Grzegorz Murzynowski

The gmdoc Bundle *

Many thanks to my TeX Guru Marcin Woliński for his TeXnical support.
For the documentation please refer to the file(s)
gmdoc.{gmd,pdf}.

```
47 ⟨*master⟩
```

(A handful of meta-settings skipped)

```
98 ⟨/ master⟩
99 ⟨*ins⟩
```

\supposedJobname
```
100 \def\supposedJobname{%
101           gmdoc%
102 }
```

```
104 \let\xA\expandafter
105 \let\nX\noexpand
106 \long\def\firstofone#1{#1}
```

```
108 \unless\ifnum\strcmp␣{\jobname}␣{\supposedJobname}␣=0
```

If we want to generate files from this file, we should call

xelatex␣--jobname=⟨*sth. else*⟩

Then the \strcmp primitive expands to some nonzero value and the conditional turns true.

```
115 \NeedsTeXFormat{LaTeX2e}[1996/12/01]
```

\gmBundleName
```
117 \def\gmBundleName{%
118     gmdoc%
119 }
```

\currentBundle
```
121 \def\currentBundle{%
122     docbundle%
123 }
```

```
125 \edef\batchfile{\gmBundleName␣.gmd}
```

```
128 \input␣docstrip.tex
```

```
\NOO 130 \def\NOO{\FromDir\gmBundleFile␣.gmd}
```

Note it's `\def` so the BundleName expands to its current value.

```
133 \let\skiplines\relax
134 \let\endskiplines\relax
135 \askforoverwritefalse
\MetaPrefixS 137 \def\MetaPrefixS{\MetaPrefix\space}
\perCentS 138 \def\perCentS{\perCent\space}
140 \begingroup
141 \endlinechar=\newlinechar
142 \catcode\newlinechar=12\relax%
144 \catcode`\^=12\relax%
145 \catcode`\ =0\relax␣% Tifinagh Letter Yay
146 \catcode`\\=12 relax␣%
147 catcode` ⁄=12 relax␣%
148 firstofone{ endgroup␣%
150     def preamBeginningLeaf{%
152       RCSInfo
153       MetaPrefixS␣This␣is␣file␣" outFileName"␣generated␣with␣
              the␣DocStrip␣utility.
154       MetaPrefixS
155       ReferenceLines␣%
156       MetaPrefix␣%
157     }% of \preamBeginningLeaf
161     def copyRightLeaf{Copyright␣©␣}%
164     def licenseNoteLeaf{%
165       This␣program␣is␣subject␣to␣the␣LaTeX␣Project␣Public␣
              License.
166       MetaPrefixS␣See␣␣
              http://www.ctan.org/tex-archive/help/Catalogue/licenses.lppl.h
167       MetaPrefixS␣for␣the␣details␣of␣that␣license.
168       MetaPrefix
169       MetaPrefixS␣LPPL␣status:␣"author-maintained".
170       MetaPrefix␣%
171     }% of \licenseNoteLeaf
173     def preamEndingLeaf{%
174       gmBundleFile.{gmd,pdf} gobble{␣or␣\file{%
              Natror-OperaOmnia.{gmd,pdf}}}.
175       MetaPrefixS␣␣%
176     }% of \preamEndingLeaf
178     def providesStatement{%
180       \NeedsTeXFormat{LaTeX2e}
181       \Provides gmFileKind{ gmOutName}
182       space space space space[ gmFileDate space␣
              gmFileVersion space␣ gmFileInfo space␣(GM)]
184     }%
186 }% of \firstofone of changed catcodes.
\beforeDot 188 \def\beforeDot#1.#2\empty{#1}
```

* This file has version number v0.993 dated 2010/09/25.

```
\firstoftwo
\secondoftwo
\def\firstoftwo#1#2{#1}
\def\secondoftwo#1#2{#2}
```

To gobble the default heading lines put by DocStrip:

```
\Name\def{ds@heading}#1{}
```

```
\csnameIf
\def\csnameIf#1{%
  \ifcsname#1\endcsname
    \csname#1\xA\endcsname
  \fi
}
```

```
\writeto
\FromDir
\writefrom
\FromDir
\WritePreamble
\def\writeto#1{\edef\destdir{#1}}
\def\FromDir{}
\def\writefrom#1{\def\FromDir{#1/}}
\def\WritePreamble#1{%
  \xA\ifx\csname␣pre@\@stripstring#1\endcsname\empty
  \else
    \edef\outFileName{\@stripstring#1}%
    \edef\gmOutName{%
      \xA\beforeDot\outFileName\empty
    }% of \gmOutName
    \edef\gmOutTitle{%
      \xA\xA\xA\detokenize\xA\xA\xA{%
        \csname␣\gmOutName␣Title\endcsname}%
    }% of \gmOutTitle
    \edef\gmOutYears{%
      \csnameIf␣{\gmOutName␣Years}%
    }%
    \edef\gmOutThanks{%
      \ifcsname␣\gmOutName␣Thanks\endcsname
        \xA\xA\xA\detokenize\xA\xA\xA{%
          \csname␣\gmOutName␣Thanks\endcsname
        }%
      \fi
    }%
    \edefInfo{Date}% \gmFileDate
    \edefInfo{Version}% \gmFileVersion
    \edefInfo{Info}% \gmFileInfo
    \StreamPut#1{\csname␣pre@\@stripstring#1\endcsname}%
  \fi}
```

First we look for the info at the leaf-level, then at standalone level, then at the bundle level. If we don't find it, it'll be empty.

```
\edefInfo
\def\edefInfo#1{%
  \Name\edef{gmFile#1}{%
    \ifcsname␣\gmOutName␣Leaf#1\endcsname␣% e.g. gmbaseLeafVersion
      \xA\xA\xA\detokenize\xA\xA\xA{%
        \csname␣\gmOutName␣Leaf#1\endcsname
      }%
    \else
      \ifcsname␣\gmOutName␣#1\endcsname␣% e.g. gmbaseVersion
```

```
        \xA\xA\xA\detokenize\xA\xA\xA{%
          \csname␣\gmOutName␣#1\endcsname
        }%
      \else
        \ifcsname␣\gmBundleFile␣#1\endcsname␣% e.g. gmutilsVersion
          \xA\xA\xA\detokenize\xA\xA\xA{%
            \csname␣\gmBundleFile␣#1\endcsname
          }%
        \fi
      \fi
    \fi
  }% of edefined macro
}% of \edefInfo

\let\gmOutName\relax
\let\gmOutTitle\relax
\let\gmOutYears\relax
\let\gmFileDate\relax
\let\gmFileVersion\relax
\let\gmFileInfo\relax
\let\gmOutThanks\relax
\let\gmBundleFile\relax
\let\gmFileKind\relax

\declarepreamble\gmdLeaf
\preamBeginningLeaf

\copyRightLeaf␣\gmOutYears
by␣Grzegorz␣'Natror'␣Murzynowski
natror␣(at)␣gmail␣(dot)␣com

\licenseNoteLeaf

For␣the␣documentation␣please␣refer␣to␣the␣file(s)
\preamEndingLeaf
\providesStatement
\endpreamble

\keepsilent
```

We declare all the preambles later and use the `\empty` Docstrip preamble.

```
\errorcontextlines=1000

\@makeother\^^A
\@makeother\^^B
\@makeother\^^C
\@makeother\^^V
```

`\gmfile`

```
\def\gmfile
#1% file name
#2% DocStrip directive(s)
#3% file extension
{%
  \file{gm#1.#3}{\from{\gmBundleFile/\NOO}{#2}}%
}
```

`\pack`

```
\def\pack#1{\gmfile{#1}{#1}{sty}}
```

```
\begingroup\catcode`\␣=9
\catcode`\^^I=9\relax
\catcode`\^^M=9\relax
\firstofone{\endgroup
```

\gmBundleFile
```
  \def\gmBundleFile{gmdoc}
```
```
  \generate{
```
```
    \usepreamble\gmdLeaf
```
\gmFileKind
```
    \def\gmFileKind{␣␣␣␣Package␣␣␣␣}
```
```
    \writeto{␣␣␣␣gmdoc␣␣␣␣}
    \pack{␣␣␣␣doc␣␣␣␣}
```
\gmFileKind
```
    \def\gmFileKind{␣␣␣␣Class␣␣␣␣}
    \gmfile{␣␣␣␣docc␣␣␣␣}{␣␣␣␣docc␣␣␣␣}{␣␣␣␣cls␣␣␣␣}
```
```
    \writefrom{␣␣␣␣gmdoc␣␣␣␣}
    \writeto{␣␣␣␣␣␣␣␣gmdoc/Source2e␣␣␣␣}
    \usepreamble\gmdStandalone
```
```
    \file{␣␣␣␣␣␣␣␣Source2e_gmdoc.tex␣␣␣␣}{\from{\NOO}{␣␣␣␣
        LaTeXsource␣␣␣␣}}
```
```
    \writeto{␣␣␣␣gmdoc/doc␣␣␣␣}
    \file{␣␣␣␣␣␣␣␣doc_gmdoc.tex␣␣␣␣}{\from{\NOO}{␣␣␣␣
        docbygmdoc␣␣␣␣}}
```
```
    \writeto{␣␣␣␣gmdoc/docstrip␣␣␣␣}
    \file{␣␣␣␣␣␣␣␣docstrip_gmdoc.tex␣␣␣␣}{\from{\NOO}{␣␣␣␣
        docstrip␣␣␣␣}}
```
```
  }
}% of changed catcodes' \firstofone
```
```
\Msg{%
    **********************************************************************}
\Msg{␣}
\Msg{␣␣To␣finish␣the␣installation␣you␣have␣to␣move}
\Msg{␣␣␣the␣generated␣files␣into␣a␣directory␣searched␣by␣TeX.}
\Msg{␣}
\Msg{␣␣To␣type-set␣the␣documentation,␣run␣the␣file␣'\NOO'}
\Msg{␣␣twice␣through␣LaTeX␣and␣maybe␣MakeIndex␣it.␣␣}
\Msg{␣}
\Msg{%
    **********************************************************************}
```
```
\csname␣fi\endcsname␣% probably for the directive's clause
\csname␣endinput\expandafter\endcsname␣%
\fi␣% of unless job name other than name of this file, which indicates the DocStrip
    pass.
```
```
⟨/ ins⟩
```

Contents

409 ⟨*doc⟩

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 a tiny adjustment in some special cases).

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

The package also works with XETEX (switches automatically).

Installation

Unpack the \jobname-tds.zip archive (this is an archive that conforms the TDS standard, see CTAN/tds/tds.pdf) in some texmf directory or just put the gmutils.sty 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.

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

Contents of the gmdoc.zip archive

The distribution of the gmutils package consists of the following three files and a TDS-compliant archive.

gmdoc.gmd
README
gmdoc.pdf
gmdoc.tds.zip

Compiling of the documentation

The last of the above files (the .pdf, i.e., *this file*) is a documentation compiled from the .gmd file by running LATEX on the gmdoc.gmd file twice (`xelatex␣gmdoc.gmd` in the directory you wish the documentation to be in), then MakeIndex on the \jobname.idx file, and then LATEX on \jobname.\gmdExt once more.

MakeIndex shell commands:

```
makeindex -r gmdoc
makeindex -r -s gmglo.ist -o gmdoc.gls gmdoc.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), gmverb.sty, the gmutils bundle, 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.

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.

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 Marcelli* ;-):

source2e_gmdoc.tex
docstrip_gmdoc.tex
doc_gmdoc.tex

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

These drivers typeset the respective files from the

…/texmf-dist/source/latex/base

directory of the TEXLive2007 distribution (they only read that directory).

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 CSes in the code maybe in the future.)

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

The doc package was and still is a great inspiration for me and I would like this humble package to be considered as a sort of homage to it[1]. 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 TEX book*'s meaning, i.e., a control sequence whose meaning is `\[e|g|x]defined`. By a **macro's parameter** I mean each of `#`⟨*digit*⟩s in its definition. When I write about a **macro's argument**, I mean the value (list of tokens) substituting the corresponding parameter of this macro. (These understandings are according to *The TEX book*, 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**.

1 As Grieg's Piano Concerto is a homage to the Schumann's.

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 'they' sees 'themself' altered in the same sentence :-) .

\heshe

Preparing of the source file

When (LA)TEX with gmdoc.sty package loaded typesets the comment lines, the code delimiter is omitted. If the comment continues a codeline, the code delimiter is printed. It's done so because ending a TEX code line with a `%` is just a concatenation with the next line sometimes. Comments longer than one line are typeset continuously with the code delimiters omitted.

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

\^^M
^^B

But there is also a gmdoc version of `\footnote` provided that sets the catcodes so that you don't bother about `^^B` in the argument, `\qfootnote`that takes the same argument(s) as the standard `\footnote` and for emphasis there is `\qemph{`⟨*text to emphasise*⟩`}`. Both of them work also in the 'straight' EOLs' scope so you don't bother. The `\arg` gmutils' command also works without `^^B`.

\qfootnote
\qemph
\arg

Moreover, if they wants to add a meta-comment i.e., a text that doesn't appear in the code layer nor in the narrative, they may use the `^^A` sequence that'll be read by TEX as ⟨*char1*⟩, which in gmdoc is active and defined to gobble the stuff between itself and the line end.

^^A

Note that `^^A` behaves much like comment char although it's active in fact: it re`\catcode`s the special characters including `\`, `{` and `}` so you don't have to worry about unbalanced braces or `\if`s in its scope. But `^^B` doesn't re`\catcode` anything (which would be useless in an argument) so any text between `^^B` and line end has to be balanced.

However, it may be a bit confusing for someone acquainted with the doc conventions. If you don't fancy the `^^B` special sequence, instead you may restore the standard meaning of the line end with the `\StraightEOL` declaration which OCSR. As almost all the control sequences, it may be used also as an environment, i.e., `\begin{StraightEOL}` … `\end{StraightEOL}`. However, if for any reason you don't want to make an environment (a group), there's a `\StraightEOL`'s counterpart, the `\QueerEOL` declaration that restores again the queer[2] 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).

\StraightEOL
\QueerEOL

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 they should prepare a main document file, a **driver** henceforth, to set all the required formattings such as `\documentclass`, paper size etc., and load this pack-

2 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.

age with a standard command i.e., `\usepackage{gmdoc}`, just as doc's documentation says:

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

```
\documentclass[⟨options⟩]{⟨document class⟩}
\usepackage[⟨options, probably none⟩]{gmdoc}
   ⟨preamble⟩
\begin{document}
   ⟨special input commands⟩
\end{document}
```

"

The main input commands

`\DocInput` To typeset a source file you may use the `\DocInput` macro that takes the (path and) name of the file *with the extension* as the only argument, e.g., `\DocInput{mybrilliantpackage.sty}`[3].

(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[*]}
```

as in doc. (With `\DocInput` you are not obliged to precede `\end{macrocode[*]}` with The Four Spaces.)

`\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`. If you wish to undeclare printing the authors names, there is `\SkipFilesAuthors` declaration.

3 I use the 'broken bar' character as a hyphen in verbatim texts and hyperlinks. If you dont't like it, see `\verbDiscretionaryHyphen` in gmverb.

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 file-marker also precedes the index entries, if the (default) codeline index option is in force.

\includeonly As with \include, you may declare \includeonly{⟨*filenames separated with commas*⟩} for the draft versions.

If you want to put the driver into the same .sty or .cls file (see chapter 640 to see how), you may write \DocInput{\jobname.sty}, or \DocInclude{\jobname},

\SelfInclude but there's also a shorthand for the latter \SelfInclude that takes no arguments. By the way, to avoid an infinite recursive input of .aux files in the case of self-inclusion an .auxx file is used instead of (main) .aux.

By the way, to say TEX to (self)include only the current file, most probably you should say \includeonly{\jobname} not \includeonly{myfile} because of the catcodes.

At the default settings, the \(Doc|Self)Included files constitute chapters if \chapter is known and parts otherwise. The \maketitles 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' \maketitles result

\ltxLookSetup 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)).

On the other hand, if you like the look offered by me but you have the files prepared

\olddocIncludes 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.

It seems possible that you wish to document the 'old-doc' files first and the 'new-doc'

\gmdocIncludes ones after, so the above declaration has its counterpart, \gmdocIncludes, that may be used anywhere, too. Before the respective \DocInclude(s), of course.

Both these declarations OCSR.

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

Talking of analogies with ltxdoc, if you like only the page layout provided by that

\ltxPageLayout 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.

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

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

If you need to add something at the beginning of input of only one file, put before

\AtBegInputOnce the respective input command an \AtBegInputOnce{⟨*the stuff to be added*⟩} declaration. It's also global which means that the groups do not limit its scope but it adds its argument only at the first input succeeding it (the argument gets wrapped in a macro that's \relaxed at the first use). \AtBegInputOnces add up, too.

\IndexInput

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).

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

Package options

As many good packages, this also provides some options:

linesnotnum

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

uresetlinecount

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

countalllines

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, which may be convenient for reference purposes, they is provided the `countalllines` option. This option switches things to use the `\inputlineno` primitive for codeline numbers so you get the numbers of the source file instead of number only of the codelines. Note however, that there are no hypertargets made to the narration lines and the value of `\ref` is the number of the most recent codeline.

countalllines*

Moreover, if they wants to get the narration lines' number printed, there is the starred version of that option, `countalllines*`. I imagine someone may use it for debug. This option is not finished in details, it causes errors with `\addvspace` because it puts a hyperlabel at every line. When it is in force, all the index entries are referenced with the line numbers and 441 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.

noindex

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

pageindex

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

The references in the change history are of the same: when index is line number, then the changes history too.

indexallmacros

By default, gmdoc excludes some 300 CSes from being indexed. They are the most common CSes, LATEX internal macros and TEX primitives. To learn what CSes are excluded actually, see lines 6211–6337.

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. 18 to feed this ambiguity with a couple of declarations.

In doc package there's a default behaviour of putting marked macro's or environment's name to a marginpar. In the standard classes it's alright but not all the classes support marginpars. That is the reason why this package enables marginpar-ing when

withmarginpar
nomarginpar

in standard classes, enables or disables it due to the respective option when with Marcin Woliński's classes and in any case provides the options `withmarginpar` and `nomarginpar`. So, in non-standard classes the default behaviour is to disable marginpars. If the marginpars are enabled in gmdoc, it will put marked control sequences and environments into marginpars (see `\TextUsage` etc.). These options do not affect common using marginpars, which depends on the document class.

codespacesblank
\CodeSpacesBlank

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

codespacesgrey
\CodeSpacesGrey

Another space formatting option is `codespacesgrey` suggested by Will Robertson. It makes the spaces of code visible only not black but grey. The name of their colour is `visspacesgrey` and by default it's defined as `{gray}{.5}`, you can change it with xcolor's `\definecolor`. There is also an OCSR declaration `\CodeSpacesGrey`.

