Grzegorz ‘Natror’ Murzynowski

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

Warszawa, 2006

Contents

a. The gmdoc.sty Package 3
Readme 3
Installation 3
Contents of the gmdoc.zip Archive 4
Compiling the Documentation . . 4
Bonus: base Drivers 4
Introduction 4
The User Interface 5
Used Terms 5
Preparing the Source File 6
The Main Input Commands 7
Package Options 8
The Packages Required 9
Macros for Marking the Macros . 10
Index Ex/Inclusions 12
The DocStrip Directives 13
The Changes History 13
The Parameters 14
The Narration Macros 17
A Queerness of \label 19
doc-Compatibility 19
The Code 20
The Package Options 20
The Dependencies and Preliminaries 22
The Core 25
Numbering (or Not) of the Lines . 33
Spacing with \everypar 33
Life Among Queer EOLs 35
Adjustment of verbatim and \verb 37
Macros for Marking The Macros . 37
Index Exclude List 53
Index Parameters 57
The DocStrip Directives 59
The Changes History 60
The Checksum 64
Macros from ltxdoc 66
\DocInclude and the ltxdoc-Like Setup 66
\SelfInclude 70
Redefinition of \maketitle 72
Miscellanea 75
doc-Compatibility 79
gmdocing doc.dtx 83
Polishing, Development and Bugs . . 85
(No) ⟨*eof*⟩ 85

b. The gmdocc Class For gmdoc Driver Files 86
Intro 86
Usage 86
The Code 87

c. gmdocDoc.tex, The Driver File . 90

d. The gmutils Package 91
Intro 91
Contents of the gmutils.zip Archive 91
\newgif and Other Globals 91
\@ifnextcat 92
\afterfi and Pals 93
Almost an Environment or Redefinition of \begin 94
Improvement of \end 94
From relsize 95
\firstofone and the Queer \catcodes 96
Metasymbols 97
Macros for Printing Macros and Filenames 98
Storing and Restoring the Meanings of CSs 99
Third Person Pronouns 101
To Save Precious Count Registers . . 101
Improvements to mwcls Sectioning Commands 102
Compatibilising Standard and mwcls Sectionings 104
Varia 106

e. The gmiflink Package 109
Introduction, usage 109
Contents of the gmiflink.zip archive 109
The Code 110

f. The gmverb Package 112
Intro, Usage 112
Contents of the gmverb.zip Archive 113
The Code 113
Preliminaries 113
The Breakables 114
Almost-Knuthian \ttverbatim . . 115
The Core: From shortvrb 115
doc- And shortvrb-Compatibility . 120

Change History 121

Index 122

a. The gmdoc.sty Package[1]

December 1, 2006

This is (a documentation of) file gmdoc.sty, intended to be used with LaTeX 2ε as a package for documenting (La)TeX files and to be documented with itself.

Written by Grzegorz 'Natror' Murzynowski,
natror at o2 dot pl

Many thanks to my TeX Guru Marcin Woliński for his TeXnical support.

```
1 ⟨package⟩
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{gmdoc}
4           [2006/12/30␣v0.99a␣a␣documenting␣package␣(GM)]
```

Readme

This package is a tool for documenting of (La)TeX 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 gmglo.ist file, a MakeIndex style for the changes' history, into some texmf/makeindex (sub)directory.

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

[1] This file has version number v0.99a dated 2006/12/30.

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 LaTeX on the gmdocDoc.tex twice, then MakeIndex on the gmdocDoc.idx and gmdocDoc.glo files, and then LaTeX 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 TeX 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.

Bonus: base Drivers

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

source2e_gmdoc.tex
docstrip_gmdoc.tex
doc_gmdoc.tex

oldcomm.sty
gmeometric.sty

These drivers typeset the respective files from the

␣.../texmf-dist/source/latex/base

directory of the TeXLive2005 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 LaTeX 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 TEX 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 authomatic 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 hommage to it[2]. 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 TEXbook's meaning, i.e., a control sequence whose meaning is `\(e/g/x)def`ined. 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) subsituting the corresponding parameter of this macro. (These understandings are according to The TEXbook, I hope: TEX 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 TEX 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 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 `\catcode`s, it's just the `%`. When the file is documented with gmdoc, such a char is re`\catcode`d and its rôle is else: it becomes the **code delimiter**.

A line containing any TEX 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**.

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 :-) .

\heshe

[2] As Grieg's Piano Concerto is a hommage to the Schumann's.

Preparing the Source File

When (LA)TEX with gmdoc.sty package loaded typesets the comment lines, the code delimiter is ommitted. If the comment continues a codeline, the code delimiter is printed. It's done so because ending a TEX code line with a % is often very important and significant. Comments longer than one line are typeset continuously with the code delimiters ommitted.

The user should just write his splendid code and brilliant commentary. In the latter she may use usual (LA)TEX commands. The only requirement is, if an argument is divided
^^B in two lines, to end such a dividing line with ^^B sequence that'll enter the (active) ⟨*char2*⟩ which shall gobble the line end.

Moreover, if he wants to add a meta-comment i.e., a text that doesn't appear in the
^^A code layer nor in the narrative, she may use the ^^A sequence that'll be read by TEX 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 occures inside an \if..., it has to have all \if...s coupled with \elses and \fis.

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
\StraightEOL 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
\QueerEOL (a group), there's a \StraightEOL's counterpart, the \QueerEOL declaration that restores again the queer[3] 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).

As in standard TEXing, 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}
"

[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. (Remember also Alice's in the Wonderland exclamations "What a queer day is today".)

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 TEX 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 LATEX'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.

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 `\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 `\AtBegInputOnce` 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 `\maketitle`s of those files result in the respective headings.

If you prefer more ltxdocish 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' `\maketitle`s 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)).

\ltxLookSetup

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.

\olddocIncludes

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.

\gmdocIncludes

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`.

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.

\ltxPageLayout

If you need to add something at the beginning of the input of file, there's the `\AtBegInput` 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.

\AtBegInput

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

\AtEndInput

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 succeding it (the argument gets wrapped in a macro that's `\relax`ed at the first use). `\AtBegInputOnce`s add up, too.

\AtBegInputOnce

One more input command is `\IndexInput` (the name and idea of effect comes from doc). 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 ⟨*char1*⟩ (`^^A` is OK).

\IndexInput

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.

linesnotnum

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
uresetlinecount 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
countalllines 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
noindex 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)
pageindex 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 888–988.
indexallmacros 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. 12 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 allright 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
withmarginpar classes and in any case provides the options `withmarginpar` and `nomarginpar`. So, in
nomarginpar 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.

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 `codespacesblank` reluctantly. Moreover, if you wish the code spaces to be blank only in
\CodeSpacesBlank 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`.

As the standard LaTeX one, my `\verb` issues an error when a line end occures in its
\verbeolOK 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 TEXbook's format, gmverb provides a macro

\dekclubs `\dekclubs` that expands to the example displayed above.

Be careful because such active chars may interfere with other things, e.g., the `|` with the vertical line marker in `tabular`s and with the tikz package. If this happens, you can

\DeleteShortVerb 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 (LA)TEX 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.

To typeset the index and the change history in balanced columns gmdoc uses the

multicol 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 exluding some macros from being indexed.)

The next concept (also taken from doc) is to ditinguish 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. 9).

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).

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.

\MakePrivateLetters

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 looks 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.

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`.

\Describe

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

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.

\CodeMarginize
\TextMarginize

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

\marginpartt

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.

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.

\gmdmarginpar

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`.

\CodeDefIndex
\CodeUsgIndex

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`.

\CodeCommonIndex*

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.

\TextUsgIndex
\TextCommonIndex

macro
environment

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 TEXnically 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 `evironment`.)

Index Ex/Inclusions

\DoNotIndex

It's understandable[4] 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, TEXnically 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 `\DoNotIndex`es as you wish (about half as many as many CSs may be declared, because for each CS ex-

[4] After reading doc's documentation ;-) .

cluded 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.

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

```
\DoIndex{\par \@@par \endgraf}
```

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.

\DefaultIndexExclusions
\UndoDefaultIndexExclusions

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 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` [command] encloses the ⟨*date*⟩ in parentheses and appends the ⟨*text*⟩ to form the printed entry in such a change history [... obsolete remark ommitted].

To cause the change information to be written out, include `\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 `\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 `\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 [... **gmglo.ist** for gmdoc]. The `\GlossaryMin`, `\GlossaryPrologue` and `\GlossaryParms` macros are analagous to the `\Index...` versions [see sec. The Parameters p. 16]. (The LaTeX 'glossary' mechanism is used for the change entries.)"

\RecordChanges
\PrintChanges
\GlossaryMin
\GlossaryPrologue
\GlossaryParms

In gmdoc `\changes` is `\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: `\changes` takes care of setting them properly. To be precise,
\MakeGlossaryControls `\changes` executes `\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 `\changes` uses them itself), use rather `\quotechar`.

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

`\ChangesStart{`⟨*version*⟩`}{`⟨*year*⟩`/`⟨*month*⟩`/`⟨*day*⟩`}`

declaration.

If the `ChangesStartDate` is set to a date contemporary to TEX i.e., not earlier than September 1982[5], then a note shall appear at the beginning of the changes history that informs the reader of ommitting the earlier changes entries.

If the date stored in `ChangesStartDate` is earlier than TEX, no notification of ommitting 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{`⟨*version?*⟩`}{1000/00/00}`

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

While scanning the CSs in the code, gmdoc counts them and prints the information
\CheckSum about their number on the terminal and in .log. Moreover, you may declare `\CheckSum{%` ⟨*number*⟩`}` before the code and TEX 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.

There it is provided as a tool for testing whether the file is corrupted. My TEX 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

```
% \chschange{v0.98j}{2006/10/19}{4372}
% \chschange{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 TEX 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 shrinkability). Subsequent blank code lines do not increase this space.

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),

\CodeTopsep 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.

The \stanzaskip's value is assigned also to the display skips and to \topsep. This \UniformSkips is done with the \UniformSkips declaration executed by default. If you want to change \NonUniformSkips 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}.)

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

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 \nostanza want not to add such a glue exceptionally. Then there's the \nostanza command.

\CodeIndent The TEX 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 '□' given by the \EOFMark macro. If you don't like such an ending, you should end the source \EOFMark file with the \NoEOF macro in a comment, i.e., \NoEOF

%⟨*some text, why not*⟩\NoEOF

This macro redefines \EOFMark 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 \CodeDelim 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 deafult.)

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

The plain (unstarred) version of \CodeDelim changes also the verb 'hyphen', the char appearing at the verbatim line breaks that is. If you don't want to change the verb 'hyphen', use \CodeDelim*.

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, \CodeEscapeChar 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.)

[5] DEK in *TEX The Program* mentions that month as of TEX Version 0 release.

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
\MakePrivateLetters 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.

\AddtoPrivateOthers 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 `\MakeShortVerb`ed. (I don't see a need of removing chars from this list, but if you do, please let me know.)

\LineNumFont 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
codelinenum called `codelinenum` so the macro printing it is `\thecodelinenum`. By default we don't change its LaTeX's definition which is equivalent `\arabic{codelinenum}`.

\IndexPrefix Three more parameter macros, are `\IndexPrefix`, `\EntryPrefix` and `\HLPrefix`.
\EntryPrefix All three are provided with the account of including multiple files in one document.
\HLPrefix 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`.

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

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

The text put at the beginning of the index is declared with a one-argument
\IndexPrologue `\IndexPrologue`. Its default text at current index option you may admire on page 122. 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 `\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.

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.

\IndexLinksBlack

Other index parameters are set with the `\IndexParms` macro defined in line 1033 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 LaTeX's `\g@addto@macro` provided by gmutils.)

\IndexParms

\gaddtomacro

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 TeX what characters of a scanned CS name it should quote before writing it to the .idx file.

\actualchar

\quotechar

\levelchar

\encapchar

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`.

\verbatimchar

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

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.).

\StopEventually

\Finale

\AlsoImplementation

\OnlyDescription

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

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`.

\verb

\cs

\env

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

\pk

\file `\file` command intended for the filenames.

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

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).

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

gmlonely \skipgmlonely 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 $\mathcal{AMS}$-TEX,
\BibTeX BIBTEX,
\SliTeX SLITEX,
\PlainTeX PLAIN TEX,
\Web WEB,
\TeXbook The TEXbook,
\eTeX ε-TEX,
\pdfeTeX pdfεTEX
\pdfTeX pdfTEX
\XeTeX XETEX (the first `E` will be reversed if 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.

[6] See gmutils for some subtle details.

\stanza
\chunkskip

To make a vertical space to separate some piece of text you are given two macros: `\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
\filedate
\fileversion
\fileinfo

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

\ProvideFileInfo
\ProvideSelfInfo

Since you may wish to process also files that you don't load, there are commands `\ProvideFileInfo{`⟨*file name with extension*⟩`}[`⟨*info*⟩`]` and `\ProvideSelfInfo[`⟨*info*⟩`]`. (⟨*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 `\ref`s will be out of those groups), so a `\ref`erence in the narrative will point at the last code line not the last section, *unlike* in the standard LaTeX.

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
\DocInclude

The main input commands compatible with doc are `\OldDocInput` and `\DocInclude`, the latter however only in the `\olddocIncludes` declaration's scope.

\olddocIncludes
macrocode

Within their scope/argument the `macrocode` environments behave as in doc, i.e. they 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 te '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`.)

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). 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`.)

oldmc

\VerbMacrocodes

For a more detailed discussion of what is doc-compatible and how, see the code section doc-Compatibiliy.

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 tomake a `\section` or `\emph`asise, 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.

```
\newif\if@linesnotnum
\DeclareOption{linesnotnum}{\@linesnotnumtrue}
```

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

```
\newif\if@uresetlinecount
\DeclareOption{uresetlinecount}{\@uresetlinecounttrue}
```

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

```
\newif\if@countalllines
\DeclareOption{countalllines}{\@countalllinestrue}
```

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

```
\newif\if@noindex
\DeclareOption{noindex}{\@noindextrue}
\newif\if@pageindex
\DeclareOption{pageindex}{\@pageindextrue}
```

It would be a great honour to me if someone would like to document LaTeX 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).

```
\newif\if@indexallmacros
\DeclareOption{indexallmacros}{\@indexallmacrostrue}
```

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

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

This switch is copied from mwbk.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`:

```
\@ifclassloaded{article}{\@marginparsusedtrue}{}
\@ifclassloaded{report}{\@marginparsusedtrue}{}
\@ifclassloaded{book}{\@marginparsusedtrue}{}
```

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

```
\DeclareOption{withmarginpar}{\@marginparsusedtrue}
\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).

```
\newif\if@codespacesblank
\DeclareOption{codespacesblank}{\@codespacesblanktrue}
\ProcessOptions
```

The Dependencies and Preliminaries

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

```
\RequirePackage{gmutils}
```

A standard package for defining colours,

```
\RequirePackage{color}
```

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

```
\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.

```
\@ifpackageloaded{hyperref}{\hypersetup{colorlinks=true,
    linkcolor=deepblue,␣urlcolor=blue,␣filecolor=blue}}{%
  \RequirePackage[colorlinks=true,␣linkcolor=deepblue,␣urlcolor=blue,
  filecolor=blue,␣pdfstartview=FitH,␣pdfview=FitBH,
  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.

```
\RequirePackage{gmiflink}
```

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

```
\RequirePackage{gmverb}
```

```
\if@noindex
  \AtBeginDocument{\gag@index}% for the latter macro see line 704.
\else
   \RequirePackage{makeidx}\makeindex
\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\`⟨*char*⟩ declaration provided by gmverb.

`\CodeDelim`
```
\def\CodeDelim{\@ifstar{\Code@Delim}{\Code@Delim@St}}
```

```
\def\Code@Delim#1{%
  {\escapechar\m@ne
```

`\code@delim`
```
    \expandafter\gdef\expandafter\code@delim\expandafter{\string#1}}}
```

```
\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:

```
\CodeDelim\%
```

We'll play with `\everypar`, a bit, and if you use such things as the `{itemize}` environment, an error would occure 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`

```
\newtoks\gmd@preverypar
\newcommand*\settexcodehangi{%
  \hangindent=\verbatimhangindent␣\hangafter=\@ne}% we'll use it in the inline
        comment case. \verbatimhangindent is provided by the gmverb package
        and = 3 em by default.
\@ifdefinable\@@settexcodehangi{\let\@@settexcodehangi=%
      \settexcodehangi}
```

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

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

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

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

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

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

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

\stanzaskip
```
\stanzaskip=\medskipamount
\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

```
\newskip\CodeTopsep
\newskip\MacroTopsep
```

And let's set them. For the æsthetical minimality[7] 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 \CodeTopsep \MacroTopsep
```
\def\UniformSkips{%
  \CodeTopsep=\stanzaskip
  \MacroTopsep=\stanzaskip
  \abovedisplayskip=\stanzaskip
%\abovedisplayshortskip remains untouched as it is 0.0 pt plus 3.0 pt by default.
  \belowdisplayskip=\stanzaskip
  \belowdisplayshortskip=.5\stanzaskip% due to DEK's idea of making the
      short below display skip half of the normal.
```

[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)

```
  \advance\belowdisplayshortskip␣by\smallskipamount
  \advance\belowdisplayshortskip␣by-1\smallskipamount%
  %
  \topsep=\stanzaskip
  \partopsep=\z@
}
```

We advance `\belowdisplayshortskip` forth and back to give it the `\smallskipamount`'s shrink- and stretchability components.

We make it the default,

```
\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.

```
\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
```
\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
```
\newcommand*\chunkskip{%
  \skip0=\MacroTopsep
  \if@codeskipput\advance\skip0␣by-\CodeTopsep\fi
  \par\addvspace{\skip0}\@codeskipputgtrue}
```

And, for a smaller part of text,

\stanza
```
\newcommand*\stanza{%
  \skip0=\stanzaskip
  \if@codeskipput\advance\skip0␣by-\CodeTopsep\fi
  \par\addvspace{\skip0}\@codeskipputgtrue}
```

Since the stanza skips are inserted automatically most often (cf. lines 190, 358, 199, 317, 374), sometimes you may need to forbid them.

\nostanza
```
\newcommand*\nostanza{%
  \@codeskipputgtrue\@afternarrgfalse\@aftercodegtrue}%
```

In the 'code to narration' case the first switch is enough but in the countercase '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
\@newlinegfalse
\if@newline
```
\newgif\if@newline
```

`\newgif` is `\newif` with global effect i.e., it defines `\...gtrue` and `\...gfalse` switchers that switch respective Boolean switch *globally*. See gmutils package for details.

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

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

This switch will be falsified at the first char of a code line. (We need a switch independent of the one indicationg 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
\Doc@Input

```
\newcommand*\DocInput{\bgroup\@makeother\_\Doc@Input}
\begingroup\catcode`\^^M=\active%
\firstofone{\endgroup%
  \newcommand*{\Doc@Input}[1]{\egroup\begingroup%
    \edef\gmd@inputname{#1}% we'll use it in some notifications.
    \let\gmd@currentlabel@before=\@currentlabel% we store it 'cause we'll do
        \xdefs of \@currentlabel to make proper references to the line numbers
        so we want to restore current \@currentlabel after our group.
    \gmd@setclubpenalty% we wrapped the assignment of \clubpenalty in a macro
        because we'll repeat it twice more.
    \@clubpenalty\clubpenalty␣\widowpenalty=3333␣% Most paragraphs of the
        code will be one-line most probably and many of the narration, too.
    \tolerance=1000␣% as in doc.
    \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.
    \catcode`\^^M=\active%
    \expandafter\@makeother\code@delim%
    \gmd@resetlinecount% due to the option uresetlinecount we reset the
        linenumber counter or do nothing.
    \@beginputhook% my first use of it is to redefine \maketitle just at this point
        not globally.
    \everypar=\expandafter{\expandafter\@codetonarrskip\the\everypar}%
^^M \let^^M=\gmd@textEOL%
    \edef\gmd@guardedinput{%
      \noexpand\@@input␣#1\relax% it's true TEX's \input.
      \noexpand\EOFMark% to pretty finish the input, see line 148.
      \noexpand^^M\code@delim%
    }% we add guardians after \inputing a file; somehow an error occurred without
        them.
    \catcode`\%=9␣% for doc-compatibility.
    \setcounter{CheckSum}{0}% we initialize the counter for the number of the
        escape chars (the assignment is \global).
    \expandafter\expandafter\expandafter^^M\gmd@guardedinput%
    \par%
    \@endinputhook% 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.
    \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.
    \endgroup%
  }% end of \Doc@Input's definition.
}% 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 115 `\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.

116 `\ifhmode\@afternarrgtrue\@codeskipputgfalse\fi%` being in the horizontal mode means we've just typeset some narration so we turn the respective 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).

117 `\@newlinegtrue%` to `\refstep` the lines' counter at the proper point.

118 `\@dsdirgtrue%` to handle the DocStrip directives.

119 `\expandafter\@trimandstore\the\everypar\@trimandstore%` we store the previous value of `\everypar` register to restore it at a proper point. See line 387 for the details.

120 `\begingroup%`

121 `\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.

122 `\aftergroup\gmd@setclubpenalty%`

123 `\let\par\@@par%` inside the verbatim group we wish `\par` to be genuine.

124 `\ttverbatim%` it does `\tt` and makes specials other or `\active`-and-breakable.

125 `\gmd@DoTeXCodeSpace%`

126 `\@makeother\|%` 'cause `\ttverbatim` doesn't do that.

127 `\MakePrivateLetters%` see line 449.

128 `\expandafter\@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.

```
129   \expandafter\@ifnextchar\expandafter{\code@delim}{%
130     \gmd@continuenarration}{%
131     \gmd@dolspaces%
```

it will launch `\gmd@typesettexcode`.

```
132   }%
```

end of `\@ifnextchar`'s else.

```
133 }%
```

end of `\gmd@textEOL`'s definition.

```
134 \def\gmd@setclubpenalty{\clubpenalty=3333␣}
```

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

\AtEndInput 135 `\def\AtEndInput{\g@addto@macro\@endinputhook}`

\@endinputhook 136 `\def\@endinputhook{}`

Simili modo

\AtBegInput 137 `\def\AtBegInput{\g@addto@macro\@beginputhook}`

\@beginputhook 138 `\def\@beginputhook{}`

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

\tableofcontents 139 `\AtBegInput{\let\gmd@@toc\tableofcontents`

```
140   \def\tableofcontents{%
```

```
\@ifQueerEOL{\StraightEOL\gmd@@toc\QueerEOL}{\gmd@@toc}}}
```

As you'll learn from lines 411 and 406, 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 `^^M` is the rôle of `\@ifQueerEOL`:

\@ifQueerEOL
```
\long\def\@ifQueerEOL#1#2{%
  \ifnum\the\catcode`\^^M=\active␣\afterelsefi#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 `\AtBegInputOnce`s, they add up.

\gmd@ABIOnce
```
\@emptify\gmd@ABIOnce
\AtBegInput\gmd@ABIOnce
```

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

Many tries of finishing the input cleanly led me to setting the guardians as in line 104 and to

\EOFMark
```
\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 1689.

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

```
\begin{obeyspaces}%
\gdef\gmd@DoTeXCodeSpace{%
\obeyspaces\let␣=\breakabletwelvespace}%
```

\CodeSpacesBlank
```
\gdef\CodeSpacesBlank{%
\let\gmd@DoTeXCodeSpace\gmobeyspaces%
\let\gmd@texcodespace=\␣}% the latter \let is for the \if...s.
```

\CodeSpacesSmall
```
\gdef\CodeSpacesSmall{%
\def\gmd@DoTeXCodeSpace{%
\obeyspaces\def␣{\,\hskip\z@}}%
\def\gmd@texcodespace{\,\hskip\z@}}%
\end{obeyspaces}
```

How the continuing of the narration should look like?

