu@RP{color}%
558   \newcommand*\czerwo{\small\color{#1}}% environment
559   \newcommand{\czer}[1]{\leavevmode{\czerwo##1}}% we leave vmode because
        if we don't, then verse's \everypar would be executed in a group and thus
        its effect lost.
560   \def\*{\czer{${*}$}}
561   \def\+{\czer{${\dag}$}}
562   \newcommand*\nieczer[1]{\textcolor{black}{##1}}

```

A command that issues a message urging to load a package if it has not been loaded.

```

563 \let\gmu@RP\RequirePackage
564 \AtBeginDocument{%
\gmu@RP 565   \renewcommand*\gmu@RP[2][ ]{%
566     \ifpackageloaded{#2}{-}{%
567       \typeout{^^J!Package'#2'not loaded!!!(\on@line)^^J}}}}

```

It's very strange to me but it seems that `c` is not defined in the basic math packages. It is missing at least in the *Symbols* book.

```

\continuum 568 \providecommand*\continuum{\gmu@RP{eufrak}\mathfrak{c}}

```

And this macro I saw in the *ltugproc* document class nad I liked it.

```

569 \providecommand*\acro[1]{\scshape\lowercase{#1}}
570 \newcommand*\IMO{\acro{IMO}}
571 \newcommand*\AKA{\acro{AKA}}

```

Probably the only use of it is loading `gmdocc.cls` 'as second class'. This command takes first argument optional, options of the class, and second mandatory, the class name. I use it in an article about `gmdoc`.

```

572 \def\secondclass{%
573   \newif\ifSecondClass

```

```
574 \SecondClasstrue
575 \@fileswithoptions\@clsextension}% [outeroff,gmeometric]{gmdocc} it's load-
    ing gmdocc.cls with all the bells and whistles except the error message.
576 \endinput
```

e. The gmiflink Package¹

Written by Grzegorz ‘Natrór’ Murzynowski,
natror at o2 dot pl

©2005, 2006 by Grzegorz ‘Natrór’ Murzynowski.

This program is subject to the L^AT_EX Project Public License.

See <http://www.ctan.org/tex-archive/help/Catalogue/licenses.lppl.html> for the details of that license.

LPPL status: "author-maintained".

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{gmiflink}
3 [2006/08/16_v0.97_Conditionally_hyperlinking_package_(GM)]
```

Introduction, usage

This package protects you against an error when a link is dangling and typesets some plain text instead of a hyperlink then. It is intended for use with the `hyperref` package. Needs *two* L^AT_EX runs.

I used it for typesetting the names of the objects in a documentation of a computer program. If the object had been defined a `\hyperlink` to its definition was made, otherwise a plain object’s name was typeset. I also use this package in automatic making of hyperlinking indexes.

The package provides the macros `\gmiflink`, `\gmifref` and `\gmhypertarget` for conditional making of hyperlinks in your document.

`\gmhypertarget` `\gmhypertarget[⟨name⟩]{⟨text⟩}` makes a `\hypertarget{⟨@name⟩}{⟨text⟩}` and a `\label{⟨@name⟩}`.

`\gmiflink` `\gmiflink[⟨name⟩]{⟨text⟩}` makes a `\hyperlink{⟨@name⟩}{⟨text⟩}` to a proper `\hypertarget` if the corresponding *label* exists, otherwise it typesets `⟨text⟩`.

`\gmifref` `\gmifref[⟨name⟩]{⟨text⟩}` makes a (hyper-) `\ref{⟨@name⟩}` to the given label if the label exists, otherwise it typesets `⟨text⟩`.

The `⟨@name⟩` argument is just `⟨name⟩` if the `⟨name⟩` is given, otherwise it’s `⟨text⟩` in all three macros.

For the example(s) of use, examine the `gmiflink.sty` file, lines 45–58.

The remarks about installation and compiling of the documentation are analogous to those in the chapter `gmdoc.sty` and therefore omitted.

Contents of the gmiflink.zip archive

The distribution of the `gmiflink` package consists of the following four files.

gmiflink.sty
README

¹ This file has version number v0.97 dated 2006/08/16.

The Code

```

4 \@ifpackageloaded{hyperref}{}{\message{^^J^^Jgmiflink package:
5   There's no use of me without hyperref package, I end my
      input.^^J}\endinput}
6 \providecommand\empty{}
   A new counter, just in case
7 \newcounter{GMhlabel}
8 \setcounter{GMhlabel}{0}

```

The macro given below creates both `\hypertarget` and `\hyperlabel`, so that you may reference both ways: via `\hyperlink` and via `\ref`. Its pattern is the `\label` macro, see L^AT_EX Source2e, file x, line 32.

But we don't want to gobble spaces before and after. First argument will be a name of the `\hypertarget`, by default the same as typeset text, i.e., argument #2.

```

\gmhypertarget 9 \DeclareRobustCommand*\gmhypertarget{%
10   \@ifnextchar{[]{\gm@hypertarget}{\@dblarg{\gm@hypertarget}}}
11 \def\gm@hypertarget[#1]#2{% If argument #1 = \empty, then we'll use #2, i.e., the
      same as name of hypertarget.
12   \refstepcounter{GMhlabel}% we \label{\gmht@firstpar}
13   \hypertarget{#1}{#2}%
14   \protected@write\@auxout{}{%
15     \string\newlabel{#1}{#2}{\thepage}{\relax}{GMhlabel.\arabic{~
      GMhlabel}}{}}}%
16 }% end of \gm@hypertarget.

```

We define a macro such that if the target exists, it makes `\ref`, else it typesets ordinary text.

```

\gmifref 17 \DeclareRobustCommand*\gmifref{\@ifnextchar{[]{\gm@ifref}{% ]
18   \@dblarg{\gm@ifref}}}
19 \def\gm@ifref[#1]#2{%
20   \expandafter\ifx\csname_r#1\endcsname\relax\relax%
21   #2\else\ref{#1}\fi%
22 }% end of \gm@ifref

```

```

\gmiflink 23 \DeclareRobustCommand*\gmiflink{\@ifnextchar{[]{\gm@iflink}{%
24   \@dblarg{\gm@iflink}}}
25 \def\gm@iflink[#1]#2{%
26   \expandafter\ifx\csname_r#1\endcsname\relax\relax%
27   #2\else\hyperlink{#1}{#2}\fi%
28 }% end of \gm@iflink

```

It's robust because when just `\newcommand*`ed, use of `\gmiflink` in an indexing macro resulted in errors: `\@ifnextchar` has to be `\noexpanded` in `\edefs`.

```
29 \endinput
```

The old version — all three were this way primarily.

```

\newcommand*\gmiflink[2][\empty]{\{%
  \def\gmht@test{\empty}\def\gmht@firstpar{#1}%
  \ifx\gmht@test\gmht@firstpar\def\gmht@firstpar{#2}\fi%
  \expandafter\ifx\csname r@\gmht@firstpar\endcsname\relax\relax%
  #2\else\hyperlink{\gmht@firstpar}{#2}\fi%
}}

```


f. The gmverb Package¹

April 30, 2007

This is (a documentation of) file `gmverb.sty`, intended to be used with L^AT_EX 2_ε as a package for a slight redefinition of the `\verb` macro and `verbatim` environment and for short verb marking such as `|\mymacro|`.

Written by Natror (Grzegorz Murzynowski),
natror at o2 dot pl

©2005, 2006 by Natror (Grzegorz Murzynowski).

This program is subject to the L^AT_EX Project Public License.

See <http://www.ctan.org/tex-archive/help/Catalogue/licenses.lppl.html> for the details of that license.

LPPL status: "author-maintained".

Many thanks to my T_EX Guru Marcin Woliński for his T_EXnical support.

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{gmverb}
3 [2007/04/24_v0.82_After_shortvrb_(FM)_but_my_way_(GM)]
```

Intro, Usage

This package redefines the `\verb` command and the `verbatim` environment so that the verbatim text can break into lines, with % (or another character chosen to be the comment char) as a ‘hyphen’. Moreover, it allows the user to define her own `verbatim`-like environments provided their contents would be not *horribly* long (as long as a macro’s argument may be at most).

This package also allows the user to declare a chosen char(s) as a ‘short verb’ e.g., to write `|\a\verbatim\example|` instead of `\verb|\a\verbatim\example|`.

The `gmverb` package redefines the `\verb` command and the `verbatim` environment in such a way that `_`, `{` and `\` are breakable, the first with no ‘hyphen’ and the other two with the comment char as a hyphen. I.e. `{\subsequent text}` breaks into `{%`

`\subsequent text}` and `\text}\mymacro` breaks into `\text}%`

`\mymacro`.

`\fixbslash` (If you don’t like linebreaking at backslash, there’s the `\fixbslash` declaration (observing the common scoping rules, hence OCSR) and an analogous declaration for the left brace: `\fixlbrace`.)

`\VerbHyphen` The default ‘hyphen’ is % since it’s the default comment char. If you wish another char to appear at the linebreak, use the `\VerbHyphen` declaration that takes `\langle char \rangle` as the only argument. This declaration is always global.

`\verbeolOK` Another difference is the `\verbeolOK` declaration (OCSR). Within its scope, `\verb`

¹ This file has version number v0.82 dated 2007/04/24.

allows an end of a line in its argument and typesets it just as a space.

As in the standard version(s), the plain `\verb` typesets the spaces blank and `\verb*` makes them visible.

`\MakeShortVerb`

Moreover, `gmverb` provides the `\MakeShortVerb` macro that takes a one-char control sequence as the only argument and turns the char used into a short verbatim delimiter, e.g., after `\MakeShortVerb*\|` (as you guess, the declaration has its starred version, which is for visible spaces, and the non-starred for the spaces blank) you may type `|\mymacro|` to get `\mymacro` instead of typing `\verb+\mymacro+`. Because the char used in this example is my favourite and used just this way by DEK in the The \TeX book's format, `gmverb` provides a macro `\dekclubs` as a shorthand for `\MakeShortVerb*\|`.

`\dekclubs`

Be careful because such active chars may interfere with other things, e.g., the `|` with the vertical marker in tables and with the `tikz` package. If this happens, you can declare e.g., `\DeleteShortVerb\|` and the previous meaning of the char used shall be restored.

`\DeleteShortVerb`

One more difference between `gmverb` and `shortverb` is that the chars `\active`ated by `\MakeShortVerb` in the math mode behave as if they were ‘other’, so you may type e.g., `$|` to get `|` and `+ \active`ated this way is in the math mode typeset properly etc.

`\OldMakeShortVerb`

However, if you don't like such a conditional behaviour, you may use `\OldMakeShortVerb` instead, what I do when I like to display short verbatims in `displaymath`.

`\dekclubs`

`\dekclubs*`

There's one more declaration provided by `gmverb`: `\dekclubs`, which is a shorthand for `\MakeShortVerb*\|` and `\dekclubs*` for `\OldMakeShortVerb*\|`.

So that, after the latter declaration, you can write

```
\[|verbatim stuff|\]
```

instead of

```
\[\hbox{|the stuff|\}
```

to get a displayed shortverb.

Both versions of `\dekclubs` OCSR.

As many good packages, this also does not support any options.

The remarks about installation and compiling of the documentation are analogous to those in the chapter `gmdoc.sty` and therefore omitted.

Contents of the `gmverb.zip` Archive

The distribution of the `gmverb` package consists of the following four files.

```
gmverb.sty
README
gmverbDoc.tex
gmverbDoc.pdf
```

The Code

Preliminaries

```
4 \RequirePackage{gmutils}[2007/04/24]
```

For `\firstofone`, `\afterfi`, `\gmokeys`paces, `\@ifnextcat` and `\noexpand's` and `\expandafter's` shorthands `\@nx` and `\@xa` resp.

```
5 \bgroup
```

```
6 \@makeother\%
```

```
7 \firstofone{\egroup
```

```
\twelvepercent 8 \def\twelvepercent{%}}
```

Someone may want to use another char for comment, but we assume here ‘orthodoxy’. Other assumptions in `gmdoc` are made. The ‘knowledge’ what char is the comment char is used to put proper ‘hyphen’ when a `verbatim` line is broken.

```
\verbhyphen 9 \let\verbhyphen\twelvepercent
```

Provide a declaration for easy changing it. Its argument should be of `\langle char \rangle` form (of course, a `\langle char \rangle12` is also allowed).