\VisSpacesGrey

If for any reason you wish the code spaces blank in general and visible and grey in `verbatim*`s, use the declaration `\VisSpacesGrey` of the gmverb package. If you like little tricks, you can also specify `codespacesgrey,␣codespacesblank` in gmdoc options (in this order).

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`.

\verbatimspecials

This package provides the `\verbatimspecials` declaration that is used in gmdocc.cls as

```
\verbatimspecials⁄«»[¿]
```

to set ⁄ (fractional slash) to the escape char, « and » to group begin and end respectively and ¿ to math shift in verbatims (also the short ones). Note however that this declaration *has no effect on the code layer*.

\verbeolOK

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

\MakeShortVerb

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

```
\MakeShortVerb*\|
```

(as you see, the declaration has the starred version, which is for visible spaces, and non-starred for blank spaces) to get `\mylittlemacro` you may type `|\mylittlemacro|` instead of `\verb+\mylittlemacro+`. Because the char used in the last example is my favourite and is used this way by DEK in *The TeX book*'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. Moreover, it provides a powerful tool for defining commands with weird optional and Knuthian arguments, `\DeclareCommand`, inspired by ancient (pre-expl3) xparse's `\DeclareDocumentCommand`.

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 their favourite options *before* loading gmdoc. If they does not, gmdoc shall load it with *my* favourite options.

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

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

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

Automatic marking of definitions

gmdoc implements automatic detection of a couple of definitions[4]. By default it detects all occurrences of the following commands in the code:

1. `\def`, `\newcount`, `\newdimen`, `\newskip`, `\newif`, `\newtoks`, `\newbox`, `\newread`, `\newwrite`, `\newlength`, `\newcommand[*]`, `\renewcommand[*]`, `\providecommand[*]`, `\DeclareRobustCommand[*]`, `\DeclareTextCommand[*]`, `\DeclareTextCommandDefault[*]`, `\DeclareDocumentCommand`, `\DeclareCommand`
2. `\newenvironment[*]`, `\renewenvironment[*]`, `\DeclareOption`,
3. `\newcounter`,

   of the xkeyval package:
4. `\define@key`, `\define@boolkey`, `\define@choicekey`, `\DeclareOptionX`,

   and of the kvoptions package:
5. `\DeclareStringOption`, `\DeclareBoolOption`, `\DeclareComplementaryOption`, `\DeclareVoidOption`.

What does 'detects' mean? It means that the main argument of detected command will be marked as defined at this point, i.e. thrown to a margin note and indexed with a 'definition' entry. Moreover, for the definitions 3–5 an alternate index entries will be created: of the CSes underlying those definitions, e.g. `\newcounter{foo}` in the code will result in indexing `foo` and `\c@foo`.

\DeclareDefining If you want to add detection of a defining command not listed above, use the `\DeclareDefining` declaration. It comes in two flavours: 'sauté' and with star. The 'sauté' version (without star and without an optional argument) declares a defining command of the kind of `\def` and `\newcommand`: its main argument, whether wrapped in braces

4 FMI: the implementation took me 752/3 hrs.

or not, is a CS. The starred version (without the optional argument) declares a defining command of the kind of `\newenvironment` and `\DeclareOption`: whose main mandatory argument is text. Both versions provide an optional argument in which you can set the keys.

type

Probably the most important key is `type`. Its default value is `cs` and that is set in the 'sauté' version. Another possible value is `text` and that is set in the starred version. You can also set three other types (any keyval setting of the type overrides the default and 'starred' setting): `dk`, `dox` or `kvo`.

`dk` stands for `\define@key` and is the type of xkeyval definitions of keys (group 4 commands). When detected, it scans further code for an optional `[`⟨*KVprefix*⟩`]`, mandatory `{`⟨*KVfamily*⟩`}` and mandatory `{`⟨*key name*⟩`}`. The default ⟨*KVprefix*⟩ is `KV`, as in xkeyval.

`dox` stands for `\DeclareOptionX` and launches scanning for an optional `[`⟨*KVprefix*⟩`]`, optional `<`⟨*KVfamily*⟩`>` and mandatory `{`⟨*option name*⟩`}`. Here the default ⟨*KVprefix*⟩ is also `KV` and the default ⟨*KVfamily*⟩ is the input file name. If you want to set another default family (e.g. if the code of `foo.sty` actually is in file `bar.dtx`), use

\DeclareDOXHead

`\DeclareDOXHead{`⟨*KVfamily*⟩`}`. This declaration has an optional first argument that is the default ⟨*KVprefix*⟩ for `\DeclareOptionX` definitions.

`kvo` stands for the kvoptions package by Heiko Oberdiek. This package provides a handful of option defining commands (the group 5 commands). Detection of such a command launches a scan for mandatory `{`⟨*option name*⟩`}` and alternate indexing of a CS`\`⟨*KVOfamily*⟩`@`⟨*option name*⟩. The default ⟨*KVOfamily*⟩ is the input file name.

\DeclareKVOFam

Again, if you want to set something else, you are given the `\DeclareKVOFam{`⟨*KVOfamily*⟩`}` that sets the default family (and prefix: ⟨*KVOfamily*⟩`@`) for all the commands of group 5.

star

Next key recognised by `\DeclareDefining` is `star`. It determines whether the starred version of a defining command should be taken into account[5]. For example, `\newcommand` should be declared with `[star=true]` while `\def` with `[star=false]`. You can also write just `[star]` instead of `[star=true]`. It's the default if the `star` key is omitted.

KVpref
KVfam

There are also `KVpref` and `KVfam` keys if you want to redeclare the xkeyval definitions with another default prefix and family.

For example, if you wish `\@namedef` to be detected (the original LaTeX version), declare

```
\DeclareDefining*[star=false]\@namedef
```

or

```
\DeclareDefining[type=text,star=false]\@namedef
```

(as stated above, `*` is equivalent to `[type=text]`).

\HideDefining

On the other hand, if you want some of the commands listed above *not* to be detected, write `\HideDefining\`⟨*command*⟩ in the commentary. If both ⟨*command*⟩ and ⟨*command**⟩ are detected, then both will be hidden. `\HideDefining` is always

\ResumeDefining

`\global`. Later you can resume detection of ⟨*command*⟩ and ⟨*command**⟩ with `\ResumeDefining`⟨*command*⟩ which is always `\global` too. Moreover, if you wish to suspend automatic detection of the defining ⟨*command*⟩ only once (the next occurrence), there is `\HideDefining*` which suspends detection of the next occurrence of ⟨*command*⟩. So, if you wish to 'hide' `\providecommand*` once, write

```
\HideDefining*\providecommand*
```

\HideAllDefining

If you wish to turn entire detection mechanism off, write `\HideAllDefining` in

5 The `star` key is provided because the default setting of `\MakePrivateLetters` is such that `*` is a letter so e.g. `\newcommand*` is scanned as one CS. However, if the `\makestarlow` declaration is in force (e.g. with the gmdocc) this is not so—`\newcommand*` is scanned as the CS `\newcommand` *and* a star.

\ResumeAllDefining the narration layer. Then you can resume detection with \ResumeAllDefining. Both declarations are \global.

The basic definition command, \def, seems to me a bit ambiguous. Definitely *not always* it defines important macros. But first of all, if you \def a CS excluded from indexing (see section Index ex/inclusions), it will not be marked even if detection of \def is on. But if the \def's argument is not excluded from indexing and you still don't
\UnDef want it to be marked at this point, you can write \HideDefining*\def or \UnDef for short.

If you don't like \def to be detected more times, you may write \HideDefining%
\HideDef \def of course, but there's a shorthand for this: \HideDef which has the starred version
\HideDef \HideDef* equivalent to \UnDef. To resume detection of \def you are provided also
\ResumeDef a shorthand, \ResumeDef (but \ResumeDefining\def also works).
\UnPdef Since I use \pdef most often, I provide also \UnPdef, analogous to \UnDef.

If you define things not with easily detectable commands, you can mark them 'manually', with the \Define declaration described in the next section.

Manual Marking of the Macros and Environments

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

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

This package provides two kinds of special formatting of the index entries: 'usage', with the reference number italic by default, and 'def' (in doc called 'main'), with the reference number roman (upright) and underlined by default. All the reference numbers, also those with no special formatting, are made hyperlinks to the page or the codeline according to the respective indexing option (see p. 12).

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' mark-
\Define ing macro is provided only for the code case. So we have the \Define, \CodeUsage
\CodeUsage and \TextUsage commands.
\TextUsage The arguments to all three are as follows:

- `#1` [*] to indicate whether we mark a single CS or more than one token(s): without star for a single CS, with star for environment names etc., the starred version executes \@sanitize,
- `[#2]` o version to be marginized and printed here,
- `#3` m version to be put to the index, and also (printed here and) marginized if the previous argument is missing.

Note that if you give a single CS to the starred version (e.g. the next \MakePri¦vateLetters is done so to hyphenate it in the text), you have to wrap it in braces because command \@sanitizes the specials including backslash.

You don't have to bother whether @ is a letter while documenting because even if not,
\MakePrivateLetters these commands do make it a letter, or more precisely, they execute \MakePrivate¦Letters 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 commands unstarred.

The \Define and \CodeUsage 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

```
%\Define*{foo}
```

right before the code line

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

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

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

They both are \outer to discourage their use inside macros because they actually re\catcode before taking their arguments.

The \TextUsage (one-argument) command is intended to mark usage of a verbatim occurrence of a TEX object in the commentary. Unlike \CodeUsage or \Define, 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 CSes, 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.

\Describe Then you have the \Describe command which puts its argument in a marginpar and indexes it as a 'usage' entry but doesn't print it in the text. It's \outer.

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

But what if you want just to make a marginpar with macro's or environment's name?

\CodeMarginize \TextMarginize 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.

The marginpars (if enabled) are 'reverse' i.e., at the left margin, and their contents is

\marginpartt flush right and typeset in a font declared with \marginpartt. By default, this declaration is \let to \tt but it may be advisable to choose a condensed font if there is any. Such a choice is made by gmdocc.cls if the Latin Modern fonts are available: in this case gmdocc.cls uses Latin Modern Typewriter Light Condensed.

If you need to put something in a marginpar without making it typewriter font,

\gmdmarginpar there's the \gmdmarginpar macro (that takes one and mandatory argument) that only flushes its contents right.

On the other hand, if you don't want to put a CS(or another verbatim text) in

\DefIndex a marginpar but only to index it, then there are \DefIndex and \CodeUsgIndex to

\CodeUsgIndex 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.

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 indexed automatically anyway). It's `\outer`.

\CodeCommonIndex

\TextUsgIndex
\TextCommonIndex

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

macro
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 `environment`.)

Index ex/inclusions

\DoNotIndex

It's understandable[6] 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 `\DoNotIndexes` as you wish (about half as many as many CSes may be declared, because for each CS excluded from indexing a special CS is declared that stores the ban sentence). Excluding the same CS more than once makes no problem.

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

\DoIndex

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

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

\DefaultIndexExclusions
\UndoDefaultIndexExclusions

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.

6 After reading doc's documentation ;-) .

The DocStrip directives

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

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

So far verbatim mode directive is only half-handled. That is, a line beginning with `%<<`⟨*END-TAG*⟩ will be typeset as a DocStrip directive, but the closing line `%`⟨*END-TAG*⟩ will be not. It doesn't seem to be hard to implement, if 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[`⟨`\cs`⟩`]{`⟨*version*⟩`}{`⟨*YYYY/MM/DD date*⟩`}{`⟨*text*⟩`}`

or, if you prefer the `\ProvidesPackage/Class` syntax,

`\chgs[`⟨`\cs`⟩`]{`⟨⟨*YYYY/MM/DD*⟩ ⟨*version*⟩ ⟨*text*⟩⟩`}`

The optional `\cs` argument may be a CS(with backslash) or a string. By default it's the most recently defined CS (see section about automatic detection of definitions).

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 omitted].

`\RecordChanges` 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

`\PrintChanges` 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

`\GlossaryMin` is supplied with doc [and gmdoc], called [… **gmglo.ist** for gmdoc]. The `\GlossaryMin`,

`\GlossaryPrologue` `\GlossaryPrologue` and `\GlossaryParms` macros are analogous to the `\Index`…

`\GlossaryParms` versions [see sec. The parameters p. 23]. (The LATEX 'glossary' mechanism is used for the change entries.)"

In gmdoc (unless you turn definitions detection off), you can put `\changes` after the line of definition of a command to set the default argument of `\changes` to that command. For example,

```
\newcommand*\dodecaphonic{…}
% \changes{v0.99e}{2007/04/29}{renamed from \cs{DodecaPhonic}}
```

results with a history (sub)entry:

v0.99e
(…)
`\dodecaphonic`:
renamed from `\DodecaPhonic`, *19*

Such a setting is in force till the next definition and *every* detected definition resets it.

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, `\changes` executes `\MakeGlossaryControls` that is defined as

\MakeGlossaryControls

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

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`. You may set this counter with a

ChangesStartDate
\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[7], then a note shall appear at the beginning of the changes history that informs the reader of omitting the earlier changes entries.

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

\ChangesStart{⟨*version?*⟩}{1000/00/00}

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

While scanning the CSes in the code, gmdoc counts them and prints the information 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.

\CheckSum

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*⟩.

There is also `\toCTAN{`⟨*date*⟩␣⟨*version*⟩`}`, a shorthand for

\toCTAN

\chgs{⟨*date*⟩␣⟨*version*⟩␣put␣to␣\acro{CTAN}␣on␣⟨*date*⟩}

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 \medskipamount by default. Subsequent blank code lines do not increase this space.

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

\CodeTopsep

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

\UniformSkips
\NonUniformSkips

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

\stanza
\chunkskip

Since \CodeTopsep glue is inserted automatically at each transition from the code (or code with an in-line comment) to the narration and reverse, it may happen that you want not to add such a glue exceptionally. Then there's the \nostanza command. You can use it before narration to remove the vskip before it or after narration to suppress the vskip after it.

\nostanza

\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, the end of a \DocInput file is marked with

\EOFMark

'□' given by the \EOFMark macro.

\everyeof

If you do use the ε-TEX's primitive \everyeof, be sure the contents of it begins with \relax because it's the token that stops the main macro scanning the code.

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

\CodeDelim

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

The unstarred version of \CodeDelim changes also the verb 'hyphen', the char appearing at the verbatim line breaks that is and affects the \narrationmark which by

\narrationmark

7 DEK in *TEX The Program* mentions that month as of TEX Version 0 release.

default typesets % followed by an en space.

\CodeDelim The starred version, \CodeDelim*, changes only the code delimiter and the char typeset remains untouched. Most probably you shouldn't use the starred version.

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.)

The tradition is that in the packages @ char is a letter i.e., of catcode 11. Frank Mittelbach in doc takes into account a possibility that a user wishes some other chars to be \MakePrivateLetters 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 \MakeShortVerbed. (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 codelinenum is called codelinenum so the macro printing it is \thecodelinenum. By default we don't change its LaTeX's definition which is equivalent to \arabic{codelinenum}.

\IndexPrefix Three more parameter macros, are \IndexPrefix, \EntryPrefix and \HLPre¦
\EntryPrefix fix. All three are provided with the account of including multiple files in one doc-
\HLPrefix ument. They are equal (almost) \@empty by default. The first may store main level index entry of which all indexed macros and environments would be sub-entries, 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 \Usg¦
\DefEntry Entry 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 {re¦lax} and a nontrivial use of it you may see in the source of chapter 640, where \def% \CommonEntryCmd{UsgEntry} makes all the index entries of the driver formatted as 'usage'.

IndexColumns
\IndexMin

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

\IndexPrologue

The text put at the beginning of the index is declared with a one-argument `\IndexPrologue`. Its default text at current index option you may admire on page 141. Of course, you may write your own `\IndexPrologue{`⟨*brand new index prologue*⟩`}`, but if you like the default and want only to add something to it, you are provided `\AtDIPrologue` one-argument declaration that adds the stuff after the default text. For instance, I used it to add a label and hypertarget that is referred to two sentences earlier.

\AtDIPrologue

\IndexLinksBlack

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

\IndexParms
\gaddtomacro

Other index parameters are set with the `\IndexParms` macro defined in line 6456 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.)

\actualchar
\quotechar
\levelchar
\encapchar

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.

\verbatimchar

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

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

\StopEventually
\Finale
\AlsoImplementation
\OnlyDescription

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

The `\OnlyDescription` and `\AlsoImplementation` declarations are intended to exclude or include the code part from the documentation. The point between the description and the implementation part should be marked with `\StopEventually{`⟨*the stuff to be executed anyway*⟩`}` and `\Finale` should be typed at the end of file. Then `\OnlyDescription` defines `\StopEventually` to expand to its argument followed by `\endinput` and
`\AlsoImplementation` defines `\StopEventually` to do nothing but pass its argument to `\Finale`.

The narration macros

\verb

To print the control sequences' names you have the `\verb` macro and its 'shortverb' version whatever you define (see the gmverb package).

\inverb

For short verbatim texts in the in-line comments gmdoc provides the `\inverb`⟨*a char*⟩…⟨*a char*⟩ (the name stands for 'in-line verbatim') command that redefines the gmverb breakables

to break with `%` at the beginning of the lower line to avoid mistaking such a broken verbatim commentary text for the code.

But nor `\verb[*]` neither `\inverb` will 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`.

\cs
\env

For usage in the in-line comments there are `\incs` and `\inenv` commands that take analogous arguments and precede the typeset command and environment names with a `%` if at the beginning of a new line. To those who like `\cmd`, there is also `\incmd`, an in-line version of the former.

\incs
\inenv
\incmd

And for line breaking at `\cs` and `\env` there is `\nlperc` to ensure `%` at the beginning of a new line and `\+` to use in `\cs` and `\env` argument for a discretionary hyphen that'll break to - at the end of the upper line and `%` at the beginning of the lower line. By default hyphenation of `\cs` and `\env` arguments is off, you can allow it only at `\-` or `\+`.

\nlperc
\+

There is also `\nlpercent` if you wish a discretionary `%` without `\incs` or `\inverb`.

\nlpercent

By default the multi-line in-line comments are typeset with a hanging indent (that is constant relatively to the current indent of the code) and justified. Since vertical alignment is determined by the parameters as they are at the moment of `\par`, no one can set the code line to be typeset ragged right (to break nicely if it's long) and the following in-line comment to be justified. Moreover, because of the hanging indent the lines of multi-line in-line comments are relatively short, you may get lots of overfulls. Therefore there is a Boolean switch `ilrr` (OCSR), whose name stands for 'In-Line Ragged-Right' and the in-line comments (and their codelines) are typeset justified in the scope of `\ilrrfalse` which is the default. When you write `\ilrrtrue`, then all in-line comments in its scope (and their codelines) will be typeset ragged right (and still with the hanging indent). Moreover, you are provided `\ilrr` and `\ilju` commands that set `\ilrrtrue` and `\ilrrfalse` for the current in-line comment only. Note you can use them anywhere within such a comment, as they set `\rightskip` basically. `\ilrr` and `\ilju` are no-ops in the stand-alone narration.

ilrr
\ilrr
\ilju

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

\pk
\file

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.