```
\def\gmd@continuenarration{%
  \endgroup
  \gmd@countnarrationline% see below.
  \expandafter\@trimandstore\the\everypar\@trimandstore
  \everypar=\expandafter{\expandafter\@codetonarrskip\the\everypar}%
  \expandafter\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 129. And in line 242 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.

```
\if@countalllines
  \def\gmd@countnarrationline{%
    \if@newline
      \grefstepcounter{codelinenum}\@newlinegfalse%
```
the `\grefstepcounter` macro, defined in gmverb, is a global version of `\refstepcounter`, observing the redefinition made to `\refstepcounter` by hyperref.
```
      \everypar=\expandafter{%
        \expandafter\@codetonarrskip\the\gmd@preverypar}%
```
the `\hyperlab-% el@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.
```
      \hyperlabel@line
      {\LineNumFont\thecodelinenum}\,\ignorespaces
    \fi}%
\else
  \@emptify\gmd@countnarrationline%
\fi
```

And typesetting the TEX code?

```
\begingroup\catcode`\^^M=\active%
\firstofone{\endgroup%
  \def\gmd@typesettexcode{%
    \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`.
```
    \everypar={\@@settexcodehangi}%
```
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     \def^^M{%
        \@newlinegtrue%
```
to `\refstep` the counter in proper place.
```
        \@dsdirgtrue%
```
to handle the DocStrip directives.
```
        \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.
```
        \ifhmode\par\@codeskipputgfalse\else%
          \if@codeskipput%
          \else\addvspace{\stanzaskip}\@codeskipputgtrue%
          \fi%
```
if we've just met a blank (code) line, we insert a `\stanzaskip` glue.
```
        \fi%
        \prevhmodegfalse%
```
we want to know later that now we are in the vmode.
```
        \@ifnextchar{\gmd@texcodespace}{%
          \@dsdirgfalse\gmd@dolspaces}{\gmd@charbychar}%
    }% end of ^^M's definition.
    \let\gmd@texcodeEOL=^^M%
```
for further checks inside `\gmd@charbychar`.
```
    \raggedright\leftskip=\CodeIndent%
    \if@aftercode\gmd@nocodeskip1{iaC}\else\if@afternarr%
        \if@codeskipput\else\gmd@codeskip1\@codeskipputgtrue%
              \@aftercodegfalse\fi%
```

```
    \else\gmd@nocodeskip1{naN}\fi\fi% if now we are switching from the narration into the code, we insert a proper vertical space.
    \@aftercodegtrue\@afternarrgfalse%
    \ifdim\gmd@ldspaceswd>\z@% and here the leading spaces.
      \leavevmode\@dsdirgfalse%
      \if@newline\grefstepcounter{codelinenum}\@newlinegfalse%
      \fi%
      \printlinenumber% if we don't want the lines to be numbered, the respective option \lets this CS to \relax.
      \hyperlabel@line%
      \mark@envir% index and/or marginize an environment if there is some to be done so, see line 656.
      \hskip\gmd@ldspaceswd%
      \advance\hangindent␣by\gmd@ldspaceswd%
      \xdef\settexcodehangi{%
        \noexpand\hangindent=\the\hangindent% and also set the hanging indent setting 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.
        \noexpand\hangafter=1\space}%
    \else%
      \glet\settexcodehangi=\@@settexcodehangi%
        % \printlinenumber here produced line numbers for blank lines which is what we don't want.
    \fi% of \ifdim
    \gmd@ldspaceswd=\z@%
    \prevhmodegfalse% we have done \par so we are not in the hmode.
    \@aftercodegtrue% we want to know later that now we are typesetting a codeline.
    \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.
  }%
}% 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.)

```
\newlength\gmd@spacewd% to store the width of a (leading) ␣12.
\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:

```
\let\gmd@texcodespace=\breakabletwelvespace
```

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

```
\def\gmd@dolspaces{%
  \ifx\gmd@texcodespace\@let@token
    \@dsdirgfalse
    \afterelsefi\settowidth{\gmd@spacewd}{\twelvespace}%
```

```
    \gmd@ldspaceswd=\z@
    \gmd@eatlspace%
  \else\afterfi% about this smart macro and other of its family see gmutils sec. 3.
    \par\gmd@typesettexcode
  \fi}
```

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

```
\def\gmd@eatlspace#1{%
  \ifx\gmd@texcodespace#1%
    \advance\gmd@ldspaceswd␣by\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.
    \afterelseiffifi\gmd@eatlspace
  \else
    \if\code@delim\noexpand#1%
      \gmd@ldspaceswd=\z@
      \gmd@continuenarration#1%
    \else␣\afterfifi\gmd@typesettexcode#1%
    \fi
  \fi}%
```

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

\prevhmodegtrue
\prevhmodegfalse
\ifprevhmode

```
\newgif\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

```
\newcommand*\gmd@charbychar[1]{%
  \ifhmode\prevhmodegtrue
  \else\prevhmodegfalse\fi
  \if\code@delim\noexpand#1%
    \afterelseiffifi\gmd@percenthack% to typeset % if a comment continues
        the codeline.
    \endgroup%
    \gmd@checkifEOLmixd% to see if next is ^^M and then do \par.
  \else% i.e., we've not met the code delimiter
    \if\code@escape@char\noexpand#1%
      \@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.
      \afterelsefifi\gmd@countheline#1\scan@macro
    \else
      \afterfifi\gmd@EOLorcharbychar#1%
    \fi
  \fi}
```

One more inner macro because `^^M` in TEX code wants to peek at the next char and possibly launch `\gmd@charbychar`. We deal with counting the lines thorougly. 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.

```
\def\gmd@EOLorcharbychar#1{%
```

```
  \ifx\gmd@texcodeEOL#1%
    \if@newline
      \if@countalllines\global\advance\c@codelinenum␣by\@ne
      \@newlinegfalse\fi
    \fi
    \afterelsefi#1% here we print #1.
  \else% i.e., #1 is not a (very active) line end,
    \afterfi
    \gmd@countthelineA#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.
 \fi}
\def\gmd@countthelineA{%
  \ifvmode
    \if@newline
      \grefstepcounter{codelinenum}\@newlinegfalse
      \hyperlabel@line
    \fi
    \printlinenumber
    \mark@envir
  \else
    \if@newline
      \grefstepcounter{codelinenum}\@newlinegfalse
      \hyperlabel@line
    \fi
  \fi}
```

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

```
\def\gmd@percenthack{%
  \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.
  \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.
  \fi}
\def\gmd@narrcheckifds@ne#1{%
  \@dsdirgfalse\@ifnextchar<{%
    \expandafter\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.

```
\def\gmd@checkifEOL{%
  \gmd@countnarrationline
  \everypar=\expandafter{\expandafter\@codetonarrskip% we add the macro
        that'll insert a vertical space if we leave the code and enter the narration.
    \the\gmd@preverypar}%
  \@ifnextchar{\gmd@textEOL}{%
    \@dsdirgfalse\par\ignorespaces}{\gmd@narrcheckifds}}%
```

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

```
\def\gmd@narrcheckifds{%
  \@dsdirgfalse\@ifnextchar<{%
    \expandafter\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.

```
\def\gmd@checkifEOLmixd{%
  \gmd@countnarrationline
  \everypar=\expandafter{\expandafter\@codetonarrskip\the%
        \gmd@preverypar}%
  \@afternarrgfalse\@aftercodegtrue
  \ifhmode\@codeskipputgfalse\fi
  \@ifnextchar{\gmd@textEOL}{%
    {\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., 305.
    \prevhmodegfalse
    \gmd@endpe\ignorespaces}{%
```

If a codeline ends with `%` (`prevhmode` == True) first `\gmd@endpe` sets the parameters at the TEX 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@endpe`s do the same and `\par` between them does nothing.

```
    \def\par{%
      \ifhmode% (I added this \ifhmode as a result of a heavy debug.)
        \@@par
        \if@afternarr
          \if@aftercode
            \if@codeskipput\else\gmd@codeskip2\@aftercodegfalse%
                  \@codeskipputgtrue\fi
          \else\gmd@nocodeskip2{naC}%
          \fi
        \else\gmd@nocodeskip2{naN}%
        \fi
        \prevhmodegfalse\gmd@endpe% when taken out of \ifhmode, this line
              caused some codeline numbers were typeset with \leftskip = 0.
        \everypar=\expandafter{%
          \expandafter\@codetonarrskip\the\gmd@preverypar}%
        \let\par\@@par%
      \fi}%
    \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.

```
\def\gmd@endpe{%
  \ifprevhmode
    \settexcodehangi%ndent
    \leftskip=\CodeIndent
```

```
  \else
    \leftskip=\TextIndent
    \hangindent=\z@
    \everypar=\expandafter{%
      \expandafter\@codetonarrskip\the\gmd@preverypar}%
  \fi}
```

Numbering (or Not) of the Lines

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

```
\if@uresetlinecount% with uresetlinecount option...
  \@relaxen\gmd@resetlinecount% ... we turn resetting the counter by \DocInput
        off...
  \newcommand*\resetlinecountwith[1]{%
    \newcounter{codelinenum}[#1]}% ... and provide a new declaration of the
          counter.
\else% With the option turned off...
  \newcounter{DocInputsCount}%
  \newcounter{codelinenum}[DocInputsCount]% ... we declare the \DocInputs'
        number counter andthe codeline counter to be reset with stepping of it.
  \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.
\fi
```

DocInputsCount
codelinenum
\gmd@resetlinecount

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

```
\newcommand*\printlinenumber{%
  \leavevmode\llap{\rlap{\LineNumFont$\phantom{999}$\llap{%
        \thecodelinenum}}%
    \hskip\leftskip}}
\def\LineNumFont{\normalfont\tiny}
\if@linesnotnum\@relaxen\printlinenumber\fi
\newcommand*\hyperlabel@line{%
  \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.
  \else
    \raisebox{2ex}[1ex][\z@]{\gmhypertarget[clnum.%
      \HLPrefix\arabic{codelinenum}]{}}%
  \fi}
```

\LineNumFont

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.

```
\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

```
\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.

```
\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 ;-).

```
\if1␣1
  \renewcommand*\gmd@codeskip[1]{%
    \hbox{\rule{1cm}{3pt}␣#1!!!}}
  \renewcommand*\gmd@nocodeskip[2]{%
    \hbox{\rule{1cm}{0.5pt}␣#1:␣#2␣}}
\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
\@aftercodegfalse
\if@aftercode

```
\newgif\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
\@afternarrgfalse
\if@afternarr

```
\newgif\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

```
\long\def\@codetonarrskip{%
  \if@codeskipput\else
    \if@afternarr\gmd@nocodeskip4{iaN}\else
      \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.

```
        {\setbox0=\lastbox}%
```

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

```
        \gmd@codeskip4\@codeskipputgtrue
        \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.)
        \else\gmd@nocodeskip4{naC}%
      \fi%
    \fi
  \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

```
\long\def\@trimandstore#1\@trimandstore{%
  \def\@trimandstore@hash{#1}%
  \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.
    \gmd@preverypar={}%
  \else
    \afterfi\expandafter\@trimandstore@ne\the\everypar\@trimandstore
  \fi}
\long\def\@trimandstore@ne#1#2\@trimandstore{%
  \def\trimmed@everypar{#2}%
  \ifx\@codetonarrskip#1%
    \gmd@preverypar=\expandafter{\trimmed@everypar}%
  \else
    \gmd@preverypar=\expandafter{\the\everypar}%
  \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:

```
\catcode`\^^B=\active% note we re\catcode ⟨char2⟩ globally, for the entire
    document.
\bgroup\catcode`\^^M=\active%
\firstofone{\egroup%
  \def\QueerCharTwo{\long\def^^B##1^^M{\ignorespaces}}}
\QueerCharTwo
\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.
```

^^B

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 comment[2] 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.

```
\catcode`\^^A=\active% note we re\catcode ⟨char1⟩ globally, for the entire document.
\bgroup\catcode`\^^M=\active%
\firstofone{\egroup%
  \def\QueerCharOne{\long\def^^A##1^^M{\ignorespaces^^M}}}
\QueerCharOne
\AtBegInput{\@ifQueerEOL{\catcode`\^^A\active}\QueerCharOne}% see note after line 399.
```

^^A

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

```
\def\StraightEOL{%
  \catcode`\^^M=5\relax
  \catcode`\^^A=14\relax
  \catcode`\^^B=14\relax
  \def\^^M{\␣}}
```

\QueerEOL

```
\def\QueerEOL{%
  \catcode`\^^M=\active
  \catcode`\^^A=\active
  \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.)
  \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 uwelcome 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`:

```
\def\gmd@parfixclosingspace{{%
    \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.

```
\newskip\gmd@closingspacewd
\newcommand*\gmd@setclosingspacewd{%
  \global\gmd@closingspacewd=\fontdimen2\font%
    plus\fontdimen3\font␣minus\fontdimen4\font\relax}
```

See also line 186 to see what we do in the codeline case when no closing space is added.

And one more detail:

```
\bgroup\catcode`\^^M=\active%
\firstofone{\egroup%
  \def\^^M{\␣\expandafter\ignorespaces^^M}}
```

To make `\^^M` behave properly. If this definition was ommitted, it 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.

```
\let\gmd@bslashEOL=\^^M
```

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:

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

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

```
\AtBegInput{\long\def\check@percent#1{%
    \expandafter\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
```
\def\AddtoPrivateOthers#1{%
  \expandafter\def\expandafter\doprivateothers\expandafter{%
    \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.

```
\begingroup
\obeylines%
\AtBegInput{\def\gm@verb@eol{\obeylines%
    \def^^M{\verb@egroup\@latex@error{%
        \noexpand\verb␣ended␣by␣end␣of␣line}%
      \@ifQueerEOL{\gmd@textEOL}{\@ehc}}}}%
\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
\quotechar
\encapchar
\levelchar
```
\edef\actualchar{\string␣@}
\def\quotechar{"}
\edef\encapchar{\twelveclub}
\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 `\edef`s that process them further are postponed till `\begin{document}`.

\CodeEscapeChar
```
\def\CodeEscapeChar#1{%
  \begingroup
  \escapechar\m@ne
```
\code@escape@char
```
  \xdef\code@escape@char{\string#1}%
  \endgroup}
```

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

```
\CodeEscapeChar\\
```

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

```
\@ifundefined{MakePrivateLetters}{%
```
\MakePrivateLetters
```
  \def\MakePrivateLetters{\makeatletter\catcode`\*=11␣}}{}
```

A tradition seems to exist to write about e.g., 'command `\section` and command `\section*`' and such an uderstanding 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
```
\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 724. So let's define the list:

\doprivateothers
```
\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 ommit some lines irrelevant with my approach.

\scan@macro
```
\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 $_{12}$ so we may just have it printed during main scan char by char, i.e., in the lines 268 and 271.

So, we step the checksum counter first,

```
  \step@checksum%
```
(see line 1193 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 $_{11}$ surely would be one of those perversities. Therefore I feel safe to take the character `a` as a benchmark letter.

```
  \ifcat␣a\noexpand#1%
    \quote@char#1%
    \xdef\macro@iname{\maybe@quote#1}%
```
global for symmetry with line 463.
```
    \xdef\macro@pname{\string#1}%
```
we'll print entire name of the macro later.

We `\string` it here and in the lines 466 and 472 to be sure it is whole $_{12}$ for easy testing for special indexentry formats, see line 508 etc.

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

Another reasonable `\catcode`s assumption seems to be that the digits are $_{12}$. 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 $_{12}$.

```
    {\uccode`9=`#1%
      \uppercase{\xdef\macro@iname{9}}%
    }%
    \quote@char#1%
    \xdef\macro@iname{\maybe@quote\macro@iname}%
    \xdef\macro@pname{\xiistring#1}%
    \afterfi␣\finish@macroscan
  \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 $_{12}$.

Now, let's explain some details, i.e., let's define them. We call the following macro having known `#1` to be $_{11}$.

```
\def\continue@macroscan#1{%
  \quote@char#1%
  \xdef\macro@iname{\macro@iname␣\maybe@quote#1}%
  \xdef\macro@pname{\macro@pname␣\string#1}%
  \@ifnextcat{a}{\continue@macroscan}{\finish@macroscan}%
}
```

The `\@ifnextcat` macro is defined analogously to `\@ifnextchar` but the test done by it is not `\ifx` but `\ifcat`.

```
\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.
    \uppercase{%
      \@ifismember␣9\of␣\indexcontrols\glet\maybe@quote\quotechar%
      \else\g@emptify\maybe@quote%
      \fi}%
  }}
```

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
```
\AtBeginDocument{\xdef\indexcontrols{%
    \bslash\levelchar\encapchar\actualchar\quotechar}}
```
\@ifismember
```
  \long\def␣\@ifismember#1\of#2{%
    \long\def\in@@##1#1##2\in@@{%
      \if\noexpand^^A##2\noexpand^^A\afterelsefi\gmd@@iffalse
      \else\afterfi\gmd@@iftrue
      \fi}%
    \expandafter\in@@#2#1\in@@}%
```

To hide the Booleans from TEX in some contexts define

\gmd@@iftrue
\gmd@@iffalse

```
\def\gmd@@iftrue{\iftrue}
\def\gmd@@iffalse{\iffalse}
```

A try to \let instead of \def resulted with the 'incomplete if' error: TEX 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\catcode s 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 secong 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.

```
\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 countercrank way because we wish to use more general indexing macro.

```
  \if\verbatimchar\macro@pname%
```

it's important that \verbatimchar comes before 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 $:

```
    \def\im@firstpar{[$]}%
  \else\def\im@firstpar{}\fi
  \expandafter␣\index@macro\im@firstpar\macro@iname\macro@pname
  \maybe@marginpar\macro@pname
  \macro@pname
```

```
  \gmd@charbychar
}
```

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.

```
\def\maybe@marginpar#1{%
  \@ifundefined{gmd/2marpar/#1}{}{%
    \expandafter\Text@Marginize\expandafter{\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.
    \expandafter\g@relaxen\csname␣gmd/2marpar/#1\endcsname% we reset the
        switch.
  }}
```

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 macro's name, usually `\macro@pname`, built with `\string`ing every char in lines 458, 466 and 472. `#3` is used only to test if the entry should be specially formatted.

\index@macro

```
\newcommand*\index@macro[3][\verbatimchar]{{%
    \@ifundefined{gmd/iexcl/#3}%
    {% #3 is not excluded from index
      \@ifundefined{gmd/defentry/#3}%
      {% #3 is not def entry
        \@ifundefined{gmd/usgentry/#3}%
        {% #3 is not usg entry
          \edef\kind@fentry{\CommonEntryCmd}}%
        {% #3 is usg entry
          \def\kind@fentry{UsgEntry}%
          \un@usgentryze{#3}}%
      }%
      {% #3 is def entry
        \def\kind@fentry{DefEntry}%
        \un@defentryze{#3}%
      }% of gmd/defentry/ test's 'else'
      \if@pageindex\@pageinclindexfalse\fi% should it be here or there? Defi-
          nitely here because we'll wish to switch the switch with a declaration.
      \if@pageinclindex
        \edef\IndexRefCs{gmdindexpagecs{\HLPrefix}{\kind@fentry}{%
            \EntryPrefix}}%
      \else
        \edef\IndexRefCs{gmdindexrefcs{\HLPrefix}{\kind@fentry}{%
            \EntryPrefix}}%
      \fi
      \edef\@tempa{\IndexPrefix#2\actualchar%
        \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).
        \encapchar\IndexRefCs}%
```

```
      \expandafter\special@index\expandafter{\@tempa}% We give the indexing macro the argument 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.)
    }{}% closing of gmd/iexcl/ test.
    }}
```

\un@defentryze
```
\def\un@defentryze#1{%
  \expandafter\g@relaxen\csname␣gmd/defentry/#1\endcsname
  \g@relaxen\last@defmark}% we care to clear the last definition marker checked by \changes.
```

\un@usgentryze
```
\def\un@usgentryze#1{%
  \expandafter\g@relaxen\csname␣gmd/usgentry/#1\endcsname}
\@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
\@pageinclindexfalse
\if@pageinclindex
\quoted@eschar
```
\newif\if@pageinclindex
\newcommand*\quoted@eschar{\quotechar\bslash}% we'll redefine it when indexing an environment.
```

Let's initialize \IndexPrefix

```
\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.

```
\if@linesnotnum\@pageindextrue\fi
\AtBeginDocument{%
  \if@pageindex
    \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.
    \let\gmdindexpagecs=\gmdindexrefcs
  \else
    \def␣\gmdindexrefcs#1#2#3#4{\gmiflink[clnum.#4]{%
        \csname#2\endcsname{#4}}}%
    \def␣\gmdindexpagecs#1#2#3#4{\hyperlink{page.#4}{%
        \csname#2\endcsname{\gmd@revprefix{#3}#4}}}%
    \def\gmd@revprefix#1{%
      \def\@tempa{#1}%
      \ifx\@tempa\@empty␣p.\,\fi}
    \providecommand*\HLPrefix{}% it'll be the hypertargets names' prefix in multi-docs. Moreover, it showed that if it was empty, hyperref saw duplicates 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 anymore 'cause during the authomatic \hypertargeting the lines are labeled
```

clnum.⟨*number*⟩. When \HLPrefix was defined as dot, MakeIndex rejected the entries as 'illegal page number'.

```
\fi}
```

The definition is postponed till \begin{document} because of the \PageIndex declaration (added for doc-compatibility), see line 1648.

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 linenumber 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
\UsgEntry

```
\def\DefEntry#1{\underline{#1}}
\def\UsgEntry#1{\textit{#1}}
```

The third option will be just \relax by default:

\CommonEntryCmd

```
\def\CommonEntryCmd{relax}
```

In line 512 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 $_{12}$ed name of the macro given it as the argument into proper macro to be \ifxed in lines 508 and 510 respectively.

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

\@ifstarl

```
\bgroup\catcode`\*=11␣%
\firstofone{\egroup
  \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*
\CodeDefIndex

```
\outer\def\CodeDefIndex{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Code@DefIndexStar}{\Code@DefIndex}}
\long\def\Code@DefIndex#1{\endgroup{%
    \escapechar\m@ne% because we will compare the macro's name with a string
                    without the backslash.
    \global\@defentryze{#1}}}
\long\def\Code@DefIndexStar#1{%
  \endgroup
  \addto@estoindex{#1}%
  \@defentryze{#1}}
```

\gmd@justadot

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

\@defentryze
\last@defmark

```
\long\def\@defentryze#1{%
  \expandafter\let\csname␣gmd/defentry/\string#1\endcsname\gmd@justadot%
                    The LaTeX \@namedef macro could not be used since it's not 'long'.
  \xdef\last@defmark{\string#1}}% we \string the argument 'cause it's a control
```

sequence most probably.

\@usgentryze
```
\long\def\@usgentryze#1{%
  \expandafter\let\csname␣gmd/usgentry/\string#1\endcsname%
        \gmd@justadot}
```

Initialize \envirs@toindex

```
\@emptify\envirs@toindex
```

Now we'll do the same for the 'usage' entries:

\CodeUsgIndex* \CodeUsgIndex
```
\outer\def\CodeUsgIndex{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Code@UsgIndexStar}{\Code@UsgIndex}}
```

The * possibility is for marking environments etc.

```
\long\def\Code@UsgIndex#1{\endgroup{%
    \escapechar\m@ne
    \global\@usgentryze{#1}}}
```

```
\long\def\Code@UsgIndexStar#1{%
  \endgroup
  \addto@estoindex{#1}%
  \@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
```
\outer\def\CodeCommonIndex{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Code@CommonIndexStar}{%
        \Code@CommonIndex}}
```

```
\long\def\Code@CommonIndex#1{\endgroup}
```

```
\long\def\Code@CommonIndexStar#1{%
  \endgroup\addto@estoindex{#1}}
```

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

\text@indexmacro
```
\long\def\text@indexmacro#1{%
  {\escapechar\m@ne␣\xdef\macro@pname{\xiistring#1}}%
  \expandafter\quote@mname\macro@pname\relax% we process the CS's name char
        by char and quote MakeIndex controls. \relax is the iterating macro's stop-
        per. The scanned CS's quoted name shall be the expansion of \macro@iname.
  \if\verbatimchar\macro@pname
    \def\im@firstpar{[$]}%
  \else\def\im@firstpar{}%
  \fi
  {\do@properindex% see line 756.
    \expandafter␣\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
```
\def\quote@mname{%
  \def\macro@iname{}%
```

```
  \quote@charbychar}
\def\quote@charbychar#1{%
  \if\relax#1% finish quoting when you meet \relax or:
  \else
    \quote@char#1%
    \xdef\macro@iname{\macro@iname␣\maybe@quote#1}%
    \afterfi\quote@charbychar
  \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*
\TextUsgIndex

```
\def\TextUsgIndex{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Text@UsgIndexStar}{\Text@UsgIndex}}
\long\def\Text@UsgIndex#1{%
  \endgroup\@usgentryze#1%
  \text@indexmacro#1}
\long\def\Text@UsgIndexStar#1{\endgroup\@usgentryze{#1}%
  \text@indexenvir{#1}}
```

\text@indexenvir

```
\long\def␣\text@indexenvir#1{%
  \edef\macro@pname{\xiistring#1}%
  \if\bslash\expandafter\@firstofmany\macro@pname\@@nil% if \stringed #1
        begins with a backslash, we will gobble it to make MakeIndex not see it.
    \edef\@tempa{\expandafter\@gobble\macro@pname}%
    \@tempswatrue
  \else
    \let\@tempa\macro@pname
    \@tempswafalse
  \fi
  \expandafter\quote@mname\@tempa\relax% we process \stinged #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.
  {\if@tempswa
    \def\quoted@eschar{\quotechar\bslash}%
    \else\@emptify\quoted@eschar\fi% we won't print any backslash before an
          environment's name, but we will before a CS's name.
    \do@properindex% see line 756.
    \index@macro\macro@iname\macro@pname}}
```

\TextCommonIndex*
\TextCommonIndex

```
\def\TextCommonIndex{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Text@CommonIndexStar}{%
        \Text@CommonIndex}}
\long\def\Text@CommonIndex#1{\endgroup
  \text@indexmacro#1}
\long\def\Text@CommonIndexStar#1{\endgroup
  \text@indexenvir{#1}}
```

As you see in the lines 519 and 515, 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* \CodeMarginize

```
\outer\def\CodeMarginize{\begingroup
  \MakePrivateLetters
  \@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`\catcode`ing group.

```
\long\def\egCode@MarginizeMacro#1{\endgroup
  \Code@MarginizeMacro#1}
\long\def\Code@MarginizeMacro#1{{\escapechar\m@ne
    \expandafter\glet\csname␣gmd/2marpar/\string#1\endcsname%
          \gmd@justadot
  }}
\long\def\egCode@MarginizeEnvir#1{\endgroup
  \Code@MarginizeEnvir{#1}}
\long\def\Code@MarginizeEnvir#1{\addto@estomarginpar{#1}}
```

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

Here it is:

\mark@envir

```
\def\mark@envir{%
  \ifx\envirs@tomarginpar\@empty
  \else
    \let\do\Text@Marginize
    \envirs@tomarginpar%
    \g@emptify\envirs@tomarginpar%
  \fi
  \ifx\envirs@toindex\@empty
  \else
    \def\do##1{% the \envirs@toindex list contains \string'ed macros or environments' names in braces and each preceded with \do.
      \if\bslash\@firstofmany##1\@@nil% if ##1 begins with a backslash, we will gobble it for MakeIndex not see it.
        \edef\@tempa{\@gobble##1}%
        \@tempswatrue
      \else
        \edef\@tempa{##1}\@tempswafalse
      \fi
      \expandafter\quote@mname\@tempa\relax% see line 632 & subs. for commentary.
      {\if@tempswa
        \def\quoted@eschar{\quotechar\bslash}%
        \else\@emptify\quoted@eschar\fi
        \index@macro\macro@iname{##1}}}%
    \envirs@toindex
    \g@emptify\envirs@toindex%
  \fi}
```

One very important thing: initialisation of the list macros:

```
\@emptify\envirs@tomarginpar
\@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* \CodeDefine

```
\outer\def\CodeDefine{\begingroup
  \MakePrivateLetters
```

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

```
  \@ifstarl{\MakePrivateOthers\Code@DefEnvir}{\Code@DefMacro}}
```

\CodeUsage* \CodeUsage

```
\outer\def\CodeUsage{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Code@UsgEnvir}{\Code@UsgMacro}}
```

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

```
\long\def\Code@DefMacro#1{%
  \Code@DefIndex#1% we use the internal macro; it'll close the group.
  \Code@MarginizeMacro#1}

\long\def\Code@UsgMacro#1{%
  \Code@UsgIndex#1% here also the internal macro; it'll close the group
  \Code@MarginizeMacro#1}
```

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

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

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

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

And in case we don't want to index:

\gag@index

```
\def\gag@index{\let\index=\@gobble
  \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:

```
\StoreMacros{\index\codeline@wrindex}
```

\ungag@index

```
\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 containig the CS to be marked. We separate indexing from marginizing to leave a possibility of doing only one of those things.