```
\VerbHyphen 10 \def\VerbHyphen#1{%
11   {\escapechar\m@ne
12    \@xa\gdef\@xa\verbhyphen\@xa{\string#1}}}
```

As you see, it’s always global.

The Breakables

Let’s define a `\discretionary` left brace such that if it breaks, it turns `{%` at the end of line. We’ll use it in almost Knuthian `\ttverbatim`—it’s part of this ‘almost’.

```
13 \bgroup\catcode'\<=1\@makeother\{\catcode'\>=2%
14 \firstofone<\egroup
\breaklbrace 15 \def\breaklbrace<\discretionary<\verbhyphen><><{}>%
\twelvebrace 16 \def\twelvebrace<{}>%
17 >% of \firstofone
18 \bgroup\catcode'\<=1\catcode'\{=\active\catcode'\>=2
19 \firstofone<\egroup
20 \def\dobreaklbrace<\catcode'\{=\active\def{<\breaklbrace>>%
21 >% end of \firstofone.
```

The `\bslash` macro defined below I use also in more ‘normal’ \TeX ing, e.g., to `\typeout` some `\outer` macro’s name.

```
22 {\catcode'\!=0\@makeother\\%
\bslash 23 !gdef!bslash{\}%
\breakbslash 24 !gdef!breakbslash{!discretionary{!verbhyphen}{\}{\}}%
25 }
```

Sometimes linebreaking at a backslash may be unwelcome. The basic case, when the first CS in a `verbatim` breaks at the lineend leaving there `%`, is covered by line 182. For the others let’s give the user a countercrank:

```
\fixbslash 26 \newcommand*\fixbslash{\let\breakbslash=\bslash}% to use due to the common
scoping rules. But for the special case of a backslash opening a verbatim scope,
we deal specially in the line 182.
```

Analogously, let’s provide a possibility of ‘fixing’ the left brace:

```
\fixlbrace 27 \newcommand*\fixlbrace{\let\breaklbrace=\twelvebrace}
28 {\catcode'\!=0%
29 !catcode'\!=!active
30 !gdef!dobreakbslash{!catcode'\!=!active\def{\!breakbslash}}%
31 }
```

The macros defined below, `\visiblebreakspaces` and `\twelveclub` we’ll use in the almost Knuthian macro making `verbatim`. This ‘almost’ makes a difference.

```

32 \bgroup\catcode'\_ =12\_%
33 \firstofone{\egroup%
\twelvespace 34 \def\twelvespace{\_}%
35 \def\breakabletwelvespace{\discretionary{\_}{\_}{\_}}
36 \bgroup\obeyspaces% it's just re\catcode'ing.
37 \firstofone{\egroup%
\activespace 38 \newcommand*\activespace{\_}%
39 \newcommand*\dobreakvisiblespace{\let\_ =\breakabletwelvespace\obeyspaces}%
% \defing it caused a stack overflow disaster with gmdoc.
40 \newcommand*\dobreakblankspace{\let\_ =\space\obeyspaces}%
41 }
42 \bgroup\@makeother\|
\twelveclub 43 \firstofone{\egroup\def\twelveclub{\_}}

```

Almost-Knuthian \ttverbatim

\ttverbatim comes from The TeXbook too, but I add into it a L^AT_EX macro changing the \catcodes and make spaces visible and breakable and left braces too.

```

\ttverbatim 44 \newcommand*\ttverbatim{%
45   \let\do=\do@noligs\_ \verbatim@nolig@list
46   \let\do=\@makeother\_ \dospecials
47   \dobreaklbrace\dobreakbslash
48   \dobreakspace
49   \tt
50   \ttverbatim@hook}

```

While typesetting stuff in the QX fontencoding I noticed there were no spaces in verbatims. That was because the QX encoding doesn't have any reasonable char at position 32. So we provide a hook in the very core of the verbatim making macros to set proper fontencoding for instance.

```

51 \@emptyify\ttverbatim@hook
11
\VerbT 52 \def\VerbT1{\def\ttverbatim@hook{\fontencoding{T1}\selectfont}}
We wish the visible spaces to be the default.
53 \let\dobreakspace=\dobreakvisiblespace

```

The Core: From shortvrb

The below is copied verbatim ;-) from doc.pdf and then is added my slight changes.

```

\MakeShortVerb* 54 \def\MakeShortVerb{%
\MakeShortVerb 55   \@ifstar
56   {\def\@shortvrbdef{\verb*}\@MakeShortVerb}%
57   {\def\@shortvrbdef{\verb}\@MakeShortVerb}}
\@MakeShortVerb 58 \def\@MakeShortVerb#1{%
59   \xa\ifx\csname\_cc\string#1\endcsname\relax
60   \@shortvrbinfo{Made\_}{#1}\@shortvrbdef
61   \add@special{#1}%
62   \AddtoPrivateOthers#1% a macro to be really defined in gmdoc.
63   \xa

```

```

64 \xdef\csname_cc\string#1\endcsname{\the\catcode'#1}%
65 \begingroup
66 \catcode'\~\active_\lccode'\~'#1%
67 \lowercase{%
68   \global\@xa\let
69   \csname_ac\string#1\endcsname~%
70   \@xa\gdef\@xa~\@xa{%
71     \@xa\ifmmode\@xa\string\@xa~%
72     \@xa\else\@xa\afterfi{\@shortvrbdef~}\fi}}% This terrible number of
        \expandafters is to make the shortverb char just other in the math
        mode (my addition).
73 \endgroup
74 \global\catcode'#1\active
75 \else
76 \@shortvrbinfo\@empty{#1_already}{\@empty\verb(*)}%
77 \fi}

\DeleteShortVerb 78 \def\DeleteShortVerb#1{%
79   \@xa\ifx\csname_cc\string#1\endcsname\relax
80   \@shortvrbinfo\@empty{#1_not}{\@empty\verb(*)}%
81   \else
82   \@shortvrbinfo{Deleted_}{#1_as}{\@empty\verb(*)}%
83   \rem@special{#1}%
84   \global\catcode'#1\csname_cc\string#1\endcsname
85   \global_\@xa\let_\csname_cc\string#1\endcsname_\relax
86   \ifnum\catcode'#1=\active
87   \begingroup
88   \catcode'\~\active_\lccode'\~'#1%
89   \lowercase{%
90     \global\@xa\let\@xa~%
91     \csname_ac\string#1\endcsname}%
92   \endgroup_\fi_\fi}

My little addition

93 \@ifpackageloaded{gmdoc}{%
94   \def\gmv@packname{gmdoc}}{%
95   \def\gmv@packname{gmverb}}

\@shortvrbinfo 96 \def\@shortvrbinfo#1#2#3{%
97   \PackageInfo{\gmv@packname}{%
98     ^^J\@empty_#1\@xa\@gobble\string#2_a_short_reference
99     for_\@xa\string#3}}

\add@special 100 \def\add@special#1{%
101   \rem@special{#1}%
102   \@xa\gdef\@xa\dospecials\@xa
103   {\dospecials_\do_#1}%
104   \@xa\gdef\@xa\@sanitize\@xa
105   {\@sanitize_\@makeother_#1}}

```

For the commentary on the below macro see the doc package's documentation. Here let's only say it's just amazing: so tricky and wicked use of `\do`. The internal macro `\rem@special` defines `\do` to expand to nothing if the `\do`'s argument is the one to be removed and to unexpandable CSs `\do` and `\do's argument` otherwise. With `\do`

defined this way the entire list is just globally expanded itself. Analogous hack is done to the `\@sanitize` list.

```
\rem@special 106 \def\rem@special#1{%
107   \def\do##1{%
108     \ifnum'#1='##1_\else_\@nx\do\@nx##1\fi}%
109   \xdef\dospecials{\dospecials}%
110   \begingroup
111   \def\@makeother##1{%
112     \ifnum'#1='##1_\else_\@nx\@makeother\@nx##1\fi}%
113   \xdef\@sanitize{\@sanitize}%
114   \endgroup}
```

And now the definition of `verbatim` itself. As you'll see (I hope), the internal macros of it look for the name of the current environment (i.e., `\@currenvir`'s meaning) to set their expectation of the environment's `\end` properly. This is done to allow the user to define his/her own environments with `\verbatim` inside them. I.e., as with the `verbatim` package, you may write `\verbatim` in the begdef of your environment and then necessarily `\endverbatim` in its enddef. Of course (or maybe surprisingly), the commands written in the begdef after `\verbatim` will also be executed at `\begin{environment}`.

```
verbatim 115 \def\verbatim{\@beginparpenalty_\predisplaypenalty_\@verbatim
\verbatim 116   \frenchspacing_\gmoveyspaces_\@xverbatim}% in the LATEX version there's %^
          \@vobeyspaces instead of \gmoveyspaces.
verbatim* 117 \@namedef{verbatim*}{\@beginparpenalty_\predisplaypenalty_\@verbatim
118   \@sxverbatim}
\endverbatim 119 \def\endverbatim{\@@par
120   \ifdim\lastskip>\z@
121     \@tempskipa\lastskip_\vskip_\- \lastskip
122     \advance\@tempskipa\parskip_\advance\@tempskipa_\- \@outerparskip
123     \vskip\@tempskipa
124     \fi
125     \addvspace\@topsepadd
126     \@endparenv}
127 \n@melet{endverbatim*}{endverbatim}
128 \begingroup_\catcode_'\!=0_%
129 \catcode_'\=[_1_\catcode_'\]=2_%
130 \catcode_'\{=\active
131 \@makeother\}%
132 \catcode_'\=\active%
\@xverbatim 133 !gdef!\@xverbatim[%
134   !edef!verbatim@edef[%
135     !def!noexpand!verbatim@end%
136     #####1!noexpand\end!noexpand{!\@currenvir}[%
137     #####1!noexpand!end[!\@currenvir]]]%
138   !verbatim@edef
139   !verbatim@end]%
140 !endgroup
\@sxverbatim 141 \let\@sxverbatim=\@xverbatim
```

F. Mittelbach says the below is copied almost verbatim from L^AT_EX source, modulo `\check@percent`.

\@verbatim 142 \def\@verbatim{%

Originally here was just \trivlist_\item[], but it worked badly in my document(s), so let's take just highlights of if.

143 \parsep\parskip

From \@trivlist:

```
144 \if@noskipsec_\leavevmode_\fi
145 \@topsepadd_\topsep
146 \ifvmode
147 \advance\@topsepadd_\partopsep
148 \else
149 \unskip_\par
150 \fi
151 \@topsep_\@topsepadd
152 \advance\@topsep_\parskip
153 \@outerparskip_\parskip
```

(End of \trivlistlist and \@trivlist highlights.)

```
154 \@@par\addvspace\@topsep
155 \if@minipage\else\vskip\parskip\fi
156 \leftmargin\parindent% please notify me if it's a bad idea.
157 \advance\@totalleftmargin\leftmargin
158 \raggedright
159 \leftskip\@totalleftmargin% so many assignments to preserve the list think-
    ing for possible future changes. However, we may be sure no internal list
    shall use \@totalleftmargin as far as no inner environments are possible
    in verbatim(*).
160 \@@par% most probably redundant.
161 \@tempwafalse
162 \def\par{% but I don't want the terribly ugly empty lines when a blank line is met.
    Let's make them gmdoc-like i.e., let a vertical space be added as in between
    stanzas of poetry. Originally \if@tempswa\hbox{}\fi, in my version will
    be
163 \ifvmode\if@tempswa\addvspace\stanzaskip\@tempwafalse\fi\fi
164 \@@par
165 \penalty\interlinepenalty_\check@percent}%
166 \everypar{\@tempwattrue\hangindent\verbatimhangindent\hangafter\@ne}%
    since several chars are breakable, there's a possibility of breaking some lines.
    We wish them to be hanging indented.
167 \obeylines
168 \ttverbatim}
```

```
169 \@ifundefined{stanzaskip}{\newlength\stanzaskip}{\}
170 \stanzaskip=\medskipamount
```

```
\verbatimhangindent 171 \newlength\verbatimhangindent
172 \verbatimhangindent=3em
173 \providecommand*\check@percent{}
```

In the gmdoc package shall it be defined to check if the next line begins with a comment char.

Similarly, the next macro shall in `gmdoc` be defined to update a list useful to that package. For now let it just gobble its argument.

```
174 \providecommand*\AddtoPrivateOthers[1]{}
```

Both of the above are `\provided` to allow the user to load `gmverb` after `gmdoc` (which would be redundant since `gmdoc` loads this package on its own, but anyway should be harmless).

Let's define the 'short' verbatim command.