\catletter
\catother
\catactive

I wish my self-documenting code to be able to be typeset each package separately or several in one document. Therefore I need some 'flexible' sectioning commands and here they are: `\division`, `\subdivision` and `\subsubdivision` so far, that by default are `\let` to be `\section`, `\subsection` and `\subsubsection` respectively.

\division
\subdivision
\subsubdivision

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[8]). If you give *two*

8 See gmutils for some subtle details.

optionals, the first will go to the running head and the other to toc. (In both cases the mandatory argument goes only to the page).

\SetFileDiv

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

gmlonely
\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 (gmutils, to be precise) provides some TEX-related logos:

\AmSTeX typesets $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX,
\BibTeX BibTEX,
\SliTeX SLiTEX,
\PlainTeX Plain TEX,
\Web Web,
\TeXbook *The TEX book*,
\TB *The TEX book*
\eTeX ε-TEX,
\pdfeTeX pdfε-TEX
\pdfTeX pdfTEX
\XeTeX XƎTEX (the first E will be reversed if the graphics package is loaded or XƎTEX is at work) and
\LaTeXpar (LA)TEX.
\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 mandatory argument which is the contents of the marginpar.

\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 \Macro|Topsep. Both of them take care of not cumulating the vspaces.

quotation

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

If you don't like it, just call \RestoreEnvironment{quotation} after loading gmdoc. Note however that other environments using quotation, such as abstract, keep their shape.

\GetFileInfo
\filedate
\fileversion
\fileinfo

The \GetFileInfo{⟨*file name with extension*⟩} command defines \filedate, \fil|eversion 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 occurs in a \StraightEOL scope).

\ProvideFileInfo

If the input file doesn't contain \Provides… in the code layer, there are commands \ProvideFileInfo{⟨*file name with extension*⟩}[⟨*info*⟩]. (⟨*info*⟩ should consist of: ⟨*year*⟩/⟨*month*⟩/⟨*day*⟩␣⟨*version number*⟩␣⟨*a short note*⟩.)

Since we may documentally input files that we don't load, doc in gmdoc e.g., we provide a declaration to be put (in the comment layer) before the line(s) containing `\Provides...`. The `\FileInfo` command takes the subsequent stuff till the closing `]` and subsequent line end, extracts from it the info and writes it to the .aux and rescans the stuff. We use an ε-TEX primitive `\scantokens` for that purpose.

\FileInfo

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 title (or heading, with `\DocInclude` at the default settings), there's `\thfileinfo` macro that puts `\fileinfo` in `\thanks`.

\filenote

\thfileinfo

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.

\gmdnoindent

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

\CDPerc

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

\CDAnd

To get 'CS' which is 'CS' in small caps (in `\acro` to be precise), you can write `\CS`. This macro is `\protected` so you can use it safely in `\changes` e.g. Moreover, it checks whether the next token is a letter and puts a space if so so you don't have to bother about `\CS\␣`.

\CS

To enumerate the list of command's arguments or macro's parameters there is the `enumargs` environment which is a version of `enumerate` with labels like `#7`. You can use `\item` or, at your option, `\mand` which is just an alias for the former. For an optional arguments use `\opt` which wraps the item label in square brackets. Moreover, to align optional and mandatory arguments digit under digit, use the `enumargs*` environment.

enumargs

\mand

\opt

enumargs*

Both environments take an optional argument which is the number of `#`s. It's 1 by default, but also can be 2 or 4 (other numbers will typeset numbers without a `#`). Please feel free to notify me if you really need more hashes in that environment.

For an example driver file see chapter The driver.

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.

\AfterMacrocode

`\AfterMacrocode{`⟨*mc number*⟩`}{`⟨*the stuff*⟩`}` defines control sequence `\gmd@mchook`⟨*mc number*⟩ with the meaning ⟨*the stuff*⟩ which is put at the end of `macrocode` and `oldmc` number ⟨*mc number*⟩ (after the group).

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 they 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
\olddocIncludes
macrocode

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

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 that in the 'new' fashion the code and narration layers philosophy is in force and that is sustained within `macrocode[*]`. Which means basically that with 'new' settings when you write

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

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

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

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

oldmc

If you are used to the traditional (doc's) `macrocode` and still wish to use gmdoc new way, you have at least two options: there is the `oldmc` environment analogous to the traditional (doc's) `macrocode` (it also has the starred version), that's the first option (I needed the traditional behaviour once in this documentation, find out where & why).

\OldMacrocodes

The other is to write `\OldMacrocodes`. That declaration (OCSR) redefines `macrocode` and `macrocode*` to behave the traditional way. (It's always executed by `\OldDocInput` and `\olddocIncludes`.)

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

The driver part

In case of a single package, such as gmutils, a driver part of the package may look as follows and you put it before `\ProvidesPackage/Class`.

```
% \skiplines we skip the driver
\ifnum\catcode`\@=12

\documentclass[outeroff, pagella, fontspec=quiet]{gmdocc}
\usepackage{eufrak}% for |\continuum| in the commentary.
\twocoltoc
\begin{document}

\DocInput{\jobname.sty}
\PrintChanges
\thispagestyle{empty}
\typeout{%
  Produce change log with^^J%
  makeindex -r -s gmglo.ist -o \jobname.gls \jobname.glo^^J
  (gmglo.ist should be put into some texmf/makeindex
    directory.)^^J}
\typeout{%
  Produce index with^^J%
  makeindex -r \jobname^^J}
```

```
\afterfi{\end{document}}

\fi% of driver pass
%\endskiplines
```

\skiplines
\endskiplines

The advantage of `\skiplines`…`\endskiplines` over `\iffalse`…`\fi` is that the latter has to contain balanced `\if`s and `\fi`s while the former hasn't because it sanitises the stuff. More precisely, it uses the `\dospecials` list, so it sanitises also the braces.

Moreover, when the `countalllines`[⋆] option is in force, `\skipfiles`…`\endskipfiles` keeps the score of skipped lines.

Note `%\iffalse` … `%\fi` in the code layer that protects the driver against being typeset.

But gmdoc is more baroque and we want to see the driver typeset—behold.

```
\ifnum\catcode`\@=12

\errorcontextlines=100

\documentclass[countalllines, codespacesgrey, outeroff, debug, mwrep,
pagella, trebuchet, cursor, fontspec=quiet]{gmdocc}

\verbLongDashes

\DoNotIndex{\gmu@tempa \gmu@tempb \gmu@tempc \gmu@tempd % \gmu@tempe \gmu@tempf}

\twocoltoc
\title{The \pk{gmdoc} Package\\ i.e., \pk{gmdoc.sty} and
  \pk{gmdocc.cls}}
\author{Grzegorz `Natror' Murzynowski}
\date{\ifcase\month\relax\or January\or February\or March\or April\or May\or
  June\or July\or August\or September\or October\or November% \or
  December\fi\ \the\year}

%\includeonly{gmoldcomm}

\begin{document}

\maketitle

\setcounter{page}{2}% hyperref cries if it sees two pages numbered 1.
\tableofcontents
\DoIndex\maketitle

\SelfInclude
\DocInclude{gmdocc}
```

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 omitted.\stanza]
\DocInclude{gmutils}
\DocInclude{gmiflink}
\DocInclude{gmverb}
```

```
\DocInclude{gmoldcomm}
\typeout{%
  Produce␣change␣log␣with^^J%
  makeindex␣-r␣-s␣gmglo.ist␣-o␣\jobname.gls␣\jobname.glo^^J
  (gmglo.ist␣should␣be␣put␣into␣some␣texmf/makeindex␣
       directory.)^^J}
\PrintChanges
\typeout{%
  Produce␣index␣with^^J%
  makeindex␣-r␣\jobname^^J}
 \PrintIndex

\afterfi{%
\end{document}
```

MakeIndex shell commands:

```
  makeindex␣-r␣gmdoc
  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" ;-) .

`}\fi%` of `\ifnum\catcode`\@=12`, of the driver that is.

The code

For debug

```
\catcode`\^^C=9\relax
```

We set the `\catcode` of this char to $_{13}$ in the comment layer.

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 CSes, 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 their splendid code and adds a brilliant comment in `%`ed lines and that's all. Of course, if they wants to make a `\section` or `\emph`asise, they has to type respective CSes.

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 beginning 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

```
\RequirePackage{gmutils}[2008/08/30]%
```
includes redefinition of `\newif` to make the switches `\protected`.

```
\RequirePackage{xkeyval}%
```
we need key-vals later, but maybe we'll make the option key-val as well.

Maybe someone wants the code lines not to be numbered.

`\if@linesnotnum`
```
\newif\if@linesnotnum
```

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

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

`\if@uresetlinecount`
```
\newif\if@uresetlinecount
```

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

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

`\if@countalllines`
`\if@printalllinenos`
```
\newif\if@countalllines
\newif\if@printalllinenos
```

`countalllines`
```
\DeclareOption{countalllines}{%
```
to use the `\inputlineno` primitive and print real line numbers in a file.
```
  \@countalllinestrue
  \@printalllinenosfalse}
```

`countalllines*`
```
\DeclareOption{countalllines*}{%
  \@countalllinestrue
  \@printalllinenostrue}
```

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

`\if@noindex`
```
\newif\if@noindex
```

`noindex`
```
\DeclareOption{noindex}{\@noindextrue}
```

`\if@pageindex`
```
\newif\if@pageindex
```

`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).

`\if@indexallmacros`
```
\newif\if@indexallmacros
```

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

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

`\if@marginparsused`
```
\@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:

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

nomarginpar
```
\DeclareOption{nomarginpar}{\@marginparsusedfalse}
```

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

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

codespacesblank
```
\DeclareOption{codespacesblank}{%
  \AtEndOfPackage{% to allow codespacesgrey, codespacesblank
  \AtBeginDocument{\CodeSpacesBlank}}}
```

codespacesgrey
```
\DeclareOption{codespacesgrey}{%
  \AtEndOfPackage{% to put the declaration into the begin-document hook after
      definition of \visiblespace.
    \AtBeginDocument{\CodeSpacesGrey}}}
```

```
\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`. Since 2008/08/08 it also makes `\if...` switches `\protected` (redefines `\newif`)

```
\RequirePackage{gmutils}[2008/08/08]
```

A standard package for defining colours,

```
\RequirePackage{xcolor}
```

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 their favourite options, we do, with *ours*. If they 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=UseNone]{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}[2010/08/12]
```

```
\Store@Macros{\@verbatim\verb}
```

```
\if@noindex
```

```
  \AtBeginDocument{\gag@index}%
\else
  \RequirePackage{makeidx}\makeindex
\fi
```

for the latter macro see line 5695.

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\`⟨*one char*⟩ declaration provided by gmverb.

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

`\Code@Delim@St`
```
\def\Code@Delim@St#1{%
  {\escapechar\m@ne
    \@xa\gdef\@xa\code@delim\@xa{\string#1}}%
  \@esphack}
```

(`\@xa` is `\expandafter`, see gmutils.)

`\Code@Delim`
```
\def\Code@Delim#1{\VerbHyphen{#1}\Code@Delim@St{#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\%
```

And to typeset this code delimiter pretty, let's `\def`:

`\narrationmark`
```
\pdef\narrationmark{{\codett\verbhyphen}{\normalfont\enspace}%
    \ignorespaces}
```

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

`\gmd@preverypar`
```
\newtoks\gmd@preverypar
```

`\settexcodehangi`
```
\newcommand*\settexcodehangi{%
  \hangindent=\verbatimhangindent␣\hangafter=\@ne}%
```

we'll use it in the in-line 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 to `\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 derivatives for typesetting poetry. A computer program code *is* poetry.

\stanzaskip
```
\stanzaskip=\medskipamount
```

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 æsthetic minimality[9] let's unify them and the 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.
  \advance\belowdisplayshortskip by\smallskipamount
  \advance\belowdisplayshortskip by-1\smallskipamount% We advance
       % \belowdisplayshortskip forth and back to give it the \smallskip|
       % amount's shrink- and stretchability components.
  \topsep=\stanzaskip
  \partopsep=\z@
}
```

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 `\stan|zaskip`, 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`.

9 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).

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{%
  \par\addvspace{%
  \glueexpr\MacroTopsep
  \if@codeskipput-\CodeTopsep\fi
  \relax
}\@codeskipputgtrue}
```

And, for a smaller part of text,

\stanza
```
\pdef\stanza{%
  \par\addvspace{%
    \glueexpr\stanzaskip
    \if@codeskipput-\CodeTopsep\fi
    \relax}\@codeskipputgtrue}
```

Since the stanza skips are inserted automatically most often (cf. lines 3248, 3727, 3268, 3608, 3780), sometimes you may need to forbid them.

\nostanza
```
\newcommand*\nostanza{%
  \par
  \if@codeskipput\unless\if@nostanza\vskip-\CodeTopsep\relax%
      \fi\fi
  \@codeskipputgtrue\@nostanzagtrue
  \@afternarrgfalse\@aftercodegtrue}%
```
In the 'code to narration' case the first switch is enough but in the counter-case 'narration to code' both the second and third are necessary while the first is not.

To count the lines where they have begun not before them
```
\newgif\if@newline
```

`\newgif` is `\newif` with a 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 indicating whether the line has or has not been counted because of two reasons: 1. line numbering is optional, 2. counting the line falsifies that switch *before* the first char.)

The core

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

```
\begingroup\catcode`\^^M=\active%
\firstofone{\endgroup%
```
\DocInput
```
  \newcommand*{\DocInput}[1]{\begingroup%
    \edef\gmd@inputname{#1}%
```
we'll use it in some notifications.
```
    \NamedInput@prepare{#1}%
```
to make this input "named", as with `\NamedInput`.
```
    \let\gmd@currentlabel@before=\@currentlabel%
```
we store it because we'll do `\xdef`s 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.

`\@xa\@makeother\csname\code@delim\endcsname%`

`\gmd@resetlinecount%` due to the option `uresetlinecount` we reset the line number counter or do nothing.

`^^M` `\QueerEOL%` It has to be before the begin-input-hook to allow change by that hook.

`\@beginputhook%` my first use of it is to redefine `\maketitle` just at this point not globally.

`\everypar=\@xa{\@xa\@codetonarrskip\the\everypar}%`

`\gmd@guardedinput` `\edef\gmd@guardedinput{%`

`\@nx\@@input␣#1\relax%` `\@nx` is `\noexpand`, see gmutils. `\@@input` is the true TEX's `\input`.

`\gmd@iihook%` cf. line 7985

`\@nx\EOFMark%` to pretty finish the input, see line 3075.

`\@nx\CodeDelim\@xanxcs{\code@delim}%` to ensure the code delimiter is the same as at the beginning of input.

`\@nx^^M\code@delim%`

`}%` we add guardians after `\input`ing a file; somehow an error occurred without them.

`\catcode`\%=9␣%` for doc-compatibility.

`\setcounter{CheckSum}{0}%` we initialise the counter for the number of the escape chars (the assignment is `\global`).

`\everyeof{\relax}%` `\@nx` moved not to spoil input of toc e.g.

`\@xa\@xa\@xa^^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 before this group. In a very special case this could cause unexpected behaviour of cross-refs, but anyway we acted globally and so acts hyperref.

`\NamedInput@finish%` to clean up after a "named" input, as with `\Named|Input`.

`\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` `\pdef\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.

`\ifhmode\@afternarrgtrue\@codeskipputgfalse\fi%` being in the horizontal mode means we've just typeset some narration so we turn the re-

spective 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).

2917 `\@newlinegtrue%` to `\refstep` the lines' counter at the proper point.

2919 `\@dsdirgtrue%` to handle the DocStrip directives.

2920 `\@xa\@trimandstore\the\everypar\@trimandstore%` we store the previous value of `\everypar` register to restore it at a proper point. See line 3816 for the details.

2923 `\begingroup%`

2929 `\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.

2933 `\aftergroup\gmd@setclubpenalty%`

2934 `\let\par\@@par%` inside the verbatim group we wish `\par` to be genuine.

2936 `\let\verbatimfont\codett␣%`

2937 `\ttverbatim%` it applies the code-layer font (`\tt` by default) and makes specials other or `\active`-and-breakable. to turn verbatim specials off in `\scan|verb` s.

2941 `\gmd@DoTeXCodeSpace%`

2942 `\@makeother\|%` because `\ttverbatim` doesn't do that.

2943 `\MakePrivateLetters%` see line 4200.

2944 `\@xa\@makeother\code@delim%` we are almost sure the code comment char is among the chars having been $_{12}$ed 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.

2953 `\texcode@hook%` we add some special stuff, e.g. in gmdocc.cls we make star low.

2955 `\@xa\@ifnextcharRS\@xa{\code@delim}{%`

2957 `\gmd@continuenarration}{%`

2958 `\gmd@dolspaces%` it will launch `\gmd@typesettexcode`.

2959 `}%` end of `\@ifnextcharRS`'s else.

2960 `}%` end of `\gmd@textEOL`'s definition.

2963 `\emptify\texcode@hook`

`\gmd@setclubpenalty` 2965 `\def\gmd@setclubpenalty{\clubpenalty=3333␣}`

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

`\AtEndInput` 2969 `\def\AtEndInput{\g@addto@macro\@endinputhook}`

`\@endinputhook` 2970 `\def\@endinputhook{}`

Simili modo

`\AtBegInput` 2973 `\def\AtBegInput{\g@addto@macro\@beginputhook}`

`\@beginputhook` 2974 `\def\@beginputhook{}`

For the index input hooking now declare a macro, we define it another way at line 7985.

2978 `\emptify\gmd@iihook`

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

```
\AtBegInput{\let\gmd@@toc\tableofcontents
  \def\tableofcontents{%
    \@ifQueerEOL
    {\StraightEOL\gmd@@toc\QueerEOL}%
    {\gmd@@toc}%
  }%
}
```

As you'll learn from lines 3987 and 3974, 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 `\@ifEOLactive`:

```
\def\@ifEOLactive{%
      % #1 what if end of line is active,
      % #2 what if not.
  \ifnum\catcode`\^^M=\active␣\@xa\@firstoftwo\else\@xa%
      \@secondoftwo\fi}

\foone\obeylines{%
  \def\@ifQueerEOL{%
      % #1 what if line end is 'queer',
      % #2 what if not 'queer'.
    \@ifEOLactive{%
      \ifx^^M\gmd@textEOL\@xa\@firstoftwo\else\@xa%
          \@secondoftwo\fi}%
    {\@secondoftwo}}% of \@ifQueerEOL
}% of \foone
```

A footnote for the 'queer' line ends scope.

```
\pdef\qfootnote{%
  \@ifQueerEOL
  {\begingroup\StraightEOL\qfootnote@}%
  {\footnote}}

\DeclareCommand\qfootnote@{o>Lm}{%
  \endgroup␣% yes, we close the group: the arguments are already parsed and
      passed to this macro.
  \edef\gmu@tempa{%
    \@nx\footnote␣\IfValueT{#1}{[#1]}}%
  \gmu@tempa{#2}%
}
```

An emphasis command for 'queer' line ends.