```
\long\def\Code@DefEnvir#1{%
  \endgroup
  \addto@estomarginpar{#1}%
  \addto@estoindex{#1}%
  \@defentryze{#1}}

\long\def\Code@UsgEnvir#1{%
  \endgroup
  \addto@estomarginpar{#1}%
  \addto@estoindex{#1}%
  \@usgentryze{#1}}

\long\def\addto@estomarginpar#1{%
  \edef\@tempa{\noexpand\do{\xiistring#1}}% we \string the argument to allow it to be a control sequence.
  \expandafter\addtomacro\expandafter\envirs@tomarginpar\expandafter{%
        \@tempa}}

\long\def\addto@estoindex#1{%
  \edef\@tempa{\noexpand\do{\xiistring#1}}
  \expandafter\addtomacro\expandafter\envirs@toindex\expandafter{%
        \@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* \TextUsage

```
\def\TextUsage{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Text@UsgEnvir}{\Text@UsgMacro}}

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

\long\def\Text@UsgEnvir#1{%
  \endgroup{\tt\xiistring#1}%
  \Text@Marginize{#1}%
  \@usgentryze{#1}% we declare the ‘usage’ kind of formatting of the entry and index the sequence #1.
  \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* \TextMarginize

```
\def\TextMarginize{\begingroup
  \MakePrivateLetters
```

```
  \@ifstarl{\MakePrivateOthers\egText@Marginize}{\egText@Marginize}}
\long\def\egText@Marginize#1{\endgroup
  \Text@Marginize#1}
```

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

```
\if@marginparsused
  \reversemarginpar
  \marginparpush\z@
  \marginparwidth8pc\relax
```

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

\gmdmarginpar
```
  \long\def\gmdmarginpar#1{%
    \marginpar{\raggedleft\strut
      \hskip0ptplus100ptminus100pt%
      #1}}%
\else
  \long\def\gmdmarginpar#1{}%
\fi
```

\Text@Marginize
```
\long\def\Text@Marginize#1{%
  \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.)

```
\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 702. 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.

```
\def\do@properindex{%
  \if@countalllines\else
    \@pageinclindextrue
    \let\special@index=\index
  \fi}
```

In doc all the 'working' TEX 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*
```
\newenvironment*{macrocode*}{%
  \if@codeskipput\else\par\addvspace\CodeTopsep\@codeskipputgtrue\fi
  \QueerEOL}%
  {\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
```
\newenvironment*{macrocode}{%
  \if@codeskipput\else\par\addvspace\CodeTopsep\@codeskipputgtrue\fi
  \CodeSpacesBlank\QueerEOL}%
  {\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 `macrocode`s, it seems to me necessary to add a possibility to typeset `macrocode`s as if they were a kind of `verbatim`, that is to leave the code layer and narration layer philosophy.

oldmc oldmc*
```
\let\oldmc\macrocode
\let\endoldmc\endmacrocode
\@namelet{oldmc*}{macrocode*}
\@namelet{endoldmc*}{endmacrocode*}
```

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

```
\addtomacro\oldmc{\@oldmacrocode@launch}%
\expandafter\addtomacro\csname␣oldmc*\endcsname{%
  \@oldmacrocode@launch}

\def\@oldmacrocode@launch{%
  \emptify\gmd@textEOL% to disable it in \gmd@docstripdirective launched
        within the code.
  \glet\stored@code@delim\code@delim
  \@makeother\^^B\CodeDelim*\^^B%
  \ttverbatim␣\gmd@DoTeXCodeSpace%
  \@makeother\|% 'cause \ttverbatim doesn't do that.
  \MakePrivateLetters% see line 449.
  \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 occure 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.

```
  \expandafter\gmd@currenvxistar\@currenvir*\relax
  \@oldmacrocode}

\bgroup\catcode`*11
\firstofone{\egroup
  \def\gm@xistar{*}}

\def\gmd@currenvxistar#1*#2\relax{%
  \edef\@currenvir{#1\if*#2\gm@xistar\fi}}
```

[8] Richard Strauss after Oscar Wilde, *Salome*.

The trick is that `#2` may be either `*`$_{12}$ or empty. If it's `*`, the test is satisfied and `\if...\fi` expands to `\gm@xistar`. If `#2` is empty, the test is also satisfied since `\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.)

```
\bgroup
\catcode`[=1␣\catcode`]=2
\catcode`\{=\active␣\@makeother\}
\@makeother\^^B
\catcode`!=0␣\catcode`\\=\active
!catcode`& =14 !catcode`*=11
!catcode`!%=!active␣!obeyspaces&
!firstofone[!egroup&
```

`\@oldmacrocode`

```
!def!@oldmacrocode[&
!bgroup!let␣=!relax& to avoid writing !noexpand␣ four times.
!xdef!oldmc@def[&
!def!noexpand!oldmc@end####1!noexpand%␣␣␣␣!noexpand\end&
!noexpand{!@currenvir}[&
####1^^B!noexpand!end[!@currenvir]!noexpand!gmd@oldmcfinis]]&
!egroup& now \oldmc@edef is defined to have one parameter delimited with
      \end{⟨current env.'s name⟩}
!oldmc@def&
!oldmc@end]&
]
```

```
\def\gmd@oldmcfinis{%
  \expandafter\CodeDelim\stored@code@delim
  \gmd@mchook}% see line 1586
```

```
\def\VerbMacrocodes{%
  \let\macrocode\oldmc
  \@namelet{macrocode*}{oldmc*}}
```

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

```
\bgroup\catcode`\%\active
\firstofone{\egroup
\def\docstrips@percent{\catcode`\%\active
  \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'.)

```
\def\gmd@codecheckifds#1#2{% note that #1 is just to gobble \gmd@charbychar
      token.
  \if@dsdir\@dsdirgfalse
    \if\noexpand<\noexpand#2\afterelsefifi\gmd@docstripdirective
    \else\afterfifi\twelvepercent#1#2%
    \fi
  \else\afterfi\twelvepercent#1#2%
  \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.

```
\newenvironment{macro}{%
  \@tempskipa=\MacroTopsep
  \if@codeskipput\advance\@tempskipa␣by-\CodeTopsep\fi
  \par\addvspace{\@tempskipa}\@codeskipputgtrue
  \begingroup\MakePrivateLetters\MakePrivateOthers%
```

we make also the 'private 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.)

```
  \gmd@ifonetoken\Hybrid@DefMacro\Hybrid@DefEnvir}%
  {\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:

```
\@namedef{macro*}{\let\gmd@ifonetoken\@secondoftwo\macro}
\expandafter\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

```
\def\gmd@ifsingle#1#2\@@nil{%
  \def\@tempa{#2}%
  \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 \expandaftered).

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

```
\def\gmd@ifonetoken#1#2#3{%
  \def\@tempb{#3}%
```

We hide #3 from TEX in case it's \if... or so. \@tempa is used in \gmd@ifsingle.

```
  \gmd@ifsingle#3\@@nil
    \afterelsefi\expandafter#1\@tempb%
  \else
    \edef\@tempa{\expandafter\string\@tempb}%
```

```
    \afterfi\expandafter#2\expandafter{\@tempa}%
  \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
```
\long\def\Hybrid@DefMacro#1{%
  \Code@DefIndex{#1}% this macro closes the group opened by \macro.
  \Text@MarginizeNext{#1}}
```

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

\Text@MarginizeNext
```
\long\def\Text@MarginizeNext#1{%
  \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-destructor built in so it `\relax`es itself after first use.

\gmd@evpaddonce
```
\long\def\gmd@evpaddonce#1{%
  \stepnummacro\gmd@oncenum
  \expandafter\long\expandafter\edef%
    \csname␣gmd/evp/NeuroOncer\gmd@oncenum\endcsname{%
      \noexpand\g@relaxen
      \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.
  \expandafter\addtomacro\csname␣gmd/evp/NeuroOncer\gmd@oncenum%
        \endcsname{#1}%
  \expandafter\addto@hook\expandafter\everypar\expandafter{%
    \csname␣gmd/evp/NeuroOncer\gmd@oncenum\endcsname}%
}
\nummacro\gmd@oncenum% We store the number uniquifying the auxiliary macro in
      a macro to save count registers (cf. gmutils sec. To Save Precious Count Reg-
      isters).
```

environment

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

```
\expandafter\let\expandafter\environment\csname␣macro*\endcsname
\expandafter\let\expandafter\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
```
\def\DoNotIndex{\bgroup\MakePrivateLetters\DoNot@Index}
\long\def\DoNot@Index#1{\egroup% we close the group,
  \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.
  \dont@index#1.}
\long\def\dont@index#1{%
  \def\@tempa{\noexpand#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.
  \if\@tempa.% a dot finishes expelling
  \else
    \if\@tempa,% The list this macro is put before may contain commas and that's O.K., we just continue the work.
      \afterelsefifi\dont@index
    \else% what is else shall off the Index be expelled.
      {\escapechar\m@ne
        \xdef\@tempa{\string#1}}%
      \expandafter\let%
      \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 506 will turn false and the subject CS shall not be indexed. We \let not \def to spare TeX's memory.
      \afterfifi\dont@index
    \fi
  \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.

```
\DoNotIndex\{ \DoNotIndex\}% the index entries of these two CSs would be rejected by MakeIndex anyway.
\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
```
  \gdef\DefaultIndexExclusions{%
    \DoNotIndex{\@ \@@par \@beginparpenalty \@empty}%
    \DoNotIndex{\@flushglue \@gobble \@input}%
    \DoNotIndex{\@makefnmark \@makeother \@maketitle}%
    \DoNotIndex{\@namedef \@ne \@spaces \@tempa}%
    \DoNotIndex{\@tempb \@tempswafalse \@tempswatrue}%
    \DoNotIndex{\@thanks \@thefnmark \@topnum}%
    \DoNotIndex{\@@ \@elt \@forloop \@fortmp \@gtempa \@totalleftmargin}%
    \DoNotIndex{\" \/ \@ifundefined \@nil \@verbatim \@vobeyspaces}%
    \DoNotIndex{\| \~ \␣ \active \advance \aftergroup \begingroup \bgroup}%
    \DoNotIndex{\mathcal \csname \def \documentstyle \dospecials \edef}%
    \DoNotIndex{\egroup}%
    \DoNotIndex{\else \endcsname \endgroup \endinput \endtrivlist}%
    \DoNotIndex{\expandafter \fi \fnsymbol \futurelet \gdef \global}%
    \DoNotIndex{\hbox \hss \if \if@inlabel \if@tempswa \if@twocolumn}%
```

[9] This idea comes from Marcin Woliński.

```
    \DoNotIndex{\ifcase}%
    \DoNotIndex{\ifcat \iffalse \ifx \ignorespaces \index \input \item}%
    \DoNotIndex{\jobname \kern \leavevmode \leftskip \let \llap \lower}%
    \DoNotIndex{\m@ne \next \newpage \nobreak \noexpand
        \nonfrenchspacing}%
    \DoNotIndex{\obeylines \or \protect \raggedleft \rightskip \rm \sc}%
    \DoNotIndex{\setbox \setcounter \small \space \string \strut}%
    \DoNotIndex{\strutbox}%
    \DoNotIndex{\thefootnote \thispagestyle \topmargin \trivlist \tt}%
    \DoNotIndex{\twocolumn \typeout \vss \vtop \xdef \z@}%
    \DoNotIndex{\, \@bsphack \@esphack \@noligs \@vobeyspaces
        \@xverbatim}%
    \DoNotIndex{\` \catcode \end \escapechar \frenchspacing \glossary}%
    \DoNotIndex{\hangindent \hfil \hfill \hskip \hspace \ht \it \langle}%
    \DoNotIndex{\leaders \long \makelabel \marginpar \markboth
        \mathcode}%
    \DoNotIndex{\mathsurround \mbox}% % \newcount \newdimen \newskip
    \DoNotIndex{\nopagebreak}%
    \DoNotIndex{\parfillskip \parindent \parskip \penalty \raise
        \rangle}%
    \DoNotIndex{\section \setlength \TeX \topsep \underline \unskip}%
    \DoNotIndex{\vskip \vspace \widetilde \\ \% \@date \@defpar}%
    \DoNotIndex{\[ \]}% see line 885.
    \DoNotIndex{\count@ \ifnum \loop \today \uppercase \uccode}%
    \DoNotIndex{\baselineskip \begin \tw@}%
    \DoNotIndex{\a \b \c \d \e \f \g \h \i \j \k \l \m \n \o \p \q}%
    \DoNotIndex{\r \s \t \u \v \w \x \y \z \A \B \C \D \E \F \G \H}%
    \DoNotIndex{\I \J \K \L \M \N \O \P \Q \R \S \T \U \V \W \X \Y \Z}%
    \DoNotIndex{\1 \2 \3 \4 \5 \6 \7 \8 \9 \0}%
    \DoNotIndex{\! \# \$ \& \' \( \) \. \: \; \< \= \> \? \_}% \+ seems to be so
        rarely used that it may be advisable to index it.
    \DoNotIndex{\discretionary \immediate \makeatletter \makeatother}%
    \DoNotIndex{\meaning \newenvironment \par \relax \renewenvironment}%
    \DoNotIndex{\repeat \scriptsize \selectfont \the \undefined}%
    \DoNotIndex{\arabic \do \makeindex \null \number \show \write \@ehc}%
    \DoNotIndex{\@author \@ehc \@ifstar \@sanitize \@title}%
    \DoNotIndex{\if@minipage \if@restonecol \ifeof \ifmmode}%
    \DoNotIndex{\lccode % %\newtoks
      \onecolumn \openin \p@ \SelfDocumenting}%
    \DoNotIndex{\settowidth \@resetonecoltrue \@resetonecolfalse \bf}%
    \DoNotIndex{\clearpage \closein \lowercase \@inlabelfalse}%
    \DoNotIndex{\selectfont \mathcode \newmathalphabet \rmdefault}%
    \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:

```
    \DoNotIndex{\@@input \@auxout \@currentlabel \@dblarg}%
    \DoNotIndex{\@ifdefinable \@ifnextchar \@ifpackageloaded}%
    \DoNotIndex{\@indexfile \@let@token \@sptoken \^}% the latter comes from
        CSs like \^^M, see sec. 660.
    \DoNotIndex{\addto@hook \addvspace}%
    \DoNotIndex{\CurrentOption}%
    \DoNotIndex{\emph \empty \firstofone}%
    \DoNotIndex{\font \fontdimen \hangindent \hangafter}%
    \DoNotIndex{\hyperpage \hyperlink \hypertarget}%
    \DoNotIndex{\ifdim \ifhmode \iftrue \ifvmode \medskipamount}%
    \DoNotIndex{\message}%
    \DoNotIndex{\NeedsTeXFormat \newcommand \newif}%
    \DoNotIndex{\newlabel}%
    \DoNotIndex{\of}%
    \DoNotIndex{\phantom \ProcessOptions \protected@edef}%
    \DoNotIndex{\protected@xdef \protected@write}%
    \DoNotIndex{\ProvidesPackage \providecommand}%
    \DoNotIndex{\raggedright}%
    \DoNotIndex{\raisebox \refstepcounter \ref \rlap}%
    \DoNotIndex{\reserved@a \reserved@b \reserved@c \reserved@d}%
    \DoNotIndex{\stepcounter \subsection \textit \textsf \thepage \tiny}%
    \DoNotIndex{\copyright \footnote \label \LaTeX}%
    \DoNotIndex{\@eha \@endparenv \if@endpe \@endpefalse \@endpetrue}%
    \DoNotIndex{\@evenfoot \@oddfoot \@firstoftwo \@secondoftwo}%
    \DoNotIndex{\@for \@gobbletwo \@idxitem \@ifclassloaded}%
    \DoNotIndex{\@ignorefalse \@ignoretrue \if@ignore}%
    \DoNotIndex{\@input@ \@input}%
    \DoNotIndex{\@latex@error \@mainaux \@nameuse}%
    \DoNotIndex{\@nomath \@oddfoot}% %\@onlypreamble should be indexed IMO.
    \DoNotIndex{\@outerparskip \@partaux \@partlist \@plus}%
    \DoNotIndex{\@sverb \@sxverbatim}%
    \DoNotIndex{\@tempcnta \@tempcntb \@tempskipa \@tempskipb}%
        I think the layout parameters even the kernel, should not be excluded:
        % \@topsep \@topsepadd \abovedisplayskip \clubpenalty etc.
    \DoNotIndex{\@writeckpt}%
    \DoNotIndex{\bfseries \chapter \part \section \subsection}%
    \DoNotIndex{\subsubsection}%
    \DoNotIndex{\char \check@mathfonts \closeout}%
    \DoNotIndex{\fontsize \footnotemark \footnotetext \footnotesize}%
    \DoNotIndex{\g@addto@macro \hfilneg \Huge \huge}%
    \DoNotIndex{\hyphenchar \if@partsw \IfFileExists \in@@}%
    \DoNotIndex{\include \includeonly \indexspace}%
    \DoNotIndex{\itshape \language \LARGE \Large \large}%
    \DoNotIndex{\lastbox \lastskip \m@th \makeglossary}%
    \DoNotIndex{\maketitle \math@fontsfalse \math@fontstrue \mathsf}%
    \DoNotIndex{\MessageBreak \noindent \normalfont \normalsize}%
    \DoNotIndex{\on@line \openout \outer}%
    \DoNotIndex{\parbox \part \rmfamily \rule \sbox}%
    \DoNotIndex{\sf@size \sffamily \skip}%
    \DoNotIndex{\textsc \textup \toks@ \ttfamily \vbox}%
%%\DoNotIndex{\begin*} maybe in the future, if the idea gets popular...
```

```
    \DoNotIndex{\hspace* \newcommand* \newenvironment*
          \providecommand*}%
    \DoNotIndex{\renewenvironment* \section* \chapter*}%
  }% of \DefaultIndexExclusions.
```

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

```
\end{MakePrivateLetters}
```

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

```
\if@indexallmacros\else
  \DefaultIndexExclusions
\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
```
\def\DoIndex{\bgroup\MakePrivateLetters\Do@Index}
```

```
\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
```
\def\UndoDefaultIndexExclusions{%
  \StoreMacro\DoNotIndex
  \let\DoNotIndex\DoIndex
  \DefaultIndexExclusions
  \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
```
\long\def\IndexPrologue#1{\@bsphack\def\index@prologue{#1}\@esphack}
```

\indexdiv
```
\def\indexdiv{\@ifundefined{chapter}{\section*}{\chapter*}}
```

\index@prologue
```
\@ifundefined{index@prologue}␣{\def\index@prologue{\indexdiv{Index}%
    \markboth{Index}{Index}%
    Numbers␣written␣in␣italic␣refer␣to␣the␣\if@pageindex␣pages␣\else
    code␣lines␣\fi␣where␣the
    corresponding␣entry␣is␣described;␣numbers␣underlined␣refer␣to␣the
    \if@pageindex\else␣code␣line␣of␣the␣\fi␣definition;␣numbers␣in
    roman␣refer␣to␣the␣\if@pageindex␣pages\else␣code␣lines␣\fi␣where
    the␣entry␣is␣used.
    \if@pageindex\else
      \ifx\HLPrefix\@empty
        The␣numbers␣preceded␣with␣‘p.’␣are␣page␣numbers.
      \else␣The␣numbers␣with␣no␣prefix␣are␣page␣numbers.
    \fi\fi
    \ifx\IndexLinksBlack\relax\else
      All␣the␣numbers␣are␣hyperlinks.
```

```
    \fi
    \gmd@dip@hook% this hook is intended to let a user add something without re-
          defining the entire prologue, see below.
  }}{}
```

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

```
\@emptify\gmd@dip@hook
\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

```
\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."

```
\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.
\newcount\c@IndexColumns␣\c@IndexColumns␣=␣3
\renewenvironment{theindex}
  {\begin{multicols}\c@IndexColumns[\index@prologue][\IndexMin]%
      \IndexLinksBlack
      \IndexParms␣\let\item\@idxitem␣\ignorespaces}%
    {\end{multicols}}
\def\IndexLinksBlack{\hypersetup{linkcolor=black}}% To make Adobe Reader
      work faster.
\@ifundefined{IndexParms}
  {\def\IndexParms{%
      \parindent␣\z@
      \columnsep␣15pt
      \parskip␣0pt␣plus␣1pt
      \rightskip␣15pt
      \mathsurround␣\z@
      \parfillskip=-15pt␣plus␣1␣fil␣% 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.
      \small
      \def\@idxitem{\par\hangindent␣30pt}%
      \def\subitem{\@idxitem\hspace*{15pt}}%
      \def\subsubitem{\@idxitem\hspace*{25pt}}%
      \def\indexspace{\par\vspace{10pt␣plus␣2pt␣minus␣3pt}}%
      \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.
      \ifnum\c@IndexColumns>\tw@\raggedright\fi% the numbers in narrow col-
            umns look better when they are \raggedright in my opinion.
    }}{}
```

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

```
{\@makeother\<\@makeother\>
  \glet\sgtleftxii=<
```

\gmd@docstripdirective
```
\gdef\gmd@docstripdirective{%
  \begingroup\let\do=\@makeother
  \do\*\do\/\do\+\do\-\do\,\do\&\do\|\do\!\do\(\do\)\do\>\do\<%
  \@ifnextchar{<}{%
    \let\do=\@makeother␣\dospecials
    \gmd@docstripverb}
  {\gmd@docstripinner}}%
\if1␣1%
\gdef\Debug@dstron{\ifnum\c@codelinenum>1178␣\ifdtraceon\fi
  \ifnum\c@codelinenum>1184␣\ifdtraceoff\@emptify\Debug@dstron\fi}%
\else
  \global\@emptify\Debug@dstron
\fi
  \gdef\gmd@docstripinner#1>{%
    \endgroup
    \def\gmd@modulehashone{%
      \Module{#1}\space
      \@afternarrgfalse\@aftercodegtrue\@codeskipputgfalse}%
    \gmd@textEOL\gmd@modulehashone\Debug@dstron}
```

A word of explanation: first of all, we close the group for changed `\catcode`s; the directive's text has its `\catcode`s 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.

```
  \catcode`\^^M=\active%
  \gdef\gmd@docstripverb<#1^^M{%
    \endgroup%
    \def\gmd@modulehashone{%
      \ModuleVerb{#1}\@afternarrgfalse\@aftercodegtrue%
      \@codeskipputgfalse}%
```

```
    \gmd@docstripshook%
    \gmd@textEOL\gmd@modulehashone^^M}%
}
```

(~Verbatim ;-) from doc:)

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

\ModuleVerb
```
\providecommand*\ModuleVerb[1]{{\mod@math@codes$\langle\langle%
      \mathsf{#1}$}}
\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
\changes
```
\def\gmd@DefineChanges{%
  \outer\long\def\changes{\@bsphack\begingroup\@sanitize
    \catcode`\\\z@␣\catcode`\␣10␣\MakePercentIgnore
    \MakePrivateLetters␣\StraightEOL
    \MakeGlossaryControls
    \changes@}}
\newcommand\changes[4][]{\PackageWarningNoLine{gmdoc}{%
    ^^JThe␣\bslash␣changes␣command␣used␣\on@line
    ^^Jwith␣no␣\string\RecordChanges\space␣declared.
    ^^JI␣shall␣not␣warn␣you␣again␣about␣it}%
  \renewcommand\changes[4][]{}}
\def\MakeGlossaryControls{%
  \def\actualchar{=}\def\quotechar{!}%
  \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@
```
\newcommand\changes@[4][\generalname]{%
  \if@RecentChange{#3}% if the date is later than the one stored in \c@Changes-
      % StartDate,
    \@tempswafalse
    \ifx\generalname#1% then we check whether a CS-entry is given in the optional
        first argument or is it unchanged.
      \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.
        \@tempswatrue
        \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 occured 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).
          \if\bslash\last@defmark\else\last@defmark\fi}%
        \ifx\last@defmark\bslash\let\@tempb\last@defmark\fi%
      \fi
    \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]...)
      \@tempswatrue
      {\escapechar\m@ne
        \xdef\@tempb{\string#1}}
    \fi
    \protected@edef\@tempa{\noexpand\glossary{%
        \if\relax\GeneralName\relax\else
          \GeneralName% it's for the \DocInclude case to precede every \changes
              of the same file with the file name, cf. line 1242.
        \fi
        #2\levelchar%
        \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 preceed it by \generalname.
          \@tempb
          \actualchar\bslash␣verb*%
          \if\verbatimchar\@tempb$\else\verbatimchar\fi
          \quotechar\bslash␣\@tempb
          \if\verbatimchar\@tempb$\else\verbatimchar\fi
        \else
          \space\actualchar\generalname
        \fi
        :\levelchar#4\encapchar␣hyperpage}}%
    \@tempa
  \fi\endgroup\@esphack}
```

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

```
\@relaxen\last@defmark
\@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 ⟨*YYYYMMDD*⟩ shape both to be easily compared and readable.

\c@ChangesStartDate
```
\newcount\c@ChangesStartDate
```

\if@RecentChange
```
\def\if@RecentChange#1{%
  \gmd@setChDate#1\@@nil\@tempcnta
  \ifnum\@tempcnta>\c@ChangesStartDate}
\def\gmd@setChDate#1/#2/#3\@@nil#4{% the last parameter will be a \count register.
  #4=#1\relax
  \multiply#4␣by\@M
  \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.
  \multiply\count8␣by100␣%
  \advance#4␣by\count8␣\count8=\z@
  \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 `{`⟨*year*⟩`/`⟨*month*⟩`/`⟨*day*⟩`}`.