```
\verb* 175 \def\verb{\relax\ifmmode\hbox\else\leavevmode\null\fi
\verb 176   \bgroup
177   \ttverbatim
178   \gm@verb@eol
179   \@ifstar{\@sverb@chbsl}{\gmobeyspaces\frenchspacing\@sverb@chbsl}}% in
      the LATEX version there's \vobeyspaces instead of \gmobeyspaces.
180 \def\@sverb@chbsl#1{\@sverb#1\check@bslash}
181 \def\@def@breakbslash{\breakbslash}% because \ is \defined as \breakbslash
      not \let.
```

For the special case of a backslash opening a (short) verbatim, in which it shouldn't be breakable, we define the checking macro.

```
\check@bslash 182 \def\check@bslash{\@ifnextchar{\@def@breakbslash}{\bslash\@gobble}{}}
183 \let\verb@balance@group\@empty
\verb@egroup 184 \def\verb@egroup{\global\let\verb@balance@group\@empty\egroup}
\gm@verb@eol 185 \let\gm@verb@eol\verb@eol@error
```

The latter is a L^AT_EX 2_ε kernel macro that `\activeates` line end and defines it to close the verb group and to issue an error message. We use a separate CS 'cause we are not quite positive to the forbidden line ends idea. (Although the allowed line ends with a forgotten closing shortverb char caused funny disasters at my work a few times.) Another reason is that `gmdoc` wishes to redefine it for its own queer purpose.

However, let's leave my former 'permissive' definition under the `\verb@eol` name.

```
186 \begingroup
187 \obeylines\obeyspaces%
\verb@eolOK 188 \gdef\verb@eolOK{\obeylines%
189 \def~M{\check@percent}%
190 }%
191 \endgroup
```

The `\check@percent` macro here is `\provided` to be `\@empty` but in `gmdoc` employed shall it be.

Let us leave (give?) a user freedom of choice:

```
\verbeolOK 192 \def\verbeolOK{\let\gm@verb@eol\verb@eolOK}
```

And back to the main matter,

```
193 \def\@sverb#1{%
194   \catcode'\#1\active\lccode'\~'\#1%
195   \gdef\verb@balance@group{\verb@egroup
196     \@latex@error{Illegal use of \bslash\verb\command}\@ehc}%
197   \aftergroup\verb@balance@group
198   \lowercase{\let~\verb@egroup}}
```



```

\verbatim@nolig@list 199 \def\verbatim@nolig@list{\do\‘\do\<\do\>\do\,\do\’\do\~}
\do@noligs 200 \def\do@noligs#1{%
201   \catcode‘#1\active
202   \begingroup
203   \lccode‘\~‘#1\relax
204   \lowercase{\endgroup\def~{\leavevmode\kern\z@\char‘#1}}

```

And finally, what I thought to be so smart and clever, now is just one of many possible uses of a general almost Rainer Schöpf’s macro:

```

\dekclubs 205 \def\dekclubs{\@ifstar{\OldMakeShortVerb*}{\MakeShortVerb*}{}}

```

But even if a shortverb is unconditional, the spaces in the math mode are not printed. So,

```

\edverbs 206 \newcommand*\edverbs{%
207   \let\gmv@dismath\[%
208   \let\gmv@edismath\]%
209   \def\[%
210     \@ifnextac\gmv@disverb\gmv@dismath}%
211   \let\edverbs\relax}%
212 \def\gmv@disverb{%
213   \gmv@dismath
214   \hbox\bgroup\def\]{\egroup\gmv@edismath}}

```

doc- And shortvrb-Compatibility

One of minor errors while T_EXing doc.dtx was caused by my understanding of a ‘shortverb’ char: at my settings, in the math mode an active ‘shortverb’ char expands to itself’s ‘other’ version thanks to \string. doc/shortvrb’s concept is different, there a ‘shortverb’ char should work as usual in the math mode. So let it may be as they wish:

```

\old@MakeShortVerb 215 \def\old@MakeShortVerb#1{%
216   \@xa\ifx\csname_cc\string#1\endcsname\relax
217   \@shortvrbinfo{Made_}{#1}\@shortvrbdef
218   \add@special{#1}%
219   \AddtoPrivateOthers#1% a macro to be really defined in gmdoc.
220   \@xa
221   \xdef\csname_cc\string#1\endcsname{\the\catcode‘#1}%
222   \begingroup
223   \catcode‘\~\active_\lccode‘\~‘#1%
224   \lowercase{%
225     \global\@xa\let\csname_ac\string#1\endcsname~%
226     \@xa\gdef\@xa~\@xa{%
227       \@shortvrbdef~}}%
228   \endgroup
229   \global\catcode‘#1\active
230   \else
231   \@shortvrbinfo\@empty{#1_already}{\@empty\verb(*)}%
232   \fi}
\OldMakeShortVerb 233 \def\OldMakeShortVerb{\begingroup
234   \let\@MakeShortVerb=\old@MakeShortVerb
235   \@ifstar{\eg@MakeShortVerbStar}{\eg@MakeShortVerb}}

```