```
\pdef\qemph{%
  \@ifQueerEOL
  {\begingroup\StraightEOL\qemph@}%
  {\emph}}

\pdef\qemph@#1{\endgroup\emph{#1}}
```

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.

```
\@emptify\gmd@ABIOnce
\AtEndOfPackage{\AtBegInput\gmd@ABIOnce}
```

```
\AtBegInputOnce 3057 \long\def\AtBegInputOnce#1{%
3070   \gaddtomacro\gmd@ABIOnce{\g@emptify\gmd@ABIOnce#1}}
```

Many tries of finishing the input cleanly led me to setting the guardians as in line 2873 and to

```
\EOFMark 3075 \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 8947.

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

```
3087 \begin{obeyspaces}%
3088 \gdef\CodeSpacesVisible{%
\gmd@DoTeXCodeSpace 3089 \def\gmd@DoTeXCodeSpace{%
3090 \obeyspaces\let␣=\breakablevisspace}}%

\CodeSpacesBlank 3097 \gdef\CodeSpacesBlank{%
3098 \let\gmd@DoTeXCodeSpace\gmobeyspaces%
3099 \let\gmd@texcodespace=\␣}% the latter \let is for the \if...s.

\CodeSpacesSmall 3102 \gdef\CodeSpacesSmall{%
\gmd@DoTeXCodeSpace 3103 \def\gmd@DoTeXCodeSpace{%
3104 \obeyspaces\def␣{\,\hskip\z@}}%
\gmd@texcodespace 3105 \def\gmd@texcodespace{\,\hskip\z@}}%

3107 \end{obeyspaces}

\CodeSpacesGrey 3109 \def\CodeSpacesGrey{%
3112   \CodeSpacesVisible
3113   \VisSpacesGrey% defined in gmverb
3114 }%
```

Note that `\CodeSpacesVisible` doesn't revert `\CodeSpacesGrey`.

```
3119 \CodeSpacesVisible
```

How the continuing of the narration should look like?

```
\gmd@continuenarration 3123 \def\gmd@continuenarration{%
3124   \endgroup
3125   \gmd@cpnarrline% see below.
3126   \@xa\@trimandstore\the\everypar\@trimandstore
3127   \everypar=\@xa{\@xa\@codetonarrskip\the\everypar}%
3128   \@xa\gmd@checkifEOL\@gobble}
```

Simple, isn't it? (We gobble the 'other' code delimiter. Despite of `\egroup` it's $_{12}$ because it was touched by `\futurelet` contained in `\@ifnextcharRS` in line 2955. And in line 3376 it's been read as $_{12}$. That's why it works in spite of that `%` is of category 'ignored'.)

```
3135 \if@countalllines
```

If the `countalllines` option is in force, we get the count of lines from the `\inputlineno` primitive. But if the option is `countalllines*`, we want to print the line number.

```
\gmd@countnarrline@ 3145   \def\gmd@countnarrline@{%
```

```
      \gmd@grefstep{codelinenum}\@newlinegfalse
      \everypar=\@xa{%
        \@xa\@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.
    }% of \gmd@countnarrline@
```

`\gmd@grefstep`

```
    \def\gmd@grefstep#1{% instead of diligent redefining all possible com-
        mands and environments we just assign the current value of the respec-
        tive TeX's primitive to the codelinenum counter. Note we decrease it by
        −1 to get the proper value for the next line. (Well, I don't quite know why,
        but it works.)
      \ifnum\value{#1}<\inputlineno
        \csname␣c@#1\endcsname\numexpr\inputlineno-1\relax
        \ifvmode\leavevmode\fi% this line is added 2008/08/10 after an all-
            night debuggery ;-) that showed that at one point \gmd@grefstep
            was called in vmode which caused adding \penalty 10000 to
            the main vertical list and thus forbidding page break during entire
            % oldmc.
        \grefstepcounter{#1}%
      \fi}% We wrap stepping the counter in an \ifnum to avoid repetition of
          the same ref-value (what would result in the "multiply defined labels"
          warning).
```

The `\grefstepcounter` macro, defined in gmverb, is a global version of `\ref|stepcounter`, observing the redefinition made to `\refstepcounter` by hyperref.

```
    \if@printalllinenos% Note that checking this switch makes only sense
        when countalllines is true.
```

`\gmd@cpnarrline`

```
      \def\gmd@cpnarrline{% count and print narration line
        \if@newline
          \gmd@countnarrline@
          \hyperlabel@line
          {\LineNumFont\thecodelinenum}\,\ignorespaces}%
        \fi}
      \else% not printalllinenos
        \emptify\gmd@cpnarrline
      \fi
```

`\gmd@ctallsetup`

```
\def\gmd@ctallsetup{% In the oldmc environments and with the \FileInfo
    declaration (when countalllines option is in force) the code is gob-
    bled as an argument of a macro and then processed at one place (at
    the end of oldmc e.g.) so if we used \inputlineno, we would have
    got all the lines with the same number. But we only set the counter
    not \refstep it to avoid putting a hypertarget.
  \setcounter{codelinenum}{\inputlineno}% it's global.
  \let\gmd@grefstep\hgrefstepcounter}

\else% not countalllines (and therefore we won't print the narration lines'
    numbers either)
  \@emptify\gmd@cpnarrline
  \let\gmd@grefstep\hgrefstepcounter% if we don't want to count all the
      lines, we only \ref-increase the counter in the code layer.
  \emptify\gmd@ctallsetup
```

```
\fi% of \if@countalllines
```

\skiplines
```
\def\skiplines{\bgroup
  \let\do\@makeother␣\dospecials␣% not \@sanitize because the latter
      doesn't recatcode braces and we want all to be quieten.
  \gmd@skiplines}

  \edef\gmu@tempa{%
    \long\def\@nx\gmd@skiplines##1\bslash␣endskiplines{%
        \egroup}}
  \gmu@tempa
```

And typesetting the TEX code?

```
\foone\obeylines{%
```
\gmd@typesettexcode
```
  \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 `\ttverbatim` 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{% TEX code EOL
```
\@newlinegtrue
```
      \@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, because 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.
      \@ifnextcharRS{\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\@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\gmd@grefstep{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 5569.
      \hskip\gmd@ldspaceswd%
      \advance\hangindent␣by\gmd@ldspaceswd%
      \xdef\settexcodehangi{%
        \@nx\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 line ends require extreme caution.
        \@nx\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.
    \if@ilgroup\aftergroup\egroup\@ilgroupfalse\fi% when we are in the in-line comment group (for ragged right or justified), we want to close it. But if we did it here, we would close the verbatim group for the code. But we set the switch false not to repeat \aftergroup\egroup.
    \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.
  }% of \gmd@typesettexcode.
}% of \foone\obeylines.
```

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.)

\gmd@spacewd
\gmd@ldspaceswd

```
\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 \breakablevisspace. So let us \let for future:

\gmd@texcodespace

```
\let\gmd@texcodespace=\breakablevisspace
```

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

\gmd@dolspaces

```
\def\gmd@dolspaces{%
  \ifx\gmd@texcodespace\@let@token
    \@dsdirgfalse
    \afterfi{\settowidth{\gmd@spacewd}{\visiblespace}%
    \gmd@ldspaceswd=\z@
```

```
    \gmd@eatlspace}%
  \else\afterfi{% about this smart macro and other of its family see gmutils sec. 3.
    \if@afternarr\if@aftercode
        \ifilrr\bgroup␣\gmd@setilrr\fi
    \fi\fi
    \par% possibly after narration
    \if@afternarr\if@aftercode
        \ifilrr\egroup\fi
    \fi\fi
    \gmd@typesettexcode}%
  \fi}
```

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

\gmd@eatlspace
```
\def\gmd@eatlspace#1{%
  \ifx\gmd@texcodespace#1%
    \advance\gmd@ldspaceswd␣by\gmd@spacewd% we don't \advance it \globally because the current group may be closed iff we meet % and then we'll won't indent the line anyway.
    \afteriffifi\gmd@eatlspace
  \else
    \if\code@delim\@nx#1%
      \gmd@ldspaceswd=\z@
      \afterfifi{\gmd@continuenarration\narrationmark}%
    \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.

```
\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
```
\def\gmd@charbychar#1{%
  \ifhmode\prevhmodegtrue
  \else\prevhmodegfalse
  \fi
  \if\code@delim\@nx#1%
    \def\next{% occurs when next a \hskip4.875pt is to be put
      \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
    \ifx\relax#1\def\next{%
      \endgroup}% special case of end of file thanks to \everyeof.
    \else
      \if\code@escape@char\@nx#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.
        \def\next{%
          \gmd@countheline#1\scan@macro}%
```

```
      \else
        \def\next{%
          \gmd@EOLorcharbychar#1}%
      \fi
    \fi
  \fi\next}
```

`\debug@special`

```
\def\debug@special#1{%
  \ifhmode\special{color␣push␣gray␣0.#1}%
  \else\special{color␣push␣gray␣0.#1000}\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 thoroughly. Increasing the counter is divided into cases and it's very low level in one case because `\refstepcounter` and `\stepcounter` added some stuff that caused blank lines, at least with hyperref package loaded.

`\gmd@EOLorcharbychar`

```
\def\gmd@EOLorcharbychar#1{%
  \ifx\gmd@texcodeEOL#1%
    \if@newline
      \@newlinegfalse
    \fi
    \afterfi{#1}% here we print #1.
  \else% i.e., #1 is not a (very active) line end,
    \afterfi
    {%
\gmd@countheline#1\gmd@charbychar}% or here we print #1. Here we would also possibly mark an environment but there's no need of it because declaring an environment to be marked requires a bit of commentary and here we are after a code ^^M with no commentary.
 \fi}
```

`\gmd@countheline`

```
\def\gmd@countheline{%
  \ifvmode
    \if@newline
      \leavevmode
      \gmd@grefstep{codelinenum}\@newlinegfalse
      \hyperlabel@line
    \fi
    \printlinenumber
    \mark@envir
  \else% not vmode
    \if@newline
      \gmd@grefstep{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).

`\gmd@percenthack`

```
\def\gmd@percenthack{%
  \ifprevhmode\aftergroup\narrationmark% We add a space after %, because I think it looks better. It's done \aftergroup to make the spaces possible after the % not to be typeset.
```

```
    \else\aftergroup\gmd@dsNarrChecker% remember that \gmd@precent|hack 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}
```

We want to handle the case of the verbatim mode's closing directive, which may by merely any text from `%` to the end of line.

`\ifgmd@dsVerb`
```
\newif\ifgmd@dsVerb
```

the informer whether we should look at such a closing at all (hope it will speed up parsing)

```
\foone{\obeylines}%
{%
```
`\gmd@dsVerbChecker`
```
  \def\gmd@dsVerbChecker%
  #1% stuff for checking normal directive
  #2% line contents
  ^^M{%
\typeout{verb␣checker␣l.\the\inputlineno}%
    \ifnum\strcmp{\detokenize{#2}}{\gmd@dsVerbDelim}=\z@%
      \global\gmd@dsVerbfalse%
```
`\gmd@modulehashone`
```
      \def\gmd@modulehashone{%
        \ModuleVerbClose{\gmd@dsVerbDelim}%
        \global\emptify\gmd@dsVerbDelim%
        \@afternarrgfalse\@aftercodegtrue%
        \@codeskipputgfalse␣%
      }%
      \@xa\@firstoftwo%
    \else␣\@xa\@secondoftwo␣%
    \fi%
    {\gmd@textEOL\gmd@modulehashone^^M}%
    {\begingroup%
      \endlinechar=\m@ne␣%
      \@XA{%
        \endgroup#1}\scantokens{#2}^^M% note that \scantokens adds char \endlinechar which we assure to be ^^M
    }%
  }% of \gmd@dsVerbChecker
}% of \obeylines
```
`\gmd@dsChecker`
```
\def\gmd@dsChecker#1{%
  \@dsdirgfalse
  \ifgmd@dsVerb
    \@xa\@firstofone
  \else
    \@xa\@secondoftwo
  \fi
  {\gmd@dsVerbChecker}%
  {#1}%
}% of \gmd@dsChecker
```
`\gmd@dsNarrChecker`
```
\def\gmd@dsNarrChecker#1{%
  \gmd@dsChecker
  {\@ifnextcharRS<{%
      \@xa\gmd@docstripdirective\@gobble}{#1}}%
```

```
}% of \gmd@dsNarrChecker
```

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.

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

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

\gmd@narrcheckifds
```
\def\gmd@narrcheckifds{%
  \gmd@dsNarrChecker{\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.

\gmd@checkifEOLmixd
```
\def\gmd@checkifEOLmixd{%
  \gmd@cpnarrline
  \everypar=\@xa{\@xa\@codetonarrskip\the\gmd@preverypar}%
  \@afternarrgfalse\@aftercodegtrue
  \ifhmode\@codeskipputgfalse\fi
  \@ifnextcharRS{\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., 3574.
    \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.

\par
```
    \def\par{% the narration \par.
      \ifhmode% (I added this \ifhmode as a result of a heavy debug.)
        \if@afternarr\if@aftercode
            \unless\if@ilgroup\bgroup\@ilgrouptrue\fi
            \ifilrr\gmd@setilrr\fi
        \fi\fi
        \@@par
        \if@afternarr
          \if@aftercode
          \if@ilgroup\egroup\fi% if we are both after code and after narration it means we are after an in-line comment. Then we probably end a group opened in line 3647
            \if@codeskipput\else\gmd@codeskip2%
                \@aftercodegfalse\fi
          \else\gmd@nocodeskip2{naC}%
          \fi
        \else\gmd@nocodeskip2{naN}%
```

```
3613          \fi
3614          \prevhmodegfalse\gmd@endpe% when taken out of \ifhmode, this
                  line caused some codeline numbers were typeset with \leftskip =
                  0.
3617          \everypar=\@xa{%
3618            \@xa\@codetonarrskip\the\gmd@preverypar}%
3619          \let\par\@@par%
3620        \fi}% of \par.
3621      \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.

`\gmd@endpe`
```
3628 \def\gmd@endpe{%
3629   \ifprevhmode
3630     \settexcodehangi% ndent
3631     \leftskip=\CodeIndent
3633   \else
3634     \leftskip=\TextIndent
3635     \hangindent=\z@
3636     \everypar=\@xa{%
3637       \@xa\@codetonarrskip\the\gmd@preverypar}%
3639   \fi}
```

Now a special treatment for an in-line comment:

`\ifilrr`
```
3643 \newif\ifilrr
```

`\ilrr`
```
3645 \def\ilrr{%
3646   \if@aftercode
3647     \unless\if@ilgroup\bgroup\@ilgrouptrue\fi% If we are 'aftercode',
           then we are in an in-line comment. Then we open a group to be able to
           declare e.g. \raggedright for that comment only. This group is closed
           in line 3604 or 3305.
3652     \ilrrtrue
3653   \fi}
```

`\if@ilgroup`
```
3655 \newif\if@ilgroup
```

`\gmd@setilrr`
```
3657 \def\gmd@setilrr{\rightskipoptplus\textwidth}
```

`\ilju`
```
3659 \def\ilju{% when in-line comments are ragged right in general but we want just
           this one to be justified.
3661   \if@aftercode
3662     \unless\if@ilgroup\bgroup\@ilgrouptrue\fi
3663     \ilrrfalse
3664   \fi}
```

`\verbcodecorr`
```
3666 \def\verbcodecorr{% a correction of vertical spaces between a verbatim and
           code. We put also a \par to allow parindent in the next commentary.
3670   \vskip-\lastskip\vskip-4\CodeTopsep\vskip3\CodeTopsep\par}
```

Numbering (or not) of the lines

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

```
3678 \if@uresetlinecount% with uresetlinecount option…
3679   \@relaxen\gmd@resetlinecount% … we turn resetting the counter by \Doc¦
         % Input off…
\resetlinecountwith 3681   \newcommand*\resetlinecountwith[1]{%
    codelinenum 3682     \newcounter{codelinenum}[#1]}% … and provide a new declaration of
                           the counter.
3684 \else% With the option turned off…
 DocInputsCount 3685   \newcounter{DocInputsCount}%
    codelinenum 3686   \newcounter{codelinenum}[DocInputsCount]% … we declare the \DocIn¦
                         puts' number counter and the codeline counter to be reset with stepping of
                         it.
\gmd@resetlinecount 3692   \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.
3696 \fi
```

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

```
\printlinenumber 3700 \newcommand*\printlinenumber{%
3701   \leavevmode\llap{\rlap{\LineNumFont$\phantom{999}$\llap{%
         \thecodelinenum}}%
3702     \hskip\leftskip}}
    \LineNumFont 3704 \def\LineNumFont{\normalfont\tiny}
3706 \if@linesnotnum\@relaxen\printlinenumber\fi
\hyperlabel@line 3708 \newcommand*\hyperlabel@line{%
3709   \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.
3712   \else
3713     \raisebox{2ex}[1ex][\z@]{\gmhypertarget[clnum.%
3714       \HLPrefix\arabic{codelinenum}]{}}%
3715   \fi}
```

Spacing with \everypar

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

```
\gmd@codeskip 3725 \newcommand*\gmd@codeskip[1]{%
3726   \@@par\addvspace\CodeTopsep
3727   \@codeskipputgtrue\@nostanzagfalse}
```

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 3738 \newgif\if@codeskipput
```

A switch to control \nostanzas:

```
3741 \newgif\if@nostanza
```

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

\gmd@nocodeskip
```
\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
  \gmd@codeskip  \renewcommand*\gmd@codeskip[1]{%
                   \hbox{\rule{1cm}{3pt}␣#1!!!}}
  \gmd@nocodeskip  \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 in-line comment) is followed by a homogeneous comment line or reverse. Let us dedicate a Boolean switch to this then.

\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.

\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
```
\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%
      \else\gmd@nocodeskip4{naC}%
      \fi
    \fi
  \fi
  \leftskip\TextIndent% this line is a patch against a bug-or-feature that in
       certain cases the narration \leftskip is left equal the code leftskip. (It
       happens when there are subsequent code lines after an in-line comment not
       ended with an explicit \par.) Before v0.99n it was just after line 3780.
  \@aftercodegfalse\@nostanzagtrue
}
```

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 `\expandafter`ing 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
\@trimandstore@hash
3805 \long\def\@trimandstore#1\@trimandstore{%
3806   \def\@trimandstore@hash{#1}%
3807   \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.
3811     \gmd@preverypar={}%
3812   \else
3813     \afterfi{\@xa\@trimandstore@ne\the\everypar%
          \@trimandstore}%
3814   \fi}

\@trimandstore@ne
\trimmed@everypar
3816 \long\def\@trimandstore@ne#1#2\@trimandstore{%
3817   \def\trimmed@everypar{#2}%
3818   \ifx\@codetonarrskip#1%
3819     \gmd@preverypar=\@xa{\trimmed@everypar}%
3820   \else
3821     \gmd@preverypar=\@xa{\the\everypar}%
3822   \fi}
```

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

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:

```
3837 \catcode`\^^B=\active% note we re\catcode ⟨char2⟩ globally, for the entire
        document.
3839 \catcode`\^^V=\active␣% the same for ^^V.
3840 \catcode`\^^U=\active␣% and for ^^U.
3841 \foone{\obeylines}%
^^B
3842 {\pdef\QueerCharTwo{%
\QueerCharTwo
3843     \protected\def^^B##1^^M{%
3845       \ifhmode\unskip\space\ignorespaces\fi}}% It shouldn't be \  not
            to drive TEX into hmode.
```

The `^^V` char is intended to mark parts of code commented out which are to be typeset verbatim. Lines are begun with narration marks (%'s by default) and the narration-verbatim-typewriter font is used.

The `^^U` char is intended for the lines commented out which are to be typeset as almost-invisible (second-class). They are marked with the narration mark as with `^^V` and a special font setting is used, which is a gray colour by default (in addition to the narration-verbatim-typewriter).