\ChangesStart
```
\def\ChangesStart#1#2{%
  \gmd@setChDate#2\@@nil\c@ChangesStartDate
  \typeout{^^JPackage␣gmdoc␣info:␣^^JChanges'␣start␣date␣#1␣memorized
    as␣\string<\the\c@ChangesStartDate\string>␣\on@line.^^J}
  \advance\c@ChangesStartDate\m@ne% we shall show the changes at the specified day and later.
  \ifnum\c@ChangesStartDate>19820900␣%[10] see below.
    \edef\@tempa{%
      \noexpand\g@addto@macro\noexpand\glossary@prologue{%
        The␣changes
        \if\relax\GeneralName\relax\else␣of␣\GeneralName\space\fi
        earlier␣than
        #1␣\if\relax#1\relax␣#2\else(#2)\fi\space␣are␣not␣shown.}}%
    \@tempa
  \fi}
```

(Explanation to line 1147.) 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.

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

In line 1147 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
```
1156 \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 `\write`ing definition of the `\changes` macro to ensure no changes are written out without `\RecordChanges`.

\RecordChanges
```
1157 \def\RecordChanges{\makeglossary\gmd@DefineChanges
1158   \@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."

```
1159 \newdimen\GlossaryMin          \GlossaryMin       = 80pt
1160 \newcount\c@GlossaryColumns    \c@GlossaryColumns = 2
```

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

theglossary
```
1161 \newenvironment{theglossary}{%
1162   \begin{multicols}\c@GlossaryColumns
1163     [\glossary@prologue][\GlossaryMin]%
1164     \GlossaryParms␣\let\item\@idxitem␣\ignorespaces}%
1165   {\end{multicols}}
```

Here is the MakeIndex style definition:

```
1166 ⟨/package⟩
1167 ⟨+gmglo⟩ preamble
1168 ⟨+gmglo⟩ "\n␣\\begin{theglossary}␣\n
1169 ⟨+gmglo⟩ \\makeatletter\n"
1170 ⟨+gmglo⟩ postamble
1171 ⟨+gmglo⟩ "\n\n␣\\end{theglossary}\n"
1172 ⟨+gmglo⟩ keyword␣"\\glossaryentry"
1173 ⟨+gmglo⟩ actual␣'='
1174 ⟨+gmglo⟩ quote␣'!'
1175 ⟨+gmglo⟩ level␣'>'
1176 ⟨*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 1177 \long\def\GlossaryPrologue#1{\@bsphack
                  1178   \def\glossary@prologue{#1}%
                  1179   \@esphack}
```

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

```
                  1180 \@ifundefined{glossary@prologue}
\glossary@prologue 1181   {\def\glossary@prologue{\indexdiv{{Change␣History}}%
                  1182       \markboth{{Change␣History}}{{Change␣History}}%
                  1183     }}{}
```

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

```
                  1184 \AtBeginDocument{%
\GlossaryParms    1185   \@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     1186 \def\PrintChanges{% to avoid a disaster among queer EOLs:
                  1187   \@ifQueerEOL
                  1188     {\StraightEOL\@input@{\jobname.gls}\QueerEOL}%
                  1189     {\@input@{\jobname.gls}}%
                  1190   \g@emptify\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        1191 \newcount\check@sum
\CheckSum         1192 \def\CheckSum#1{\@bsphack\global\check@sum#1\relax\@esphack}
CheckSum          1193 \newcounter{CheckSum}
                  1194 \newcommand*\step@checksum{\stepcounter{CheckSum}}
```

And we'll use it in the line 454 (`\stepcounter` is `\global`). See also the `\chschange` declaration, l. 1229.

However, the check sum mechanism in gmdoc behaves slightly different than in doc which is nicely visible while gmdocing 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 `macrocode`s, 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.

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

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

\check@checksum
```
\def\check@checksum{\relax
  \ifnum\check@sum=\z@
    \typeout{**********************************}%
    \typeout{*␣The␣input␣file␣\gmd@inputname\space␣has␣no␣Checksum
      stated.}%
    \typeout{*␣The␣current␣checksum␣is␣\the\c@CheckSum.}%
    \gmd@chschangeline% a check sum changes history entry, see below.
    \typeout{*␣(package␣gmdoc␣info.)}
    \typeout{**********************************}%
  \else
    \ifnum\check@sum=\c@CheckSum
      \typeout{**********************}%
      \typeout{*␣The␣input␣file␣\gmd@inputname:␣Checksum␣passed.}%
      \gmd@chschangeline
      \typeout{*␣(package␣gmdoc␣info.)}
      \typeout{**********************}%
    \else
      \typeout{********!*!*!*!*!*!*!*!*!*!*!}%
      \typeout{*!␣The␣input␣file␣\gmd@inputname:}
      \typeout{*!␣The␣CheckSum␣stated:␣\the\check@sum\space<>␣my
        count:␣\the\c@CheckSum.}
      \gmd@chschangeline
      \typeout{*!␣(package␣gmdoc␣info.)}
      \typeout{********!*!*!*!*!*!*!*!*!*!*!}%
    \fi
  \fi
  \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.

```
\def\gmd@chschangeline{%
  \typeout{\twelvepercent\space\string\chschange{%
      \fileversion}{\the\year/\the\month/\the\day}{\the\c@CheckSum}}
  \typeout{\twelvepercent\space\string\chschange{\fileversion}{%
```

\chschange

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

```
1227        \expandafter\@gobbletwo\the\year/\the\month/\the\day}{%  with two
                digit year in case you use \ChangesStart.
1228        \the\c@CheckSum}}}
```

And here the meaning of such a line is defined:

```
1229 \newcommand*\chschange[3]{%
1230   \csname␣changes\endcsname{#1}{#2}{CheckSum␣#3}% \csname... 'cause \cha-
         % nges is \outer.
1231   \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
```
1232 \newcommand*\ltxPageLayout{%
```

"Increase the text width slightly so that width the standard fonts 72 columns of code may appear in a `macrocode` environment."

```
1233   \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 `\addtolength`s with `\setlength`s.

```
1234  \setlength\marginparwidth{95pt}%
1235  \setlength\oddsidemargin{82pt}%
1236  \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.

```
1237 \newcommand*\DocInclude{\bgroup\@makeother\_\Doc@Include}% First, we make
         % _ 'other' in order to allow it in the filenames.
```

\Doc@Include
```
1238 \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.

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

\HLPrefix
```
\gdef\HLPrefix{\filesep}%
```

\EntryPrefix
```
\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
```
\def\GeneralName{#2\actualchar\pk{#2}␣}%
```
for the changes'history main level entry.

```
\relax
\clearpage
\docincludeaux
\def\currentfile{gmdoc-IncludeFileNotFound.000}%
\let\fullcurrentfile\currentfile
\IfFileExists{#1#2.fdd}{\edef\currentfile{#2.fdd}}{%
```
it's not .fdd,
```
  \IfFileExists{#1#2.dtx}{\edef\currentfile{#2.dtx}}{%
```
it's not .dtx either,
```
    \IfFileExists{#1#2.sty}{\edef\currentfile{#2.sty}}{%
```
it's not .sty,
```
      \IfFileExists{#1#2.cls}{\edef\currentfile{#2.cls}}{%
```
it's not .cls,
```
        \IfFileExists{#1#2.tex}{\edef\currentfile{#2.tex}}{%
```
it's not .tex,
```
          \IfFileExists{#1#2.fd}{\edef\currentfile{#2.fd}}{%
```
so it must be .fd or error.
```
            \PackageError{gmdoc}{\string\DocInclude\space␣file
              #1#2.fdd/dtx/sty/cls/tex/fd␣not␣found.}}}}}}}%
\edef\fullcurrentfile{#1\currentfile}%
\ifnum\@auxout=\@partaux
  \@latexerr{\string\DocInclude\space␣cannot␣be␣nested}\@eha
\else␣\@docinclude{#1}#2␣\fi}%
```
Why is #2 delimited with ␣ not braced as we are used to, one may ask.

\@docinclude
```
\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).

```
\clearpage
\if@filesw␣\gmd@writemauxinpaux{#2.aux}\fi%
```
this strange macro with a long name is another thing to allow _ in the filenames (see line 1289).

```
\@tempswatrue
\if@partsw␣\@tempswafalse\edef\@tempb{#2}%
  \@for␣\@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswatrue\fi}%
\fi
\if@tempswa␣\let\@auxout\@partaux
  \if@filesw
    \immediate\openout\@partaux␣#2.aux\relax%
```
Yes, only #2. It's to create and process the partial .aux files always in the main document's (driver's) directory.
```
    \immediate\write\@partaux{\relax}%
  \fi
```

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

```
    \StoringAndRelaxingDo\gmd@doIndexRelated
    \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.
    \else\let\maketitle=\InclMaketitle
    \fi% In the default setup we redefine \maketitle to typeset a common chapter
        or part heading.
    \if@ltxDocInclude\xdef@filekey\fi
    \GetFileInfo{\currentfile}% it's my (GM) addition with the account of using
        file info in the included files' title/heading etc.
    \incl@DocInput{\fullcurrentfile}% originally just \currentfile.
    \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.

```
    \RestoringDo\gmd@doIndexRelated
    \clearpage
    \gmd@writeckpt{#1#2}%
    \if@filesw␣\immediate\closeout\@partaux␣\fi
  \else\@nameuse{cp@#1#2}%
  \fi
  \let\@auxout\@mainaux}% end of \@docinclude.
```

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

\xdef@filekey
```
\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.
    \xdef\filekey{\filekey,␣\thefilediv={\ttfamily\currentfile}}}}
```

To allow _ in the filenames we must assure _ will be $_{12}$ while reading the filename. Therefore define

```
\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.)

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

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

```
\def\gmd@writeckpt#1{%
```

```
  \immediate\write\@partaux{%
    \string\bgroup\string\@makeother\string\_%
    \string\firstofone\@charlb\string\egroup}
  \@writeckpt{#1}%
  \immediate\write\@partaux{\@charrb}}
\gmd@doIndexRelated \def\gmd@doIndexRelated{%
  \do\tableofcontents␣\do\makeindex␣\do\EnableCrossrefs
  \do\PrintIndex␣\do\printindex␣\do\RecordChanges␣\do\PrintChanges
  \do\theglossary␣\do\endtheglossary}
\@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 \def\aalph#1{\@aalph{\csname␣c@#1\endcsname}}
\@aalph \def\@aalph#1{%
  \ifcase#1\or␣a\or␣b\or␣c\or␣d\or␣e\or␣f\or␣g\or␣h\or␣i\or
        j\or␣k\or␣l\or␣m\or␣n\or␣o\or␣p\or␣q\or␣r\or␣s\or
        t\or␣u\or␣v\or␣w\or␣x\or␣y\or␣z\or␣A\or␣B\or␣C\or
        D\or␣E\or␣F\or␣G\or␣H\or␣I\or␣J\or␣K\or␣L\or␣M\or
        N\or␣O\or␣P\or␣Q\or␣R\or␣S\or␣T\or␣U\or␣V\or␣W\or
        X\or␣Y\or␣Z\else\@ctrerr\fi}
```

A macro that initialises things for `\DocInclude`.

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

We set the things for including the files only once.

```
  \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.

```
  \ifx\filediv\relax
    \ifx\filedivname\relax% (nor \filediv neither \filedivname is defined by
          the user)
      \@ifundefined{chapter}{%
        \SetFileDiv{part}}%
      {\SetFileDiv{chapter}}%
    \else% (\filedivname is defined by the user, \filediv is not)
      \SetFileDiv{\filedivname}% why not? Inside is \edef so it'll work.
    \fi
  \else% (\filediv is defined by the user
    \ifx\filedivname\relax% and \filedivname is not)
      \PackageError{gmdoc}{You've␣redefined␣\string\filediv\space
        without␣redefining␣\string\filedivname.}{Please␣redefine␣the
        two␣macros␣accordingly.␣You␣may␣use␣\string\SetFileDiv{name
          without␣bslash}.}%
    \fi
  \fi
  \@addtoreset{codelinenum}{\filedivname}% remember it has a \global effect
        in fact. For each file we'll reset codelinenum.
  \def\thefilediv{\aalph{\filedivname}}% The files will be numbered with let-
        ters, lowercase first.
```

```
\expandafter\let\csname␣the\filedivname\endcsname=\thefilediv% This line lets \the⟨chapter⟩ etc. equal \thefilediv.
\def\filesep{\thefilediv-}% File separator (identifier) for the index.
\let\filekey=\@gobble
\g@addto@macro\index@prologue{%
  \gdef\@oddfoot{\parbox{\textwidth}{\strut\footnotesize
      \raggedright{\bfseries␣File␣Key:}␣\filekey}}% The footer for the pages of index.
  \glet\@evenfoot\@oddfoot}% anyway, it's intended to be oneside.
\g@addto@macro\glossary@prologue{%
  \gdef\@oddfoot{\strut␣Change␣History\hfill\thepage}% The footer for the changes history.
  \glet\@evenfoot\@oddfoot}%
\gdef\@oddfoot{% The footer of the file pages will be its name and, if there is a file info, also the date and version.
  \expandafter\ifx\csname␣ver@\currentfile\endcsname\relax
    File␣\thefilediv:␣{\ttfamily\currentfile}␣%
  \else
    \GetFileInfo{\currentfile}%
    File␣\thefilediv:␣{\ttfamily\filename}␣%
    Date:␣\filedate\␣%
    Version␣\fileversion
  \fi
  \hfill\thepage}%
\glet\@evenfoot\@oddfoot% see line 1339.
\expandafter\def\csname\filedivname␣name\endcsname{File}% we redefine the name of the proper division to 'File'.
\ifx\filediv\section
  \let\division=\subsection
  \let\subdivision=\subsubsection% If \filediv is higher than \section we don't change the two divisions (they are \section and \subsection by default). \section seems to me the lowest reasonable sectioning command for the file. If \filediv is lower you should rather rethink the level of a file in your documentation not redefine the two divisions.
\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
```
\def\SetFileDiv#1{%
  \edef\filedivname{#1}%
  \expandafter\let\expandafter\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 `\catcode`s trick because it uses `\jobname` inside so the filename will be all $_{12}$ anyway.

\SelfInclude
```
\newcommand*{\SelfInclude}[2][.tex]{% As you guess, the optional #1 is the jobname'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.
```

```
  \AtBegInputOnce{#2}%
  \gdef\HLPrefix{\filesep}%
  \gdef\EntryPrefix{\filesep}% we define two rather kernel parameters etc. as
        in \DocInclude.
  \relax
  \clearpage
  \docincludeaux
  \edef\currentfile{\jobname#1}%
  \let\fullcurrentfile\currentfile
  \def\GeneralName{\jobname\actualchar\pk{\jobname}␣}% for the changes his-
        tory main level entry.
  \ifnum\@auxout=\@partaux
    \@latexerr{\string\DocInclude\space␣cannot␣be␣nested}\@eha
  \else
    \if@filesw
      \gmd@writemauxinpaux{\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.
    \fi
    \@tempswatrue
    \if@partsw␣\@tempswafalse\edef\@tempb{\jobname}%
      \@for
      \@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswatrue\fi}%
    \fi
    \if@tempswa␣\let\@auxout\@partaux
      \if@filesw
        \immediate\openout\@partaux␣\jobname.auxx\relax
        \immediate\write\@partaux{\relax}
      \fi
```

"We need to save (and later restore)..."

```
      \StoringAndRelaxingDo% provided by gmutils
      \gmd@doIndexRelated
      \if@ltxDocInclude\part{\currentfile}%
      \else\let\maketitle=\InclMaketitle
      \fi
      \if@ltxDocInclude\xdef@filekey\fi
      \GetFileInfo{\currentfile}% it's my (GM) addition with the account of
            using file info in the included files' title etc.
      \incl@DocInput{\fullcurrentfile}% originally just \currentfile, no dif-
            ference in \SelfInclude.
      \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.

```
      \RestoringDo
      \gmd@doIndexRelated
```

```
      \clearpage% among others, causes the \writes to be executed which is crucial
          for proper toc-ing e.g.
      \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.
      \if@filesw␣\immediate\closeout\@partaux␣\fi
    \else\@nameuse{cp@\jobname.x}% note .x: it's used for the same reason as
        above.
    \fi
    \let\@auxout\@mainaux
  \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.

```
\newif\if@ltxDocInclude
```

\ltxLookSetup
```
\newcommand*\ltxLookSetup{%
  \SetFileDiv{part}%
  \ltxPageLayout
  \@ltxDocIncludetrue
}
```

```
\@onlypreamble\ltxLookSetup
```

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

```
\let\incl@DocInput=\DocInput
```

```
\@emptify\currentfile% for the pages outside the \DocInclude's scope. In force
    for all includes.
```

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

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

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

\gmdocIncludes
```
\newcommand*\gmdocIncludes{%
  \let\incl@DocInput=\DocInput
  \AtBegInput{\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.

```
\AtBegInput{%
```

\maketitle
```
  \providecommand*\maketitle{\par
    \begingroup␣\def␣\thefootnote␣{\fnsymbol␣{footnote}}%
    \setcounter␣{footnote}\z@
```

```
    \def\@makefnmark{\hbox␣to\z@{$\m@th^{\@thefnmark}$\hss}}%
    \long\def\@makefntext##1{\parindent␣1em\noindent
      \hbox␣to1.8em{\hss$\m@th^{\@thefnmark}$}##1}%
    \if@twocolumn␣\twocolumn␣[\@maketitle␣]%
    \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."

```
    \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:"

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

"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."

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

And let's provide `\@maketitle` just in case: an error occurred without it at TEXing 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
```
  \providecommand*\@maketitle{%
    \newpage\null␣\vskip␣2em\relax%
    \begin{center}%
      \titlesetup
      \let␣\footnote␣\thanks
      {\LARGE␣\@title␣\par}%
      \vskip␣1.5em%
      {\large␣\lineskip␣.5em%
        \begin{tabular}[t]{c}%
          \strut␣\@author
        \end{tabular}\par}%
      \vskip␣1em%
      {\large␣\@date}%
    \end{center}%
    \par␣\vskip␣1.5em\relax}%
```

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

\title
\author
\date
\thanks
```
  \providecommand*\title[1]{\gdef\@title{#1}}
  \providecommand*\author[1]{\gdef\@author{#1}}
  \providecommand*\date[1]{\gdef\@date{#1}}
  \providecommand*\thanks[1]{\footnotemark
    \protected@xdef\@thanks{\@thanks
      \protect\footnotetext[\the\c@footnote]{#1}}%
  }%
```

```
    \providecommand*\and{% ␣%␣\begin{tabular}
      \end{tabular}%
      \hskip␣1em␣\@plus.17fil%
      \begin{tabular}[t]{c}}% ␣%␣\end{tabular} And finally, let's initialize \tit-
                                                % lesetup if it is not yet.
\titlesetup     \providecommand*\titlesetup{}%
}% 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 hte `\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.

```
\@relaxen\filediv
\@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  \def\InclMaketitle{%
      {\def\and{,␣}% we make \and just a comma.
        {\let\thanks=\@gobble% for the toc version of the heading we discard \thanks.
          \protected@xdef\incl@titletotoc{\@title\if@fshda\protect\space
            (\@author)\fi}% we add the author iff the 'files have different authors'
                          % (@fshda)
        }%
\thanks       \def\thanks##1{\footnotemark
          \protected@xdef\@thanks{\@thanks% to keep the previous \thanks if there
                                    were any.
            \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?
        \@emptify\@thanks
        \protected@xdef\incl@filedivtitle{%
          [{\incl@titletotoc}]% braces to allow [ and ] in the title to toc.
          {\protect\@title
            {\smallerr% this macro is provided by the gmutils package after the relsize
                          package.
              \if@fshda\\[0.15em]\protect\@author
                \if\relax\@date\relax\else,␣\fi
              \else
                \if\relax\@date\relax\else\\[0.15em]\fi
              \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 1492.

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.

```
1484            \protect\@date}}% end of \incl@filedivtitle's brace (2nd or 3rd ar-
                    gument).
1485      }% 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).

```
1486      }% end of the comma-\and's group.
1487      \expandafter\filediv\incl@filedivtitle
1488      \@thanks
1489      \g@relaxen\@author␣\g@relaxen\@title␣\g@relaxen\@date
1490      \g@emptify\@thanks
1491  }% 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.

```
1492  \newif\if@fshda
```

(its name comes from *f*ile*s* *h*ave *d*ifferent *a*uthors).

\PrintFilesAuthors
```
1493  \newcommand*\PrintFilesAuthors{\@fshdatrue}
```

And the counterpart, if you change your mind:

```
1494  \newcommand*\SkipFilesAuthors{\@fshdafalse}
```

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:

\IndexInput
```
1495  \def\IndexInput#1{%
1496    \StoreMacro\code@delim
1497    \CodeDelim*\^^A%
1498    \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
```
1499  \newenvironment{gmlonely}{\StraightEOL}{}
```

\skipgmlonely
```
1500  \newcommand\skipgmlonely[1][]{%
1501    \def\@tempa{%
1502      \def\gmd@skipgmltext{\g@emptify\gmd@skipgmltext#1}}%
1503    \@tempa
1504    \expandafter\AtBegInput\expandafter{\@tempa}%
1505    \renewenvironment{gmlonely}{%
1506      \StraightEOL
1507      \@fileswfalse% to forbid writing to .toc, .idx etc.
1508      \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`.

```
\bgroup\catcode`\_=8␣% we ensure the standard \catcode of _.
\firstofone{\egroup
  \newcommand*\catother{${}_{12}$}%
```

Similarly, if we need to say some char is of category 13 ('active'), we'll write $_{13}$, got by `\catactive`

```
  \newcommand*\catactive{${}_{13}$}%
```

and a letter, $_{11}$

```
  \newcommand*\catletter{${}_{11}$}% .
}
```

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

```
\newenvironment*{copyrnote}{%
  \StraightEOL\everypar{\hangindent3em\relax\hangafter1␣}%
  \par\addvspace\medskipamount\parindent\z@\obeylines}{%
  \@codeskipputgfalse\stanza}
```

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

```
\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:

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

When a verbatim text occurres 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

```
\newcommand*\nlpercent{%
  \@ifstar{\def\@tempa{{\tt\twelvepercent}}%
    \@emptify\@tempb\nl@percent}%
  {\@emptify\@tempa
    \def\@tempb{\gmboxedspace}\nl@percent}}
\newcommand*\gmboxedspace{\hbox{\normalfont{␣}}}
\newcommand*\nl@percent[1][]{%
  \unskip
  \discretionary{\@tempa}{{\tt\twelvepercent\gmboxedspace}}{\@tempb}%
  \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:

```
\@ifundefined{+}{}{\typeout{^^Jgmdoc.sty:␣redefining␣\bslash+.}}
\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 $\mathcal{AMS}$-TEX, BIBTEX and SLITEX, as well as the usual TEX and LATEX. There's even a PLAIN TEX and a WEB."

```
\@ifundefined{AmSTeX}
  {\def\AmSTeX{\leavevmode\hbox{$\mathcal␣A\kern-.2em\lower.376ex%
        \hbox{$\mathcal␣M$}\kern-.2em\mathcal␣S$-\TeX}}}{}
\@ifundefined{BibTeX}
  {\def\BibTeX{{\rmfamily␣B\kern-.05em%
        \textsc{i\kern-.025em␣b}\kern-.08em%
        \TeX}}}{}
\@ifundefined{SliTeX}
  {\def\SliTeX{{\rmfamily␣S\kern-.06emL\kern-.18em\raise.32ex\hbox
        {\scshape␣i}\kern␣-.03em\TeX}}}{}
\@ifundefined{PlainTeX}{\def\PlainTeX{\textsc{Plain}\kern2pt\TeX}}{}
\@ifundefined{Web}{\def\Web{\textsc{Web}}}{}
```

There's also the (LA)TEX logo got with the `\LaTeXpar` macro provided by gmutils. And here The TEXbook's logo:

```
\def\TeXbook{The␣\TeX␣book}
\@ifundefined{eTeX}{\def\eTeX{$\varepsilon$\TeX}}{}% The εTEX manual uses
      'ε-TEX' but that would look strange in pdfεTEX.
\@ifundefined{pdfeTeX}{\def\pdfeTeX{pdf\eTeX}}{}
\@ifundefined{pdfTeX}{\def\pdfTeX{pdf\TeX}}{}
\@ifundefined{XeTeX}{\def\XeTeX{X\kern-.125em\relax
    \@ifundefined{reflectbox}{%
      \lower.5ex\hbox{E}\kern-.1667em\relax}{%
      \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
    \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.
\@ifundefined{ds}{\def\ds{DocStrip}}{}
```

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

```
\def\GetFileInfo#1{%
  \def\filename{#1}%
  \def\@tempb##1␣##2␣##3\relax##4\relax{%
    \def\filedate{##1}%
    \def\fileversion{##2}%
    \def\fileinfo{##3}}%
  \edef\@tempa{\csname␣ver@#1\endcsname}%
  \expandafter\@tempb\@tempa\relax?␣?␣\relax\relax}
```

Since we may documentally input files that we don't load, as doc e.g., let's define a macro for explicit providing the file info. It's written in analogy to `\ProvidesFile`, source 2_ϵ, file L v1.1g, l. 102.

```
\def\ProvideFileInfo#1{%
```

```
  \begingroup
    \catcode`\␣10␣\catcode\endlinechar␣10␣%
    \@makeother\/\@makeother\&%
    \kernel@ifnextchar[{\gmd@providefii{#1}}{\gmd@providefii{#1}[]}%
  }
\def\gmd@providefii#1[#2]{%
```

(we *don't* write the file info to .log)

```
  \expandafter\xdef\csname␣ver@#1\endcsname{#2}%
  \endgroup}
```

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

\ProvideSelfInfo
```
\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
```
\newcommand*\filenote{This␣file␣has␣version␣number␣\fileversion{}␣
      dated␣\filedate{}.}
```

And exactly as `\thanks`:

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

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
```
\newcommand*\CDAnd{\CodeDelim\&}
```
\CDPerc
```
\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.

```
\@ifdefinable\division{}% just to test (this LaTeX check issues an error if the first
      argument is already defined).
\@ifundefined{section}{%
```
\division
```
  \@relaxen\division}{%
```
\division
```
  \let\division=\section}
```

```
\@ifdefinable\subdivision{}% just to test (see above).
\@ifundefined{subsection}{%
```
\subdivision
```
  \@relaxen\subdivision}{%
```
\subdivision
```
  \let\subdivision=\subsection}
```

The `\let`s are inside `\@ifundefined`s 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):

```
\newcounter{gmd@mc}
\def\gmd@mchook{\stepcounter{gmd@mc}%
  \gmd@mcdiag
  \csname␣gmd@mchook\the\c@gmd@mc\endcsname}
```

\AfterMacrocode
```
\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 expalin 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
\mcdiagOn
\mcdiagOff

```
1590 \@emptify\gmd@mcdiag
1591 \def\mcdiagOn{\def\gmd@mcdiag{%
1592     \typeout{^^J\bslash␣end{macrocode(*)}␣No.\the\c@gmd@mc
1593     \space\on@line,␣cln.\the\c@codelinenum.}}}
1594 \def\mcdiagOff{\@emptify\gmd@mcdiag}
```

doc-Compatibility

My TEX 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 `\let`s 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 106.

At the very beginning of a doc-prepared file we meet a nice command `\Character-Table`. My TEX Guru says it's a bit old fashioned these days so let's just make it notify the user:

\CharacterTable

```
1595 \def\CharacterTable{\begingroup
1596   \@makeother\{\@makeother\}%
1597   \Character@Table}
1598 \begingroup
1599 \catcode`\<=1␣\catcode`\>=2␣%
1600 \@makeother\{\@makeother\}%
1601 \firstofone<\endgroup
1602   \def\Character@Table#1{#2}<\endgroup
1603     \message<^^J^^J␣gmdoc.sty␣package:^^J
1604     ====␣The␣input␣file␣contains␣the␣\bslash␣CharacterTable.^^J
1605     ====␣If␣you␣really␣need␣to␣check␣the␣correctness␣of␣the␣chars,^^J
1606     ====␣please␣notify␣the␣author␣of␣gmdoc.sty␣at␣the␣email␣
            address^^J
1607     ====␣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 1685.