```

\eg@MakeShortVerbStar 236 \def\eg@MakeShortVerbStar#1{\MakeShortVerb*#1\endgroup}
\eg@MakeShortVerb      237 \def\eg@MakeShortVerb#1{\MakeShortVerb#1\endgroup}
                        238 \endinput% for the Tradition.

```

Change History

gmdoc v0.96

General:

Checksum 2395, [4](#)

gmdoc v0.98d

General:

An entry to show the change history works: watch and admire. Some sixty `\changes` entries irrelevant for the users-other-than-myself are hidden due to the trick described on p. [64](#).

gmdoc v0.99a

General:

Checksum 4479, [4](#)

gmdoc v0.99b

General:

Thanks to the `\edverbs` declaration in the class, displayed shortverbs simplified; Emacs mode changed to doctex. Author's true name more exposed, [88](#)

gmdoc v0.99c

`\^^M`:

a bug fix: redefinition of it left solely to `\QueerEOL`, [38](#)

General:

A bug fixed in `\DocInput` and all `\expandafters` changed to `\@xa` and `\noexpands` to `\@nx`, [88](#)

The T_EX-related logos now are declared with `\DeclareLogo` provided in `gmutils`, [88](#)

`\DocInput`:

added ensuring the code delimiter to be the same at the end as at the beginning, [27](#)

gmdoc v0.99d

General:

`\@namelet` renamed to `\n@melet` to solve a conflict with the `beamer` class (in `gmutils` at first), [88](#)

`\afterfi` & `pals` made two-argument, [88](#)

`\FileInfo`:

added, [77](#)

gmdoc v0.99e

General:

a bug fixed in `\DocInput` and `\IndexInput`, [88](#)

Checksum 4574, [4](#)

gmdocc v0.74

`\edverbs`:

used to simplify displaying shortverbs, [92](#)

gmdocc v0.75

General:

Checksum 130, [89](#)

gmdocDoc vNo

General:

Checksum 41, [93](#)

gmutils v0.74

General:

Added macros to make sectioning commands of `mwcls` and standard classes compatible. Now my sectionings allow two optionals in both worlds and with `mwcls` if there's only one optional, it's the title to toc and running head not just to the latter, [117](#)

The catcodes of `\begin` and `\end` argument(s) don't have to agree strictly anymore: an environment is properly closed if the `\begin`'s and `\end`'s arguments result in the same `\csname`, [99](#)

gmutils v0.75

`\@ifnextac`:

added, [97](#)

`\@ifnextcat`:

`\let` for #1 changed to `\def` to allow things like `\noexpand~`, [96](#)

gmutils v0.76

General:

A 'fixing' of `\dots` was rolled back since it came out they were O.K. and that was the QX encoding that prints them very tight, [117](#)

`\freeze@actives`:

added, [115](#)

gmutils v0.77

General:

`\afterfi` & `pals` made two-argument as the Marcin Woliński's analogoi are. At this occasion some redundant macros of that family are deleted, [117](#)

gmutils v0.78

General:	added, 114
<code>\@namelet</code> renamed to <code>\n@melet</code> to	<code>gmverb v0.79</code>
solve a conflict with the <code>beamer</code> class.	<code>\edverbs:</code>
The package contents regrouped, 117	added, 129
<code>gmutils v0.79</code>	<code>gmverb v0.80</code>
<code>\not@onlypreamble:</code>	<code>\edverbs:</code>
All the actions are done in a group and	debugged, i.e. <code>\hbox</code> added back and
therefore <code>\xdef</code> used instead of <code>\edef</code>	redefinition of <code>\[</code> , 129
because this command has to use <code>\do</code>	<code>\ttverbatim:</code>
(which is contained in the	<code>\ttverbatim@hook</code> added, 124
<code>\@preamblecmds</code> list) and	<code>gmverb v0.81</code>
<code>\not@onlypreamble</code> itself should be	General:
able to be let to <code>\do</code> , 106	<code>\afterfi</code> made two-argument, 130
<code>gmutils v0.80</code>	<code>gmverb v0.82</code>
General:	General:
Checksum 1689, 94	Checksum 663, 121
<code>\hfillneg:</code>	

Index

Numbers written in *italic* refer to the code lines 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. The numbers with no prefix are page numbers. All the numbers are hyperlinks.

- *, a-452, a-562, a-1056,
 a-1706, d-64, d-65,
 d-66, d-68, d-70,
 d-71, d-72, d-76,
 d-505, d-560
- \+, a-1056, a-1593, d-561
- \-, a-1056, d-190, f-199
- \<...>, d-177, *84*
- \@@codeline@wrindex, a-709
- \@@nil, a-627, a-668, a-837,
 a-842, a-1135,
 a-1137, a-1145,
 a-1528, d-182, d-183,
 d-185, d-517, d-538,
 d-539
- \@par, a-125, a-317, a-328,
 a-361, a-890
- \@settexcodehangi, a-50,
 a-50, a-185, a-218
- \@M, a-1139
- \@MakeShortVerb, a-1718,
 f-56, f-57, f-58, f-234
- \@NoEOF, a-1726, a-1728
- \@aalph, a-1307, a-1308
- \@addtoreset, a-1333
- \@aftercodegfalse, a-202,
 a-320, a-370, a-382
- \@aftercodegtrue, a-82,
 a-204, a-222, a-309,
 a-370, a-1071, a-1077
- \@afternarrgfalse, a-82,
 a-204, a-309, a-371,
 a-1071, a-1077
- \@afternarrgtrue, a-118,
 a-371
- \@badend, d-97
- \@beginputhook, *a-98*,
 a-139, a-140
- \@begnamedgroup, d-83,
 d-84, d-90, d-93
- \@car, d-470
- \@cclv, d-552
- \@ccclvi, d-542
- \@charlb, a-1299
- \@charrb, a-1301
- \@checkend, d-94
- \@clsextension, d-575
- \@clubpenalty, a-92
- @codeskipput, a-362, *35*
- \@codeskipputgfalse,
 a-118, a-190, a-310,
 a-362, a-1071,
 a-1078, a-1579
- \@codeskipputgtrue, a-76,
 a-80, a-82, a-192,
 a-202, a-320, a-361,
 a-362, a-377, a-764,
 a-768, a-831, a-834
- \@codespacesblanktrue,
 a-24
- \@codetonarrskip, a-99,
 a-167, a-174, a-299,
 a-308, a-327, a-339,
 a-372, a-392
- \@countalllinestrue, a-10
- \@ctrerr, a-1314
- \@currenvir, a-792, a-805,
 a-806, d-87, d-96,
 f-136, f-137
- \@currenvir*, a-786
- \@currenvline, d-88
- \@currsize, d-102, d-103,
 d-104, d-105, d-106,
 d-107, d-108, d-109,
 d-110, d-111
- \@debugtrue, b-13
- \@def@breakbslash, f-181,
 f-182
- \@defentryze, a-570,
 a-574, a-576, a-714
- \@docinclude, a-1261, a-1262
- \@dsdirgfalse, a-197,
 a-206, a-231, a-259,
 a-295, a-302, a-304,
 a-822
- \@dsdirgtrue, a-120, a-188
- \@emptify, a-147, a-179,
 a-540, a-581, a-637,
 a-677, a-682, a-683,
 a-1023, a-1063,
 a-1065, a-1132,
 a-1306, a-1416,
 a-1476, a-1584,
 a-1585, a-1629,
 a-1633, a-1711,
 a-1712, a-1713, d-26,
 d-27, d-28, f-51
- \@endinputhook, *a-112*,
 a-137, a-138
- \@enumctr, d-407, d-408, d-414
- \@enumdepth, d-403, d-406,
 d-407
- \@fileswfalse, a-1568
- \@fileswithoptions, d-575
- \@firstofmany, a-627,
 a-668, a-1528, d-517
- \@fshdafalse, a-1494, a-1496
- \@fshdatrue, a-1494, a-1495
- \@gif, d-9, d-10, d-12
- \@gmccnochangesfalse, b-16
- \@gmccnochangestrue,
 b-16, b-17
- \@ifQueerEOL, a-144,
 a-145, a-402, a-408,
 a-439, a-1051, a-1189
- \@ifismember, a-479, a-485
- \@ifl@aded, d-268
- \@ifncat, d-46, d-47, d-59
- \@ifnextac, d-60, f-210
- \@ifnextcat, a-461, a-475,
 d-42, d-61
- \@ifnextspace, d-69
- \@ifnotmw, b-53, d-311,
 d-313, d-369, d-401
- \@ifstarl, a-564, a-567,
 a-584, a-594, a-619,
 a-642, a-649, a-686,
 a-689, a-728, a-741,
 a-1654

<code>\@indexallmacrofalse</code> , a-15	<code>\@relaxen</code> , a-72 , a-342 ,	<code>\abovedisplayskip</code> , a-62
<code>\@indexallmacrotrue</code> ,	a-354 , a-997 , a-1131 ,	<code>\acro</code> , d-569 , d-570 , d-571
a-15 , a-16	a-1160 , a-1289 ,	<code>\activespace</code> , f-38
<code>\@itemdepth</code> , d-419 , d-422 ,	a-1316 , a-1465 ,	<code>\actualchar</code> , 18 , a-441 ,
d-423	a-1466 , a-1618 ,	a-484 , a-529 , a-1097 ,
<code>\@itemitem</code> , d-423 , d-424	a-1622 , a-1698 ,	a-1121 , a-1126 ,
<code>\@latexerr</code> , a-1260 , a-1375	a-1727 , d-30 , d-31 , d-32	a-1244 , a-1373 , a-1704
<code>\@linesnotnumalse</code> , a-5	<code>\@reserveda</code> , d-70 , d-72	<code>\add@special</code> , f-61 , f-100 ,
<code>\@linesnotnumtrue</code> , a-5 , a-6	<code>\@shortvrbbdef</code> , f-56 , f-57 ,	f-218
<code>\@ltxDocIncludefalse</code> ,	f-60 , f-72 , f-217 , f-227	<code>\addto@estoindex</code> , a-573 ,
a-1408	<code>\@shortvrbinf</code> , f-60 , f-76 ,	a-590 , a-597 , a-713 ,
<code>\@ltxDocIncludetrue</code> ,	f-80 , f-82 , f-96 , f-217 ,	a-718 , a-723
a-1408 , a-1412	f-231	<code>\addto@estomarginpar</code> ,
<code>\@makefntext</code> , a-1427	<code>\@starttoc</code> , d-522	a-657 , a-712 , a-717 ,
<code>\@marginparsusedfalse</code> ,	<code>\@sverb@chbsl</code> , f-179 , f-180	a-720
a-17 , a-22	<code>\@toodeep</code> , d-404 , d-420	<code>\addto@macro</code> , d-21 , d-25
<code>\@marginparsusedtrue</code> ,	<code>\@topnewpage</code> , d-357	<code>\addtomacro</code> , a-722 , a-725 ,
a-17 , a-18 , a-19 ,	<code>\@topsep</code> , f-151 , f-152 , f-154	a-775 , a-776 , a-862 , d-25
a-20 , a-21	<code>\@topsepadd</code> , f-125 , f-145 ,	<code>\addtonummacro</code> , d-294
<code>\@newlinegfalse</code> , a-83 ,	f-147 , f-151	<code>\AddtoPrivateOthers</code> , 17 ,
a-172 , a-207 , a-269 ,	<code>\@trimandstore</code> , a-121 ,	a-431 , f-62 , f-174 , f-219
a-279 , a-286	a-166 , a-383 , a-383 ,	<code>\afterfi</code> , a-146 , a-232 ,
<code>\@newlinegtrue</code> , a-83 ,	a-388 , a-390	a-235 , a-271 , a-273 ,
a-119 , a-187	<code>\@trimandstore@hash</code> ,	a-388 , a-430 , a-461 ,
<code>\@nobreakfalse</code> , d-527	a-384 , a-385	a-469 , a-487 , a-488 ,
<code>\@noindexfalse</code> , a-11	<code>\@trimandstore@ne</code> , a-388 ,	a-615 , a-826 , a-843 ,
<code>\@noindextrue</code> , a-11 , a-12	a-390	a-846 , b-45 , d-63 ,
<code>\@cnx</code> , a-102 , d-5 , d-37 , d-51 ,	<code>\@uresetlinecountfalse</code> , a-7	d-73 , d-74 , d-77 ,
d-63 , d-218 , d-225 ,	<code>\@uresetlinecounttrue</code> ,	d-180 , d-205 , f-72
d-243 , d-259 , d-260 ,	a-7 , a-8	<code>\afterfifi</code> , a-246 , a-260 ,
d-263 , d-272 , d-274 ,	<code>\@usgentryze</code> , a-579 ,	a-262 , a-823 , a-824 ,
d-275 , d-442 , d-497 ,	a-587 , a-591 , a-621 ,	a-878 , a-884 , d-78 ,
f-108 , f-112	a-623 , a-719 , a-737 ,	d-229 , d-232
<code>\@oarg</code> , d-197 , d-198 , d-199	a-1658 , a-1663	<code>\afterfififi</code> , d-82
<code>\@oargsq</code> , d-197 , d-199 , d-207	<code>\@whilenum</code> , d-542 , d-552	<code>\afteriffifi</code> , a-241 ,