```
^^V
^^U
\gmd@UVdefs
\gmd@UV@percent
3860   \pdef\gmd@UVdefs␣{%
3861     \def\gmd@UV@percent{\global\let\verb@balance@group\@empty␣
           ␣%
   %%      \hyphenchar\font=\gmv@storedhyphenchar % it works back for the
current paragraph so destroys our special hyphenchar.
3864       \egroup\endgroup␣%
3865     }% of \gmd@UV@percent
```

```
3867      \@xa\def\@xa\verb@egroup@UV\@xa{%
3868        \gmd@UV@percent␣^^M%
3869      }% of \verb@egroup@UV.

3871      \addtomacro\gmd@UV@percent{\narrationmark}%
\gmd@QueerUV 3873      \pdef\gmd@QueerUV␣##1{%
3874        \scantokens\@xa{\code@delim␣%
3875          \fooatletter{\@ifQueerEOL\@gobble}{}%
3876        }% of \scantokens

3878        \par%
3879        {\codett\verbhyphen}\narrationmark␣%
3880        \begingroup␣%
3881        \catcode`\^^M=\active␣%
3883        \let\verb@egroup=\verb@egroup@UV␣%
3884        \verb^^M%
3886        ##1% nothing in ^^V case and a gray setting in the ^^U case.

3888        \begingroup␣%
3889        \@xa\lccode\@xa`\@xa~\@xa`\code@delim%
3890        \lowercase{\endgroup\let~\gmd@UV@percent␣}%
3891        \@xa\catcode\@xa`\code@delim\active␣%
3892      }% of \gmd@QueerUV
3893    }% of \gmd@UVdefs

  \QueerU 3896    \pdef\QueerU␣{%
3897      \unless\ifdefined\gmd@QueerU%
3899        \gmd@UVdefs␣%
\gmd@QueerU 3900        \pdef\gmd@QueerU␣{\gmd@QueerUV␣{\QueerUFont␣}}%
3901        \let^^U\gmd@QueerU%
3902        \catcode`\^^U=\active%
3903      \fi␣%
3904    }% of \QueerU

  \QueerV 3906    \pdef\QueerV␣{%
3907      \unless\ifdefined\gmd@QueerV␣%
3909        \gmd@UVdefs␣%
\gmd@QueerV 3910        \pdef\gmd@QueerV␣{\gmd@QueerUV␣{}}%
3911        \let^^V\gmd@QueerV%
3912        \catcode`\^^V=\active%
3913      \fi␣%
3914    }% of \QueerV

3916 }% of \foone

3918 \QueerCharTwo
3919 \QueerV
3920 \QueerU
\QueerUFont 3921 \def\QueerUFont{\color{black!35}}

3924 \AtBegInput{\@ifEOLactive{\catcode`\^^B\active}{}\QueerCharTwo}%
```

We repeat redefinition of ⟨*char2*⟩ at begin of the documenting input, because doc.dtx suggests that some packages (namely inputenc) may re\catcode such unusual characters.

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

I also liked the doc's idea of 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.
\foone\obeylines{%
  \def\QueerCharOne{%
    \def^^A{%
      \bgroup\let\do\@makeother\dospecials\gmd@gobbleuntilM}}%
  \def\gmd@gobbleuntilM#1^^M{\egroup\ignorespaces^^M}%
}
\QueerCharOne
\AtBeginInput{\@ifEOLactive{\catcode`\^^A%
    \active}\QueerCharOne}% see note after line 3924.
```

^^A, \QueerCharOne, \gmd@gobbleuntilM

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 line-breaking of the command arguments. Another—making possible executing some code lines during the documentation pass.

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

\StraightEOL

```
\foone\obeylines{%
  \def\QueerEOL{%
    \catcode`\^^M=\active%
    \let^^M\gmd@textEOL%
    \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}%
}
```

\QueerEOL

To make ^^M behave more like a 'normal' line end I command it to add a ␣10 at first. It works but has one unwelcome feature: if the line has nearly \textwidth, this closing space may cause line breaking and setting a blank line. To fix this I \advance the \parfillskip:

```
\def\gmd@parfixclosingspace{{%
    \advance\parfillskip␣by-\gmd@closingspacewd
    \if@aftercode\ifilrr␣\gmd@setilrr␣\fi\fi
    \par}%
  \if@ilgroup\aftergroup\egroup\@ilgroupfalse\fi% we are in the verbatim group so we close the in-line comment group after it if the closing is not yet set.
```

\gmd@parfixclosingspace

```
}
```

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.

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

See also line 3241 to see what we do in the codeline case when no closing space is added.

And one more detail:

```
\foone\obeylines{%
  \if␣1␣1%
\gmd@bslashEOL
    \protected\def\gmd@bslashEOL{\␣\@xa\ignorespaces^^M}%
  }% of \foone. Note we interlace here \if with a group.
\else%
\gmd@bslashEOL
  \protected\def\gmd@bslashEOL{%
    \ifhmode\unskip\fi\␣\ignorespaces}
  \fi
```

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

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

Adjustments of `verbatim` and `\verb`

To make `verbatim[*]` typeset its contents with the TEX code's indentation:

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

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

```
\check@percent
\AtBegInput{\long\def\check@percent#1{%
    \gmd@cpnarrline% to count the verbatim lines and possibly print their numbers. This macro is used only by the verbatim end of line.
    \@xa\ifx\code@delim#1\else\afterfi{#1}\fi}}
```

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

```
\AddtoPrivateOthers
\def\AddtoPrivateOthers#1{%
  \@xa\def\@xa\doprivateothers\@xa{%
    \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%
```

```
\gm@verb@eol
\verb@egroup
\AtBegInput{\def\gm@verb@eol{\obeylines%
    \def^^M{\verb@egroup\@latex@error{%
        \@nx\verb␣ended␣by␣end␣of␣line}%
      \@ifEOLactive{^^M}{\@ehc}}}%
\endgroup
```

To distinguish the code typewriter from the narrative typewriter:

(2010/08/14, v0.993:) due to troubles with bad fontification in the narration layer I implement the counterpart to `\narrativett`: `\codett`, which is `\tt` by default so it even may be transparent to the users.

```
\verbatimfont
\def\verbatimfont{\narrativett}
    \codett
\def\codett{\tt}

    \texttt
\pdef\texttt#1{{\narrativett#1}}
```

To rescan the verbatim's contents and show its effect, the gmverb package provides a modifier of the inner macros to make them throw the verbatim contents as a contents of a macro. Let's do that.

```
\VerbatimPitch

  \ResultsIn
\def\ResultsIn{results␣in:}

\DeclareEnvironment{verbatim@p}{}
{\begingroup
  \verbatim
}
{\endverbatim
  \endgroup
  \ResultsIn
  \[\parbox{0,85\textwidth}{%
    \newlinechar=\endlinechar
    \StraightEOL
    \scantokens\@xa{\VerbatimContents}%
  }% of parbox
  \]%
}
```

(Note that gmverb provides a reverse: macro that first executes its

Macros for marking of 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 escape char) 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␣@}
\edef\quotechar{\string␣"}
\edef\encapchar{\xiiclub}
\edef\levelchar{\string␣!}
```

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 4187 \def\CodeEscapeChar#1{%
                4188   \begingroup
                4189   \escapechar\m@ne
\code@escape@char 4190   \xdef\code@escape@char{\string#1}%
                4191   \endgroup}
```

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

```
                4197 \CodeEscapeChar\\
```

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

```
                4200 \@ifundefined{MakePrivateLetters}{%
\MakePrivateLetters 4201   \def\MakePrivateLetters{\makeatletter\catcode`\*=11␣}}{}
```

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

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

```
\MakePrivateOthers 4209 \def\MakePrivateOthers{\let\do=\@makeother␣\doprivateothers}
```

We use this macro to re`\catcode` the space for marking the environments' names and the caret for marking chars such as `^^M`, see line 5759. So let's define the list:

```
\doprivateothers 4213 \def\doprivateothers{\do\␣\do\^}
```

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

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

```
                4227 \foone\obeylines{%
\scan@macro     4228   \def\scan@macro#1{%
                4229     \ifx#1^^M\@xa#1\else\afterfi{\scan@macro@#1}\fi%
                4230   }% of \scan@macro,
                4231 }% of \foone.
```

```
\scan@macro@ 4234 \def\scan@macro@#1{%
```
we are sure to scan at least one token which is not the line end 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 3444 and 3448.

So, we step the checksum counter first,

```
                4241   \step@checksum%
```
(see line 7063 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.

```
4250  \ifcat␣a\@nx#1%
4251    \quote@char#1%
4252    \xdef\macro@iname{\gmd@maybequote#1}% global for symmetry with line
            4270.
4254    \xdef\macro@pname{\string#1}%  we'll print entire name of the macro
            later.
```

We `\string` it here and in the lines 4274 and 4286 to be sure it is whole $_{12}$ for easy testing for special index entry formats, see line 5181 etc. Here we are sure the result of `\string` is $_{12}$ since its argument is $_{11}$.

```
4261    \afterfi{\@ifnextcat{a}{\gmd@finishifstar#1}{%
            \finish@macroscan}}%
4262  \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 $_{12}$.

```
4269    {\uccode`9=`#1%
4270      \uppercase{\xdef\macro@iname{9}}%
4271    }%
4272    \quote@char#1%
4273    \xdef\macro@iname{\gmd@maybequote\macro@iname}%
4274    \xdef\macro@pname{\xiistring#1}%
4275    \afterfi␣\finish@macroscan
4276  \fi}% of \scan@macro@. The \xiistring macro, provided by gmutils, is used
          instead of original \string because we wish to get ␣$_{12}$('other' space).
```

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

`\continue@macroscan`

```
4283 \def\continue@macroscan#1{%
4284   \quote@char#1%
4285   \xdef\macro@iname{\macro@iname␣\gmd@maybequote#1}%
4286   \xdef\macro@pname{\macro@pname␣\string#1}%  we know#1 to be 11, so
           we don't need \xiistring.
4289   \@ifnextcat{a}{\gmd@finishifstar#1}{\finish@macroscan}%
4290 }
```

As you may guess, `\@ifnextcat` is defined analogously to `\@ifnextchar` but the test it does is `\ifcat` (not `\ifx`). (Note it wouldn't work for an active char as the 'pattern'.)

We treat the star specially since in usual LaTeX it should finish the scanning of a CS name—we want to avoid scanning `\command*argum` as one CS.

`\gmd@finishifstar`

```
4299 \def\gmd@finishifstar#1{%
4300   \if*\@nx#1\afterfi\finish@macroscan% note we protect #1 against ex-
           pansion. In gmdoc verbatim scopes some chars are active (e.g. \ ).
4303   \else\afterfi\continue@macroscan
4304   \fi}
```

If someone *really* uses * as a letter please let me know.

\quote@char 4308 \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.
4311 \uppercase{%
4312 \@ifinmeaning␣9\of␣\indexcontrols
4313 {\glet\gmd@maybequote\quotechar}%
4314 {\g@emptify\gmd@maybequote}%
4315 }%
4316 }}

This macro is used for catching chars that are MakeIndex's controls. How does it work?

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

In \quote@char the second argument for gmutils \@ifinmeaning is \index¦controls defined as the (expanded and 'other') sequence of the MakeIndex controls. \@ifinmeaning defines its inner macro \gmd@in@@ to take two parameters separated by the first and the second \@ifinmeaning'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. \gmd@in@@ is put before a string consisting of \@ifinmeaning's second and first parameters (in such a reversed order) and \gmd@in@@ itself. In such a sequence it looks for something fitting its parameter pattern. \gmd@in@@ is sure to find the parameters delimiter (\gmd@in@@ itself) and the separator, \ifismember's #1 i.e., the investigated char, because they are just there. But the investigated char may be found not near the end, where we put it, but among the MakeIndex controls' list. Then the rest of this list and \ifismember's #1 put by us become the second argument of \gmd@in@@. What \gmd@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 \gmd@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.

Since version 0.99g \@ifinmeaning is used also in testing whether a detector is already present in the carrier in the mechanism of automatic detection of definitions (line 4521).

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 4370 \AtBeginDocument{\xdef\indexcontrols{%
4371 \bslash\levelchar\encapchar\actualchar\quotechar}}

\ifgmd@glosscs 4375 \newif\ifgmd@glosscs% we use this switch to keep the information whether a history entry is a CS or not.

\finish@macroscan 4379 \newcommand*\finish@macroscan{%

First we check if the current CS is not just being defined. The switch may be set true in line 4418

```
    \ifgmd@adef@cshook% if so, we throw it into marginpar and index as a def entry…
      \gmu@ifundefined{gmd/iexcl/\macro@pname\space}{% … if it's not excluded from indexing.
        \@xa\Code@MarginizeMacro\@xa{\macro@pname}%
        \@xa\@defentryze\@xa{\macro@pname}{1}}{}% here we declare the kind of index entry and define \last@defmark used by \changes
      \global\gmd@adef@cshookfalse% we falsify the hook that was set true just for this CS.
    \fi
```

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

```
    \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 $:
\im@firstpar      \def\im@firstpar{[$%
        ]}%
\im@firstpar    \else\def\im@firstpar{}%
    \fi
    \@xa␣\index@macro\im@firstpar\macro@iname\macro@pname
    \maybe@marginpar\macro@pname
    \if\xiispace\macro@pname\relax\gmd@texcodespace
    \else
      {\noverbatimspecials\Restore@Macro\verb
        \@xa\scanverb\@xa{\macro@pname}}% we typeset scanned CS.
    \fi
    \let\next\gmd@charbychar
    \gmd@detectors% for automatic detection of definitions. Defined and explained in the next section. It redefines \next if detects a definition command and thus sets the switch of line 4379 true.
    \next
}
```

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.

```
\maybe@marginpar \def\maybe@marginpar#1{%
    \gmu@ifundefined{gmd/2marpar/\@xa\detokenize\@xa{#1}}{}{%
      \edef\gmu@tempa{%
        \unexpanded{\Text@Marginize*}%
        {\bslash\@xa\unexpanded\@xa{#1}}%
      }\gmu@tempa
```

`\macro@pname`, which will be the only possible argument to `\maybe@marg¦% inpar`, contains the macro's name without the escape char so we added it here.

```
      \@xa\g@relaxen
```

```
4446      \csname␣gmd/2marpar/\@xa\detokenize\@xa{#1}\endcsname% we re-
              set the switch.
4447    }}
```

Since version 0.99g we introduce automatic detection of definitions, it will be implemented in the next section. The details of indexing CSes are implemented in the section after it.

Automatic detection of definitions

To begin with, let's introduce a general declaration of a defining command. `\Declare¦Defining` comes in two flavours: 'sauté', and with star. The 'sauté' version without an optional argument declares a defining command of the kind of `\def` and `\newcommand`: whether wrapped in braces or not, its main argument is a CS. The star version without the optional argument declares a defining command of the kind of `\newenvironment` and `\DeclareOption`: whose main mandatory argument is text. Both versions provide an optional argument in which you can set the keys. Probably the most important key is `star`. It determines whether the starred version of a defining command should be taken into account. For example, `\newcommand` should be declared with `[star=true]` while `\def` with `[star=false]`. You can also write just `[star]` instead of `[star=true]`. It's the default if the `star` key is omitted.

Another key is `type`. Its possible values are the (backslashless) names of the defining commands, see below.

We provide now more keys for the xkeyvalish definitions: `KVpref` (the key prefix) and `KVfam` (the key family). If not set by the user, they are assigned the default values as in xkeyval: `KVpref` letters `KV` and `KVfam` the input file name. The latter assignment is done only for the `\DeclareOptionX` defining command because in other xkeyval definitions (`\define@[...]key`) the family is mandatory.

`\DeclareDefining` and the detectors

Note that the main argument of the next declaration should be a CS *without star*, unless you wish to declare only the starred version of a command. The effect of this command is always global.

`\DeclareDefining`

```
4489 \outer\def\DeclareDefining{\begingroup
4490   \MakePrivateLetters
4491   \gmu@ifstar
4492     {\gdef\gmd@adef@defaulttype{text}\Declare@Dfng}%
4493     {\gdef\gmd@adef@defaulttype{cs}\Declare@Dfng}%
4494 }
```

The keys except `star` depend of `\gmd@adef@currdef`, therefore we set them having known both arguments

`\Declare@Dfng`

```
4498 \newcommand*\Declare@Dfng[2][]{%
4499   \endgroup
4500   \Declare@Dfng@inner{#1}{#2}%
4501   \ifgmd@adef@star% this switch may be set false in first \Declare@Dfng@inner
          (it's the star key).
4503     \Declare@Dfng@inner{#1}{#2*}% The catcode of * doesn't matter since
          it's in \csname...\endcsname everywhere.
4507   \fi}
```

`\Declare@Dfng@inner`

```
4510 \def\Declare@Dfng@inner#1#2{%
4511   \edef\gmd@resa{%
4512     \@nx\setkeys[gmd]{adef}{type=\gmd@adef@defaulttype}}%
```

```
    \gmd@resa
    {\escapechar\m@ne
\gmd@adef@currdef     \xdef\gmd@adef@currdef{\string#2}%
    }%
    \gmd@adef@setkeysdefault
    \setkeys[gmd]{adef}{#1}%
    \@xa\@ifinmeaning
      \csname␣gmd@detect@\gmd@adef@currdef\endcsname
      \of\gmd@detectors{}{%
        \@xa\gaddtomacro\@xa\gmd@detectors\@xa{%
          \csname␣gmd@detect@\gmd@adef@currdef\endcsname}}% we add a CS
                % \gmd@detect@⟨def name⟩ (a detector) to the meaning of the detectors' carrier. And we define it to detect the #2 command.
    \@xa\xdef\csname␣gmd@detectname@\gmd@adef@currdef%
         \endcsname{%
      \gmd@adef@currdef}%
    \edef\gmu@tempa{% this \edef is to expand \gmd@adef@TYPE.
      \global\@nx\@namedef{gmd@detect@\gmd@adef@currdef}{%
        \@nx\ifx
          \@xanxcs{gmd@detectname@\gmd@adef@currdef}%
          \@nx\macro@pname
          \@nx\n@melet{next}{gmd@adef@\gmd@adef@TYPE}%
          \@nx\n@melet{gmd@adef@currdef}{gmd@detectname@%
               \gmd@adef@currdef}%
        \@nx\fi}}%
    \gmu@tempa
    \SMglobal\Store@MacroSt␣{gmd@detect@\gmd@adef@currdef}% we store the CS to allow its temporary discarding later.
}
```