Let us now look over other original doc's control sequences and let's 'domesticate' them if they are not yet.

kici kici

\DescribeMacro
\DescribeEnv

The \DescribeMacro and \DescribeEnv commands seem to correspond with my \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

```
\outer\def\DescribeMacro{%
  \begingroup\MakePrivateLetters
  \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\catcodeing shall not be done to it.

\DescribeEnv

```
\outer\def\DescribeEnv{%
  \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

```
\outer\def\Describe{%
  \begingroup\MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Describe@Env}{\Describe@Macro}}
```

The below two definitions are adjusted ~s of \Text@UsgMacro and \Text@UsgEnvir.

```
\long\def\Describe@Macro#1{%
  \endgroup
  \strut\Text@Marginize#1%
  \@usgentryze#1% we declare kind of formatting the entry
  \text@indexmacro#1\ignorespaces}
```

```
\def\Describe@Env#1{%
  \endgroup
  \strut\Text@Marginize{#1}%
  \@usgentryze{#1}% we declare the 'usage' kind of formatting the entry and index
        the sequence #1.
  \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 æsthetic 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:

```
\let\MacroFont\tt
```

\CodeIndent
\MacroIndent

We have provided \CodeIndent in line 52. And it corresponds with doc's \MacroIndent so

\MacroIndent

```
\let\MacroIndent\CodeIndent
```

And similarly the other skips:

\MacrocodeTopsep

```
\let\MacrocodeTopsep\CodeTopsep
```

\MacroTopsep

Note that \MacroTopsep is defined in gmdoc and has the same rôle as in doc.

```
\let\SpecialEscapechar\CodeEscapeChar
```

\theCodelineNo
\LineNumFont

\theCodelineNo is not used in gmdoc. Instead of it there is \LineNumFont declaration and a possibility to redefine \thecodelinenum as for all the counters. Here the \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.

```
\AtEndInput{\@ifundefined{theCodelineNo}{}{\PackageInfo{gmdoc}{The
      \string\theCodelineNo\space␣macro␣has␣no␣effect␣here,␣please␣use
      \string\LineNumFont\space␣for␣setting␣the␣font␣and/or
      \string\thecodelinenum\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

```
\def\noeffect@info#1{\@ifundefined{#1}{}{\PackageInfo{gmdoc}{^^J%
      The␣\bslash#1␣macro␣is␣not␣supported␣by␣this␣package^^J
      and␣therefore␣has␣no␣effect␣but␣this␣notification.^^J
      If␣you␣think␣it␣should␣have,␣please␣contact␣the␣maintainer^^J
      indicated␣in␣the␣package's␣legal␣note.^^J}}}
```

The four macros formatting the macro and environment names, namely \PrintDescribeMacro, \PrintMacroName, \PrintDescribeEnv and \PrintEnvName are not supported by gmdoc. They seem to me to be too internal to take care of them. Note that in the name of (æsthetical) minimality and (my) convenience I deprive you of easy knobs to set strange formats for verbatim bits: I think they are not advisable.

\PrintDescribeMacro
\PrintMacroName
\PrintDescribeEnv
\PrintEnvName

Let us just notify the user.

```
\AtEndInput{%
  \noeffect@info{PrintDescribeMacro}%
  \noeffect@info{PrintMacroName}%
  \noeffect@info{PrintDescribeEnv}%
  \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.

```
\def\CodelineNumbered{\AtBeginDocument{\gag@index}}
\@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

```
\let\CodelineIndex\@pageindexfalse
\@onlypreamble\CodelineIndex
```

\PageIndex

```
\let\PageIndex\@pageindextrue
\@onlypreamble\PageIndex
```

The next two declarations I find useful and smart:

\DisableCrossrefs

```
\def\DisableCrossrefs{\@bsphack\gag@index\@esphack}
```

\EnableCrossrefs
```
\def\EnableCrossrefs{\@bsphack\ungag@index
  \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
```
\newcommand*\AlsoImplementation{\@bsphack%
  \long\def\StopEventually##1{\gdef\Finale{##1}}% we define \Finale just to
        expand to the argument of \StopEventually not to 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).
    \@esphack}
\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
```
\def\OnlyDescription{\@bsphack\long\def\StopEventually##1{%
```

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

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

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

```
\@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
```
\let\SpecialMainIndex=\CodeDefIndex
```

\SpecialMainEnvIndex
```
\def\SpecialMainEnvIndex{\csname␣CodeDefIndex\endcsname*}% we don't type
      \CodeDefIndex explicitly here 'cause it's \outer, remember?
```

\SpecialIndex
```
\let\SpecialIndex=\CodeCommonIndex
```

\SpecialUsageIndex
```
\let\SpecialUsageIndex=\TextUsgIndex
```

\SpecialEnvIndex
```
\def\SpecialEnvIndex{\csname␣TextUsgIndex\endcsname*}
```

\SortIndex
```
\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 occures 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 1666 `\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.)

* 1667
```
\DeclareRobustCommand*\*{\leavevmode\lower.8ex\hbox{$\,\widetilde{\␣}%
    \,$}}
```

\IndexPrologue \IndexPrologue is defined in line 1001. And other doc index commands too.

\main 1668 `\@ifundefined{main}{}{\let\DefEntry=\main}`

\usage 1669 `\@ifundefined{usage}{}{\let\UsgEntry=\usage}`

About how the DocStrip directives are supported by gmdoc, see section The DocStrip.... 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 1670 `\newcounter{StandardModuleDepth}`

For now let's just mark the macro for further development

\DocstyleParms 1671 `\noeffect@info{DocstyleParms}`

For possible further development or to notify the user once and forever:

\DontCheckModules 1672 `\@emptify\DontCheckModules␣\noeffect@info{DontCheckModules}`
\CheckModules 1673 `\@emptify\CheckModules␣\noeffect@info{CheckModules}`

\Module The \Module macro *is* provided exactly as in doc.

\AltMacroFont 1674 `\@emptify\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 1675 `\def\MakePercentIgnore{\catcode`\%9\relax}`
\MakePercentComment 1676 `\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".

TEXing the original doc with my humble[12] package was a challenge and a milestone experience in my TEX 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

[12] What a *false* modesty! ;-)

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 1679).

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
```
1677 \def\OldDocInput{%
1678   \AtBegInputOnce{\StraightEOL
1679     \let\@MakeShortVerb=\old@MakeShortVerb
1680     \let\gmd@@macro\macro
1681     \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 occures.)
\VerbMacrocodes
1682     \VerbMacrocodes}%
1683   \bgroup\@makeother\_% it's to allow _ in the filenames. The next macro will close
          the group.
1684   \Doc@Input}
```

We don't swith 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 TEX all the night nad what a bliss when the 'Succesfully formated 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 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

```
1685 %␣\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 TEXing doc what inspired adding the `\StraightEOL` and `\QueerEOL` declarations.

Polishing, Development and Bugs

• `\MakePrivateLetters` theoretically may interfere with `\active`ating 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 hypertarget 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 140).

• 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 below[14]:

```
\bgroup\catcode`\^^M\active␣\firstofone{\egroup%
  \def\@NoEOF#1^^M{%
    \@relaxen\EOFMark\expandafter\noexpand\endinput^^M}}
\def\NoEOF{\QueerEOL\@NoEOF}
```

\NoEOF

As you probably see, `\NoEOF` has also the `\endinput` effect.

```
\endinput
⟨/package⟩
```

[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 BachoTEX 2006 Pearl Session where he presented `\input`ing a file inside `\edef`.

b. The gmdocc Class For gmdoc Driver Files[1]

Written by Grzegorz 'Natror' Murzynowski,
natror at o2 dot pl


```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{gmdocc}
             [2006/12/01␣v0.73␣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 (LA)TEX 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 gmdocing of the (LA)TEX 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.

noindex

Simili modo, the `nochanges` option is provided to turn creating the change history off.

nochanges

[1] This file has version number v0.73 dated 2006/12/01.

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 `\relax`es 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 LaTeX 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'es
mwrep name: `mwrep` for mwrep and `mwbk` for mwbk. For the symmetry there's also `mwart` option
mwbk (equivalent to the default setting).
mwart The existence test is done for any MW class option as it is in the default case.

\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) analogous 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 corresponds 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 writing outto .idx file off.

```
\newif\if@gmccnochanges
\DeclareOption{nochanges}{\@gmccnochangestrue}% This option turns the writing outto .glo file off.
\def\gmTheGeometry{geometry}
\DeclareOption{gmeometric}{%
  \IfFileExists{gmeometric.sty}{%
    \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.
\ProcessOptions
\ifgmcc@mwcls
  \IfFileExists{\gmcc@baseclass.cls}{}{\gmcc@mwclsfalse}% As announced, we do the ontological test to any mwcls.
\fi
\ifgmcc@mwcls
  \LoadClass[fleqn,␣oneside,␣noindentfirst,␣11pt,␣withmarginpar,
  sfheadings]{\gmcc@baseclass}%
\else
  \LoadClass[fleqn,␣11pt]{article}% Otherwise the standard article is loaded.
\fi
\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 you may display single verbatim lines with `\[...\]`:

```
\[\hbox{|\verbatim\stuff|}\] .
```

```
\IfFileExists{lmodern.sty}{% We also examine the ontological status of this package
  \RequirePackage{lmodern}% and if it shows to be satisfactory (the package shows *to be*), we load it and set the proper font encoding.
  \RequirePackage[QX]{fontenc}%
}{}
\RequirePackage[margin=2.7cm,␣left=4cm,
   right=2.2cm]{\gmTheGeometry}% Now we set the page layout.
\def\gmdoccMargins{%
  \geometry{margin=2.7cm,␣left=4cm,␣right=2.2cm}}
\if@debug% For debugging we load also the trace package that was very helpful to me.
  \RequirePackage{trace}%
  \errorcontextlines=100␣% And we set an error info parameter.
\fi
```

\ifdtraceon
\ifdtraceoff

```
\newcommand*\ifdtraceon{\if@debug\afterfi\traceon\fi}
\newcommand*\ifdtraceoff{\if@debug\traceoff\fi}
```

We load the core package:

```
\RequirePackage{gmdoc}
```

```
\@ifpackageloaded{lmodern}{% The Latin Modern font family provides a light condensed typewriter font that seems to be the most suitable for the marginpar CS marking.
  \def\marginpartt{\normalfont\fontseries{lc}\ttfamily}}%
\raggedbottom
\setcounter{secnumdepth}{0}% We wish only the parts and chapters to be numbered.
\renewcommand*\thesection{\arabic{section}}% isn't it redundant at the above setting?
\@ifnotmw{}{%
  \@ifclassloaded{mwart}{% We set the indentation of Contents:
    \SetTOCIndents{{}{\quad}{\quad}{\quad}{\quad}{\quad}{\quad}}}{% for mwart
    \SetTOCIndents{{}{\bf9.\enspace}{\quad}{\quad}{\quad}{\quad}{\quad}}}% and for the two other mwcls.
  \pagestyle{outer}}% We set the page numbers to be printed in the outer and bottom corner of the page.
\def\titlesetup{\bfseries\sffamily}% We set the title(s) to be boldface and sans serif.
\if@gmccnochanges\let\RecordChanges\relax\fi% If the nochanges option is on, we discard writing outto the .glo file.
\RecordChanges% We turn the writing the \changes outto the .glo file if not the above.
\dekclubs% We declare the club sign | to be a shorthand for \verb*
\smartunder% and we declare the _ char to behave as usual in the math mode and outside math to be just an uderscore.
\exhyphenpenalty\hyphenpenalty% 'cause mwcls set it =10000 due to Polish customs.
\RequirePackage{amssymb}
\def\EOFMark{\rightline{\ensuremath{\square}}}
\endinput
```

c. gmdocDoc.tex, The Driver File

```
\documentclass[outeroff,mwrep]{gmdocc}
\twocoltoc
\title{The␣\pk{gmdoc}␣Package\\␣i.e.,␣\pk{gmdoc.sty}␣and
  \pk{gmdocc.cls}}
\author{Grzegorz␣`Natror'␣Murzynowski}
\date{Warszawa,␣2006}
\begin{document}
\maketitle
\setcounter{page}{2}% hyperref cries if it sees two pages numbered 1.
\tableofcontents
\DoIndex\maketitle
\DocInclude{gmdoc}
\DocInclude{gmdocc}
\SelfInclude{\def\CommonEntryCmd{UsgEntry}%
  \filediv[\file{gmdocDoc.tex},␣The␣Driver
  File]{\file{gmdocDoc.tex},␣\gmhypertarget{The␣Driver}␣File}%
  \label{Driver}}
```

For your convenience I decided to add the documentations of the three auxiliary packages:

```
\skipgmlonely[\stanza␣The␣remarks␣about␣installation␣and␣compiling
  of␣the␣documentation␣are␣analogous␣to␣those␣in␣the␣chapter
  \pk{gmdoc.sty}␣and␣therefore␣ommitted.\stanza]
\DocInclude{gmutils}
\DocInclude{gmiflink}
\DocInclude{gmverb}
\typeout{%
  Produce␣change␣log␣with^^J%
  makeindex␣-r␣-s␣gmglo.ist␣-o␣\jobnamewoe.gls␣\jobnamewoe.glo^^J
  (gmglo.ist␣should␣be␣put␣into␣some␣texmf/makeindex␣directory.)^^J}
\PrintChanges
\typeout{%
  Produce␣index␣with^^J%
  makeindex␣-r␣\jobnamewoe^^J}
 \PrintIndex
\end{document}
```

MakeIndex shell commands:

```
  makeindex␣-r␣gmdocDoc
  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 Package[1]

Written by Grzegorz 'Natror' Murzynowski,
natror at o2 dot pl

Many thanks to my TeX Guru Marcin Woliński for his TeXnical support.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmutils}
      [2006/11/29␣v0.74␣some␣rather␣TeXnical␣macros,␣some␣of␣them␣
            tricky␣(GM)]
```

Intro

The gmutils.sty package provides some macros that are analogous to the standard LaTeX 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 (La)TeX-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 ommitted.

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

`\newgif` and Other Globals

The `\newgif` declaration's effect is used even in the LaTeX 2_ε source by redefining some particular user defined ifs (UD-ifs henceforth) step by step. The goal is to make the UD-if's assignment global. I needed it at least twice during gmdoc writing so I make it

[1] This file has version number v0.74 dated 2006/11/29.

a macro. It's an almost verbatim copy of LaTeX's `\newif` modulo the letter *g* and the `\global` prefix. (File d: ltdefns.dtx Date: 2004/02/20 Version v1.3g, lines 139–150)

\newgif
```
\def\newgif#1{%
  {\escapechar\m@ne
    \global\let#1\iffalse
    \@gif#1\iftrue
    \@gif#1\iffalse
  }}
```

'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.

```
\def\@gif#1#2{%
  \expandafter\gdef\csname\expandafter\@gobbletwo\string#1%
  g% the letter g for '\global'.
  \expandafter\@gobbletwo\string#2\endcsname
  {\global\let#1#2}}
```

After `\newgif\iffoo` you may type `{\foogtrue}` and the `\iffoo` 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 LaTeX approach.

\grefstepcounter
```
\newcommand*\grefstepcounter[1]{%
  {\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 `\expandafter`s e.g.

\glet
```
\def\glet{\global\let}
```

\@ifnextcat

As you guess, we `\def \@ifnextcat` à la `\@ifnextchar`, see LaTeX 2ε 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 TeXbook, 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.

\@ifnextcat
```
\long\def\@ifnextcat#1#2#3{%
```

```
  \let\reserved@d=#1%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
  \futurelet\@let@token\@ifncat}
\def\@ifncat{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifncat
  \else
    \ifcat\reserved@d\noexpand\@let@token
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
{\def\:{\let\@sptoken=␣}␣\:␣% this makes \@sptoken a space token.
\def\:{\@xifncat}␣\expandafter\gdef\:␣{\futurelet\@let@token\@ifncat}}
```

Note the trick to achieve 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).

`\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 `\expandafter`s 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 TEX 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.

\afterfi
```
\long\def\afterfi#1\fi{\fi#1}
```

One more of that family:

\afterelsefifi
```
\long\def\afterelsefifi#1\else#2\fi#3\fi{\fi\fi#1}
```

... and some other:

\afterelsefi
```
\long\def\afterelsefi#1\else#2\fi{\fi#1}
```
\afterfifi
```
\long\def\afterfifi#1\fi#2\fi{\fi\fi#1}
```
\afterelseiffifi
```
\long\def\afterelseiffifi#1\else#2\if#3\fi#4\fi{\fi#1}
```

Note, if you fancy this smart trick, that some '`else`' cases are covered by proper non-`else` `\after...` macros, e.g., `\afterfielsefi`'s task would be fulfilled by `\afterfifi` and `\afterelsefifi` covers also the '`\afterelsefielsefi`' case.

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 LaTeX'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}}%
    \@ifnextchar*
    \begingroup\@endpefalse\reserved@a}
```

\@begnamedgroup

```
\@ifdefinable\@begnamedgroup{\relax}
\def\@begnamedgroup#1{%
  \@ifnextchar%% not to ignore blanks after group
  \begingroup\@endpefalse
  \def\@currenvir{#1}%
  \edef\@currenvline{\on@line}%
  \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

\bnamegroup

```
\let\bnamegroup\@begnamedgroup
```

And for the ending

\enamegroup

```
\def\enamegroup#1{\end{#1}}
```

And we make it the starred version of \begin.

\old@begin
\begin
\begin*

```
\let\old@begin\begin
\def\begin{\@ifstar{\@begnamedgroup}{\old@begin}}
```

Improvement of \end

It's very clever and useful that \end checks whether its argument is ifx-equivalent @currenvir. 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'.

```
\def\@checkend#1{%
  \edef\reserved@a{\expandafter\string\csname#1\endcsname}%
  \edef\exii@currenvir{\expandafter\string\csname\@currenvir%
      \endcsname}%
```

```
\ifx\reserved@a\exii@currenvir\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, LaTeX 2_ε version of these macros was written by Donald Arseneau asnd@triumf.ca and Matt Swift swift@bu.edu after the LaTeX 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 \smaller \larger \smallerr \largerr

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` to `\LARGE` size. Negative n selects smaller fonts. `\smaller` == `\relsize{-1}`; `\larger` == `\relsize{1}`. `\smallerr`(my addition) == `\relsize{-2}`; `\largerr` guess yourself.

(Since `\DeclareRobustCommand` doesn't issue an error if its argument has been defined and it only informs about redefining, loading relsize remains allowed.)

\relsize

```
\DeclareRobustCommand*\relsize[1]{%
  \ifmmode\@nomath\relsize\else
    \begingroup
     \@tempcnta% assign number representing current font size
       \ifx\@currsize\normalsize 4\else\\\% funny order is to have most ...
        \ifx\@currsize\small 3\else\\\\\\\% ...likely sizes checked first
         \ifx\@currsize\footnotesize 2\else
          \ifx\@currsize\large 5\else
           \ifx\@currsize\Large 6\else
            \ifx\@currsize\LARGE 7\else
             \ifx\@currsize\scriptsize 1\else
              \ifx\@currsize\tiny 0\else
               \ifx\@currsize\huge 8\else
                \ifx\@currsize\Huge 9\else
                4\rs@unknown@warning% unknown state: \normalsize as starting point
    \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
```

Change the number by the given increment:

```
    \advance\@tempcnta#1\relax
```

watch out for size underflow:

```
     \ifnum\@tempcnta<\z@ \rs@size@warning{small}{\string\tiny}%
           \@tempcnta\z@ \fi
     \expandafter\endgroup
     \ifcase\@tempcnta\\% set new size based on altered number
        \tiny \or \scriptsize \or \footnotesize \or \small \or %
             \normalsize \or
```

```
          \large␣\or␣\Large␣\or␣\LARGE␣\or␣\huge␣\or␣\Huge␣\else
           \rs@size@warning{large}{\string\Huge}\Huge
\fi\fi}% end of \relsize.
\rs@size@warning \providecommand*\rs@size@warning[2]{\PackageWarning{gmutils␣
      (relsize)}{%
 Size␣requested␣is␣too␣#1.\MessageBreak␣Using␣#2␣instead}}
\rs@unknown@warning \providecommand*\rs@unknown@warning{\PackageWarning{gmutils␣
      (relsize)}{Current␣font␣size
 is␣unknown!␣(Why?!?)\MessageBreak␣Assuming␣\string\normalsize}}
```

And a handful of shorthands:

```
\larger \DeclareRobustCommand*\larger[1][\@ne]{\relsize{+#1}}
\smaller \DeclareRobustCommand*\smaller[1][\@ne]{\relsize{-#1}}
\textlarger \DeclareRobustCommand*\textlarger[2][\@ne]{{\relsize{+#1}#2}}
\textsmaller \DeclareRobustCommand*\textsmaller[2][\@ne]{{\relsize{-#1}#2}}
\largerr \DeclareRobustCommand*\largerr{\relsize{+2}}
\smallerr \DeclareRobustCommand*\smallerr{\relsize{-2}}
```

`\firstofone` and the Queer `\catcodes`

Remember that once a macro's argument has been read, its `\catcode`s are assigned forever and ever. That's what is `\firstofone` for. It allows you to change the `\catcode`s locally for a definition *outside* the changed `\catcode`s' group. Just see the below usage of this macro 'with TEX's eyes', as my TEX Guru taught me.

```
\firstofone \long\def\firstofone#1{#1}
```

And this one is defined, I know, but it's not `\long` with the standard definition.

```
\gobble \long\def\gobble#1{}
\gobbletwo \let\gobbletwo\@gobbletwo

\bgroup\catcode`\_=8␣%
\firstofone{\egroup
\subs   \let\subs=_}

\bgroup\@makeother\_%
\firstofone{\egroup
\twelveunder   \def\twelveunder{_}}
```

Now, let's define such a smart _ (underscore) which will be usual $\__8$ in the math mode and $\__{12}$ ('other') outside math.

```
\bgroup\catcode`\_=\active
\firstofone{\egroup
\smartunder   \newcommand*\smartunder{%
     \catcode`\_=\active
     \def_{\ifmmode\subs\else\_\fi}}}% We define it as \_ not just as \twelveunder
          because some font encodings don't have _ at the \char`\_ position.

\begingroup\catcode`\!=0
\@makeother\\
!firstofone{!endgroup%
\twelvebackslash   !newcommand*!twelvebackslash{\}}
\bslash \@ifundefined{bslash}{\let\bslash=\twelvebackslash}{}
```

```
\begingroup \@makeother\%
\firstofone{\endgroup
  \def\twelvepercent{%}}
\begingroup␣\@makeother\&%
\firstofone{\endgroup%
  \def\twelveand{&}}
\begingroup\@makeother\␣%
\firstofone{\endgroup%
  \def\twelvespace{␣}}
```

\twelvepercent \twelveand \twelvespace

Metasymbols

I fancy also another Knuthian trick for typesetting ⟨*metasymbols*⟩ in The TEXbook. 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 hypenation by defining a `\language` which has no patterns associated with it and use this to typeset the words within the angle brackets."

```
\ifx\l@nohyphenation\undefined
  \newlanguage\l@nohyphenation
\fi
\DeclareRobustCommand*\meta[1]{%
```

\meta

"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."

```
  \ensuremath\langle
  \ifmmode␣\expandafter␣\nfss@text␣\fi
  {%
    \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.

```
    \edef\meta@hyphen@restore{%
      \hyphenchar\the\font\the\hyphenchar\font}%
    \hyphenchar\font\m@ne
    \language\l@nohyphenation
    #1\/%
    \meta@hyphen@restore
  }\ensuremath\rangle
}
```

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.

```
\def\meta@font@select{\it}
```

The below \meta's drag[2] is a version of The TEXbook's one.

\<...> 135 \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 136 \def\discre#1#2#3{\kern0sp\discretionary{#1}{#2}{#3}\penalty10000%
\hskip0sp\relax}

\discret 137 \def\discret#1{\kern0sp\discretionary{#1}{#1}{#1}\penalty10000%
\hskip0sp\relax}

A tiny little macro that acts like \- outside the math mode and has its original meaning inside math.

138 \def\:{\ifmmode\afterelsefi\mskip\medmuskip\else\afterfi\discret{}\fi}

\vs 139 \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 140 \def\printspaces#1{{\let~=\vs␣\let\␣=\vs␣\gm@pswords#1␣\@@nil}}
141 \def\gm@pswords#1␣#2\@@nil{%
142 \if\relax#1\relax\else#1\fi
143 \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 144 \DeclareRobustCommand*\sfname[1]{\textsf{\printspaces{#1}}}
\file 145 \let\file\sfname% it allows the spaces in the filenames (and prints them as ␣).