<code>\@oldmacrocode</code> , a-787 , a-801	<code>\@xa</code> , a-44 , d-4 , d-13 , d-15 ,	a-254 , d-79
<code>\@oldmacrocode@launch</code> ,	d-22 , d-35 , d-59 ,	<code>\afteriffififi</code> , d-81
a-775 , a-777 , a-778	d-95 , d-96 , d-116 ,	<code>\afteriffiffifi</code> , d-80
<code>\@onlypreamble</code> , a-1414 ,	d-165 , d-194 , d-215 ,	<code>\AfterMacrocode</code> , a-1628
a-1684 , a-1686 ,	d-218 , d-240 , d-243 ,	<code>\AKA</code> , d-571
a-1688 , d-528	d-252 , d-256 , d-259 ,	<code>\all@other</code> , d-494 , d-495
<code>\@pageinclindexfalse</code> ,	d-260 , d-380 , d-386 ,	<code>\AlsoImplementation</code> , 18 ,
a-523 , a-541	d-408 , d-424 , d-437 ,	a-1692 , a-1695
<code>\@pageinclindextrue</code> ,	d-495 , d-497 , d-501 ,	<code>\AltMacroFont</code> , a-1713
a-541 , a-760	d-524 , d-539 , f-12 ,	<code>\AmSTeX</code> , 19 , a-1595
<code>\@pageindexfalse</code> , a-13 ,	f-59 , f-63 , f-68 , f-70 ,	<code>\and</code> , a-1459 , a-1468
a-1685	f-71 , f-72 , f-79 , f-85 ,	<code>\arg</code> , d-204 , d-205
<code>\@pageindextrue</code> , a-13 ,	f-90 , f-98 , f-99 , f-102 ,	<code>article</code> , b-10
a-14 , a-544 , a-1687	f-104 , f-216 , f-220 ,	<code>\AtBeginDocument</code> , a-37 ,
<code>\@parg</code> , d-200 , d-201 , d-202	f-225 , f-226	a-71 , a-483 , a-545 ,
<code>\@pargp</code> , d-200 , d-202 , d-208	<code>\@xifncat</code> , d-49 , d-59	a-700 , a-1186 ,
<code>\@parindent</code> , d-410 , d-411 ,	<code>^^A</code> , 7 , a-406	a-1683 , b-32 , d-203 ,
d-413 , d-426 , d-427 ,	<code>^^B</code> , 7 , a-400	d-271 , d-298 , d-564
d-429	<code>^^M</code> , a-427	<code>\AtBegInput</code> , 9 , a-139 ,
<code>\@pkgeextension</code> , d-269	<code>^^M</code> , a-100 , a-186	a-142 , a-148 , a-402 ,
<code>\@preamblecmds</code> , d-264 ,	<code>\aalph</code> , a-1307 , a-1334	a-408 , a-429 , a-436 ,
d-265 , d-273 , d-276		a-1421 , a-1422 , a-1565

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmlink.sty, f=gverb.sty

<code>\AtBegInputOnce</code> , 9, 10, a-149, a-1365, a-1717	<code>\ChangesStart</code> , 15, a-1144	<code>\CodeDefIndex</code> , 13, a-565, a-1699
<code>\AtDIPrologue</code> , 18, a-1024	<code>ChangesStartDate</code> , 15	<code>\CodeDefIndex*</code> , a-565
<code>\AtEndInput</code> , 9, a-137, a-1197, a-1669, a-1678	<code>\Character@Table</code> , a-1636, a-1641	<code>\CodeDefine</code> , 12, a-684
<code>\author</code> , a-1453, c-5	<code>\CharacterTable</code> , a-1634	<code>\CodeDefine*</code> , a-684
<code>\AVerySpecialMacro</code> , a-1724	<code>\check@bslash</code> , f-180, f-182	<code>\CodeDelim</code> , 17, a-41, a-46, a-105, a-1614, a-1615
<code>\begin</code> , d-92, d-93	<code>\check@checksum</code> , a-1197, a-1198	<code>\CodeDelim*</code> , a-781, a-1555
<code>\begin*</code> , d-93	<code>\check@percent</code> , a-429, f-165, f-173, f-189	<code>\CodeEscapeChar</code> , 17, a-445, a-450, a-766, a-770, a-1668
<code>\belowdisplaysortskip</code> , a-64, a-65, a-66	<code>\check@sum</code> , a-1193, a-1194, a-1199, a-1208, a-1217, a-1224	<code>\CodeIndent</code> , 16, a-52, a-53, a-200, a-334, a-428, a-1666, b-32, 83
<code>\belowdisplayskip</code> , a-63	<code>\CheckModules</code> , a-1712	<code>\codeline@windex</code> , a-696, a-704, a-707, a-708
<code>\BibTeX</code> , 19, a-1597	<code>Checksum</code> , a-1195	<code>\CodelineIndex</code> , a-1685, a-1686
<code>\bigskipamount</code> , d-515	<code>\Checksum</code> , 15, a-1194, a-1233	<code>codelinenum</code> , 17, a-347
<code>\bnamegroup</code> , d-90	<code>\chscchange</code> , a-1226, a-1228, a-1231	<code>\CodelineNumbered</code> , a-1683, a-1684, 83
<code>\boldmath</code> , d-470	<code>\chunkskip</code> , 16, 20, a-73	<code>\CodeMarginize</code> , 13, a-647
<code>\box</code> , d-483	<code>\cleardoublepage</code> , d-310	<code>\CodeMarginize*</code> , a-647
<code>\breakabletwelvespace</code> , a-154, a-228, f-35, f-39	<code>\clubpenalty</code> , a-92, a-136	<code>\CodeSpacesBlank</code> , 11, a-94, a-155, a-769
<code>\breakbslash</code> , f-24, f-26, f-30, f-181	<code>\cmd</code> , d-194	<code>codespacesblank</code> , 11, a-24
<code>\breaklbrace</code> , f-15, f-20, f-27	<code>\cmd@to@cs</code> , d-194, d-195	<code>\CodeSpacesSmall</code> , a-158
<code>\bslash</code> , a-484, a-504, a-530, a-542, a-627, a-636, a-668, a-676, a-1092, a-1106, a-1107, a-1121, a-1123, a-1592, a-1631, a-1643, a-1674, d-150, d-189, f-23, f-26, f-182, f-196	<code>\Code@CommonIndex</code> , a-594, a-595	<code>\CodeTopsep</code> , 16, a-57, a-60, a-75, a-79, a-361, a-764, a-766, a-768, a-770, a-830, a-1667
<code>\c@ChangesStartDate</code> , a-1133, a-1136, a-1145, a-1147, a-1148, a-1149	<code>\Code@CommonIndexStar</code> , a-594, a-596	<code>\CodeUsage</code> , 12, a-687
<code>\c@Checksum</code> , a-1195, a-1203, a-1208, a-1218, a-1227, a-1230	<code>\Code@DefEnvir</code> , a-686, a-710	<code>\CodeUsage*</code> , a-687
<code>\c@codelinenum</code> , a-268, a-344, a-699, a-1062, a-1063, a-1632	<code>\Code@DefIndex</code> , a-567, a-568, a-691, a-849	<code>\CodeUsgIndex</code> , 13, a-582
<code>\c@footnote</code> , a-1457, a-1475	<code>\Code@DefIndexStar</code> , a-567, a-571, a-852	<code>\CodeUsgIndex*</code> , a-582
<code>\c@GlossaryColumns</code> , a-1162, a-1162, a-1164	<code>\Code@DefMacro</code> , a-686, a-690	<code>\color</code> , d-558
<code>\c@gmd@mc</code> , a-1627, a-1631	<code>\Code@Delim</code> , a-41, a-42, a-45	<code>\columnsep</code> , a-1037
<code>\c@IndexColumns</code> , a-1027, a-1027, a-1029, a-1048	<code>\code@delim</code> , a-44, a-96, a-105, a-106, a-130, a-131, a-243, a-253, a-291, a-430, a-780, a-1554, a-1558, a-1559	<code>\CommonEntryCmd</code> , 18, a-514, a-561, c-14
<code>\c@secnumdepth</code> , d-319	<code>\Code@Delim@St</code> , a-41, a-45	<code>\continue@macroscan</code> , a-461, a-471, a-475
<code>\catactive</code> , 19, a-1573	<code>\code@escape@char</code> , a-258, a-448	<code>\continuum</code> , d-568
<code>\catletter</code> , 19, a-1574	<code>\Code@MarginizeEnvir</code> , a-656, a-657	<code>\copy</code> , d-454, d-475, d-489
<code>\catother</code> , 19, a-1572	<code>\Code@MarginizeMacro</code> , a-651, a-652, a-692, a-695	<code>copyrnote</code> , 20, a-1576
<code>\CDAnd</code> , 20, a-1614	<code>\Code@UsgEnvir</code> , a-689, a-715	<code>\count</code> , a-1140, a-1141, a-1142, d-295, d-296, d-297, d-459, d-460, d-461, d-462, d-463, d-464, d-465, d-466, d-541, d-542, d-543, d-544, d-547, d-551, d-552, d-553, d-554
<code>\CDPerc</code> , 20, a-1615	<code>\Code@UsgIndex</code> , a-584, a-585, a-694, a-732	<code>countalllines</code> , 10, a-10
<code>\changes</code> , a-1086, a-1091, a-1095	<code>\Code@UsgIndexStar</code> , a-584, a-588	<code>\cs</code> , 19, d-189, d-192, d-194
<code>\changes@</code> , a-1090, a-1099	<code>\Code@UsgMacro</code> , a-689, a-693	<code>\currentfile</code> , a-1248, a-1249, a-1250,
	<code>\CodeCommonIndex</code> , a-592, a-1701	
	<code>\CodeCommonIndex*</code> , 13	

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmlink.sty,
f=gmverb.sty

a-1251, a-1252,	<code>\DescribeEnv</code> , a-1650 , 82	<code>\egRestore@Macro</code> , d-236,
a-1253, a-1254,	<code>\DescribeMacro</code> , a-1647 , 82	d-237
a-1255, a-1258,	<code>\dimen</code> , d-531, d-532, d-533	<code>\egRestore@MacroSt</code> ,
a-1275, a-1279,	<code>\DisableCrossrefs</code> ,	d-236, d-238
a-1290, a-1346,	a-1689, a-1691	<code>\egStore@Macro</code> , d-211, d-212
a-1347, a-1349,	<code>\discre</code> , a-1593, d-178 , d-181	<code>\egStore@MacroSt</code> , d-211,
a-1371, a-1372,	<code>\discret</code> , d-179 , d-180	d-213
a-1392, a-1396, a-1416	<code>\divide</code> , d-461, d-464, d-465	<code>\egText@Marginize</code> , a-741,
<code>\czer</code> , d-559, d-560, d-561	<code>\division</code> , 19 , a-1358,	a-742
<code>\czerwo</code> , d-558, d-559	a-1616, a-1618 , a-1619	<code>\emptify</code> , a-141, a-779,
	<code>\Do@Index</code> , a-996, a-997	d-27 , d-27
<code>\dag</code> , d-561	<code>\do@noligs</code> , f-45, f-200	<code>\EnableCrossrefs</code> , a-1303,
<code>\date</code> , a-1454 , <i>c-6</i>	<code>\do@properindex</code> , a-605,	a-1690
<code>\day</code> , a-1227, a-1229	a-638, a-758	<code>\enamegroup</code> , d-91
<code>debug</code> , b-13 , 90	<code>\dobreakblankspace</code> , f-40	<code>\encapchar</code> , 18 , a-443 ,
<code>\Debug@dstron</code> , a-1062,	<code>\dobreakbslash</code> , f-30, f-47	a-484, a-531, a-1098,
a-1063, a-1065, a-1072	<code>\dobreaklbrace</code> , f-20 , f-47	a-1128
<code>\DeclareLogo</code> , a-1597,	<code>\dobreakspace</code> , f-48, f-53	<code>\endenvironment</code> , a-868
a-1600, a-1602,	<code>\dobreakvisible</code> , f-39 , f-53	<code>\endlinechar</code> , a-1541
a-1603, a-1604,	<code>\Doc@Include</code> , a-1239, a-1240	<code>\endlist</code> , d-417, d-432
a-1605, a-1606,	<code>\Doc@Input</code> , a-85 , a-88, a-1723	<code>\endmacro</code> , a-834, a-836
a-1607, a-1608,	<code>\DocInclude</code> , 8 , 10 , 21 ,	<code>\endmacro*</code> , a-836
d-436, d-445, d-469,	a-1239 , a-1256,	<code>\endmacrocode</code> , a-772
d-472	a-1260, a-1375, <i>c-12</i> ,	<code>\endoldmc</code> , a-772
<code>\DeclareOption</code> , a-6, a-8,	<i>c-13</i> , <i>c-21</i> , <i>c-22</i> , <i>c-23</i>	<code>\endtheglossary</code> , a-1305
a-10, a-12, a-14,	<code>\docincludeaux</code> , a-1247,	<code>\endverbatim</code> , f-119
a-16, a-21, a-22,	a-1315 , a-1316, a-1370	<code>\enoughpage</code> , d-529
a-24, b-7, b-8, b-9,	<code>\DocInput</code> , 8 , a-85 , a-1415,	<code>\enspace</code> , b-56
b-10, b-11, b-13,	a-1420, a-1559	<code>\ensuremath</code> , b-67, d-164,
b-14, b-17, b-19	<code>DocInputsCount</code> , a-346	d-174
<code>\DeclareRobustCommand*</code> ,	<code>\docstrips@percent</code> , a-785	<code>\EntryPrefix</code> , 17 , a-525,
a-1706, d-98, d-126,	<code>\DocstyleParms</code> , a-1710	a-527, a-540, a-1047,