```
\def\gmd@adef@setkeysdefault{%
  \setkeys[gmd]{adef}{star,prefix,KVpref}}
```

Note we don't set `KVfam`. We do not so because for `\define@key`-likes family is a mandatory argument and for `\DeclareOptionX` the default family is set to the input file name in line 4719.

star
```
\define@boolkey[gmd]{adef}{star}[true]{}
```

The `prefix@`⟨*command*⟩ key-value will be used to create additional index entry for detected definiendum (a **definiendum** is the thing defined, e.g. in `\newenvironment{foo}` the env. `foo`). For instance, `\newcounter` is declared with `[prefix=\bslash␣c@]` in line 4984 and therefore `\newcounter{foo}` occurring in the code will index both `foo` and `\c@foo` (as definition entries).

prefix
```
\define@key[gmd]{adef}{prefix}[]{%
  \edef\gmd@resa{%
    \def\@xanxcs{gmd@adef@prefix@\gmd@adef@currdef␣}{%
      #1}}%
  \gmd@resa}
```

`\def\gmd@KVprefdefault{KV}%` in a separate macro because we'll need it in `\ifx`.

A macro `\gmd@adef@KVprefixset@`⟨*command*⟩ if defined, will falsify an `\ifnum` test that will decide whether create additional index entry together with the tests for `prefix`⟨*command*⟩ and

KVpref
```
\define@key[gmd]{adef}{KVpref}[\gmd@KVprefdefault]{%
  \edef\gmd@resa{#1}%
  \ifx\gmd@resa\gmd@KVprefdefault
  \else
    \@namedef{gmd@adef@KVprefixset@\gmd@adef@currdef}{1}%
    \gmd@adef@setKV% whenever the KVpreffix is set (not default), the declared
                   command is assumed to be keyvalish.
  \fi
  \edef\gmd@resa{#1}% because \gmd@adef@setKV redefined it.
  \edef\gmd@resa{%
    \def\@xanxcs{gmd@adef@KVpref@\gmd@adef@currdef}{%
      \ifx\gmd@resa\empty
      \else#1@\fi}}% as in xkeyval, if the KV prefix is not empty, we add @ to it.
  \gmd@resa}
```

Analogously to `KVpref`, `KVfam` declared in `\DeclareDefining` will override the family scanned from the code and, in `\DeclareOptionX` case, the default family which is the input file name (only for the command being declared).

KVfam
```
\define@key[gmd]{adef}{KVfam}[]{%
  \edef\gmd@resa{#1}%
  \@namedef{gmd@adef@KVfamset@\gmd@adef@currdef}{1}%
  \edef\gmd@resa{%
    \def\@xanxcs{gmd@adef@KVfam@\gmd@adef@currdef}{%
      \ifx\gmd@resa\empty
      \else#1@\fi}}%
  \gmd@resa
  \gmd@adef@setKV}% whenever the KVfamily is set, the declared command is
                 assumed to be keyvalish.
```

type
```
\define@choicekey[gmd]{adef}{type}
  [\gmd@adef@typevals\gmd@adef@typenr]
  {% the list of possible types of defining commands
    def,
    newcommand,
    cs,% equivalent to the two above, covers all the cases of defining a CS, including the Plain TeX \new›... and LaTeX \newlength.
    newenvironment,
    text,% equivalent to the one above, covers all the commands defining its first mandatory argument that should be text, \DeclareOption e.g.
    define@key,% special case of more arguments important; covers the xkeyval defining commands.
    dk,% a shorthand for the one above.
    DeclareOptionX,% another case of special arguments configuration, covers the xkeyval homonym.
    dox,% a shorthand for the one above.
    kvo% one of option defining commands of the kvoptions package by Heiko Oberdiek (a package available o CTAN in the oberdiek bundle).
  }
  {% In fact we collapse all the types just to four so far:
    \ifcase\gmd@adef@typenr% if def
```

```
      \gmd@adef@settype{cs}{0}%
    \or% when newcommand
      \gmd@adef@settype{cs}{0}%
    \or% when cs
      \gmd@adef@settype{cs}{0}%
    \or% when newenvironment
      \gmd@adef@settype{text}{0}%
    \or% when text
      \gmd@adef@settype{text}{0}%
    \or% when define@key
      \gmd@adef@settype{dk}{1}%
    \or% when dk
      \gmd@adef@settype{dk}{1}%
    \or% when DeclareOptionX
      \gmd@adef@settype{dox}{1}%
    \or% when dox
      \gmd@adef@settype{dox}{1}%
    \or% when kvo
      \gmd@adef@settype{text}{1}% The kvoptions option definitions take
                                first mandatory argument as the option name and they define a keyval
                                key whose macro's name begins with the prefix/family, either default
                                or explicitly declared. The kvoptions prefix/family is supported in gm-
                                doc with [KVpref=,␣KVfam=⟨family⟩].
    \fi}
```

```
\def\gmd@adef@settype#1#2{%
  \def\gmd@adef@TYPE{#1}%
  \ifnum1=#2␣% now we define (or not) a quasi-switch that fires for the keyvalish
       definition commands.
    \gmd@adef@setKV
  \fi}
```

```
\def\gmd@adef@setKV{%
  \edef\gmd@resa{%
    \def\@xanxcs{gmd@adef@KV@\gmd@adef@currdef}{1}%
  }%
  \gmd@resa}
```

We initialise the carrier of detectors:

```
\emptify\gmd@detectors
```

The definiendum of a command of the `cs` type is the next control sequence. Therefore we only need a self-relaxing hook in `\finish@macroscan`.

`\ifgmd@adef@cshook`

```
\newif\ifgmd@adef@cshook
```

```
\def\gmd@adef@cs{\global\gmd@adef@cshooktrue\gmd@charbychar}
```

For other kinds of definitions we'll employ active chars of their arguments' opening braces, brackets and sergeants. In gmdoc code layer scopes the left brace is active so we only add a hook to its meaning (see line **??** in gmverb) and here we switch it according to the type of detected definition.

```
\def\gmd@adef@text{\gdef\gmd@lbracecase{1}\gmd@charbychar}
```

```
\foone{%
  \catcode`\[\active
```

```
  \catcode`\<\active}
{%
```

The detector of xkeyval `\define@[…]key`:

```
  \def\gmd@adef@dk{%
    \let[\gmd@adef@scanKVpref
    \catcode`\[\active
    \gdef\gmd@lbracecase{2}%
    \gmd@adef@dfKVpref\gmd@KVprefdefault% We set the default value of
        the xkeyval prefix. Each time again because an assignment
        in \gmd@adef@dfKVpref is global.
    \gmd@adef@checklbracket}
```

The detector of xkeyval `\DeclareOptionX`:

```
  \def\gmd@adef@dox{%
    \let[\gmd@adef@scanKVpref
    \let<\gmd@adef@scanDOXfam
    \catcode`[\active
    \catcode`<\active
    \gdef\gmd@lbracecase{1}%
    \gmd@adef@dfKVpref\gmd@KVprefdefault% We set the default values of
        the xkeyval prefix…
    \edef\gmd@adef@fam{\gmd@inputname}% … and family.
    \gmd@adef@dofam
    \gmd@adef@checkDOXopts}%
}
```

The case when the right bracket is next to us is special because it is already touched by `\futurelet` (of CSes scanning macro's `\@ifnextcat`), therefore we need a 'future' test.

```
\def\gmd@adef@checklbracket{%
  \@ifnextchar[%
  \gmd@adef@scanKVpref\gmd@charbychar}% note that the prefix scanning
      macro gobbles its first argument (undelimited) which in this case is [.
```

After a `\DeclareOptionX`-like defining command not only the prefix in square brackets may occur but also the family in sergeants. Therefore we have to test presence of both of them.

```
\def\gmd@adef@checkDOXopts{%
  \@ifnextchar[\gmd@adef@scanKVpref%
  {\@ifnextchar<\gmd@adef@scanDOXfam\gmd@charbychar}}

\def\gmd@adef@scanKVpref#1#2]{%
  \gmd@adef@dfKVpref{#2}%
  [#2]\gmd@charbychar}

\def\gmd@adef@dfKVpref#1{%
  \ifnum1=0\csname␣gmd@adef@KVprefixset@\gmd@adef@currdef%
      \endcsname
    \relax
  \else
    \edef\gmu@resa{%
    \gdef\@xa\@nx
    \csname␣gmd@adef@KVpref@\gmd@adef@currdef\endcsname{%
```

```
      \ifx\relax#1\relax
      \else#1@%
      \fi}}%
    \gmu@resa
  \fi}

\def\gmd@adef@scanDOXfam{%
  \ifnum12=\catcode`\>\relax
    \let\next\gmd@adef@scanfamoth
  \else
    \ifnum13=\catcode`\>\relax
      \let\next\gmd@adef@scanfamact
    \else
      \PackageError{gmdoc}{>␣neither␣`other'␣nor␣`active'!␣
          Make␣it
        `other'␣with␣\bslash␣AddtoPrivateOthers\bslash\>.}{}%
    \fi
  \fi
  \next}

\def\gmd@adef@scanfamoth#1>{%
    \edef\gmd@adef@fam{\@gobble#1}% there is always \gmd@charbychar
         first.
    \gmd@adef@dofam
    <\gmd@adef@fam>%
  \gmd@charbychar}

\foone{\catcode`\>\active}
  {\def\gmd@adef@scanfamact#1>{%
      \edef\gmd@adef@fam{\@gobble#1}% there is always \gmd@charbychar
           first.
      \gmd@adef@dofam
      <\gmd@adef@fam>%
      \gmd@charbychar}%
  }
```

The hook of the left brace consists of `\ifcase` that logically consists of three subcases:

0 —the default: do nothing in particular;

1 —the detected defining command has one mandatory argument (is of the `text` type, including kvoptions option definition);

2–3 —we are after detection of a `\define@key`-like command so we have to scan *two* mandatory arguments (case 2 is for the family, case 3 for the key name).

```
\def\gm@lbracehook{%
  \ifcase\gmd@lbracecase\relax
  \or% when 1
    \afterfi{%
      \gdef\gmd@lbracecase{0}%
      \gmd@adef@scanname}%
  \or% when 2—the first mandatory argument of two (\define@[...]key)
    \afterfi{%
    \gdef\gmd@lbracecase{3}%
     \gmd@adef@scanDKfam}%
  \or% when 3—the second mandatory argument of two (the key name).
```

```
    \afterfi{%
      \gdef\gmd@lbracecase{0}%
      \gmd@adef@scanname}%
  \fi}
\def\gmd@lbracecase{0}% we initialise the hook caser.
```

And we define the inner left brace macros:

```
\foone{\catcode`\[1␣\catcode`\]2␣\catcode`\}12␣}
  [% Note that till line 4849 the square brackets are grouping and the right brace is
      'other'.
```

Define the macro that reads and processes the `\define@key` family argument. It has the parameter delimited with 'other' right brace. An active left brace that has launched this macro had been passed through iterating `\gmd@charbychar` that now stands next right to us.

```
  \def\gmd@adef@scanDKfam#1}[%
    \edef\gmd@adef@fam[\@gobble#1]% there is always \gmd@charbychar
        first.
    \gmd@adef@dofam
    \gmd@adef@fam}%
  \gmd@charbychar]

  \def\gmd@adef@scanname#1}[%
    \@makeother\[%
    \@makeother\<%
```

The scanned name begins with `\gmd@charbychar`, we have to be careful.

```
    \gmd@adef@deftext[#1]%
    \@gobble#1}%
    \gmd@charbychar]
  ]

\def\gmd@adef@dofam{%
  \ifnum1=0\csname␣gmd@adef@KVfamset@\gmd@adef@currdef%
      \endcsname
    \relax% a family declared with \DeclareDefining overrides the one cur-
        rently scanned.
  \else
    \edef\gmu@resa{%
      \gdef\@xa\@nx
      \csname␣gmd@adef@KVfam@\gmd@adef@currdef\endcsname
      {\ifx\gmd@adef@fam\empty
        \else\gmd@adef@fam␣@%
        \fi}}%
    \gmu@resa
  \fi}

\def\gmd@adef@deftext#1{%
  \@xa\def\@xa\macro@pname\@xa{\@gobble#1}% we gobble \gmd@charbychar,
      cf. above.
  \edef\macro@pname{\@xa\detokenize\@xa{\macro@pname}␣}% note the
      space at the end.
  \edef\macro@pname{\@xa\@xiispaces\macro@pname\@nil}%
  \@xa\Text@Marginize\@xa{\macro@pname}%
```

```
\gmd@adef@indextext
\edef\gmd@adef@altindex{%
  \csname␣gmd@adef@prefix@\gmd@adef@currdef␣\endcsname}%
```

and we add the xkeyval header if we are in xkeyval definition.

```
\ifnum1=0\csname␣gmd@adef@KV@\gmd@adef@currdef␣\endcsname%
    \relax% The
    CS \gmd@adef@KV@⟨def. command⟩ is defined {1} (so \ifnum gets
    1=01\relax—true) iff ⟨def. command⟩ is a keyval definition. In
    that case we check for the KVprefix and KVfamily. (Otherwise
    \gmd@adef@KV@⟨def. command⟩ is undefined so \ifnum gets
    1=0\relax—false.)
  \edef\gmd@adef@altindex{%
    \gmd@adef@altindex
    \csname␣gmd@adef@KVpref@\gmd@adef@currdef␣\endcsname}%
  \edef\gmd@adef@altindex{%
    \gmd@adef@altindex
    \csname␣gmd@adef@KVfam@\gmd@adef@currdef␣\endcsname}%
\fi
\ifx\gmd@adef@altindex\empty
\else% we make another index entry of the definiendum with prefix/KVheader.
  \edef\macro@pname{\gmd@adef@altindex\macro@pname}%
  \gmd@adef@indextext
\fi}

\def\gmd@adef@indextext{%
  \@xa\@defentryze\@xa{\macro@pname}{0}% declare the definiendum has
      to have a definition entry and should appear without backslash in the
      changes history.
  \gmd@doindexingtext% redefine \do to an indexing macro.
  \@xa\do\@xa{\macro@pname}}
```

So we have implemented automatic detection of definitions. Let's now introduce some.

Default defining commands

Some commands are easy to declare as defining:

```
\DeclareDefining[star=false]\def
```
\pdef
```
\DeclareDefining[star=false]\pdef% it's a gmutils' shorthand for \protected
    % \def.
```
\provide
```
\DeclareDefining[star=false]\provide% a gmutils' conditional \def.
```
\pprovide
```
\DeclareDefining[star=false]\pprovide% a gmutils' conditional \pdef.
```

But \def definitely *not always* defines an important macro. Sometimes it's just a scratch assignment. Therefore we define the next declaration. It turns the next occurrence of \def off (only the next one).

```
\def\UnDef{{%
    \gmd@adef@selfrestore\def
}}

\def\UnPdef{{\gmd@adef@selfrestore\pdef}}

\Store@Macro\UnDef% because the 'hiding' commands relax it.

\def\HideDef{%
```

```
    \gmu@ifstar\UnDef{\HideDefining\def\relaxen\UnDef}}

\def\ResumeDef{%
  \ResumeDefining\def
  \Restore@Macro\UnDef}
```

Note that I *don't* declare `\gdef`, `\edef` neither `\xdef`. In my opinion their use as 'real' definition is very rare and then you may use `\Define` implemented later.

```
\DeclareDefining[star=false]\newcount
\DeclareDefining[star=false]\newdimen
\DeclareDefining[star=false]\newskip
\DeclareDefining[star=false]\newif
\DeclareDefining[star=false]\newtoks
\DeclareDefining[star=false]\newbox
\DeclareDefining[star=false]\newread
\DeclareDefining[star=false]\newwrite
\DeclareDefining[star=false]\newlength
\DeclareDefining[star=false]\DeclareDocumentCommand
\DeclareDefining[star=false]\DeclareCommand

\DeclareDefining\newcommand
\DeclareDefining\renewcommand
\DeclareDefining\providecommand
\DeclareDefining\DeclareRobustCommand
\DeclareDefining\DeclareTextCommand
\DeclareDefining\DeclareTextCommandDefault

\DeclareDefining*\newenvironment
\DeclareDefining*\renewenvironment
\DeclareDefining*[star=false]\DeclareOption

%\DeclareDefining*\@namedef

\DeclareDefining*[prefix=\bslash␣c@]\newcounter%
```

this prefix provides indexing also `\c@`⟨*counter*⟩.

```
\DeclareDefining[type=dk,␣prefix=\bslash]\define@key
\DeclareDefining[type=dk,␣prefix=\bslash␣if]\define@boolkey%
```

the alternate index entry will be `\if`⟨*KVpref*⟩`@`⟨*KVfam*⟩`@`⟨*key name*⟩

```
\DeclareDefining[type=dk,␣prefix=\bslash]\define@choicekey

\DeclareDefining[type=dox,␣prefix=\bslash]\DeclareOptionX%
```

the alternate index entry will be ⟨*KVpref*⟩`@`⟨*KVfam*⟩`@`⟨*option name*⟩.

For `\DeclareOptionX` the default KVfamily is the input file name. If the source file name differs from the name of the goal file (you TEX a .dtx not .sty e.g.), there is the next declaration. It takes one optional and one mandatory argument. The optional is the `KVpref`, the mandatory the `KVfam`.