The below macro I use to format the packages' names.

146 \@ifundefined{pk}{%
\pk 147 \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 148 \@ifundefined{cs}{\DeclareRobustCommand*{\cs}[2][\bslash]{{%
149 \def\-{\discretionary{{\rmfamily-}}{}{}}%
150 \def\{{\char`\{}\def\}{\char`\}}\ttfamily␣#1#2}}}{}

\env 151 \@ifundefined{env}{\DeclareRobustCommand*{\env}[1]{\cs[]{#1}}}{}

And one for encouraging linebreaks e.g., before long verbatim words.

\possfil 152 \newcommand*\possfil{\hfil\penalty1000\hfilneg}

The five macros below are taken from the ltxdoc.dtx.

[2] Think of the drags that transform a very nice but rather standard 'auntie' ('Tante' in Deutsch) into a most adorable Queen ;-) .

"\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
```
\def\cmd#1{\cs{\expandafter\cmd@to@cs\string#1}}
\def\cmd@to@cs#1#2{\char\number`#2\relax}
```

\marg{text} prints {⟨*text*⟩}, 'mandatory argument'.

\marg
```
\def\marg#1{{\ttfamily\char`\{}\meta{#1}{\ttfamily\char`\}}}
```

\oarg{text} prints [⟨*text*⟩], 'optional argument'. Also \oarg[text] does that.

\oarg
```
\def\oarg{\@ifnextchar[\@oargsq\@oarg}
\def\@oarg#1{{\ttfamily[}\meta{#1}{\ttfamily]}}
\def\@oargsq[#1]{\@oarg{#1}}
```

\parg{te,xt} prints (⟨*te,xt*⟩), 'picture mode argument'.

\parg
```
\def\parg{\@ifnextchar(\@pargp\@parg}
\def\@parg#1{{\ttfamily(}\meta{#1}{\ttfamily)}}
\def\@pargp(#1){\@parg{#1}}
```

But we can have all three in one command.

```
\AtBeginDocument{%
```
\arg
```
  \let\math@arg\arg
  \def\arg{\ifmmode\math@arg\else\afterfi
    \@ifnextchar[\@oargsq{\@ifnextchar(\@pargp\marg}\fi}%
}
```

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 meanig back (when grouping is not recommended):

\StoreMacro
```
\def\StoreMacro{\bgroup\makeatletter\egStore@Macro}
\long\def\egStore@Macro#1{\egroup\Store@Macro{#1}}
\long\def\Store@Macro#1{%
  \expandafter\let\csname␣/gml/store\string#1\endcsname#1}
```

We make the \StoreMacro command a three-step to allow usage of the most inner macro also in the next command.

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
```
\long\def\StoreMacros{\bgroup\makeatletter\Store@Macros}
\long\def\Store@Macros#1{\egroup
  \let\gml@StoreCS\Store@Macro
  \gml@storemacros#1.}
```

And the inner iterating macro:

```
\long\def\gml@storemacros#1{%
  \def\@tempa{\noexpand#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.
  \if\@tempa.% a dot finishes storing.
  \else
```

```
    \if\@tempa,% The list this macro is put before may contain commas and that's
          O.K., we just continue the work.
      \afterelsefifi\gml@storemacros
    \else% what is else this shall be stored.
      \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.
      \afterfifi\gml@storemacros
    \fi
  \fi}
```

And for the restoring

\RestoreMacro
```
\def\RestoreMacro{\bgroup\makeatletter\egRestore@Macro}
\long\def\egRestore@Macro#1{\egroup\Restore@Macro{#1}}
\long\def\Restore@Macro#1{%
    \expandafter\let\expandafter#1\csname␣/gml/store\string#1%
          \endcsname}
```

\RestoreMacros
```
\long\def\RestoreMacros{\bgroup\makeatletter\Restore@Macros}
\long\def\Restore@Macros#1{\egroup
  \let\gml@StoreCS\Restore@Macro% we direct the core CS towards restoring and
        call the same iterating macro as in line 174.
  \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:

```
\def\StoredMacro{\bgroup\makeatletter\Stored@Macro}
\long\def\Stored@Macro#1{\egroup\Restore@Macro#1#1}
```

It happended (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
```
\long\def\StoringAndRelaxingDo{%
  \def\do##1{\expandafter\let\csname␣/gml/store\string##1%
        \endcsname##1%
    \let##1\relax}}
```

And here is the counter-definition for restore.

\RestoringDo
```
\long\def\RestoringDo{%
  \def\do##1{%
    \expandafter\let\expandafter##1\csname␣/gml/store\string##1%
          \endcsname}}
```

And to store a cs as explicitly named cs, i.e. to `\let` one csname another:

```
\def\@namelet#1#2{%
  \edef\@tempa{%
    \let\expandafter\noexpand\csname#1\endcsname
    \expandafter\noexpand\csname#2\endcsname}%
  \@tempa}
```

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.

```
\newcounter{gm@PronounGender}
\newcommand*\gm@atppron[2]{%
  \stepcounter{gm@PronounGender}% remember \stepcounter is global.
  \ifodd\arabic{gm@PronounGender}#1\else#2\fi}
\newcommand*\heshe{\gm@atppron{he}{she}}
\newcommand*\hisher{\gm@atppron{his}{her}}
\newcommand*\himher{\gm@atppron{him}{her}}
\newcommand*\hishers{\gm@atppron{his}{hers}}
\newcommand*\HeShe{\gm@atppron{He}{She}}
\newcommand*\HisHer{\gm@atppron{His}{Her}}
\newcommand*\HimHer{\gm@atppron{Him}{Her}}
\newcommand*\HisHers{\gm@atppron{His}{Hers}}
```

To Save Precious Count Registers

It's a contribution to TEX's ecology ;-). You can use as many CSs as you wish and you may use only 256 count registers (although in eTEX there are 2^{16} count registers, which makes the following a bit obsolete).

```
\newcommand*\nummacro[1]{\gdef#1{0}}
\newcommand*\stepnummacro[1]{%
  \@tempcnta=#1\relax
  \advance\@tempcnta␣by1\relax
  \xdef#1{\the\@tempcnta}}% Because of some mysterious reasons explicit \count0
        interferred with page numbering when used in \gmd@evpaddonce in gmdoc.
\newcommand*\addtonummacro[2]{%
  \count0=#1\relax
  \advance\count0by#2\relax
  \xdef#1{\the\count\z@}}
```

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 TEX's count CS and LATEX's counter. The macros `\stepnummacro` and `\addtonummacro` are analogous to LATEX'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 LaTeX'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. pdfTeX 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`.

```
\AtBeginDocument{% because we don't know when exactly hyperref is loaded and
      maybe after this package.
  \@ifpackageloaded{hyperref}{\newcounter{NoNumSecs}%
    \setcounter{NoNumSecs}{617}% to make \refing to an unnumbered section
        visible (and funny?).
    \def\gm@hyperrefstepcounter{\refstepcounter{NoNumSecs}}%
    \DeclareRobustCommand*\gm@targetheading[1]{%
      \hypertarget{#1}{#1}}}% end of then
  {\def\gm@hyperrefstepcounter{}%
    \def\gm@targetheading#1{#1}}% end of else
}% of \AtBeginDocument
```

Auxiliary macros for the kernel sectioning macro:

```
\def\gm@dontnumbersectionsoutofmainmatter{%
  \if@mainmatter\else␣\HeadingNumberedfalse␣\fi}
\def\gm@clearpagesduetoopenright{%
  \if@openright\cleardoublepage\else␣\clearpage\fi}
```

To avoid `\def`ing 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`

```
\long\def\@ifnotmw#1#2{\@ifundefined{mw@sectionxx}{#1}{#2}}
\@ifnotmw{%
  \StoreMacro\def␣\def\def#14#2{\RestoreMacro\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

[3] A. Berg, *Wozzeck*.

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:

```
\def\mw@sectionxx#1#2[#3]#4{%
  \edef\mw@HeadingLevel{\csname␣#1@level\endcsname
        \space}% space delimits level number!
  \ifHeadingNumbered
      \ifnum␣\mw@HeadingLevel>\c@secnumdepth␣\HeadingNumberedfalse␣\fi
```

line below is in ifundefined to make it work in classes other than mwbk

```
      \@ifundefined{if@mainmatter}{}{%
            \gm@dontnumbersectionsoutofmainmatter}
  \fi
%   \ifHeadingNumbered
%     \refstepcounter{#1}%
%     \protected@edef\HeadingNumber{\csname the#1\endcsname\relax}%
%   \else
%     \let\HeadingNumber\@empty
%   \fi
  \def\HeadingRHeadText{#2}%
  \def\HeadingTOCText{#3}%
  \def\HeadingText{#4}%
  \def\mw@HeadingType{#1}%
  \if\mw@HeadingBreakBefore
    \if@specialpage\else\thispagestyle{closing}\fi
    \@ifundefined{if@openright}{}{\gm@clearpagesduetoopenright}%
    \if\mw@HeadingBreakAfter
      \thispagestyle{blank}\else
      \thispagestyle{opening}\fi
       \global\@topnum\z@
  \fi% of \if\mw@HeadingBreakBefore
```

placement of `\refstep` suggested by me (GM)

```
  \ifHeadingNumbered
    \refstepcounter{#1}%
    \protected@edef\HeadingNumber{\csname␣the#1\endcsname\relax}%
  \else
    \let\HeadingNumber\@empty
    \gm@hyperrefstepcounter
  \fi% of \ifHeadingNumbered
  \if\mw@HeadingRunIn
    \mw@runinheading
  \else
    \if\mw@HeadingWholeWidth
      \if@twocolumn
        \if\mw@HeadingBreakAfter
```

```
        \onecolumn
        \mw@normalheading
        \pagebreak\relax
              \if@twoside
                \null
                \thispagestyle{blank}%
                \newpage
              \fi% of \if@twoside
        \twocolumn
        \else
          \@topnewpage[\mw@normalheading]%
        \fi% of \if\mw@HeadingBreakAfter
      \else
        \mw@normalheading
        \if\mw@HeadingBreakAfter\pagebreak\relax\fi
      \fi% of \if@twocolumn
    \else
      \mw@normalheading
      \if\mw@HeadingBreakAfter\pagebreak\relax\fi
    \fi% of \if\mw@HeadingWholeWidth
  \fi% of \if\mw@HeadingRunIn
  }
```

(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.)

```
\@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those
      written with two optionals.
  \def\gm@secini{gm@la}%
\gm@secxx  \def\gm@secxx#1#2[#3]#4{%
    \ifx\gm@secstar\@empty
      \@namelet{gm@true@#1mark}{#1mark}% a little trick to allow a special ver-
            sion of the heading just to the running head.
      \@namedef{#1mark}##1{% we redefine \⟨sec⟩mark to gobble its argument and
            to launch the stored true marking command on the appropriate argu-
            ment.
        \csname␣gm@true@#1mark\endcsname{#2}%
        \@namelet{#1mark}{gm@true@#1mark}% after we've done what we wanted
              we restore original \#1mark.
      }%
      \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.
    \fi
    \expandafter\expandafter\csname\gm@secini#1\endcsname
    \gm@secstar{#4}}%
}{% 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.
  \def\gm@secini{gm@mw}%
  \let\gm@secmarkh\@gobble% in mwcls there's no need to make tricks for special
        version to running headings.
\gm@secxx  \def\gm@secxx#1#2[#3]#4{%
    \expandafter\expandafter\csname\gm@secini#1\endcsname
    \gm@secstar[#2][#3]{#4}}%
}

\def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
\def\gm@secx#1[#2]{%
  \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}% if there's only
        one optional, we double it not the mandatory argument.

\def\gm@straightensec#1{% the parameter is for the command's name.
  \@ifundefined{#1}{}{% we don't change the ontological status of the command
        because someone may test it.
    \@namelet{\gm@secini#1}{#1}%
    \@namedef{#1}{%
      \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
        \def\gm@secstar{}\gm@sec{#1}}}}%
}%

\let\do\gm@straightensec
\do{part}\do{chapter}\do{section}\do{subsection}\do{subsubsection}
\@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?
```

Varia

LaTeX 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
```
\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
```
\long\def\addto@macro#1#2{%
  \toks@\expandafter{#1#2}%
  \edef#1{\the\toks@}%
}% (\toks@ is a scratch register, namely \toks0.)
```

And for use in the very document,

\addtomacro
```
\let\addtomacro=\addto@macro
```

'(LA)TEX' in my opinion better describes what I work with/in than just 'LaTeX'.

\LaTeXpar
```
\DeclareRobustCommand*{\LaTeXpar}{(L\kern-.36em%
        {\sbox\z@␣T%
         \vbox␣to\ht\z@{\hbox{\check@mathfonts
                              \fontsize\sf@size\z@
                              \math@fontsfalse\selectfont
                              A}%
                       \vss}%
        }%
        \kern-.07em% originally −, 15 em
        )\TeX}
```

\@emptify \emptify
```
\newcommand*\@emptify[1]{\let#1=\@empty}
\@ifdefinable\emptify{\let\emptify\@emptify}
```

Note the two following commands are in fact one-argument.

\g@emptify \gemptify
```
\newcommand*\g@emptify{\global\@emptify}
\@ifdefinable\gemptify{\let\gemptify\g@emptify}
```

\@relaxen \relaxen
```
\newcommand*\@relaxen[1]{\let#1=\relax}
\@ifdefinable\relaxen{\let\relaxen\@relaxen}
```

Note the two following commands are in fact one-argument.

\g@relaxen \grelaxen
```
\newcommand*\g@relaxen{\global\@relaxen}
\@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
```
\def\gmshowlists{\showboxdepth=1000␣\showboxbreadth=1000␣\showlists}
```

\nameshow
```
\newcommand*\nameshow[1]{\expandafter\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
```
\def\xiistring#1{%
```

```
  \if\noexpand#1\twelvespace
    \twelvespace
  \else
    \string#1%
  \fi}
```

A very neat macro provided by doc. I copy it ~verbatim.

* `\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 \activeate the space but also will (re)define it as the \␣ primitive. So define \gmobeyspaces that obeys this requirement.

(This definition is repeated in gmverb.)

```
\begin{catcode}`\␣\active
```

\gmobeyspaces `\gdef\gmobeyspaces{\catcode`\␣\active\let␣\␣}`

```
\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.

```
\bgroup\catcode`\^^M\active% the comment signs here are crucial.
\firstofone{\egroup%
```

\defobeylines `\newcommand*\defobeylines{\catcode`\^^M=13␣\def^^M{\par}}}%`

Another thing I dislike in LaTeX yet is doing special things for \...skip's, 'cause I like the Knuthian simplicity. So I sort of restore Knuthian meanings:

\deksmallskip `\def\deksmallskip{\vskip\smallskipamount}`
\undeksmallskip `\def\undeksmallskip{\vskip-\smallskipamount}`
\dekmedskip `\def\dekmedskip{\vskip\medskipamount}`
\dekbigskip `\def\dekbigskip{\vskip\bigskipamount}`

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 `\long\def\@firstofmany#1#2\@@nil{#1}`

A mark for the **TODO!**s:

\TODO
```
\newcommand*{\TODO}[1][]{{%
    \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 LaTeX macro instead.

\twocoltoc
```
\newcommand*\twocoltoc{%
  \RequirePackage{multicol}%
```

\@starttoc
```
  \def\@starttoc##1{%
    \begin{multicols}{2}\makeatletter\@input␣{\jobname␣.##1}%
      \if@filesw␣\expandafter␣\newwrite␣\csname␣tf@##1\endcsname
        \immediate␣\openout␣\csname␣tf@##1\endcsname␣\jobname␣.##1%
              \relax
```

```
      \fi
      \@nobreakfalse\end{multicols}}}
\@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
```
\newcommand\enoughpage[1]{%
  \par
  \dimen0=\pagegoal
  \advance\dimen0␣by-\pagetotal
  \ifdim\dimen0<#1\relax\newpage\fi}
```

The `\dots` didn't come out well. My small investigation revealed a mysterious replacement of the original LaTeX definition of `\textellipsis` with

```
> \textellipsis=macro:
->\PD1-cmd \textellipsis \PD1\textellipsis .
```

So, let's ensure `\dots` are given the proper kerning:

\ltxtextellipsis
```
\DeclareTextCommandDefault\ltxtextellipsis{%
  .\kern\fontdimen3\font
  .\kern\fontdimen3\font
  .\kern\fontdimen3\font}
```

\dots
```
\DeclareRobustCommand*\dots{%
  \ifmmode\mathellipsis\else\ltxtextellipsis\fi}
\let\ldots\dots
```

Two shorthands for debugging:

\tOnLine
```
\newcommand*\tOnLine{\typeout{\on@line}}
```

\OnAtLine
```
\let\OnAtLine\on@line
```

An equality sign properly spaced:

\equals
```
\newcommand*\equals{${}={}$}
```

And for the LaTeX's pseudo-code statements:

\eequals
```
\newcommand*\eequals{${}=={}$}
```

The job name without extension.

```
\def\gm@jobn#1.#2\@@nil{#1}
```

\jobnamewoe
```
\def\jobnamewoe{\expandafter\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 extrension.

```
\endinput
```

e. The gmiflink Package[1]

Written by Grzegorz 'Natror' Murzynowski,
natror at o2 dot pl


```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmiflink}
      [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* LaTeX 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 authomatic 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 ommitted.

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.

gmiflinkDoc.tex
gmiflinkDoc.pdf

The Code

```
\@ifpackageloaded{hyperref}{}{\message␣{^^J^^J␣gmiflink␣package:
    There's␣no␣use␣of␣me␣without␣hyperref␣package,␣I␣end␣my␣
          input.^^J}\endinput}
\providecommand\empty{}
```

A new counter, just in case

```
\newcounter{GMhlabel}
\setcounter{GMhlabel}{0}
```

The macro given below creates both hypertarget and hyperlabel, so that you may reference both ways: via `\hyperlink` and via `\ref`. It's pattern is the `\label` macro, see LaTeX 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
```
\DeclareRobustCommand*\gmhypertarget{%
  \@ifnextchar[{\gm@hypertarget}{\@dblarg{\gm@hypertarget}}}
\def\gm@hypertarget[#1]#2{% If argument #1 = \empty, then we'll use #2, i.e., the
      same as name of hypertarget.
  \refstepcounter{GMhlabel}% we \label{\gmht@firstpar}
  \hypertarget{#1}{#2}%
  \protected@write\@auxout{}{%
    \string\newlabel{#1}{{#2}{\thepage}{\relax}{GMhlabel.\arabic{%
        GMhlabel}}{}}}%
}% end of \gm@hypertartget.
```

We define a macro such that if the target exists, it makes `\ref`, else it typesets ordinary text.

\gmifref
```
\DeclareRobustCommand*\gmifref{\@ifnextchar[{\gm@ifref}{% ]
    \@dblarg{\gm@ifref}}}
\def\gm@ifref[#1]#2{%
  \expandafter\ifx\csname␣r@#1\endcsname\relax\relax%
  #2\else\ref{#1}\fi%
}% end of \gm@ifref
```

\gmiflink
```
\DeclareRobustCommand*\gmiflink{\@ifnextchar[{\gm@iflink}{%
    \@dblarg{\gm@iflink}}}
\def\gm@iflink[#1]#2{%
  \expandafter\ifx\csname␣r@#1\endcsname\relax\relax%
  #2\else\hyperlink{#1}{#2}\fi%
}% 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 `\noexpand`ed in `\edef`s.

```
\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 Package[1]

December 1, 2006

This is (a documentation of) file gmverb.sty, intended to be used with LaTeX 2ε as a package for a slight redefinition of the `\verb` macro and `verbatim` environment and for short verb marking such as `|\mymacro|`.

Written by Grzegorz 'Natror' Murzynowski,
natror at o2 dot pl

Many thanks to my TeX Guru Marcin Woliński for his TeXnical support.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmverb}
    [2006/12/01␣v0.78␣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
\fixlbrace left brace: `\fixlbrace`.)

The default 'hyphen' is `%` since it's the default comment char. If you wish another
\VerbHyphen char to appear at the linebreak, use the `\VerbHyphen` declaration that takes `\`⟨*char*⟩ 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 v0.78 dated 2006/12/01.

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 achieve `\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 TEXbook's \dekclubs format, gmverb provides a macro `\dekclubs` as a shorthand for `\MakeShortVerb*\|`.

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 \DeleteShortVerb 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` in the math mode behave as if they were 'other', so you may type e.g., `$|$` to achieve | and `+ \active`ated this way is in the math mode typeset properly etc.

\dekclubs There's one more declaration provided by gmverb: `\dekclubs`, which is a shorthand for `\MakeShortVerb*\|` .

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 ommitted.

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

\firstofone
\afterfi

```
\long\def\firstofone#1{#1}
\long\def\afterfi#1\fi{\fi#1}
```

The standard `\obeyspaces` command is only re`\catcode`ing of the space to be $_{13}$. Since we'll know a bit of where these macros are used, we know we have also to (re)define such an active space to be some space.

\gmobeyspaces

```
\begin{catcode}`\␣\active
\gdef\gmobeyspaces{\catcode`\␣\active\let␣\␣}
\end{catcode}
```

(The above three preliminary definitions are present also in gmutils.)

```
\bgroup
\@makeother\%
\firstofone{\egroup
  \def\twelvepercent{%}}
```

\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
```
\let\verbhyphen\twelvepercent
```

Provide a declaration for easy changing it. Its argument should be of $\backslash\langle char\rangle$ form (of course, a $\langle char\rangle_{12}$is also allowed).

\VerbHyphen
```
\def\VerbHyphen#1{%
  {\escapechar\m@ne
    \expandafter\gdef\expandafter\verbhyphen\expandafter{\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'.

```
\bgroup␣\catcode`\<=1␣\@makeother\{␣\catcode`\>=2␣%
\firstofone<\egroup
```
\breaklbrace
\twelvelbrace
```
   \def\breaklbrace<\discretionary<{\verbhyphen><><{>>%
   \def\twelvelbrace<{>%
>% of \firstofone
```

```
\bgroup␣\catcode`\<=1␣\catcode`\{=\active␣\catcode`\>=2
\firstofone<\egroup
   \def\dobreaklbrace<\catcode`\{=\active␣\def{<\breaklbrace>>%
>% end of \firstofone.
```

The \bslash macro defined below I use also in more 'normal' TEXing, e.g., to \typeout some \outer macro's name.

```
{\catcode`\!=0␣\@makeother\\%
```
\bslash
```
!gdef!bslash{\}%
```
\breakbslash
```
!gdef!breakbslash{!discretionary{!verbhyphen}{\}{\}}%
}
```

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 183. For the others let's give the user a countercrank:

\fixbslash
```
\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 183.

Analogously, let's provide a possibility of 'fixing' the left brace:

\fixlbrace
```
\newcommand*\fixlbrace{\let\breaklbrace=\twelvelbrace}
```

```
{\catcode`\!=0␣%
!catcode`!\=!active
!gdef!dobreakbslash{!catcode`!\=!active␣!def\{!breakbslash}}%
}
```

The macros defined below, \visiblebreakspaces and \twelveclub we'll use in the almost Knuthian macro making verbatim. This 'almost' makes a difference.

```
\bgroup\catcode`\␣=12␣%
\firstofone{\egroup%
\def\twelvespace{␣}%
\def\breakabletwelvespace{\discretionary{␣}{}{␣}}}
\bgroup\obeyspaces% it's just re\catcode'ing.
\firstofone{\egroup%
\newcommand*\activespace{␣}%
\newcommand*\dobreakvisiblespace{\let␣=\breakabletwelvespace\obeyspaces}%
      %\defing it caused a stack overflow disaster with gmdoc.
\newcommand*\dobreakblankspace{\let␣=\space\obeyspaces}%
}
\bgroup\@makeother\|
\firstofone{\egroup\def\twelveclub{|}}
```

Almost-Knuthian `\ttverbatim`

`\ttverbatim` comes from The TEXbook too, but I add into it a LATEX macro changing the `\catcode`s and make spaces visible and breakable and left braces too.

```
\newcommand*\ttverbatim{%
  \let\do=\do@noligs␣\verbatim@nolig@list
  \let\do=\@makeother␣\dospecials
  \dobreaklbrace\dobreakbslash
  \dobreakspace
  \tt}
```

We wish the visible spaces to be the default.

```
\let\dobreakspace=\dobreakvisiblespace
```

The Core: From shortvrb

The below is copied verbatim ;-) from doc.pdf and then is added my slight changes.

```
\def\MakeShortVerb{%
  \@ifstar
  {\def\@shortvrbdef{\verb*}\@MakeShortVerb}%
  {\def\@shortvrbdef{\verb}\@MakeShortVerb}}
\def\@MakeShortVerb#1{%
  \expandafter\ifx\csname␣cc\string#1\endcsname\relax
  \@shortvrbinfo{Made␣}{#1}\@shortvrbdef
  \add@special{#1}%
  \AddtoPrivateOthers#1% a macro to be really defined in gmdoc.
  \expandafter
  \xdef\csname␣cc\string#1\endcsname{\the\catcode`#1}%
  \begingroup
  \catcode`\~\active␣\lccode`\~`#1%
  \lowercase{%
    \global\expandafter\let
    \csname␣ac\string#1\endcsname~%
    \expandafter\gdef\expandafter~\expandafter{%
      \expandafter\ifmmode\expandafter\string\expandafter~%
```

```
      \expandafter\else\expandafter\afterfi\@shortvrbdef~\fi}}%
  \endgroup
  \global\catcode`#1\active
 \else
 \@shortvrbinfo\@empty{#1 already}{\@empty\verb(*)}%
 \fi}
\def\DeleteShortVerb#1{%
  \expandafter\ifx\csname cc\string#1\endcsname\relax
  \@shortvrbinfo\@empty{#1 not}{\@empty\verb(*)}%
  \else
  \@shortvrbinfo{Deleted }{#1 as}{\@empty\verb(*)}%
  \rem@special{#1}%
  \global\catcode`#1\csname cc\string#1\endcsname
  \global \expandafter\let \csname cc\string#1\endcsname \relax
  \ifnum\catcode`#1=\active
  \begingroup
  \catcode`\~\active \lccode`\~`#1%
  \lowercase{%
    \global\expandafter\let\expandafter~%
    \csname ac\string#1\endcsname}%
  \endgroup \fi \fi}
```

My little addition

```
\@ifpackageloaded{gmdoc}{%
  \def\gmv@packname{gmdoc}}{%
  \def\gmv@packname{gmverb}}
\def\@shortvrbinfo#1#2#3{%
  \PackageInfo{\gmv@packname}{%
   ^^J\@empty #1\expandafter\@gobble\string#2 a short reference
    for \expandafter\string#3}}
\def\add@special#1{%
  \rem@special{#1}%
  \expandafter\gdef\expandafter\dospecials\expandafter
  {\dospecials \do #1}%
  \expandafter\gdef\expandafter\@sanitize\expandafter
  {\@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.