d-127, d-128, d-129,	<code>\documentclass</code> , <i>c-1</i>	a-1243 , a-1367
d-130, d-131, d-163,	<code>\DoIndex</code> , 14 , a-996 ,	<code>enumerate*</code> , d-402
d-186, d-188, d-189,	a-1000, <i>c-11</i>	<code>\env</code> , 19 , d-192
d-192, d-302, d-505,	<code>\DoNot@Index</code> , a-869, a-870	<code>\environment</code> , a-867
e-9, e-17, e-23	<code>\DoNotIndex</code> , 14 , a-869 ,	<code>environment</code> , 13 , a-867
<code>\DeclareTextCommand</code> , d-442	a-999, a-1000, a-1002	<code>\envirs@toindex</code> , a-581,
<code>\DeclareTextCommandDefault</code> ,	<code>\dont@index</code> , a-872, a-873 ,	a-665, a-679, a-680,
d-444	a-878, a-884, a-997	a-683, a-725
<code>\def@other</code> , d-492	<code>\DontCheckModules</code> , a-1711	<code>\envirs@tomarginpar</code> ,
<code>\DefaultIndexExclusions</code> ,	<code>\doprivateothers</code> , a-432,	a-659, a-662, a-663,
14 , a-889 , a-994, a-1001	a-433, a-453, a-454	a-682, a-722
<code>\DefEntry</code> , 17 , a-559 , a-1707	<code>\ds</code> , 20 , a-1613	<code>\EOFMark</code> , 16 , <i>a-104</i> , a-151 ,
<code>\definecolor</code> , a-28	<code>\edef@other</code> , d-500, d-503	a-1558, a-1727, b-67, 90
<code>\defobeylines</code> , d-511	<code>\edverbs</code> , b-62, f-206 , f-211	<code>\equals</code> , d-536
<code>\dekbigskip</code> , d-515	<code>\eequals</code> , d-537	<code>\errorcontextlines</code> , b-43
<code>\dekclubs</code> , 11 , <i>b-61</i> , f-205 , 122	<code>\eg@MakeShortVerb</code> , f-235,	<code>\eTeX</code> , 20 , a-1605 , a-1606
<code>\dekclubs*</code> , 122	f-237	<code>\evensidemargin</code> , a-1238
<code>\dekmedskip</code> , d-514	<code>\eg@MakeShortVerbStar</code> ,	<code>\everypar</code> , a-99, <i>a-99</i> ,
<code>\dekssmallskip</code> , d-512	f-235, f-236	a-121, a-166, a-167,
<code>\DeleteShortVerb</code> , 11 ,	<code>\egCode@MarginizeEnvir</code> ,	a-173, a-185, a-299,
f-78 , 122	a-649, a-655	a-308, a-326, a-338,
<code>\Describe</code> , 13 , a-1652	<code>\egCode@MarginizeMacro</code> ,	a-388, a-395, a-863,
<code>\Describe@Env</code> , a-1649,	a-649, a-650	a-1577, f-166
a-1651, a-1654, a-1660		<code>\exhyphenpenalty</code> , b-64
<code>\Describe@Macro</code> , a-1649,		<code>\exii@currenvir</code> , d-96, d-97
a-1654, a-1655		

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmlink.sty, f=gmverb.sty

<code>\f@encoding</code> , d-549	<code>\glet</code> , a-113, a-218, a-479,	<code>\gmcc@mwclstrue</code> , b-6
<code>\f@series</code> , d-470	a-653, a-780, a-1053,	<code>\gmd@@iffalse</code> , a-487, a-492
<code>\file</code> , 19, c-15, c-16, d-187	a-1341, a-1344,	<code>\gmd@@iftrue</code> , a-488, a-491
<code>\filedate</code> , 20, a-1351,	a-1355, d-19	<code>\gmd@@macro</code> , a-1719, a-1720
a-1500, a-1549	<code>\glossary@prologue</code> ,	<code>\gmd@@toc</code> , a-142, a-144
<code>\filediv</code> , a-1317, a-1327,	a-1151, a-1165,	<code>\gmd@ABIOnce</code> , a-147 ,
a-1357, a-1363,	a-1180, a-1183 , a-1342	a-148, a-150
a-1465, a-1489, c-15	<code>\GlossaryMin</code> , 15, a-1161,	<code>\gmd@bslashEOL</code> , a-418, a-427
<code>\filedivname</code> , a-1318,	a-1165	<code>\gmd@charbychar</code> , a-197,
a-1323, a-1326,	<code>\GlossaryParms</code> , 15,	a-223, a-250, a-250,
a-1328, a-1333,	a-1166, a-1187	a-274, a-500
a-1334, a-1335,	<code>\GlossaryPrologue</code> , 15,	<code>\gmd@checkifEOL</code> , a-168, a-297
a-1356, a-1362, a-1466	a-1179	<code>\gmd@checkifEOLmixd</code> ,
<code>\FileInfo</code> , 20, a-1505	<code>\gm@atppron</code> , d-278 , d-281,	a-256, a-306
<code>\fileinfo</code> , 20, a-1502	d-282, d-283, d-284,	<code>\gmd@chschangeline</code> ,
<code>\filekey</code> , a-1290, a-1337,	d-285, d-286, d-287,	a-1204, a-1211,
a-1340	d-288	a-1219, a-1225
<code>\filename</code> , a-1350, a-1498	<code>\gm@clearpagesduetoopenright</code> ,	<code>\gmd@closingspacewd</code> ,
<code>\filenote</code> , 20, a-1549 , a-1550	d-309, d-328	a-189, a-420, a-421 ,
<code>\filesep</code> , a-1242, a-1243,	<code>\gm@def@other@tempa</code> , d-493	a-423
a-1306, a-1336,	<code>\gm@dontnumbersectionsoutofmainmatter</code> ,	<code>\gmd@closecheckifds</code> , a-821
a-1366, a-1367	d-307, d-320	<code>\gmd@codeskip</code> , a-202,
<code>\fileversion</code> , 20, a-1227,	<code>\gm@gobmacro</code> , d-495, d-497	a-320, a-361 , a-365,
a-1228, a-1352,	<code>\gm@hyperrefstepcounter</code> ,	a-377
a-1501, a-1549	d-301, d-304, d-339	<code>\gmd@continuenarration</code> ,
<code>\Finale</code> , 18, a-1693, a-1698	<code>\gm@hypertarget</code> , e-10, e-11	a-132 , a-163 , a-245
<code>\finish@macroscan</code> , a-461,	<code>\gm@iflink</code> , e-23, e-24, e-25	<code>\gmd@countnarrationline</code> ,
a-469, a-475, a-493	<code>\gm@ifnac</code> , d-61, d-62	a-165, a-170 , a-179 ,
<code>\fixbslash</code> , f-26 , 121	<code>\gm@ifref</code> , e-17, e-18, e-19	a-298, a-307
<code>\fixlbrace</code> , f-27 , 121	<code>\gm@jobn</code> , d-538, d-539	<code>\gmd@counttheline</code> , a-260,
<code>\fontencoding</code> , f-52	<code>\gm@letspace</code> , d-67, d-73	a-274, a-276
<code>\fontseries</code> , b-49	<code>\gm@notprerr</code> , d-270, d-275	<code>\gmd@currentlabel@before</code> ,
<code>\freeze@actives</code> , d-540	<code>gm@PronounGender</code> , d-277	a-90 , a-113
<code>\fullcurrentfile</code> , a-1249,	<code>\gm@pswords</code> , d-182, d-183 ,	<code>\gmd@currenvxistar</code> ,
a-1258, a-1280,	d-185	a-786, a-791
a-1372, a-1397	<code>\gm@sec</code> , d-389, d-396, d-397	<code>\gmd@DefineChanges</code> ,
<code>\g@emptyify</code> , a-150, a-480,	<code>\gm@secini</code> , d-370, d-380,	a-1085 , a-1159
a-663, a-680, a-1192,	d-383, d-386, d-394	<code>\gmd@dip@hook</code> , a-1021,
a-1433, a-1434,	<code>\gm@secmarkh</code> , d-384	a-1023 , a-1024
a-1492, a-1563, d-28 ,	<code>\gm@secstar</code> , d-372, d-378,	<code>\gmd@docrescan</code> , a-1521,
d-29	d-381, d-387, d-396,	a-1529
<code>\g@relaxen</code> , a-505, a-536,	d-397	<code>\gmd@docrescanfile</code> ,
a-537, a-539, a-860,	<code>\gm@secx</code> , d-389, d-390	a-1530, a-1531,
a-1491, d-32 , d-33	<code>\gm@secxx</code> , d-371 , d-385 , d-391	a-1533, a-1536
<code>\gaddtomacro</code> , 18, a-150,	<code>\gm@straightensec</code> , d-392,	<code>\gmd@docstripdirective</code> ,
a-428, d-20	d-399	a-296, a-305, a-823,
<code>\gag@index</code> , a-37, a-706 ,	<code>\gm@targetheading</code> , d-302,	a-1054
a-1683, a-1689	d-305	<code>\gmd@docstripinner</code> ,
<code>\gemptify</code> , d-29 , d-29	<code>\gm@verb@eol</code> , a-436 , f-178,	a-1060, a-1067
<code>\GeneralName</code> , a-1115,	f-185 , f-192	<code>\gmd@docstripshook</code> , a-1079
a-1116, a-1132,	<code>\gm@xistar</code> , a-790, a-792	<code>\gmd@docstripverb</code> ,
a-1153, a-1244 , a-1373	<code>\gmboxedspace</code> , a-1586,	a-1059, a-1074
<code>\generalname</code> , a-1099,	a-1587, a-1590, a-1593	<code>\gmd@doIndexRelated</code> ,
a-1102, a-1126, a-1158	<code>\gmcc@baseclass</code> , b-4 , b-7,	a-1274, a-1282,
<code>\geometry</code> , b-40	b-8, b-9, b-24, b-28	a-1302 , a-1391, a-1400
<code>\GetFileInfo</code> , 20, a-1279,	<code>gmcc@mwcls</code> , b-5	<code>\gmd@dolspaces</code> , a-133 ,
a-1349, a-1396, a-1497	<code>\gmcc@mwclsfalse</code> , b-10, b-24	a-197, a-229

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmlink.sty, f=gmverb.sty

`\gmd@DoTeXCodeSpace`,
 a-127, [a-153](#), a-156,
 a-159, a-782
`\gmd@eatlspace`, a-234,
 [a-238](#), a-241
`\gmd@endpe`, a-312, a-314,
 a-325, a-330, [a-331](#)
`\gmd@EOlorcharbychar`,
 a-262, [a-265](#)
`\gmd@evpaddonce`, a-855, [a-856](#)
`\gmd@fileinfo`, a-1509, a-1517
`\gmd@guardedinput`, [a-101](#),
 a-110
`\gmd@iedir`, a-871, a-883,
 a-997
`\gmd@ifonetoken`, a-833,
 a-835, [a-840](#), a-1649,
 a-1720
`\gmd@ifsingle`, [a-837](#), a-842
`\gmd@iihook`, a-103, a-141,
 a-1556
`\gmd@inputname`, [a-89](#),
 a-1201, a-1210, a-1216
`\gmd@justadot`, [a-575](#),
 a-577, a-580, a-653,
 a-871
`\gmd@ldspaceswd`, a-205,
 a-212, a-213, a-220,
 [a-227](#), a-233, a-240,
 a-244
`gmd@mc`, [a-1624](#)
`\gmd@mcdiag`, a-1626,
 [a-1629](#), a-1630, a-1633
`\gmd@mchhook`, a-1625
`\gmd@modulehashone`,
 a-1069, a-1072,
 a-1076, a-1080
`\gmd@narrcheckifds`,
 a-302, a-303
`\gmd@narrcheckifds@ne`,
 a-292, a-294
`\gmd@nocodeskip`, a-201,
 a-203, a-321, a-323,
 [a-363](#), a-367, a-374,
 a-379
`\gmd@oldmcfinis`, a-806
`\gmd@oncenum`, a-857,
 a-859, a-861, a-862,
 a-864, a-866
`\gmd@parfixclosingspace`,
 a-184, [a-419](#)
`\gmd@percenthack`, a-254,
 [a-290](#)
`\gmd@preverypar`, [a-47](#),
 a-174, a-300, a-308,
 a-327, a-339, a-386,
 a-393, a-395
`\gmd@providefii`, a-1543,
 [a-1545](#)
`\gmd@resetlinecount`,
 a-97, a-342, [a-348](#)
`\gmd@revprefix`, a-553, [a-554](#)
`\gmd@setChDate`, a-1135,
 a-1137, a-1145
`\gmd@setclosingspacewd`,
 [a-422](#)
`\gmd@setclubpenalty`,
 a-91, a-123, a-124, a-136
`\gmd@skipgmltext`, [a-1563](#),
 a-1563, a-1569
`\gmd@spacewd`, [a-226](#),
 a-232, a-240
`\gmd@texcodeEOL`, [a-199](#), a-266
`\gmd@texcodespace`, [a-157](#),
 a-161, a-196, [a-228](#),
 a-230, a-239
`\gmd@textEOL`, a-100,
 [a-117](#), a-301, a-311,
 a-439, a-779, a-1072,
 a-1080
`\gmd@typesettexcode`,
 [a-183](#), a-236, a-246
`\gmd@writeckpt`, a-1284,
 a-1296, a-1402
`\gmd@writeFI`, a-1520, [a-1525](#)
`\gmd@writemauxinpau`,
 a-1264, [a-1291](#), a-1378
`\gmdindexpagecs`, a-548, [a-552](#)
`\gmdindexrefcs`, a-547,
 a-548, [a-550](#)
`\gmdmarginpar`, *13*, *20*,
 [a-748](#), a-753, a-756
`\gmdnoindent`, *20*, [a-1581](#)
`\gmdoccMargins`, b-39
`\gmdocIncludes`, *9*, [a-1419](#)
`gmeometric`, [b-18](#)
`gmгло.ist`, 65
`\gmhypertarget`, a-358,
 c-16, *e-9*, *118*
`\gmiflink`, a-550, *e-23*, *118*
`\gmifref`, *e-17*, *118*
`\gml@StoreCS`, d-222,
 d-231, d-247
`\gml@storemacros`, d-223,
 d-224, d-229, d-232,
 d-248
`gmlonely`, *19*, [a-1560](#)
`\gmobeyspaces`, *a-156*,