```
\newcommand*\DeclareDOXHead[2][\gmd@KVprefdefault]{%
  \csname␣DeclareDefining\endcsname
  [type=dox,␣prefix=\bslash,␣KVpref=#1,␣KVfam=#2]%
  \DeclareOptionX
}
```

An example:

```
\DeclareOptionX[Berg]<Lulu>{EvelynLear}{}
```

Check in the index for `EvelynLear` and `\Berg@Lulu@EvelynLear`. Now we set in the comment layer `\DeclareDOXHead[Webern]{Lieder}` and

ChneOelze
```
\DeclareOptionX<AntonW>{ChneOelze}
```

The latter example shows also overriding the option header by declaring the default. By the way, both the example options are not declared in the code actually.

Now the Heiko Oberdiek's kvoptions package option definitions:

\DeclareStringOption
\DeclareBoolOption
areComplementaryOption
\DeclareVoidOption
```
\DeclareDefining[type=kvo,␣prefix=\bslash,␣KVpref=]%
     \DeclareStringOption
\DeclareDefining[type=kvo,␣prefix=\bslash,␣KVpref=]%
     \DeclareBoolOption
\DeclareDefining[type=kvo,␣prefix=\bslash,␣KVpref=]%
     \DeclareComplementaryOption
\DeclareDefining[type=kvo,␣prefix=\bslash,␣KVpref=]%
     \DeclareVoidOption
```

The kvoptions option definitions allow setting the default family/prefix for all definitions forth so let's provide analogon:

```
\def\DeclareKVOFam#1{%
  \def\do##1{%
    \csname␣DeclareDefining\endcsname
    [type=kvo,␣prefix=\bslash,␣KVpref=,␣KVfam=#1]##1}%
  \do\DeclareStringOption
  \do\DeclareBoolOption
  \do\DeclareComplementaryOption
  \do\DeclareVoidOption
}
```

As a nice exercise I recommend to think why this list of declarations had to be preceded (in the comment layer) with `\HideAllDefining` and for which declarations of the above `\DeclareDefining\DeclareDefining` did not work. (The answers are commented out in the source file.)

One remark more: if you define (in the code) a new defining command (I did: a shorthand for `\DeclareOptionX[gmcc]<>`), declare it as defining (in the commentary) *after* it is defined. Otherwise its first occurrence shall fire the detector and mark next CS or worse, shall make the detector expect some arguments that it won't find.

Suspending ('hiding') and resuming detection

Sometimes we want to suspend automatic detection of definitions. For `\def` we defined suspending and resuming declarations in the previous section. Now let's take care of detection more generally.

The next command has no arguments and suspends entire detection of definitions.

```
\def\HideAllDefining{%
  \ifnumo=o\csname␣gmd@adef@allstored\endcsname
    \SMglobal\Store@Macro\gmd@detectors
    \global\@namedef{gmd@adef@allstored}{1}%
  \fi
  \global\emptify\gmd@detectors}% we make the carrier \empty not \re¦
       lax to be able to declare new defining command in the scope of \Hide¦
       All...
```

The \ResumeAllDefining command takes no arguments and restores the meaning of the detectors' carrier stored with \HideAllDefining

```
\def\ResumeAllDefining{%
  \ifnum1=0\csname␣gmd@adef@allstored\endcsname\relax
    \SMglobal\Restore@Macro\gmd@detectors
    \SMglobal\Restore@Macro\UnDef
    \global\@namedef{gmd@adef@allstored}{0}%
  \fi}
```

Note that \ResumeAllDefining discards the effect of any \DeclareDefining that could have occurred between \HideAllDefining and itself.

The \HideDefining command takes one argument which should be a defining command (always without star). \HideDefining suspends detection of this command (also of its starred version) until \ResumeDefining of the same command or \ResumeAllDefining.

```
\def\HideDefining{\begingroup
  \MakePrivateLetters
  \gmu@ifstar\Hide@DfngOnce\Hide@Dfng}

\def\Hide@Dfng#1{%
  \escapechar\m@ne
  \gn@melet{gmd@detect@\string#1}{relax}%
  \gn@melet{gmd@detect@\string#1*}{relax}%
  \ifx\def#1\global\relaxen\UnDef\fi
  \endgroup}

\def\Hide@DfngOnce#1{%
  \gmd@adef@selfrestore#1%
  \endgroup}

\def\gmd@adef@selfrestore#1{%
  \@ifundefined{gmd@detect@\strip@bslash{#1}}{%
    \SMglobal\@xa\Store@Macro
    \csname␣gmd@detect@\strip@bslash{#1}\endcsname}{}%
  \global\@nameedef{gmd@detect@\strip@bslash{#1}}{%
    \@nx\gmu@if␣x%
    {\@xanxcs{gmd@detectname@\strip@bslash{#1}}%
      \@nx\macro@pname}% we compare the detect(ed) name with \macro@pname.
    {\def\@nx\next{% this \next will be executed in line 4423.
        \SMglobal\Restore@Macro␣% they both are \protected.
        \@xanxcs{gmd@detect@\string#1}%
        \@nx\gmd@charbychar}%
      \@nx}%
    {}% or do nothing if the CS' names are unequal.
  }% of \@nameedef.
}% of \gmd@adef@selfrestore.
```

The \ResumeDefining command takes a defining command as the argument and resumes its automatic detection. Note that it restores also the possibly undefined detectors of starred version of the argument but that is harmless I suppose until we have millions of CSes.

```
\def\ResumeDefining{\begingroup
  \MakePrivateLetters
```

```
  \gmd@ResumeDfng}

\def\gmd@ResumeDfng#1{%
  \escapechar\m@ne
  \SMglobal\Restore@MacroSt{gmd@detect@\string#1}%
  \SMglobal\Restore@MacroSt{gmd@detect@\string#1*}%
  \endgroup}
```

Indexing of CSes

The inner macro indexing macro. `#1` is the `\verb`'s delimiter; `#2` is assumed to be the macro's name with MakeIndex-control chars quoted. `#3` is a macro storing the $_{12}$ macro's name, usually `\macro@pname`, built with `\string`ing every char in lines 4254, 4274 and 4286. `#3` is used only to test if the entry should be specially formatted.

`\index@macro`

```
\newcommand*\index@macro[3][\verbatimchar]{{%
    \gmu@ifundefined{gmd/iexcl/\@xa\detokenize\@xa{#3␣}}%
    {% #3 is not excluded from index
      \gmu@ifundefined{gmd/defentry/\@xa\detokenize\@xa{#3␣}}%
      {% #3 is not def entry
        \gmu@ifundefined{gmd/usgentry/\@xa\detokenize\@xa{#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?
            Definitely here because we'll wish to switch the switch with a declaration.
      \if@pageinclindex
        \edef\gmu@tempa{gmdindexpagecs{\HLPrefix}{%
            \kind@fentry}{\EntryPrefix}}%
      \else
        \edef\gmu@tempa{gmdindexrefcs{\HLPrefix}{%
            \kind@fentry}{\EntryPrefix}}%
      \fi
      \edef\gmu@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\gmu@tempa}%
      \@xa\special@index\@xa{\gmu@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.
    }}

\def\un@defentryze#1{%
```

```
    \ifcsname␣gmd/defentry/\@xa\detokenize\@xa{#1␣}\endcsname
    \@xa\g@relaxen\csname␣gmd/defentry/\@xa\detokenize\@xa{#1␣}%
        \endcsname
    \fi
    \ifx\gmd@detectors\empty
      \g@relaxen\last@defmark
    \fi}%
```
the last macro (assuming `\fi` is not a macro :-) is only used by `\changes`. If we are in the scope of automatic detection of definitions, we want to be able not to use `\Define` but write `\changes` after a definition and get proper entry. Note that in case of automatic detection of definitions `\last@defmark`'s value keeps until the next definition.

```
\def\un@usgentryze#1{%
    \ifcsname␣gmd/usgentry/\@xa\detokenize\@xa{#1␣}\endcsname
    \@xa\g@relaxen\csname␣gmd/usgentry/\@xa\detokenize\@xa{#1␣}%
        \endcsname
    \fi}
```

```
\@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:

`\if@pageinclindex`
```
\newif\if@pageinclindex
```

`\quoted@eschar`
```
\newcommand*\quoted@eschar{\quotechar\bslash}%
```
we'll redefine it when indexing an environment.

Let's initialise `\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\gmu@tempa{#1}%
      \ifx\gmu@tempa\@empty␣p.\,\fi}
```

`\HLPrefix`
```
    \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 because during the automatic \hypertargeting the lines are labelled clnum.⟨*number*⟩. When \HLPrefix was defined as dot, MakeIndex rejected the entries as 'illegal page number'.

```
5294    \fi}
```

The definition is postponed till \begin{document} because of the \PageIndex declaration (added for doc-compatibility), see line 8599.

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 line number 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.

```
5309 \def\DefEntry#1{\underline{#1}}
5310 \def\UsgEntry#1{\textit{#1}}
```

The third option will be just \relax by default:

```
5312 \def\CommonEntryCmd{relax}
```

In line 5185 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 driver part to be 'usage', see the source of chapter 640.

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 5181 and 5183 respectively.

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

```
5328 \outer\def\DefIndex{\begingroup
5329   \MakePrivateLetters
5330   \gmu@ifstar
5331   {\@sanitize\MakePrivateOthers%
5332     \Code@DefIndexStar}%
5333   {\Code@DefIndex}}

5338 \long\def\Code@DefIndex#1{\endgroup{%
5339     \escapechar\m@ne% because we will compare the macro's name with a string
                          without the backslash.
5341     \@defentryze{#1}{1}}}

5345 \long\def\Code@DefIndexStar#1{%
5346   \endgroup{%
5347   \addto@estoindex{#1}%
5348   \@defentryze{#1}{0}}%
5349 }

5351 \def\gmd@justadot{.}

5353 \long\def\@defentryze#1#2{%
5354   \@xa\glet\csname␣gmd/defentry/\detokenize{#1␣}\endcsname%
          \gmd@justadot% The
```

LaTeX `\@namedef` macro could not be used since it's not 'long'. The space to sound with the checker.

```
\ifcat\relax\@xa\@nx\@firstofmany#1\@nil
```

if we meet a CS, then maybe it's a CS to be 'defentryzed' or maybe it's a 'verbatim special' CS. The only way to distinguish those cases is to assume there shouldn't be a verbatim containing only a 'verbatim special' CS.

```
    \@xa\def\@xa\gmu@tempa\@xa{\@allbutfirstof#1\@nil}%
    \ifx\gmu@tempa\@empty
      \afterfifi\@firstoftwo%
```

if `#1` is a single CS, we `\xiistring` it. Otherwise we `\detokenize` it.

```
    \else\afterfifi\@secondoftwo
    \fi
  \else\@xa\@secondoftwo
  \fi
\last@defmark {\xdef\last@defmark{\xiistring#1}}%
```

we `\string` the argument just in case it's a control sequence. But when it can be a CS, we `\@defentryze` in a scope of `\escapechar=-1`, so there will never be a backslash at the beginning of `\last@defmark`'s meaning (unless we `\@defentryze \\`).

```
  {\xdef\last@defmark{\detokenize{#1}}}%
  \@xa\gdef\csname␣gmd/isaCS/\last@defmark\endcsname{#2}%
```

`#2` is either 0 or 1. It is the information whether this entry is a CS or not.

```
}% of \@defentryze.

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

Initialise `\envirs@toindex`

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

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

```
\outer\def\CodeUsgIndex{\begingroup
  \MakePrivateLetters
  \gmu@ifstar
  {\@sanitize\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:

```
\outer\def\CodeCommonIndex{\begingroup
```

```
  \MakePrivateLetters
  \gmu@ifstar
  {\MakePrivateOthers\@sanitize\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.

```
\long\def\text@indexmacro#1{%
  {\escapechar\m@ne␣\xdef\macro@pname{\xiistring#1}}%
  \@xa\quote@mname\macro@pname\relax% we process the CS's name char by
        char and quote MakeIndex controls. \relax is the iterating macro's 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 5858.
    \@xa␣\index@macro\im@firstpar\macro@iname\macro@pname}}
```

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

```
\def\quote@mname{%
  \def\macro@iname{}%
  \quote@charbychar}

\def\quote@charbychar#1{%
  \ifx\relax#1% finish quoting when you meet \relax or:
  \else
    \ifnumo\ifcat\@nx#1\@nx~1\fi\ifcat\@nx#1\relax1\fi>0␣% we can
          meet active char and/or control sequences (made by) verbatim specials,
          therefore we check whether #1 is an active char and if it is a CS.
      \afterfifi{% we can meet an active char or a CS iff we use verbatim spe-
            cials.
        \ifdefined\verbatim@specials@list
        \afterfi{%
          \begingroup
          \escapechar\@xa\@xa\@xa`\@xa\@firstofmany%
                \verbatim@specials@list\@nil
          \@xa\endgroup
          \@xa\quote@charbychar\detokenize{#1}% for a CS \detokenize
                adds a space but if so, it will be ignored by the argument scanner.
        }% of \afterfi.
        \else\PackageError{gmdoc}{Please␣report␣a\space␣bug␣in
          \bslash␣quote@charbychar␣in␣line␣4934}{}%
        \fi% of \ifdefined\verbatim@specials@list.
      }% of \afterfifi.
    \else
      \quote@char#1%
```

```
      \xdef\macro@iname{\macro@iname␣\gmd@maybequote#1}%
      \afterfifi\quote@charbychar
    \fi
  \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.

```
\def\TextUsgIndex{\begingroup
  \MakePrivateLetters
  \gmu@ifstar{\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}}

\long\def\text@indexenvir#1{%
  {\verbatim@specials
  \edef\macro@pname{\xiistring#1}%
  \if\bslash\@xa\@firstofmany\macro@pname\@nil% if \stringed #1 begins with a backslash, we will gobble it to make MakeIndex not see it.
    \edef\gmu@tempa{\@xa\@gobble\macro@pname}%
    \@tempswatrue
  \else
    \let\gmu@tempa\macro@pname
    \@tempswafalse
  \fi
  \@xa\quote@mname\gmu@tempa\relax% we process \stringed #1 char by char and quote MakeIndex controls. \relax is the iterating macro's stopper. The quoted \stringed #1 shall be the meaning 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 5858.
    \index@macro\macro@iname\macro@pname}}

\def\TextCommonIndex{\begingroup
  \MakePrivateLetters
  \gmu@ifstar{\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 5192 and 5188, the markers of special formatting are reset after first use.

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

```
\outer\def\CodeMarginize{\begingroup
  \MakePrivateLetters
  \gmu@ifstar
    {\MakePrivateOthers\egCode@MarginizeEnvir}
    {\egCode@MarginizeMacro}}
```

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

```
\long\def\egCode@MarginizeMacro#1{\endgroup
  \Code@MarginizeMacro#1}

\long\def\Code@MarginizeMacro#1{% #1 is always a CS.
    \escapechar\m@ne
    \@xa\glet\csname␣gmd/2marpar/\xiistring#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 triggered at the beginning of the nearest codeline.

Here it is:

```
\def\mark@envir{%
  \ifx\envirs@tomarginpar\@empty
  \else
    \def\do{\Text@Marginize*}%
    \envirs@tomarginpar%
    \g@emptify\envirs@tomarginpar%
  \fi
  \ifx\envirs@toindex\@empty
  \else
    {\verbatim@specials
      \gmd@doindexingtext
      \envirs@toindex
      \g@emptify\envirs@toindex}%
  \fi}

\def\gmd@doindexingtext{%
  \def\do##1{% the \envirs@toindex list contains \stringed macros or environments' names in braces and each preceded with \do. We extract the definition because we use it also in line 4900.
    \if\bslash\@firstofmany##1\@nil% if ##1 begins with a backslash, we will gobble it for MakeIndex not see it.
    \edef\gmd@resa{\@gobble##1}%
    \@tempswatrue
    \else
    \edef\gmd@resa{##1}\@tempswafalse
    \fi
    \@xa\quote@mname\gmd@resa\relax% see line 5508 & subs. for commentary.
    {\if@tempswa
```

```
5600          \def\quoted@eschar{\quotechar\bslash}%
5601          \else\@emptify\quoted@eschar
5602        \fi
5603        \index@macro\macro@iname{##1}}}%
5604 }
```

One very important thing: initialisation of the list macros:

```
5608 \@emptify\envirs@tomarginpar
5609 \@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.

```
5616 \outer\def\Define{% note that since it's \outer, it doesn't have to be \pro¦
         tected.
5618   \begingroup
5619   \MakePrivateLetters
```

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

```
5624   \gmu@ifstar{\@sanitize%
5625     \Code@DefEnvir}{\Code@DefMacro}}

5627 \outer\def\CodeUsage{\begingroup
5628   \MakePrivateLetters
5629   \gmu@ifstar{%
5630     \@sanitize%
5631     \MakePrivateOthers
5632     \Code@UsgEnvir}{\Code@UsgMacro}}
```

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

`\Code@DefMacro`

```
5635 \DeclareCommand\Code@DefMacro\long{om}{%
5636   \Code@DefIndex#2% we use the internal macro; it'll close the group.
5637   \IfValueTF{#1}%
5638   {\Code@MarginizeMacro#1}%
5639   {\Code@MarginizeMacro#2}%
5640 }
```

`\Code@UsgMacro`

```
5644 \DeclareCommand\Code@UsgMacro\long{om}{%
5645   \Code@UsgIndex#2% here also the internal macro; it'll close the group
5647   \IfValueTF{#1}%
5648   {\Code@MarginizeMacro#1}%
5649   {\Code@MarginizeMacro#2}%
5650 }
```

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

```
5655 \def\codeline@wrindex#1{\if@filesw
5656   \immediate\write\@indexfile
5657   {\string\indexentry{#1}%
5658     {\HLPrefix\number\c@codelinenum}}\fi}

5662 \def\codeline@glossary#1{% It doesn't need to establish a group since it is al-
                                ways called in a group.
```

```
  \if@pageinclindex
    \edef\gmu@tempa{gmdindexpagecs{\HLPrefix}{relax}{%
        \EntryPrefix}}%
  \else
    \edef\gmu@tempa{gmdindexrefcs{\HLPrefix}{relax}{%
        \EntryPrefix}}% relax stands for the formatting command. But
                       we don't want to do anything special with the change history entries.
  \fi
  \protected@edef\gmu@tempa{%
    \@nx\protected@write\@nx\@glossaryfile{}%
    {\string\glossaryentry{#1\encapchar\gmu@tempa}%
    {\HLPrefix\number\c@codelinenum}}}%
  \gmu@tempa
}
```

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

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

And in case we don't want to 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:

```
\Store@Macros{\index\codeline@wrindex}

\def\ungag@index{\Restore@Macros␣{\index\@@codeline@wrindex}}
```

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

\Code@DefEnvir
```
\DeclareCommand\Code@DefEnvir\long{om}{%
  \endgroup
  {%
    \IfValueTF{#1}%
    {\addto@estomarginpar{#1}}%
    {\addto@estomarginpar{#2}}%
    \addto@estoindex{#2}%
    \@defentryze{#2}{0}}}
```

\Code@UsgEnvir
```
\DeclareCommand\Code@UsgEnvir\long{om}{%
  \endgroup
```

```
  {%
    \IfValueTF{#1}%
    {\addto@estomarginpar{#1}}%
    {\addto@estomarginpar{#2}}%
    \addto@estoindex{#2}%
    \@usgentryze{#2}}}

\long\def\addto@estomarginpar#1{%
\gaddtomacro\envirs@tomarginpar{\do{#1}}}

\long\def\addto@estoindex#1{%
  \gaddtomacro\envirs@toindex{\do{#1}}}
```

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 CSes appearing explicitly and the latter for the strings and CSes appearing implicitly.