```
\def\rem@special#1{%
  \def\do##1{%
    \ifnum`#1=`##1 \else \noexpand\do\noexpand##1\fi}%
  \xdef\dospecials{\dospecials}%
  \begingroup
```

```
  \def\@makeother##1{%
    \ifnum`#1=`##1␣\else␣\noexpand\@makeother\noexpand##1\fi}%
  \xdef\@sanitize{\@sanitize}%
  \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
\verbatim
verbatim*
\endverbatim
\@xverbatim
\@sxverbatim
\@verbatim

```
\def\verbatim{\@beginparpenalty␣\predisplaypenalty␣\@verbatim
  \frenchspacing␣\gmobeyspaces␣\@xverbatim}% in the LaTeX version there's %%
        \@vobeyspaces instead of \gmobeyspaces.
\@namedef{verbatim*}{\@beginparpenalty␣\predisplaypenalty␣\@verbatim
  \@sxverbatim}
\def\endverbatim{\@@par
  \ifdim\lastskip␣>\z@
    \@tempskipa\lastskip␣\vskip␣-\lastskip
    \advance\@tempskipa\parskip␣\advance\@tempskipa␣-\@outerparskip
    \vskip\@tempskipa
  \fi
  \addvspace\@topsepadd
  \@endparenv}
\expandafter\let\csname␣endverbatim*\endcsname␣=\endverbatim
\begingroup␣\catcode␣`!=0␣%
\catcode␣`[=␣1␣\catcode`]=2␣%
\catcode`\{=\active
\@makeother\}%
\catcode`\\=\active%
!gdef!@xverbatim[%
  !edef!verbatim@edef[%
    !def!noexpand!verbatim@end%
      ####1!noexpand\end!noexpand{!@currenvir}[%
      ####1!noexpand!end[!@currenvir]]]%
  !verbatim@edef
  !verbatim@end]%
!endgroup
\let\@sxverbatim=\@xverbatim
```

F. Mittelbach says the below is copied almost verbatim from LaTeX source, modulo `\check@percent`.

```
\def\@verbatim{%
```

Originally here was just `\trivlist␣\item[]`, but it worked badly in my document(s), so let's take just highlights of if.

```
  \parsep\parskip
```

From `\@trivlist`:

```
  \if@noskipsec␣\leavevmode␣\fi
```

```
  \@topsepadd␣\topsep
  \ifvmode
    \advance\@topsepadd␣\partopsep
  \else
    \unskip␣\par
  \fi
  \@topsep␣\@topsepadd
  \advance\@topsep␣\parskip
  \@outerparskip␣\parskip
```

(End of `\trivlistlist` and `\@trivlist` highlights.)

```
  \@@par\addvspace\@topsep
  \if@minipage\else\vskip\parskip\fi
  \leftmargin\parindent% please notify me if it's a bad idea.
  \advance\@totalleftmargin\leftmargin
  \raggedright
  \leftskip\@totalleftmargin% so many assignments to preserve the list thinking 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(*).
  \@@par% most probably redundant.
  \@tempswafalse
  \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
    \ifvmode\if@tempswa\addvspace\stanzaskip\@tempswafalse\fi\fi
    \@@par
    \penalty\interlinepenalty␣\check@percent}%
  \everypar{\@tempswatrue\hangindent\verbatimhangindent\hangafter\@ne}% since several chars are breakable, there's a possibility of breaking some lines. We wish them to be hanging indented.
  \obeylines
  \ttverbatim}
\@ifundefined{stanzaskip}{\newlength\stanzaskip}{}
\stanzaskip=\medskipamount
\newlength\verbatimhangindent
\verbatimhangindent=3em
\providecommand*\check@percent{}
```

\verbatimhangindent

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.

```
\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*  \def\verb{\relax\ifmmode\hbox\else\leavevmode\null\fi
\verb    \bgroup
         \ttverbatim
         \gm@verb@eol
         \@ifstar{\@sverb@chbsl}{\gmobeyspaces\frenchspacing\@sverb@chbsl}}% in
             the LaTeX version there's \@vobeyspaces instead of \gmobeyspaces.
         \def\@sverb@chbsl#1{\@sverb#1\check@bslash}
         \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  \def\check@bslash{\@ifnextchar{\@def@breakbslash}{\bslash\@gobble}{}}
               \let\verb@balance@group\@empty
\verb@egroup   \def\verb@egroup{\global\let\verb@balance@group\@empty\egroup}
\gm@verb@eol   \let\gm@verb@eol\verb@eol@error
```

The latter is a LaTeX 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.

```
               \begingroup
               \obeylines\obeyspaces%
\verb@eolOK    \gdef\verb@eolOK{\obeylines%
               \def^^M{␣\check@percent}%
               }%
               \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     \def\verbeolOK{\let\gm@verb@eol\verb@eolOK}
```

And back to the main matter,

```
               \def\@sverb#1{%
                 \catcode`#1\active␣\lccode`\~`#1%
                 \gdef\verb@balance@group{\verb@egroup
                   \@latex@error{Illegal␣use␣of␣\bslash␣verb␣command}\@ehc}%
                 \aftergroup\verb@balance@group
                 \lowercase{\let~\verb@egroup}}
\verbatim@nolig@list  \def\verbatim@nolig@list{\do\`\do\<\do\>\do\,\do\'\do\-}
\do@noligs     \def\do@noligs#1{%
                 \catcode`#1\active
                 \begingroup
                 \lccode`\~=`#1\relax
                 \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
```
\def\dekclubs{\MakeShortVerb*\|}
```

doc- And shortvrb-Compatibility

One of minor errors while TEXing 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
```
\def\old@MakeShortVerb#1{%
  \expandafter\ifx\csname␣cc\string#1\endcsname\relax
  \@shortvrbinfo{Made␣}{#1}\@shortvrbdef
  \add@special{#1}%
  \AddtoPrivateOthers#1% a macro to be really defined in gmdoc.
  \expandafter
  \xdef\csname␣cc\string#1\endcsname{\the\catcode`#1}%
  \begingroup
  \catcode`\~\active␣\lccode`\~`#1%
  \lowercase{%
    \global\expandafter\let
    \csname␣ac\string#1\endcsname~%
    \expandafter\gdef\expandafter~\expandafter{%
      \@shortvrbdef~}}%
  \endgroup
  \global\catcode`#1\active
 \else
 \@shortvrbinfo\@empty{#1␣already}{\@empty\verb(*)}%
 \fi}
```

\OldMakeShortVerb
```
\def\OldMakeShortVerb{\begingroup
  \let\@MakeShortVerb=\old@MakeShortVerb
  \@ifstar{\eg@MakeShortVerbStar}{\eg@MakeShortVerb}}
```

\eg@MakeShortVerbStar
```
\def\eg@MakeShortVerbStar#1{\MakeShortVerb*#1\endgroup}
```

\eg@MakeShortVerb
```
\def\eg@MakeShortVerb#1{\MakeShortVerb#1\endgroup}
```