 [d-507](#), f-116, f-179
`\gmshowlists`, [d-34](#)
`\gmTheGeometry`, b-18,
 b-21, b-38
`\gmu@def`, d-312, d-314, d-315
`\gmu@RP`, d-557, d-563,
 d-565, d-568
`\gmu@smtempa`, d-217,
 d-219, d-242, d-244
`\gmu@tempa`, d-500, d-501,
 d-503, d-504
`\gmvdismath`, f-207, f-210,
 f-213
`\gmvdisverb`, f-210, f-212
`\gmvedismath`, f-208, f-214
`\gmvpackname`, [f-94](#), f-95, f-97
`\gobble`, [d-133](#)
`\gobbletwo`, [d-134](#)
`\grefstepcounter`, a-172,
 a-207, a-279, a-286, [d-17](#)
`\grelaxen`, [d-33](#), d-33
`\HeadingNumber`, d-336, d-338
`\HeadingNumberedfalse`,
 d-308, d-319
`\HeadingRHeadText`, d-322
`\HeadingText`, d-324
`\HeadingTOCText`, d-323
`\HeShe`, [d-285](#)
`\heshe`, 7, [d-281](#)
`\hfillneg`, [d-516](#)
`\HimHer`, [d-287](#)
`\himher`, [d-283](#)
`\HisHer`, [d-286](#)
`\hisher`, [d-282](#)
`\HisHers`, [d-288](#)
`\hishers`, [d-284](#)
`\HLPrefix`, *17*, a-359,
 a-525, a-527, a-557,
 a-699, a-1014,
 a-1242, a-1366
`\Hybrid@DefEnvir`, a-833,
 a-851
`\Hybrid@DefMacro`, a-833,
 a-848
`hyperindex`, 43
`\hyperlabel@line`, a-175,
 a-210, a-280, a-287,
 a-355
`\hypersetup`, a-29, a-1033
`\hyphenpenalty`, b-64, d-185
`\if*`, a-792
`\if@aftercode`, a-201,
 a-319, [a-370](#), a-375
`\if@afternarr`, a-201,
 a-318, [a-371](#), a-374
`\if@codeskipput`, a-75,
 a-79, a-191, a-202,
 a-320, [a-362](#), a-373,
 a-764, a-768, a-830
`\if@codespacesblank`,
 [a-23](#), a-94

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty,
 f=gmverb.sty

<code>\if@countalllines</code> , a-9, a-169, a-268, a-759	<code>\incl@DocInput</code> , a-1280, a-1397, a-1415, a-1418, a-1420	<code>\levelchar</code> , 18, a-444, a-484, a-1098, a-1118, a-1128
<code>\if@debug</code> , b-12, b-41, b-45, b-46	<code>\incl@filedivtitle</code> , a-1477, a-1489	<code>\LineNumFont</code> , 17, a-176, a-351, a-353, a-1671, 83
<code>\if@dsdir</code> , a-84, a-84, a-822	<code>\incl@titletotoc</code> , a-1470, a-1478	<code>\lineskip</code> , a-1444
<code>\if@filesw</code> , a-696, a-1264, a-1270, a-1285, a-1377, a-1386, a-1403, d-524	<code>\InclMaketitle</code> , a-1276, a-1393, a-1467	<code>linesnotnum</code> , 10, a-6
<code>\if@fshda</code> , a-1470, a-1481, a-1494	<code>\index@macro</code> , a-497, a-507, a-606, a-639, a-678	<code>\list</code> , d-408, d-424
<code>\if@gmccnochanges</code> , b-16, b-59	<code>\index@prologue</code> , a-1003, a-1005, a-1029, a-1338	<code>\listparindent</code> , d-413, d-429
<code>\if@indexallmacros</code> , a-15, a-993	<code>indexallmacros</code> , 10, a-16	<code>\liturgiques</code> , d-556
<code>\if@linesnotnum</code> , a-5, a-354, a-544	<code>IndexColumns</code> , 18, a-1027	<code>\LoadClass</code> , b-27, b-30
<code>\if@ltxDocInclude</code> , a-1275, a-1278, a-1281, a-1392, a-1395, a-1398, a-1408	<code>\indexcontrols</code> , a-479, a-483	<code>\ltxLookSetup</code> , 9, a-1409, a-1414
<code>\if@mainmatter</code> , d-308	<code>\indexdiv</code> , a-1004, a-1005, a-1183	<code>\ltxPageLayout</code> , 9, a-1234, a-1411
<code>\if@marginparsused</code> , a-17, a-744	<code>\indexentry</code> , a-698	<code>\macro</code> , a-828, a-835, a-1719, a-1720, d-497
<code>\if@newline</code> , a-83, a-171, a-207, a-267, a-278, a-285	<code>\IndexInput</code> , 10, a-1553	<code>macro</code> , 13, a-828
<code>\if@noindex</code> , a-11, a-36	<code>\IndexLinksBlack</code> , 18, a-1018, a-1030, a-1033	<code>macro*</code> , a-835
<code>\if@noskipsec</code> , f-144	<code>\IndexMin</code> , 18, a-1026, a-1026, a-1029	<code>\macro@iname</code> , a-459, a-464, a-467, a-473, a-497, a-606, a-608, a-614, a-639, a-678
<code>\if@openright</code> , d-310	<code>\IndexParms</code> , 18, a-1031, a-1035, a-1187	<code>\macro@pname</code> , a-460, a-468, a-474, a-494, a-497, a-498, a-499, a-599, a-600, a-601, a-606, a-626, a-627, a-628, a-631, a-639
<code>\if@pageinclindex</code> , a-524, a-541	<code>\IndexPrefix</code> , 17, a-529, a-543	<code>\macrocode</code> , a-771
<code>\if@pageindex</code> , a-13, a-356, a-523, a-546, a-701, a-1007, a-1010, a-1011, a-1013	<code>\IndexPrologue</code> , 18, a-1003, 85	<code>macrocode</code> , 8, 21, a-767
<code>\if@RecentChange</code> , a-1100, a-1134	<code>\IndexRefCs</code> , a-525, a-527, a-531	<code>macrocode*</code> , a-763
<code>\if@specialpage</code> , d-327	<code>\interlinepenalty</code> , f-165	<code>\MacrocodeTopsep</code> , a-1667
<code>\if@twoside</code> , d-350	<code>\itemindent</code> , d-410, d-426	<code>\MacroFont</code> , a-1665, 82
<code>\if@uresetlinecount</code> , a-7, a-341	<code>itemize*</code> , d-418	<code>\MacroIndent</code> , a-1666, 83
<code>\ifdtraceoff</code> , a-1063, b-46	<code>\jobnamewoe</code> , c-26, c-31, d-539	<code>\MacroTopsep</code> , a-58, a-61, a-74, a-829, a-834, 83
<code>\ifdtraceon</code> , a-1062, b-45	<code>\kernel@ifnextchar</code> , a-1543	<code>\main</code> , a-1707
<code>\ifgmcc@mwcls</code> , b-5, b-23, b-26	<code>\kind@fentry</code> , a-514, a-516, a-520, a-525, a-527	<code>\MakeGlossaryControls</code> , 15, a-1089, a-1096
<code>\ifHeadingNumbered</code> , d-318, d-334	<code>\l@nohyphenation</code> , d-160, d-161, d-171	<code>\MakePercentComment</code> , a-1715
<code>\ifodd</code> , d-280	<code>\labelsep</code> , d-412, d-428	<code>\MakePercentIgnore</code> , a-1087, a-1714
<code>\ifprevhmode</code> , a-249, a-291, a-332	<code>\labelwidth</code> , d-411, d-412, d-427, d-428	<code>\MakePrivateLetters</code> , 12, 17, a-129, a-452, a-566, a-583, a-593, a-618, a-641, a-648, a-685, a-688, a-727, a-740, a-784, a-832, a-869, a-996, a-1088, a-1648, a-1653
<code>\ifSecondClass</code> , d-573	<code>\larger</code> , d-126, 99	<code>\MakePrivateOthers</code> , a-453, a-567, a-584, a-594, a-619, a-642, a-649, a-686, a-689, a-728, a-741, a-832, a-1651, a-1654
<code>\im@firstpar</code> , a-495, a-496, a-497, a-602, a-603, a-606	<code>\largerr</code> , d-130, 99	
<code>\IMO</code> , d-570	<code>\last@defmark</code> , a-537, a-578, a-1103, a-1106, a-1107, a-1131	
	<code>\LaTeXe</code> , d-434, d-469	
	<code>\LaTeXpar</code> , 20, d-472	
	<code>\leftmargin</code> , d-409, d-425, f-156, f-157	

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmlink.sty,
f=gmverb.sty

`\MakeShortVerb`, [11](#), [f-54](#),
[f-237](#), [122](#)
`\MakeShortVerb*`, [f-54](#),
[f-205](#), [f-236](#)
`\maketitle`, [8](#), [a-983](#),
[a-1276](#), [a-1393](#),
[a-1423](#), [c-8](#), [c-11](#), [74](#)
`\marg`, [d-196](#), [d-208](#)
`\marginparpush`, [a-746](#)
`\marginpartt`, [13](#), [a-756](#),
[a-757](#), [b-49](#)
`\marginparwidth`, [a-747](#),
[a-1236](#)
`\mark@envir`, [a-211](#), [a-283](#),
[a-658](#)
`\math@arg`, [d-204](#), [d-205](#)
`\mathfrak`, [d-568](#)
`\mathindent`, [b-32](#)
`\maybe@marginpar`, [a-498](#),
[a-502](#)
`\maybe@quote`, [a-459](#),
[a-467](#), [a-473](#), [a-479](#),
[a-480](#), [a-614](#)
`\mcdiagOff`, [a-1633](#)
`\mcdiagOn`, [a-1630](#)
`\medmskip`, [d-180](#)
`\meta`, [d-163](#), [d-177](#), [d-196](#),
[d-198](#), [d-201](#), [84](#)
`\meta@font@select`, [d-167](#),
[d-176](#)
`\meta@hyphen@restore`,
[d-168](#), [d-173](#)
`\mod@math@codes`, [a-1082](#),
[a-1083](#), [a-1084](#)
`\Module`, [a-1070](#), [a-1082](#)
`\ModuleVerb`, [a-1077](#), [a-1083](#)
`\month`, [a-1227](#), [a-1229](#)
`\mskip`, [d-180](#)
`\multiply`, [a-1139](#), [a-1141](#),
[d-460](#), [d-463](#)
`\mw@HeadingBreakAfter`,
[d-329](#), [d-346](#), [d-361](#),
[d-365](#)
`\mw@HeadingBreakBefore`,
[d-326](#)
`\mw@HeadingLevel`, [d-316](#),
[d-319](#)
`\mw@HeadingRunIn`, [d-341](#)
`\mw@HeadingType`, [d-325](#)
`\mw@HeadingWholeWidth`,
[d-344](#)
`\mw@normalheading`, [d-348](#),
[d-357](#), [d-360](#), [d-364](#)
`\mw@runinheading`, [d-342](#)
`\mw@sectionxx`, [d-315](#)
`mwart`, [b-7](#), [90](#)
`mwbk`, [b-9](#), [90](#)

`mwrep`, [b-8](#), [c-1](#), [90](#)
`\n@melet`, [a-773](#), [a-774](#),
[d-257](#), [d-373](#), [d-376](#),
[d-394](#), [f-127](#)
`\nameshow`, [d-35](#)
`NeuroOnc`, [a-859](#)
`\newcount`, [a-1027](#), [a-1133](#),
[a-1162](#), [a-1193](#)
`\newcounter`, [a-344](#), [a-346](#),
[a-347](#), [a-1195](#),
[a-1624](#), [a-1709](#),
[d-277](#), [d-299](#), [e-7](#)
`\newdimen`, [a-1026](#), [a-1161](#)
`\newgif`, [a-83](#), [a-84](#), [a-249](#),
[a-362](#), [a-370](#), [a-371](#), [d-6](#)
`\newlanguage`, [d-161](#)
`\newlength`, [a-51](#), [a-52](#),
[a-54](#), [a-226](#), [a-227](#),
[f-169](#), [f-171](#)
`\newlinechar`, [a-1532](#)
`\newskip`, [a-57](#), [a-58](#), [a-421](#)
`\newtoks`, [a-47](#)
`\newwrite`, [a-1530](#), [d-524](#)
`\nfss@text`, [d-165](#)
`\nieczer`, [d-562](#)
`\nl@percent`, [a-1584](#),
[a-1586](#), [a-1588](#)
`\nlpercent`, [a-1582](#)
`nochanges`, [b-17](#), [89](#)
`\noeffect@info`, [a-1673](#),
[a-1679](#), [a-1680](#),
[a-1681](#), [a-1682](#),
[a-1710](#), [a-1711](#),
[a-1712](#), [a-1713](#)
`\NoEOF`, [16](#), [a-1534](#), [a-1728](#)
`noindex`, [10](#), [a-12](#), [b-14](#), [89](#)
`nomarginpar`, [10](#), [a-22](#)
`\NonUniformSkips`, [16](#), [a-72](#)
`\nostanza`, [16](#), [a-81](#)
`\not@onlypreamble`, [d-262](#),
[d-265](#), [d-266](#), [d-267](#),
[d-268](#), [d-269](#)
`\nummacro`, [a-866](#), [d-289](#)
`\oarg`, [d-197](#)
`\obeyspaces`, [a-154](#), [a-160](#),
[a-799](#), [f-36](#), [f-39](#), [f-40](#),
[f-187](#)
`\oddsidemargin`, [a-1237](#)
`\old@begin`, [d-92](#), [d-93](#)
`\old@MakeShortVerb`,
[a-1718](#), [f-215](#), [f-234](#)
`\olddocIncludes`, [9](#), [21](#),
[a-1417](#)
`\OldDocInput`, [8](#), [21](#),
[a-1418](#), [a-1716](#)
`\oldLaTeX`, [d-433](#)

`\oldLaTeXe`, [d-434](#)
`\OldMakeShortVerb`, [f-233](#), [122](#)
`\OldMakeShortVerb*`, [f-205](#)
`\oldmc`, [a-771](#), [a-775](#)
`oldmc`, [21](#), [a-771](#)
`oldmc*`, [a-771](#)
`\oldmc@def`, [a-803](#), [a-808](#)
`\oldmc@end`, [a-804](#), [a-809](#)
`\OnAtLine`, [d-535](#)