```
\endinput% for the Tradition.
```

Change History

gmdoc v0.96
- General:
  - CheckSum 2395, 3

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. 62.

gmdoc v0.99a
- General:
  - CheckSum 4479, 3

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, 108
  - 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`, 94

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-450, a-560, a-1054, a-1667, d-363
\+, a-1054, a-1532
\-, a-1054, d-149, f-200
\<...>, d-135, *82*
\@@codeline@wrindex, a-707
\@@nil, a-625, a-666, a-835, a-840, a-1133, a-1135, a-1143, d-140, d-141, d-143, d-374, d-402, d-403
\@@par, a-123, a-314, a-325, a-358, a-888
\@@settexcodehangi, a-50, a-50, a-182, a-215
\@M, a-1137
\@MakeShortVerb, a-1679, f-57, f-58, f-59, f-227
\@NoEOF, a-1687, a-1689
\@aalph, a-1305, a-1306
\@addtoreset, a-1331
\@aftercodegfalse, a-199, a-317, a-367, a-379
\@aftercodegtrue, a-82, a-201, a-219, a-306, a-367, a-1069, a-1075
\@afternarrgfalse, a-82, a-201, a-306, a-368, a-1069, a-1075
\@afternarrgtrue, a-116, a-368
\@badend, d-55
\@beginputhook, *a-98*, a-137, a-138
\@begnamedgroup, d-41, d-42, d-48, d-51
\@charlb, a-1297
\@charrb, a-1299
\@checkend, d-52
\@clubpenalty, a-92
@codeskipput, a-359, *34*
\@codeskipputgfalse, a-116, a-187, a-307, a-359, a-1069, a-1076, a-1518
\@codeskipputgtrue, a-76, a-80, a-82, a-189, a-199, a-317, a-358, a-359, a-374, a-762, a-766, a-829, a-832
\@codespacesblanktrue, a-24
\@codetonarrskip, a-99, a-164, a-171, a-296, a-305, a-324, a-336, a-369, a-389
\@countalllinestrue, a-10
\@ctrerr, a-1312
\@currenvir, a-790, a-803, a-804, d-45, d-54, f-137, f-138
\@currenvir*, a-784
\@currenvline, d-46
\@currsize, d-60, d-61, d-62, d-63, d-64, d-65, d-66, d-67, d-68, d-69
\@debugtrue, b-13
\@def@breakbslash, f-182, f-183
\@defentryze, a-568, a-572, a-574, a-712
\@docinclude, a-1259, a-1260
\@dsdirgfalse, a-194, a-203, a-228, a-256, a-292, a-299, a-301, a-820
\@dsdirgtrue, a-118, a-185
\@emptify, a-144, a-176, a-538, a-579, a-635, a-675, a-680, a-681, a-1021, a-1061, a-1063, a-1130, a-1304, a-1414, a-1474, a-1523, a-1524, a-1590, a-1594, a-1672, a-1673, a-1674, d-347, d-348, d-349
\@endinputhook, *a-110*, a-135, a-136
\@fileswfalse, a-1507
\@firstofmany, a-625, a-666, d-374
\@fshdafalse, a-1492, a-1494
\@fshdatrue, a-1492, a-1493
\@gif, d-7, d-8, d-10
\@gmccnochangesfalse, b-16
\@gmccnochangestrue, b-16, b-17
\@ifQueerEOL, a-141, a-142, a-399, a-405, a-437, a-1049, a-1187
\@ifismember, a-477, a-483
\@ifncat, d-22, d-23, d-35
\@ifnextcat, a-459, a-473, d-18
\@ifnotmw, b-53, d-241, d-242, d-298, d-330
\@ifstarl, a-562, a-565, a-582, a-592, a-617, a-640, a-647, a-684, a-687, a-726, a-739, a-1615
\@indexallmacrosfalse, a-15
\@indexallmacrostrue, a-15, a-16
\@latexerr, a-1258, a-1373
\@linesnotnumalse, a-5
\@linesnotnumtrue, a-5, a-6
\@ltxDocIncludefalse, a-1406
\@ltxDocIncludetrue, a-1406, a-1410
\@makefntext, a-1425
\@marginparsusedfalse, a-17, a-22

\@marginparsusedtrue, a-17, a-18, a-19, a-20, a-21
\@namelet, a-771, a-772, d-202, d-302, d-305, d-323
\@newlinegfalse, a-83, a-169, a-204, a-266, a-276, a-283
\@newlinegtrue, a-83, a-117, a-184
\@nobreakfalse, d-384
\@noindexfalse, a-11
\@noindextrue, a-11, a-12
\@oarg, d-156, d-157, d-158
\@oargsq, d-156, d-158, d-165
\@oldmacrocode, a-785, a-799
\@oldmacrocode@launch, a-773, a-775, a-776
\@onlypreamble, a-1412, a-1645, a-1647, a-1649, d-385
\@pageinclindexfalse, a-521, a-539
\@pageinclindextrue, a-539, a-758
\@pageindexfalse, a-13, a-1646
\@pageindextrue, a-13, a-14, a-542, a-1648
\@parg, d-159, d-160, d-161
\@pargp, d-159, d-161, d-165
\@relaxen, a-72, a-339, a-351, a-995, a-1129, a-1158, a-1287, a-1314, a-1463, a-1464, a-1579, a-1583, a-1659, a-1688, d-351, d-352, d-353
\@shortvrbdef, f-57, f-58, f-61, f-73, f-209, f-220
\@shortvrbinfo, f-61, f-77, f-81, f-83, f-97, f-209, f-224
\@starttoc, d-379
\@sverb@chbsl, f-180, f-181
\@topnewpage, d-286
\@topsep, f-152, f-153, f-155
\@topsepadd, f-126, f-146, f-148, f-152
\@trimandstore, a-119, a-163, a-380, a-380, a-385, a-387
\@trimandstore@hash, a-381, a-382
\@trimandstore@ne, a-385, a-387
\@uresetlinecountfalse, a-7
\@uresetlinecounttrue, a-7, a-8
\@usgentryze, a-577, a-585, a-589, a-619, a-621, a-717, a-735, a-1619, a-1624
\@xifncat, d-25, d-35
^^A, *6*, a-403
^^B, *6*, a-397
\^^M, a-424
^^M, a-100, a-183

\aalph, a-1305, a-1332
\abovedisplayskip, a-62
\activespace, f-42
\actualchar, *17*, a-439, a-482, a-527, a-1095, a-1119, a-1124, a-1242, a-1371, a-1665
\add@special, f-62, f-101, f-210
\addto@estoindex, a-571, a-588, a-595, a-711, a-716, a-721
\addto@estomarginpar, a-655, a-710, a-715, a-718
\addto@macro, d-332, d-336
\addtomacro, a-720, a-723, a-773, a-774, a-860, d-336
\addtonummacro, d-224
\AddtoPrivateOthers, *16*, a-429, f-63, f-175, f-211
\afterelsefi, a-143, a-229, a-268, a-459, a-485, a-841, d-38, d-138
\afterelsefifi, a-257, a-821, a-876, d-37, d-180
\afterelseiffifi, a-238, a-251, d-40
\afterfi, a-143, a-232, a-270, a-385, a-428, a-467, a-486, a-613, a-824, a-844, b-45, d-36, d-138, d-164, f-5, f-73
\afterfifi, a-243, a-259, a-822, a-882, d-39, d-183
\AfterMacrocode, a-1589
\AlsoImplementation, *17*, a-1653, a-1656
\AltMacroFont, a-1674
\AmSTeX, 18, a-1534
\and, a-1457, a-1466
\arg, d-163, d-164
article, b-10
\AtBeginDocument, a-37, a-71, a-481, a-543, a-698, a-1184, a-1644, b-32, d-162, d-228
\AtBegInput, *8*, a-137, a-139, a-145, a-399, a-405, a-427, a-434, a-1419, a-1420, a-1504
\AtBegInputOnce, 7, *8*, a-146, a-1363, a-1678
\AtDIPrologue, *16*, a-1022
\AtEndInput, *8*, a-135, a-1195, a-1630, a-1639
\author, a-1451, *c-5*
\AVerySpecialMacro, a-1685

\begin, d-50, d-51
\begin*, d-51
\belowdisplayshortskip, a-64, a-65, a-66
\belowdisplayskip, a-63
\BibTeX, 18, a-1537
\bigskipamount, d-373
\bnamegroup, d-48
\breakabletwelvespace, a-151, a-225, f-39, f-43
\breakbslash, f-28, f-30, f-34, f-182
\breaklbrace, f-19, f-24, f-31
\bslash, a-482, a-502, a-528, a-540, a-625, a-634, a-666, a-674, a-1090, a-1104, a-1105, a-1119, a-1121, a-1531, a-1592, a-1604, a-1635, d-108, d-148, f-27, f-30, f-183, f-197

\c@ChangesStartDate, a-1131, a-1134, a-1143, a-1145, a-1146, a-1147
\c@CheckSum, a-1193, a-1201, a-1206, a-1216, a-1225, a-1228
\c@codelinenum, a-265, a-341, a-697, a-1060, a-1061, a-1593
\c@footnote, a-1455, a-1473
\c@GlossaryColumns, a-1160, a-1160, a-1162
\c@gmd@mc, a-1588, a-1592

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty

\c@IndexColumns, a-1025, a-1025, a-1027, a-1046
\c@secnumdepth, d-248
\catactive, *18*, a-1512
\catletter, *18*, a-1513
\catother, *18*, a-1511
\CDAnd, *19*, a-1575
\CDPerc, *19*, a-1576
\changes, a-1084, a-1089, a-1093
\changes@, a-1088, a-1097
\ChangesStart, 14, a-1142
ChangesStartDate, *14*
\Character@Table, a-1597, a-1602
\CharacterTable, a-1595
\check@bslash, f-181, f-183
\check@checksum, a-1195, a-1196
\check@percent, a-427, f-166, f-174, f-190
\check@sum, a-1191, a-1192, a-1197, a-1206, a-1215, a-1222
\CheckModules, a-1673
CheckSum, a-1193
\CheckSum, *14*, a-1192, a-1231
\chschange, a-1224, a-1226, a-1229
\chunkskip, *15*, *19*, a-73
\cleardoublepage, d-240
\clubpenalty, *a-92*, a-134
\cmd, d-153
\cmd@to@cs, d-153, d-154
\Code@CommonIndex, a-592, a-593
\Code@CommonIndexStar, a-592, a-594
\Code@DefEnvir, a-684, a-708
\Code@DefIndex, a-565, a-566, a-689, a-847
\Code@DefIndexStar, a-565, a-569, a-850
\Code@DefMacro, a-684, a-688
\Code@Delim, a-41, a-42, a-45
\code@delim, a-44, *a-96*, *a-104*, a-128, a-129, a-240, a-250, a-288, a-428, a-778, a-1496, a-1498
\Code@Delim@St, a-41, a-45
\code@escape@char, a-255, a-446
\Code@MarginizeEnvir, a-654, a-655
\Code@MarginizeMacro, a-649, a-650, a-690, a-693
\Code@UsgEnvir, a-687, a-713
\Code@UsgIndex, a-582, a-583, a-692, a-730
\Code@UsgIndexStar, a-582, a-586
\Code@UsgMacro, a-687, a-691
\CodeCommonIndex, a-590, a-1662
\CodeCommonIndex*, *12*
\CodeDefIndex, *12*, a-563, a-1660
\CodeDefIndex*, a-563
\CodeDefine, *11*, a-682
\CodeDefine*, a-682
\CodeDelim, *15*, a-41, a-46, a-1575, a-1576
\CodeDelim*, a-779, a-1497
\CodeEscapeChar, *15*, a-443, *a-448*, a-764, a-768, a-1629
\CodeIndent, *15*, a-52, a-53, a-197, a-331, a-426, a-1627, b-32, *80*
\codeline@wrindex, a-694, a-702, a-705, a-706
\CodelineIndex, a-1646, a-1647
codelinenum, *16*, a-344
\CodelineNumbered, a-1644, a-1645, *81*
\CodeMarginize, *11*, a-645
\CodeMarginize*, a-645
\CodeSpacesBlank, *9*, *a-94*, a-152, a-767
codespacesblank, *9*, a-24
\CodeSpacesSmall, a-155
\CodeTopsep, *15*, a-57, a-60, a-75, a-79, a-358, a-762, a-764, a-766, a-768, a-828, a-1628
\CodeUsage, *11*, a-685
\CodeUsage*, a-685
\CodeUsgIndex, *12*, a-580
\CodeUsgIndex*, a-580
\columnsep, a-1035
\CommonEntryCmd, *16*, a-512, a-559, *c-14*
\continue@macroscan, a-459, a-469, a-473
copyrnote, *18*, a-1515
\count, a-1138, a-1139, a-1140, d-225, d-226, d-227
countalllines, *9*, a-10
\cs, *17*, d-148, d-151, d-153
\currentfile, a-1246, a-1247, a-1248, a-1249, a-1250, a-1251, a-1252, a-1253, a-1256, a-1273, a-1277, a-1288, a-1344, a-1345, a-1347, a-1369, a-1370, a-1390, a-1394, a-1414

\date, a-1452, *c-6*
\day, a-1225, a-1227
debug, b-13, *87*
\Debug@dstron, a-1060, a-1061, a-1063, a-1070
\DeclareOption, a-6, a-8, a-10, a-12, a-14, a-16, a-21, a-22, a-24, b-7, b-8, b-9, b-10, b-11, b-13, b-14, b-17, b-19
\DeclareRobustCommand*, a-1667, d-56, d-84, d-85, d-86, d-87, d-88, d-89, d-121, d-144, d-147, d-148, d-151, d-232, d-337, d-363, d-395, e-9, e-17, e-23
\DeclareTextCommandDefault, d-391
\DefaultIndexExclusions, *13*, a-887, a-992, a-999
\DefEntry, *16*, a-557, a-1668
\definecolor, a-28
\defobeylines, d-369
\dekbigskip, d-373
\dekclubs, *10*, *b-61*, f-206, *113*
\dekmedskip, d-372
\deksmallskip, d-370
\DeleteShortVerb, *10*, f-79, *113*
\Describe, *11*, a-1613
\Describe@Env, a-1610, a-1612, a-1615, a-1621
\Describe@Macro, a-1610, a-1615, a-1616
\DescribeEnv, a-1611, *80*
\DescribeMacro, a-1608, *80*
\dimen, d-388, d-389, d-390
\DisableCrossrefs, a-1650, a-1652
\discre, a-1532, d-136, d-139
\discret, d-137, d-138

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty

\division, *18*, a-1356, a-1577, a-1579, a-1580
\Do@Index, a-994, a-995
\do@noligs, f-49, f-201
\do@properindex, a-603, a-636, a-756
\dobreakblankspace, f-44
\dobreakbslash, f-34, f-51
\dobreaklbrace, f-24, f-51
\dobreakspace, f-52, f-54
\dobreakvisiblespace, f-43, f-54
\Doc@Include, a-1237, a-1238
\Doc@Input, a-85, a-88, a-1684
\DocInclude, *7*, 9, *19*, a-1237, a-1254, a-1258, a-1373, *c-12*, *c-13*, *c-21*, *c-22*, *c-23*
\docincludeaux, a-1245, a-1313, a-1314, a-1368
\DocInput, *7*, a-85, a-1413, a-1418, a-1498
DocInputsCount, a-343
\docstrips@percent, a-783
\DocstyleParms, a-1671
\documentclass, *c-1*
\DoIndex, *13*, a-994, a-998, *c-11*
\DoNot@Index, a-867, a-868
\DoNotIndex, *12*, a-867, a-997, a-998, a-1000
\dont@index, a-870, a-871, a-876, a-882, a-995
\DontCheckModules, a-1672
\doprivateothers, a-430, a-431, a-451, a-452
\dots, d-395, d-397
\ds, 18, a-1554

\eequals, d-401
\eg@MakeShortVerb, f-228, f-230
\eg@MakeShortVerbStar, f-228, f-229
\egCode@MarginizeEnvir, a-647, a-653
\egCode@MarginizeMacro, a-647, a-648
\egRestore@Macro, d-186, d-187
\egStore@Macro, d-167, d-168
\egText@Marginize, a-739, a-740
\emptify, a-777, d-348, d-348
\EnableCrossrefs, a-1301, a-1651
\enamegroup, d-49
\encapchar, *17*, a-441, a-482, a-529, a-1096, a-1126
\endenvironment, a-866
\endlinechar, a-1565
\endmacro, a-832, a-834
\endmacro*, a-834
\endmacrocode, a-770
\endoldmc, a-770
\endtheglossary, a-1303
\endverbatim, f-120, f-128
\enoughpage, d-386
\enspace, b-56
\ensuremath, b-65, d-122, d-132
\EntryPrefix, *16*, a-523, a-525, a-538, a-1045, a-1241, a-1365
\env, *17*, d-151
\environment, a-865
environment, *12*, a-865
\envirs@toindex, a-579, a-663, a-677, a-678, a-681, a-723
\envirs@tomarginpar, a-657, a-660, a-661, a-680, a-720
\EOFMark, *15*, *a-103*, a-148, a-1688, b-65
\equals, d-400
\errorcontextlines, b-43
\eTeX, 18, a-1546, a-1547
\evensidemargin, a-1236
\everypar, a-99, *a-99*, a-119, a-163, a-164, a-170, a-182, a-296, a-305, a-323, a-335, a-385, a-392, a-861, a-1516, f-167
\exhyphenpenalty, b-63
\exii@currenvir, d-54, d-55

\file, *18*, *c-15*, *c-16*, d-145
\filedate, *19*, a-1349, a-1558, a-1573
\filediv, a-1315, a-1325, a-1355, a-1361, a-1463, a-1487, *c-15*
\filedivname, a-1316, a-1321, a-1324, a-1326, a-1331, a-1332, a-1333, a-1354, a-1360, a-1464
\fileinfo, *19*, a-1560
\filekey, a-1288, a-1335, a-1338
\filename, a-1348, a-1556
\filenote, *19*, a-1573, a-1574
\filesep, a-1240, a-1241, a-1304, a-1334, a-1364, a-1365
\fileversion, 19, a-1225, a-1226, a-1350, a-1559, a-1573
\Finale, *17*, a-1654, a-1659
\finish@macroscan, a-459, a-467, a-473, a-491
\fixbslash, f-30, *112*
\fixlbrace, f-31, *112*
\fontseries, b-49
\fullcurrentfile, a-1247, a-1256, a-1278, a-1370, a-1395

\g@emptify, a-147, a-478, a-661, a-678, a-1190, a-1431, a-1432, a-1490, a-1502, d-349, d-350
\g@relaxen, a-503, a-534, a-535, a-537, a-858, a-1489, d-353, d-354
\gaddtomacro, 17, a-147, a-426, d-331
\gag@index, a-37, a-704, a-1644, a-1650
\gemptify, d-350, d-350
\GeneralName, a-1113, a-1114, a-1130, a-1151, a-1242, a-1371
\generalname, a-1097, a-1100, a-1124, a-1156
\geometry, b-40
\GetFileInfo, *19*, a-1277, a-1347, a-1394, a-1555
\glet, a-111, a-215, a-477, a-651, a-778, a-1051, a-1339, a-1342, a-1353, d-17
\glossary@prologue, a-1149, a-1163, a-1178, a-1181, a-1340
\GlossaryMin, *13*, a-1159, a-1163
\GlossaryParms, *13*, a-1164, a-1185
\GlossaryPrologue, *13*, a-1177
\gm@atppron, d-208, d-211, d-212, d-213, d-214, d-215, d-216, d-217, d-218
\gm@clearpagesduetoopenright, d-239, d-257

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty

\gm@dontnumbersectionsoutofmain, d-237, d-249
\gm@hyperrefstepcounter, d-231, d-234, d-268
\gm@hypertarget, e-10, e-11
\gm@iflink, e-23, e-24, e-25
\gm@ifref, e-17, e-18, e-19
\gm@jobn, d-402, d-403
gm@PronounGender, d-207
\gm@pswords, d-140, d-141, d-143
\gm@sec, d-318, d-325, d-326
\gm@secini, d-299, d-309, d-312, d-315, d-323
\gm@secmarkh, d-313
\gm@secstar, d-301, d-307, d-310, d-316, d-325, d-326
\gm@secx, d-318, d-319
\gm@secxx, d-300, d-314, d-320
\gm@straightensec, d-321, d-328
\gm@targetheading, d-232, d-235
\gm@verb@eol, a-434, f-179, f-186, f-193
\gm@xistar, a-788, a-790
\gmboxedspace, a-1525, a-1526, a-1529, a-1532
\gmcc@baseclass, b-4, b-7, b-8, b-9, b-24, b-28
gmcc@mwcls, b-5
\gmcc@mwclsfalse, b-10, b-24
\gmcc@mwclstrue, b-6
\gmd@@iffalse, a-485, a-490
\gmd@@iftrue, a-486, a-489
\gmd@@macro, a-1680, a-1681
\gmd@@toc, a-139, a-141
\gmd@ABIOnce, a-144, a-145, a-147
\gmd@bslashEOL, a-415, a-425
\gmd@charbychar, a-194, a-220, a-247, a-247, a-271, a-498
\gmd@checkifEOL, a-165, a-294
\gmd@checkifEOLmixd, a-253, a-303
\gmd@chschangeline, a-1202, a-1209, a-1217, a-1223
\gmd@closingspacewd, a-186, a-417, a-418, a-420
\gmd@codecheckifds, a-819
\gmd@codeskip, a-199, a-317, a-358, a-362, a-374
\gmd@continuenarration, *a-130*, a-160, a-242
\gmd@countnarrationline, a-162, a-167, a-176, a-295, a-304
\gmd@countheline, a-257, a-271, a-273
\gmd@currentlabel@before, a-90, a-111
\gmd@currenvxistar, a-784, a-789
\gmd@DefineChanges, a-1083, a-1157
\gmd@dip@hook, a-1019, a-1021, a-1022
\gmd@docstripdirective, a-293, a-302, a-821, a-1052
\gmd@docstripinner, a-1058, a-1065
\gmd@docstripshook, a-1077
\gmd@docstripverb, a-1057, a-1072
\gmd@doIndexRelated, a-1272, a-1280, a-1300, a-1389, a-1398
\gmd@dolspaces, *a-131*, a-194, a-226
\gmd@DoTeXCodeSpace, a-125, a-150, a-153, a-156, a-780
\gmd@eatlspace, a-231, a-235, a-238
\gmd@endpe, a-309, a-311, a-322, a-327, a-328
\gmd@EOLorcharbychar, a-259, a-262
\gmd@evpaddonce, a-853, a-854
\gmd@guardedinput, a-101, a-108
\gmd@iedir, a-869, a-881, a-995
\gmd@ifonetoken, a-831, a-833, a-838, a-1610, a-1681
\gmd@ifsingle, a-835, a-840
\gmd@inputname, a-89, a-1199, a-1208, a-1214
\gmd@justadot, a-573, a-575, a-578, a-651, a-869
\gmd@ldspaceswd, a-202, a-209, a-210, a-217, a-224, a-230, a-237, a-241
gmd@mc, a-1585
\gmd@mcdiag, a-1587, a-1590, a-1591, a-1594
\gmd@mchook, a-1586
\gmd@modulehashone, a-1067, a-1070, a-1074, a-1078
\gmd@narrcheckifds, a-299, a-300
\gmd@narrcheckifds@ne, a-289, a-291
\gmd@nocodeskip, a-198, a-200, a-318, a-320, a-360, a-364, a-371, a-376
\gmd@oldmcfinis, a-804
\gmd@oncenum, a-855, a-857, a-859, a-860, a-862, a-864
\gmd@parfixclosingspace, a-181, a-416
\gmd@percenthack, a-251, a-287
\gmd@preverypar, a-47, a-171, a-297, a-305, a-324, a-336, a-383, a-390, a-392
\gmd@providefii, a-1567, a-1569
\gmd@resetlinecount, *a-97*, a-339, a-345
\gmd@revprefix, a-551, a-552
\gmd@setChDate, a-1133, a-1135, a-1143
\gmd@setclosingspacewd, a-419
\gmd@setclubpenalty, a-91, a-121, a-122, a-134
\gmd@skipgmltext, a-1502, a-1502, a-1508
\gmd@spacewd, a-223, a-229, a-237
\gmd@texcodeEOL, a-196, a-263
\gmd@texcodespace, a-154, a-158, a-193, a-225, a-227, a-236
\gmd@textEOL, a-100, a-115, a-298, a-308, a-437, a-777, a-1070, a-1078
\gmd@typesettexcode, a-180, a-233, a-243
\gmd@writeckpt, a-1282, a-1294, a-1400
\gmd@writemauxinpaux, a-1262, a-1289, a-1376
\gmdindexpagecs, a-546, a-550

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty

\gmdindexrefcs, a-545, a-546, a-548
\gmdmarginpar, *12*, *18*, a-746, a-751, a-754
\gmdnoindent, *19*, a-1520
\gmdoccMargins, b-39
\gmdocIncludes, *8*, a-1417
gmeometric, b-18
gmglo.ist, 63
\gmhypertarget, a-355, *c-16*, e-9, *109*
\gmiflink, a-548, e-23, *109*
\gmifref, e-17, *109*
\gml@StoreCS, d-173, d-182, d-192
\gml@storemacros, d-174, d-175, d-180, d-183, d-193
gmlonely, *18*, a-1499
\gmobeyspaces, *a-153*, d-365, f-7, f-117, f-180
\gmshowlists, d-355
\gmTheGeometry, b-18, b-21, b-38
\gmv@packname, f-95, f-96, f-98
\gobble, d-91
\gobbletwo, d-92
\grefstepcounter, a-169, a-204, a-276, a-283, d-15
\grelaxen, d-354, d-354

\HeadingNumber, d-265, d-267
\HeadingNumberedfalse, d-238, d-248
\HeadingRHeadText, d-251
\HeadingText, d-253
\HeadingTOCText, d-252
\HeShe, d-215
\heshe, *5*, d-211
\HimHer, d-217
\himher, d-213
\HisHer, d-216
\hisher, d-212
\HisHers, d-218
\hishers, d-214
\HLPrefix, *16*, a-356, a-523, a-525, a-555, a-697, a-1012, *a-1240*, a-1364
\Hybrid@DefEnvir, a-831, a-849
\Hybrid@DefMacro, a-831, a-846
hyperindex, 42
\hyperlabel@line, a-172, a-207, a-277, a-284, a-352
\hypersetup, a-29, a-1031
\hyphenpenalty, b-63, d-143

\if*, a-790
\if@aftercode, a-198, a-316, a-367, a-372
\if@afternarr, a-198, a-315, a-368, a-371
\if@codeskipput, a-75, a-79, a-188, a-199, a-317, a-359, a-370, a-762, a-766, a-828
\if@codespacesblank, a-23, a-94
\if@countalllines, a-9, a-166, a-265, a-757
\if@debug, b-12, b-41, b-45, b-46
\if@dsdir, a-84, a-84, a-820
\if@filesw, a-694, a-1262, a-1268, a-1283, a-1375, a-1384, a-1401, d-381
\if@fshda, a-1468, a-1479, a-1492
\if@gmccnochanges, b-16, b-59
\if@indexallmacros, a-15, a-991
\if@linesnotnum, a-5, a-351, a-542
\if@ltxDocInclude, a-1273, a-1276, a-1279, a-1390, a-1393, a-1396, a-1406
\if@mainmatter, d-238
\if@marginparsused, a-17, a-742
\if@newline, a-83, a-168, a-204, a-264, a-275, a-282
\if@noindex, a-11, a-36
\if@noskipsec, f-145
\if@openright, d-240
\if@pageinclindex, a-522, a-539
\if@pageindex, a-13, a-353, a-521, a-544, a-699, a-1005, a-1008, a-1009, a-1011
\if@RecentChange, a-1098, a-1132
\if@specialpage, d-256
\if@twoside, d-279
\if@uresetlinecount, a-7, a-338
\ifdtraceoff, a-1061, b-46
\ifdtraceon, a-1060, b-45
\ifgmcc@mwcls, b-5, b-23, b-26
\ifHeadingNumbered, d-247, d-263
\ifodd, d-210
\ifprevhmode, a-246, a-288, a-329
\im@firstpar, a-493, a-494, a-495, a-600, a-601, a-604
\incl@DocInput, a-1278, a-1395, a-1413, a-1416, a-1418
\incl@filedivtitle, a-1475, a-1487
\incl@titletotoc, a-1468, a-1476
\InclMaketitle, a-1274, a-1391, a-1465
\index@macro, a-495, a-505, a-604, a-637, a-676
\index@prologue, a-1001, a-1003, a-1027, a-1336
indexallmacros, *9*, a-16
IndexColumns, *16*, a-1025
\indexcontrols, a-477, a-481
\indexdiv, a-1002, a-1003, a-1181
\indexentry, a-696
\IndexInput, *8*, a-1495
\IndexLinksBlack, 17, a-1016, a-1028, a-1031
\IndexMin, *16*, a-1024, a-1024, a-1027
\IndexParms, *17*, a-1029, a-1033, a-1185
\IndexPrefix, *16*, a-527, a-541
\IndexPrologue, *16*, a-1001, *83*
\IndexRefCs, a-523, a-525, a-529
\interlinepenalty, f-166

\jobnamewoe, *c-26*, *c-31*, d-403

\kernel@ifnextchar, a-1567
\kind@fentry, a-512, a-514, a-518, a-523, a-525

\l@nohyphenation, d-118, d-119, d-129
\larger, d-84, *95*
\largerr, d-88, *95*

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty

\last@defmark, a-535, a-576, a-1101, a-1104, a-1105, a-1129
\LaTeXpar, 18, d-337
\ldots, d-397
\leftmargin, f-157, f-158
\levelchar, *17*, a-442, a-482, a-1096, a-1116, a-1126
\LineNumFont, *16*, a-173, a-348, a-350, a-1632, *81*
\lineskip, a-1442
linesnotnum, *8*, a-6
\LoadClass, b-27, b-30
\ltxLookSetup, *8*, a-1407, a-1412
\ltxPageLayout, *8*, a-1232, a-1409
\ltxtextellipsis, d-391, d-396

\macro, a-826, a-833, a-1680, a-1681
macro, *12*, a-826
macro*, a-833
\macro@iname, a-457, a-462, a-465, a-471, a-495, a-604, a-606, a-612, a-637, a-676
\macro@pname, a-458, a-466, a-472, a-492, a-495, a-496, a-497, a-597, a-598, a-599, a-604, a-624, a-625, a-626, a-629, a-637
\macrocode, a-769
macrocode, *7*, *19*, a-765
macrocode*, a-761
\MacrocodeTopsep, a-1628
\MacroFont, a-1626, *80*
\MacroIndent, a-1627, 80
\MacroTopsep, a-58, a-61, a-74, a-827, a-832, *80*
\main, a-1668
\MakeGlossaryControls, *14*, a-1087, a-1094
\MakePercentComment, a-1676
\MakePercentIgnore, a-1085, a-1675
\MakePrivateLetters, *11*, *16*, a-127, a-450, a-564, a-581, a-591, a-616, a-639, a-646, a-683, a-686, a-725, a-738, a-782, a-830, a-867, a-994, a-1086, a-1609, a-1614
\MakePrivateOthers, a-451, a-565, a-582, a-592, a-617, a-640, a-647, a-684, a-687, a-726, a-739, a-830, a-1612, a-1615
\MakeShortVerb, *10*, f-55, f-230, *113*
\MakeShortVerb*, f-55, f-206, f-229
\maketitle, *7*, a-981, a-1274, a-1391, a-1421, *c-8*, *c-11*, *72*
\marg, d-155, d-165
\marginparpush, a-744
\marginpartt, *11*, a-754, a-755, b-49
\marginparwidth, a-745, a-1234
\mark@envir, a-208, a-280, a-656
\math@arg, d-163, d-164
\mathellipsis, d-396
\mathindent, b-32
\maybe@marginpar, a-496, a-500
\maybe@quote, a-457, a-465, a-471, a-477, a-478, a-612
\mcdiagOff, a-1594
\mcdiagOn, a-1591
\medmuskip, d-138
\meta, d-121, d-135, d-155, d-157, d-160, *82*
\meta@font@select, d-125, d-134
\meta@hyphen@restore, d-126, d-131
\mod@math@codes, a-1080, a-1081, a-1082
\Module, a-1068, a-1080
\ModuleVerb, a-1075, a-1081
\month, a-1225, a-1227
\mskip, d-138
\multiply, a-1137, a-1139
\mw@HeadingBreakAfter, d-258, d-275, d-290, d-294
\mw@HeadingBreakBefore, d-255
\mw@HeadingLevel, d-245, d-248
\mw@HeadingRunIn, d-270
\mw@HeadingType, d-254
\mw@HeadingWholeWidth, d-273
\mw@normalheading, d-277, d-286, d-289, d-293
\mw@runinheading, d-271
\mw@sectionxx, d-244
mwart, b-7, *87*
mwbk, b-9, *87*
mwrep, b-8, *c-1*, *87*

\nameshow, d-356
NeuroOncer, *a-857*
\newcount, a-1025, a-1131, a-1160, a-1191
\newcounter, a-341, a-343, a-344, a-1193, a-1585, a-1670, d-207, d-229, e-7
\newdimen, a-1024, a-1159
\newgif, a-83, a-84, a-246, a-359, a-367, a-368, d-4
\newlanguage, d-119
\newlength, a-51, a-52, a-54, a-223, a-224, f-170, f-172
\newskip, a-57, a-58, a-418
\newtoks, a-47
\newwrite, d-381
\nfss@text, d-123
\nl@percent, a-1523, a-1525, a-1527
\nlpercent, a-1521
nochanges, b-17, *86*
\noeffect@info, a-1634, a-1640, a-1641, a-1642, a-1643, a-1671, a-1672, a-1673, a-1674
\NoEOF, *15*, a-1689
noindex, *9*, a-12, b-14, *86*
nomarginpar, *9*, a-22
\NonUniformSkips, *15*, a-72
\nostanza, *15*, a-81
\nummacro, a-864, d-219

\oarg, d-156
\obeyspaces, a-151, a-157, a-797, f-40, f-43, f-44, f-188
\oddsidemargin, a-1235
\old@begin, d-50, d-51
\old@MakeShortVerb, a-1679, f-207, f-227
\olddocIncludes, *8*, *19*, a-1415
\OldDocInput, *7*, *19*, a-1416, a-1677
\OldMakeShortVerb, f-226
\oldmc, a-769, a-773
oldmc, *20*, a-769

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty

oldmc*, a-769
\oldmc@def, a-801, a-806
\oldmc@end, a-802, a-807
\OnAtLine, d-399
\OnlyDescription, *17*, a-1657
outeroff, b-11, *c-1*, *87*

\PackageError, a-1254, a-1325
\PackageInfo, a-1630, a-1634, f-98
\PackageWarning, d-80, d-82
\PackageWarningNoLine, a-1089
\pagebreak, d-278, d-290, d-294
\pagegoal, d-388
\PageIndex, a-1648, a-1649
pageindex, *9*, a-14
\pagestyle, b-57
\pagetotal, d-389
\par, a-76, a-80, a-109, a-123, a-187, a-233, a-299, a-309, a-312, a-325, a-417, a-762, a-764, a-766, a-768, a-829, a-832, a-929, a-1041, a-1044, a-1421, a-1440, a-1445, a-1449, a-1517, a-1519
\parg, d-159
\parsep, f-144
\partopsep, a-68, f-148
\PassOptionsToPackage, b-15
\pdfeTeX, 18, a-1547
\pdfTeX, 18, a-1548
\pk, *17*, a-1242, a-1371, *c-3*, *c-4*, *c-20*, d-147
\PlainTeX, 18, a-1543
\possfil, d-152
\predisplaypenalty, f-116, f-118
\prevhmodegfalse, a-192, a-218, a-246, a-249, a-310, a-322
\prevhmodegtrue, a-246, a-248
\PrintChanges, *13*, a-1186, a-1190, a-1302, *c-28*
\PrintDescribeEnv, *81*
\PrintDescribeMacro, *81*
\PrintEnvName, *81*
\PrintFilesAuthors, *7*, a-1493
\PrintIndex, a-1048, a-1302, *c-32*
\printindex, a-1049, a-1302
\printlinenumber, a-206, a-279, a-347, a-351
\PrintMacroName, *81*
\printspaces, d-140, d-144
\ProvideFileInfo, *19*, a-1563, a-1572
\ProvidesClass, b-2
\ProvideSelfInfo, *19*, a-1572
\ps@plain, a-1434
\ps@titlepage, a-1434

\quad, b-55, b-56
\QueerCharOne, a-403, a-404, a-405
\QueerCharTwo, a-397, a-398, a-399
\QueerEOL, *6*, a-141, a-411, a-763, a-767, a-1049, a-1188, a-1419, a-1689
quotation, *19*, a-1519
\quote@char, a-456, a-464, a-470, a-475, a-611
\quote@charbychar, a-607, a-608, a-613
\quote@mname, a-598, a-605, a-632, a-672
\quotechar, *17*, a-440, a-477, a-482, a-528, a-540, a-634, a-674, a-1095, a-1121
\quoted@eschar, a-528, a-540, a-634, a-635, a-674, a-675

\raggedbottom, b-50
\RecordChanges, *13*, a-1091, a-1157, a-1158, a-1302, b-59, b-60
\reflectbox, a-1552
\relaxen, d-352, d-352
\relsize, d-56, d-57, d-84, d-85, d-86, d-87, d-88, d-89, *95*
\rem@special, f-84, f-102, f-107
\renewcommand, a-1093
\renewcommand*, a-362, a-364, b-52
\RequirePackage, a-26, a-27, a-31, a-34, a-35, a-39, a-1023, b-34, b-35, b-37, b-42, b-47, b-64, d-378
\resetlinecountwith, a-340
\Restore@Macro, d-187, d-188, d-192, d-195
\Restore@Macros, d-190, d-191
\RestoreMacro, a-1000, a-1498, d-186, d-243
\RestoreMacros, a-707, d-190
\RestoringDo, a-1280, a-1397, d-199
\reversemarginpar, a-743
\rightline, b-65
\rs@size@warning, d-73, d-78, d-80
\rs@unknown@warning, d-70, d-82

\scan@macro, a-257, a-453
\scshape, a-1542
\SelfInclude, *7*, a-1362, *c-14*
\SetFileDiv, *18*, a-1318, a-1319, a-1321, a-1327, a-1359, a-1408
\settexcodehangi, a-48, a-50, a-211, a-215, a-330
\SetTOCIndents, b-55, b-56
\sfname, d-144, d-145
\sgtleftxii, a-1051
\showboxbreadth, d-355
\showboxdepth, d-355
\showlists, d-355
\SkipFilesAuthors, *7*, a-1494
\skipgmlonely, *18*, a-1500, *c-18*
\SliTeX, 18, a-1541
\smaller, d-85, *95*
\smallerr, a-1478, d-89, *95*
\smallskipamount, a-65, a-66, d-370, d-371
\smartunder, *b-62*, d-101
\SortIndex, a-1665
\special@index, a-530, a-700, a-702, a-759
\SpecialEnvIndex, a-1664
\SpecialEscapechar, a-1629
\SpecialIndex, a-1662
\SpecialMainEnvIndex, a-1661
\SpecialMainIndex, a-1660
\SpecialUsageIndex, a-1663
\square, b-65
StandardModuleDepth, a-1670
\stanza, *15*, *19*, a-77, a-1518, *c-18*, *c-20*
\stanzaskip, *14*, a-54, a-55, a-56, a-60, a-61, a-62, a-63, a-64, a-67, a-78, a-189, f-164, f-170, f-171
\step@checksum, a-454, a-1194
\stepnummacro, a-855, d-220
\StopEventually, *17*, a-1654, a-1657

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty

\Store@Macro, d-168, d-169, d-173
\Store@Macros, d-171, d-172
\stored@code@delim, a-778
\Stored@Macro, d-194, d-195
\StoredMacro, d-194
\StoreMacro, a-997, a-1496, d-167, d-243
\StoreMacros, a-706, d-171
\StoringAndRelaxingDo, a-1272, a-1388, d-196
\StraightEOL, *6*, a-141, a-406, a-1049, a-1086, a-1188, a-1499, a-1506, a-1516, a-1678
\subdivision, *18*, a-1357, a-1581, a-1583, a-1584
\subitem, a-1042
\subs, d-95, d-103
\subsubitem, a-1043

\tableofcontents, a-139, a-140, a-1301, *c-10*
\TeXbook, 18, a-1545
\Text@CommonIndex, a-640, a-641
\Text@CommonIndexStar, a-640, a-643
\text@indexenvir, a-622, a-623, a-644, a-736, a-1625
\text@indexmacro, a-596, a-620, a-642, a-731, a-1620
\Text@Marginize, a-502, a-659, a-729, a-734, a-741, a-753, a-853, a-1618, a-1623
\Text@MarginizeNext, a-848, a-851, a-852
\Text@UsgEnvir, a-726, a-732
\Text@UsgIndex, a-617, a-618
\Text@UsgIndexStar, a-617, a-621
\Text@UsgMacro, a-726, a-727
\TextCommonIndex, *12*, a-638
\TextCommonIndex*, a-638
\TextIndent, *15*, a-51, a-333, a-375
\textlarger, d-86
\TextMarginize, *11*, a-737
\TextMarginize*, a-737
\textsmaller, d-87
\TextUsage, *11*, a-724
\TextUsage*, a-724
\TextUsgIndex, *12*, a-615, a-1663
\TextUsgIndex*, a-615
\textvisiblespace, d-139
\textwidth, a-1233, a-1337
\thanks, a-1439, a-1453, a-1467, a-1471, a-1574
\theCodelineNo, a-1631, *81*
\thecodelinenum, a-173, a-348, a-1633
\thefilediv, a-1288, a-1332, a-1333, a-1334, a-1345, a-1348
\theglossary, a-1161, a-1303
theglossary, a-1161
theindex, a-1026
\thesection, b-52
\thfileinfo, *19*, a-1574
\title, a-1450, *c-3*
\titlesetup, a-1438, a-1461, b-58
\TODO, d-375
\tolerance, *a-93*
\tOnLine, d-398
\traceoff, b-46
\traceon, b-45
\trimmed@everypar, a-388, a-390
\ttverbatim, *a-124*, a-780, f-48, f-169, f-178
\twelveand, d-114
\twelvebackslash, d-107, d-108
\twelveclub, a-441, a-1096, f-47
\twelvelbrace, f-20, f-31
\twelvepercent, a-822, a-824, a-1224, a-1226, a-1522, a-1529, a-1532, d-111, f-12, f-13
\twelvespace, a-229, d-117, d-358, d-359, f-38
\twelveunder, d-98
\twocoltoc, *c-2*, d-377, d-385

\un@defentryze, a-519, a-533
\un@usgentryze, a-515, a-536
\undeksmallskip, d-371
\UndoDefaultIndexExclusions, 13, a-996
\ungag@index, a-707, a-1651
\UniformSkips, *15*, a-59, a-70, a-71, a-72
uresetlinecount, *9*, a-8
\usage, a-1669
\UsgEntry, *16*, a-558, a-1669

\varepsilon, a-1546
\verb, *17*, a-436, f-58, f-77, f-81, f-83, f-176, f-224
\verb*, f-57, f-176
\verb@balance@group, f-184, f-185, f-196, f-198
\verb@egroup, a-435, f-185, f-196, f-199
\verb@eol@error, f-186
\verb@eolOK, f-189, f-193
\verbatim, f-116
verbatim, f-116
verbatim*, f-118
\verbatim@edef, f-135, f-139
\verbatim@end, f-136, f-140
\verbatim@nolig@list, f-49, f-200
\verbatimchar, *17*, a-492, a-505, a-599, a-1120, a-1122, a-1666, *82*
\verbatimhangindent, a-49, f-167, f-172, f-173
\verbeolOK, *9*, f-193, *112*
\VerbHyphen, a-45, f-14, *112*
\verbhyphen, f-13, f-16, f-19, f-28
\VerbMacrocodes, *20*, a-1682
\vs, d-139, d-140, d-143

\Web, 18, a-1544
\widowpenalty, *a-92*
withmarginpar, *9*, a-21

\xdef@filekey, a-1276, a-1279, a-1287, a-1393, a-1396
\XeTeX, 18, a-1549
\xiistring, a-466, a-597, a-624, a-719, a-722, a-728, a-733, a-754, d-357

\year, a-1225, a-1227

File Key: a=gmdoc.sty, b=gmdocc.cls, c=gmdocDoc.tex, d=gmutils.sty, e=gmiflink.sty, f=gmverb.sty