`\OnlyDescription`, [18](#), [a-1696](#)
`outeroff`, [b-11](#), [c-1](#), [90](#)
`\PackageError`, [a-1256](#), [a-1327](#)
`\PackageInfo`, [a-1669](#),
[a-1673](#), [f-97](#)
`\PackageWarning`, [d-122](#), [d-124](#)
`\PackageWarningNoLine`,
[a-1091](#)
`\pagebreak`, [d-349](#), [d-361](#),
[d-365](#)
`\pagegoal`, [d-531](#)
`\PageIndex`, [a-1687](#), [a-1688](#)
`pageindex`, [10](#), [a-14](#)
`\pagestyle`, [b-57](#)
`\pagetotal`, [d-532](#)
`\par`, [a-76](#), [a-80](#), [a-111](#),
[a-125](#), [a-190](#), [a-236](#),
[a-302](#), [a-312](#), [a-315](#),
[a-328](#), [a-420](#), [a-764](#),
[a-766](#), [a-768](#), [a-770](#),
[a-831](#), [a-834](#), [a-931](#),
[a-1043](#), [a-1046](#),
[a-1423](#), [a-1442](#),
[a-1447](#), [a-1451](#),
[a-1578](#), [a-1580](#)
`\parg`, [d-200](#)
`\parsep`, [f-143](#)
`\partopsep`, [a-68](#), [d-409](#),
[d-425](#), [f-147](#)
`\PassOptionsToPackage`,
[b-15](#)
`\pdfTeX`, [20](#), [a-1606](#)
`\pdfTeX`, [20](#), [a-1607](#)
`\pk`, [19](#), [a-1244](#), [a-1373](#),
[c-3](#), [c-4](#), [c-20](#), [d-188](#)
`\PlainTeX`, [19](#), [a-1602](#)
`\possfil`, [d-193](#)
`\predisplaypenalty`,
[f-115](#), [f-117](#)
`\prevhmodegfalse`, [a-195](#),
[a-221](#), [a-249](#), [a-252](#),
[a-313](#), [a-325](#)
`\prevhmodegtrue`, [a-249](#), [a-251](#)
`\PrintChanges`, [15](#), [a-1188](#),
[a-1192](#), [a-1304](#), [c-28](#)
`\PrintDescribeEnv`, [83](#)
`\PrintDescribeMacro`, [83](#)

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmlink.sty,
f=gmverb.sty

<code>\PrintEnvName</code> , 83	b-34, b-35, b-37,	<code>\SpecialUsageIndex</code> , a-1702
<code>\PrintFilesAuthors</code> , 8 ,	b-42, b-47, b-66,	<code>\square</code> , b-67
a-1495	d-521, d-563, f-4	<code>StandardModuleDepth</code> , a-1709
<code>\PrintIndex</code> , a-1050 ,	<code>\resetlinecountwith</code> , a-343	<code>\stanza</code> , 16 , 20 , a-77 ,
a-1304 , c-32	<code>\Restore@Macro</code> , d-237,	a-1579 , c-18 , c-20
<code>\printindex</code> , a-1051, a-1304	d-239, d-247, d-250	<code>\stanzaskip</code> , 16 , a-54,
<code>\printlinenumber</code> , a-209,	<code>\Restore@Macros</code> , d-245, d-246	a-55 , a-56, a-60,
a-282 , a-350 , a-354	<code>\Restore@MacroSt</code> , d-238,	a-61 , a-62, a-63,
<code>\PrintMacroName</code> , 83	d-241	a-64 , a-67, a-78,
<code>\printspaces</code> , d-182 , d-186	<code>\RestoreMacro</code> , a-1002,	a-192 , f-163, f-169, f-170
<code>\ProvideFileInfo</code> , 20 ,	a-1559 , d-235 , d-314	<code>\step@checksum</code> , a-456, a-1196
a-1539 , a-1548	<code>\RestoreMacros</code> , a-709, d-245	<code>\stepnummacro</code> , a-857, d-290
<code>\ProvidesClass</code> , b-2	<code>\RestoringDo</code> , a-1282,	<code>\StopEventually</code> , 18 ,
<code>\ProvideSelfInfo</code> , 20 , a-1548	a-1399 , d-254	a-1693 , a-1696
<code>\ps@plain</code> , a-1436	<code>\reversemarginpar</code> , a-745	<code>\Store@Macro</code> , d-212,
<code>\ps@titlepage</code> , a-1436	<code>\rightline</code> , b-67	d-214, d-222
 	<code>\romannumeral</code> , d-407, d-423	<code>\Store@Macros</code> , d-220, d-221
<code>\quad</code> , b-55, b-56	<code>\rs@size@warning</code> , d-115,	<code>\Store@MacroSt</code> , d-213, d-216
<code>\QueerCharOne</code> , a-406 ,	d-120, d-122	<code>\stored@code@delim</code> , a-780
a-407 , a-408	<code>\rs@unknown@warning</code> ,	<code>\Stored@Macro</code> , d-249, d-250
<code>\QueerCharTwo</code> , a-400 ,	d-112, d-124	<code>\StoredMacro</code> , d-249
a-401 , a-402	 	<code>\StoreMacro</code> , a-999,
<code>\QueerEOL</code> , 7 , a-144, a-414 ,	<code>\scan@macro</code> , a-260, a-455	a-1554 , d-210 , d-314
a-765 , a-769, a-1051,	<code>\scshape</code> , a-1601, d-569	<code>\StoreMacros</code> , a-708, d-220
a-1190 , a-1421, a-1728	<code>\secondclass</code> , d-572	<code>\StoringAndRelaxingDo</code> ,
quotation, 20 , a-1580	<code>\SecondClasstrue</code> , d-574	a-1274 , a-1390, d-251
<code>\quote@char</code> , a-458, a-466,	<code>\SelfInclude</code> , 9 , a-1364 , c-14	<code>\StraightEOL</code> , 7 , a-144,
a-472 , a-477 , a-613	<code>\SetFileDiv</code> , 19 , a-1320,	a-409 , a-1051,
<code>\quote@charbychar</code> , a-609,	a-1321 , a-1323,	a-1088 , a-1190,
a-610 , a-615	a-1329 , a-1361 , a-1410	a-1560 , a-1567,
<code>\quote@mname</code> , a-600,	<code>\settexcodehangi</code> , a-48,	a-1577 , a-1717
a-607 , a-634, a-674	a-50 , a-214, a-218, a-333	<code>\subdivision</code> , 19 , a-1359,
<code>\quotechar</code> , 18 , a-442 ,	<code>\SetTOCIndents</code> , b-55, b-56	a-1620 , a-1622 , a-1623
a-479 , a-484, a-530,	<code>\sfname</code> , d-186 , d-187	<code>\subitem</code> , a-1044
a-542 , a-636, a-676,	<code>\sgtleftxii</code> , a-1053	<code>\subs</code> , d-137 , d-145
a-1097 , a-1123	<code>\showboxbreadth</code> , d-34	<code>\subsubitem</code> , a-1045
<code>\quoted@eschar</code> , a-530,	<code>\showboxdepth</code> , d-34	
a-542 , a-636, a-637,	<code>\ShowFont</code> , d-548	<code>\tableofcontents</code> , a-142 ,
a-676 , a-677	<code>\showlists</code> , d-34	a-143 , a-1303, c-10
 	<code>\SkipFilesAuthors</code> , 8 , a-1496	<code>\TeXbook</code> , 20 , a-1604
<code>\raggedbottom</code> , b-50	<code>\skipgmlonely</code> , 19 , a-1561 ,	<code>\Text@CommonIndex</code> , a-642,
<code>\RecordChanges</code> , 15 ,	c-18	a-643
a-1093 , a-1159 ,	<code>\SliTeX</code> , 19 , a-1600	<code>\Text@CommonIndexStar</code> ,
a-1160 , a-1304, b-59,	<code>\smaller</code> , d-127 , 99	a-642 , a-645
b-60	<code>\smallerr</code> , a-1480, d-131 , 99	<code>\text@indexenvir</code> , a-624,
<code>\reflectbox</code> , a-1611	<code>\smallskipamount</code> , a-65,	a-625 , a-646, a-738,
<code>\relaxen</code> , d-31 , d-31	a-66 , d-512, d-513	a-1664
<code>\relsize</code> , d-98 , d-99,	<code>\smartunder</code> , b-63 , d-143	<code>\text@indexmacro</code> , a-598 ,
d-126, d-127, d-128,	<code>\SortIndex</code> , a-1704	a-622 , a-644, a-733,
d-129, d-130, d-131, 99	<code>\special@index</code> , a-532,	a-1659
<code>\rem@special</code> , f-83, f-101,	a-702 , a-704 , a-761	<code>\Text@Marginize</code> , a-504,
f-106	<code>\SpecialEnvIndex</code> , a-1703	a-661 , a-731, a-736,
<code>\renewcommand</code> , a-1095	<code>\SpecialEscapechar</code> , a-1668	a-743 , a-755 , a-855,
<code>\renewcommand*</code> , a-365,	<code>\SpecialIndex</code> , a-1701	a-1657 , a-1662
a-367 , b-52, d-565	<code>\SpecialMainEnvIndex</code> ,	<code>\Text@MarginizeNext</code> ,
<code>\RequirePackage</code> , a-26,	a-1700	a-850 , a-853, a-854
a-27 , a-31, a-34,	<code>\SpecialMainIndex</code> , a-1699	<code>\Text@UsgEnvir</code> , a-728, a-734
a-35 , a-39, a-1025,		

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmlink.sty, f=gmverb.sty

`\Text@UsgIndex`, a-619, [a-620](#)
`\Text@UsgIndexStar`,
a-619, [a-623](#)
`\Text@UsgMacro`, a-728, [a-729](#)
`\textcolor`, d-562
`\TextCommonIndex`, *13*, [a-640](#)
`\TextCommonIndex*`, [a-640](#)
`\TextIndent`, *16*, [a-51](#),
a-336, a-378
`\textlarger`, [d-128](#)
`\TextMarginize`, *13*, [a-739](#)
`\TextMarginize*`, [a-739](#)
`\textsmaller`, [d-129](#)
`\textstyle`, d-471
`\TextUsage`, *12*, [a-726](#)
`\TextUsage*`, [a-726](#)
`\TextUsgIndex`, *13*, [a-617](#),
a-1702
`\TextUsgIndex*`, [a-617](#)
`\textvisiblespace`, d-181
`\textwidth`, a-1235, a-1339
`\thanks`, a-1441, [a-1455](#),
a-1469, [a-1473](#), a-1550
`\theCodelineNo`, a-1670, *83*
`\thecodelinenum`, a-176,
a-351, a-1672
`\thefilediv`, a-1290,
a-1334, a-1335,
a-1336, a-1347, a-1350
`\theglossary`, [a-1163](#), a-1305
`theglossary`, [a-1163](#)
`theindex`, [a-1028](#)
`\thesection`, b-52
`\thfileinfo`, *20*, [a-1550](#)
`\thr@@`, d-403, d-419
`\title`, [a-1452](#), *c-3*
`\titlesetup`, a-1440,
[a-1463](#), b-58
`\TODO`, [d-518](#)
`\tolerance`, *a-93*
`\tOnLine`, [d-534](#)

`\traceoff`, b-46
`\traceon`, b-45
`\trimmed@everypar`, a-391,
a-393
`\ttverbatim`, *a-126*, a-782,
[f-44](#), f-168, f-177
`\ttverbatim@hook`, f-50,
f-51, f-52
`\twelveand`, [d-156](#)
`\twelvebackslash`, [d-149](#),
d-150
`\twelveclub`, a-443,
a-1098, [f-43](#)
`\twelvelbrace`, [f-16](#), f-27
`\twelvepercent`, a-824,
a-826, a-1226,
a-1228, a-1534,
a-1583, a-1590,
a-1593, [d-153](#), [f-8](#), f-9
`\twelvespace`, a-232, d-37,
d-38, [d-159](#), [f-34](#)
`\twelveunder`, [d-140](#)
`\twocoltoc`, *c-2*, [d-520](#), d-528

`\un@defentryze`, a-521, [a-535](#)
`\un@usgentryze`, a-517, [a-538](#)
`\undeksmallskip`, [d-513](#)
`\UndoDefaultIndexExclusions`,
14, [a-998](#)
`\unex@namedef`, [d-499](#)
`\unex@nameuse`, [d-502](#)
`\ungag@index`, [a-709](#), a-1690
`\UniformSkips`, *16*, [a-59](#),
a-70, a-71, a-72
`uresetlinecount`, *10*, [a-8](#)
`\usage`, [a-1708](#)
`\usecounter`, d-414
`\UsgEntry`, *17*, [a-560](#), a-1708

`\varepsilon`, a-1605, d-471
`\verb`, *19*, a-438, f-57, f-76,
f-80, f-82, [f-175](#), f-231

`\verb*`, f-56, [f-175](#)
`\verb@balance@group`,
f-183, f-184, [f-195](#), f-197
`\verb@egroup`, a-437, [f-184](#),
f-195, f-198
`\verb@eol@error`, f-185
`\verb@eolOK`, [f-188](#), f-192
`\verbatim`, f-115
`verbatim`, [f-115](#)
`verbatim*`, [f-117](#)
`\verbatim@edef`, f-134, f-138
`\verbatim@end`, f-135, f-139
`\verbatim@nolig@list`,
f-45, [f-199](#)
`\verbatimchar`, *18*, a-494,
a-507, a-601, a-1122,
a-1124, [a-1705](#), *85*
`\verbatimhangindent`,
a-49, f-166, [f-171](#), f-172
`\verbeolOK`, *11*, [f-192](#), *121*
`\VerbHyphen`, a-45, [f-10](#), *121*
`\verbhyphen`, [f-9](#), f-12, f-15,
f-24
`\VerbMacrocodes`, *21*, [a-1721](#)
`\VerbT`, b-65, [f-52](#)
`\vs`, [d-181](#), d-182, d-185

`\wd`, d-452, d-455, d-476,
d-481, d-489, d-490
`\Web`, *20*, [a-1603](#)
`\widowpenalty`, *a-92*
`withmarginpar`, *10*, [a-21](#)

`\xdef@filekey`, a-1278,
a-1281, [a-1289](#),
a-1395, a-1398
`\XeTeX`, *20*, [a-1608](#)
`\xiistring`, a-468, a-599,
a-626, a-721, a-724,
a-730, a-735, a-756, [d-36](#)

`\year`, a-1227, a-1229