```
\def\TextUsage{\begingroup
  \MakePrivateLetters
  \gmu@ifstar{\@sanitize\MakePrivateOthers
    \Text@UsgEnvir}{\Text@UsgMacro}}
```

\Text@UsgMacro

```
\DeclareCommand\Text@UsgMacro\long{om}{%
  \endgroup
  \IfValueTF{#1}%
  {\Text@Marginize*{#1}{\scanverb*{#1}}}%
  {\Text@Marginize*{#2}{\scanverb*{#2}}}%
  \begingroup\Code@UsgIndex#2% we declare the kind of formatting of the entry.
  \text@indexmacro#2}
```

\Text@UsgEnvir

```
\DeclareCommand\Text@UsgEnvir\long{om}{%
  \endgroup
  \IfValueTF{#1}%
  {\Text@Marginize*{#1}{\scanverb*{#1}}}%
  {\Text@Marginize*{#2}{\scanverb*{#2}}}%
  \@usgentryze{#2}% we declare the 'usage' kind of formatting of the entry and index the sequence #1.
  \text@indexenvir{#2}}
```

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.

```
\pdef\TextMarginize{\@bsphack\begingroup
  \MakePrivateLetters
  \gmu@ifstar{%
    \MakePrivateOthers\egText@MarginizeEnv}{%
      \egText@MarginizeCS}}

\long\def\egText@MarginizeEnv#1{\endgroup
  \Text@Marginize*{#1}%
  \@esphack}

\long\def\egText@MarginizeCS#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.

```
  \long\def\gmdmarginpar#1{%
    \marginpar{\raggedleft\strut
      \hskipoptplus100optminus100opt%
      #1}}%
\else
  \long\def\gmdmarginpar#1{}%
\fi
\let\gmu@tempa\all@stars
\@xa\addtomacro\@xa\gmu@tempa\@xa{\all@unders}
\@xa\DeclareCommand\@xa\Text@Marginize\@xa!%
\@xa{\@xa␣Q\@xa{\gmu@tempa}m}{%
  \gmdmarginpar{%
    \addtomacro\verb@lasthook{\marginpartt}%
    \IfValueTF{#1}{\scanverb#1}{\scanverb}{#2}}%
}% of \Text@Marginize.
```

Note that the above command will just gobble its arguments 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\narrativett
```

If we print also the narration lines' numbers, then the index entries for CSes and environments marked in the commentary should have codeline numbers not page numbers and that is `\let` in line 5689. On the other hand, if we don't print narration lines' numbers, 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@printalllinenos\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
  \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"[10].

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

oldmc
```
\let\oldmc\macrocode
\let\endoldmc\endmacrocode
```
oldmc*
```
\n@melet{oldmc*}{macrocode*}
\n@melet{endoldmc*}{endmacrocode*}
```

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

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

```
\def\@oldmacrocode@launch{%
  \emptify\gmd@textEOL% to disable it in \gmd@docstripdirective launched
       within the code.
  \gmd@ctallsetup
  \glet\stored@code@delim\code@delim
  \@makeother\^^B\CodeDelim*\^^B%
  \ttverbatim␣\gmd@DoTeXCodeSpace
  \@makeother\|% because \ttverbatim doesn't do that.
  \MakePrivateLetters% see line 4200.
  \docstrips@percent␣\@makeother\>%
```

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

```
  \@xa\gmd@currenvxistar\@currenvir*\relax
```

10 Richard Strauss after Oscar Wilde, *Salome*.

```
  \@oldmacrocode}

\foone{\catcode`*11 }
{\def\gmu@xistar{*}}

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

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

```
\foone{%
\catcode`[=1 \catcode`]=2
\catcode`\{=\active \@makeother\}
\@makeother\^^B
\catcode`/=0 \catcode`\\=\active
\catcode`&=14 \catcode`*=11
\catcode`\%=\active \obeyspaces}& %
[& here the \foone's second pseudo-argument begins

/def/@oldmacrocode[&
/bgroup/let =/relax& to avoid writing /@nx four times.
/xdef/oldmc@def[&
/def/@nx/oldmc@end####1/@nx%    /@nx\end&
/@nx{/@currenvir}[&
####1^^B/@nx/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{%
  \def\gmu@tempa{\end{\@currenvir}}%
   \@xa\gmu@tempa\@xa\def\@xa\gmd@lastenvir\@xa{\@currenvir}%
  \@xa\CodeDelim\@xa*\stored@code@delim
  \gmd@mchook}% see line 8235

\def\OldMacrocodes{%
  \let\macrocode\oldmc
  \n@melet{macrocode*}{oldmc*}}
```

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

```
\foone{\catcode`\%\active}
{\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.
```

```
\typeout{@@@@@␣codecheckifds␣hash␣1:␣»\unexpanded{#1}«,␣2:␣»%
    \unexpanded{#2}«}%
\ifnum␣␣\if@dsdir␣1\else␣0\fi\ifgmd@dsVerb␣1\fi>\z@
  \afterfi{%
    \gmd@dsChecker{%
      \if\@nx<\@nx#2\afterfi\gmd@docstripdirective
      \else\afterfi{\xiipercent#1#2}%
      \fi}% of the checker's arg
    }% of \afterfi
\else\afterfi{\xiipercent#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.

macro
```
\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}%
```

endmacro

```
  {\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` occur `macro`s 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}
```

endmacro*

```
\@xa\let\csname␣endmacro*\endcsname\endmacro
```

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

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

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

```
\def\gmd@ifsingle#1#2\@nil{%
  \def\gmu@tempa{#2}%
  \ifx\gmu@tempa\@empty}
```

Note it expands to an open `\if...` test (unbalanced with `\fi`) so it has to be used as all the `\if`s, 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.

```
\def\gmd@ifonetoken#1#2#3{%
  \def\gmu@tempb{#3}% We hide #3 from TeX in case it's \if... or
       so. \gmu@tempa is used in \gmd@ifsingle.
  \gmd@ifsingle#3\@nil
    \afterfi{\@xa#1\gmu@tempb}%
  \else
    \edef\gmu@tempa{\@xa\string\gmu@tempb}%
    \afterfi{\@xa#2\@xa{\gmu@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.

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

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

\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.

```
\long\def\gmd@evpaddonce#1{%
  \global\advance\gmd@oncenum\@ne
  \@xa\long\@xa\edef%
    \csname␣gmd/evp/NeuroOncer\the\gmd@oncenum\endcsname{%
      \@nx\g@relaxen
      \csname␣gmd/evp/NeuroOncer\the\gmd@oncenum%
           \endcsname}% Why does it work despite it shouldn't? Because
           when the CS got with \csname ›...\endcsname is undefined, it's
           equivalent to \relax and therefore unexpandable. That's why it
           passes \edef and is able to be assigned.
  \@xa\addtomacro\csname␣gmd/evp/NeuroOncer\the\gmd@oncenum%
       \endcsname{#1}%
  \@xa\addto@hook\@xa\everypar\@xa{%
    \csname␣gmd/evp/NeuroOncer\the\gmd@oncenum\endcsname}%
}
```

`\gmd@oncenum`

```
\newcount\gmd@oncenum
```

`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 æ sthetic and psychological reasons.

```
\@xa\let\@xa\environment\csname␣macro*\endcsname
\@xa\let\@xa\endenvironment\csname␣endmacro*\endcsname
```

Index exclude list

We want some CSes 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 CSes. 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 to `\relax`).[11]

```
\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\gmu@tempa{\@nx#1}% My TEX Guru's trick to deal with \fi and such, i.e., to hide from TEX when it is processing a test's branch without expanding.
  \if\gmu@tempa.% a dot finishes expelling
  \else
    \if\gmu@tempa,% The list this macro is put before may contain commas and that's O.K., we just continue the work.
      \afterfifi\dont@index
    \else% what is else shall off the Index be expelled.
      {\escapechar\m@ne
        \xdef\gmu@tempa{\string#1␣}}% its to sound with \detokenizes in tests.
      \@xa\let%
      \csname␣gmd/iexcl/\gmu@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 5178 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 because I suppose one will document not LATEX source but normal packages.

```
\DoNotIndex\{ \DoNotIndex\}% the index entries of these two CSes 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.
\gdef\DefaultIndexExclusions{%
```

11 This idea comes from Marcin Woliński.

```
\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}%
\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 6203.
\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 because 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|provide)command` 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 CSes like \^^M, see sec. 668.
\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
        IMHO.
    \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 }%
    \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 CSes, someone may like it in general but he/she may need one or two expelled to be indexed back. So

```
\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, they may use the `\DefaultIndexExclusions` declaration themself. This declaration OCSR, but anyway let's provide the counterpart. It OCSR, too.

```
\def\UndoDefaultIndexExclusions{%
  \Store@Macro\DoNotIndex
  \let\DoNotIndex\DoIndex
  \DefaultIndexExclusions
  \Restore@Macro\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."

```
\long\def\IndexPrologue#1{\@bsphack\def\index@prologue{#1}%
  \@esphack}
\def\indexdiv{\@ifundefined{chapter}{\section*}{\chapter*}}
\@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 redefining 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}}
```

Now we can rollback the `\ampulexdef` made to `\verb`:

```
\AtDIPrologue{%
  \ampulexdef\verb\ttverbatim\verbatim@specials
  {\ttverbatim\verbatim@specials}}
```

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."

`\IndexMin`
```
\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.

`\c@IndexColumns`
```
\newcount\c@IndexColumns \c@IndexColumns = 3
```

`theindex`
```
\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 `\dot|fill`) between the item and the entries. But in gmdoc we define no such special delimiters, so we add an infinite 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 columns look better when they are `\raggedright` in my opinion.
```
   }}{}
```

```
\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

```
\foone{\@makeother\<\@makeother\>
  \glet\sgtleftxii=<}
{
  \def\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}}%
  \def\gmd@docstripinner#1>{%
    \endgroup
    \def\gmd@modulehashone{%
      \Module{#1}\space
      \@afternarrgfalse\@aftercodegtrue\@codeskipputgfalse}%
    \gmd@textEOL\gmd@modulehashone}
```

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.

```
}
\foone{\@makeother\<\@makeother\>
  \glet\sgtleftxii=<
  \catcode`\^^M=\active}%
{%
  \def\gmd@docstripverb<#1^^M{%
    \endgroup%
    \def\gmd@modulehashone{%
      \ModuleVerb{#1}\@afternarrgfalse\@aftercodegtrue%
      \@codeskipputgfalse}%
  %%   \global\gmd@dsVerbtrue% see below.
```

```
      \xdef\gmd@dsVerbDelim{\detokenize{#1}}%
      \gmd@textEOL\gmd@modulehashone^^M}%
}

\edef\gmd@dsVerbDelim{\detokenize{%
      #@$$$$#_()(_)*981-71092519-40^^A^^B}}
```

So far proper handling of the checks for the closing directive is too expensive to implement so we only provide a macro to be put in a line before the closing directive. The problem is of course with the verbatim commands that are very difficult to rescan (`\scantokens` doesn't do the job).

It's not necessary to put it right before the line with the closing directive. The only requirement is that the lines between this macro and the closing directive don't contain any recatcode'ing in the narration layer.

```
\pdef\dsVerbClose{%
  \global\gmd@dsVerbtrue}
```

(~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\ModuleVerbClose#1{{%
      \xiipercent
      \mod@math@codes$\mathsf{#1}$
      {\normalfont[\ds␣verbatim␣closing␣dir.]}}}

\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 omit 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 recode 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.

```
\def\gmd@DefineChanges{%
  \outer\long\def\changes{%
    \gmd@changes@init
    \changes@}}

\def\gmd@changes@init{%
  \@bsphack\begingroup\@sanitize
  \catcode`\\\z@␣\catcode`\␣10␣\MakePercentIgnore
\catcode`\^=7
  \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}%
\changes  \renewcommand\changes[4][]{%
  }}

\def\MakeGlossaryControls{%
  \edef\actualchar{\string=}\edef\quotechar{\string!}%
  \edef\levelchar{\string>}\edef\encapchar{\xiiclub}}% for the glossary the 'actual', 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@Chang|
      % esStartDate,
    \@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 \gmu@tempb scratch macro.
        \@tempswatrue
        \edef\gmu@tempb{% it's a bug fix: while typesetting traditional .dtxes,
            % \last@defmark came out with \ at the beginning (which resulted with \\⟨name⟩ in the change log) but while typesetting the 'new' way, it occurred without 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\gmu@tempb\last@defmark%
            \fi%
        \n@melet{gmd@glossCStest}{gmd/isaCS/\last@defmark}%
      \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, they should type \changes[generalname]...)
      \@tempswatrue
      {\escapechar\m@ne
        \xdef\gmu@tempb{\string#1}}%
```

```
      \if\bslash\@xa\@firstofmany\string#1\relax\@nil% we check
           whether #1 is a CS...
        \def\gmd@glossCStest{1}% ... and tell the glossary if so.
      \fi
    \fi
    \@ifundefined{gmd@glossCStest}{\def\gmd@glossCStest{0}}{}%
    \protected@edef\gmu@tempa{\@nx\gmd@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 7270.
        \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 precede it by \generalname.
          \gmu@tempb
          \actualchar\bslash␣verb*%
          \if\verbatimchar\gmu@tempb$\else\verbatimchar\fi
          \if1\gmd@glossCStest\quotechar\bslash\fi␣\gmu@tempb
          \if\verbatimchar\gmu@tempb$\else\verbatimchar\fi
        \else
          \space\actualchar\generalname
        \fi
        :\levelchar%
        #4%
      }}%
    \gmu@tempa
    \grelaxen\gmd@glossCStest
  \fi% of \if@recentchange
  \endgroup\@esphack}
```

Let's initialise `\last@defmark` and `\GeneralName`.

```
\@relaxen\last@defmark
\@emptify\GeneralName
```

```
\def\ChangesGeneral{\grelaxen\last@defmark}% If automatic detection of
     definitions is on, the default entry of \changes is the meaning of \last@defmark,
     the last detected definiendum that is. The declaration defined here serves to
     start a scope of 'general' \changes' entries.
```

```
\AtBegInput{\ChangesGeneral}
```

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
```

```
\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.
```

```
6812    #4=\numexpr#1*\@M+#2*100+#3\relax
```

(2010/06/23, changed:) from TEX's arithmetic to `\numexpr`

```
6815 }
```

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*⟩}.

```
6821 \def\ChangesStart#1#2{%
6824   \gmd@setChDate#2\@nil\c@ChangesStartDate
6825   \typeout{^^JPackage␣gmdoc␣info:␣^^JChanges'␣start␣date␣#1␣
           memorised
6826     as␣\string<\the\c@ChangesStartDate\string>␣\on@line.^^J}
6827   \advance\c@ChangesStartDate\m@ne% we shall show the changes at the spec-
           ified day and later.
6829   \ifnum\c@ChangesStartDate>19820900␣% [12] see below.
6833     \edef\gmu@tempa{%
6834       \@nx\g@addto@macro\@nx\glossary@prologue{%
6835         The␣changes
6836         \if\relax\GeneralName\relax\else␣of␣\GeneralName%
                \space\fi
6837         earlier␣than
6838         #1␣\if\relax#1\relax␣#2\else(#2)\fi\space␣are␣not␣
                shown.}}%
6839     \gmu@tempa
6840   \fi}
```

(Explanation to line 6829.) 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: they should only date the 'public' changes with the four digit year and the 'personal' ones with two digit year and set `\ChangesStart{}{1000/0/0}` or so.

In line 6829 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 macro name (i.e., that in the most recent `\begin{macro}` command [or `\Define`]). 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."

```
6878 \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`.

```
6890 \def\RecordChanges{\makeglossary\gmd@DefineChanges
```

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

```
\@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."

\GlossaryMin
```
\newdimen\GlossaryMin          \GlossaryMin       = 80pt
```

c@GlossaryColumns

\c@GlossaryColumns
```
\newcount\c@GlossaryColumns  \c@GlossaryColumns = 2
```

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

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

Here is the MakeIndex style definition:

```
⟨/ doc⟩
⟨gmglo⟩ preamble
⟨gmglo⟩ "\n␣\\begin{theglossary}␣\n
⟨gmglo⟩ \\makeatletter\n"
⟨gmglo⟩ postamble
⟨gmglo⟩ "\n\n␣\\end{theglossary}\n"
⟨gmglo⟩ keyword␣"\\glossaryentry"
⟨gmglo⟩ actual␣'='
⟨gmglo⟩ quote␣'!'
⟨gmglo⟩ level␣'>'
⟨*doc⟩
```

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."

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

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

```
\@ifundefined{glossary@prologue}
  {\def\glossary@prologue{\indexdiv{{Change␣History}}%
```

```
      \markboth{{Change␣History}}{{Change␣History}}%
    }}{}
```

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

```
\AtBeginDocument{%
  \@ifundefined{GlossaryParms}{\let\GlossaryParms%
      \IndexParms}{}}
```

\GlossaryParms

"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

```
\def\PrintChanges{% to avoid a disaster among queer EOLs:
  \@ifQueerEOL
    {\StraightEOL\@input@{\jobname.gls}\QueerEOL}%
    {\@input@{\jobname.gls}}%
    \g@emptify\PrintChanges}
```

```
\pdef\toCTAN{%
          % #1 ⟨year/month/day⟩␣⟨version number⟩
  \gmd@changes@init
 \gmd@toCTAN@}
```

```
\def\gmd@toCTAN@#1{%
  \edef\gmu@tempa{\gmd@chgs@parse#1␣\@nil}%
  \edef\gmu@tempa{%
    \unexpanded{\changes@[\generalname]}%
    {\@xa\@firstofthree\gmu@tempa}%
    {\@xa\@secondofthree\gmu@tempa}%
    {put␣to␣\acro{CTAN}␣on␣\@xa\@secondofthree\gmu@tempa}}%
  \gmu@tempa}
```

To make writing changes easier, to allow copying the date & version string from the `\ProvidesPackage/Class` optional argument.

```
\outer\pdef\chgs{\gmd@changes@init\gmd@chgs}
```

\gmd@chgs

```
\DeclareCommand\gmd@chgs{%
  o␣% the optional CS the change refers to
  >!m␣% change's date, version and text
}{%
  \IfValueTF{#1}{%
    \edef\gmu@tempa{\@nx\changes@[\unexpanded{#1}]%
      \@xa\unexpanded\@xa{\gmd@chgs@parse#2\@nil}}}%
  {\edef\gmu@tempa{\@nx\changes@
      \@xa\unexpanded\@xa{\gmd@chgs@parse#2\@nil}}}%
  \gmu@tempa}% of \gmd@chgs
```

```
\long\def\gmd@chgs@parse#1␣#2␣#3\@nil{{#2}{#1}{#3}}%
```

```
\outer\pdef\CH{%
  \gmd@changes@init\gmd@chgsplus}
```

\gmd@chgsplus
```
\DeclareCommand\gmd@chgsplus{\SameAs\gmd@chgs}{%
  \DCUse\gmd@chgs{#1}{#2}%
  \gmd@threeway{#1}#2\@nil
}
```

This is just formatting of the main

```
\long\def\gmd@threeway
#1% opt. CS that \CH refers to
#2␣% (delimd. with a blank) date
#3␣% (delimd. with a blank) version
#4\@nil␣% text
{%
  \par␣(#2,␣#3\IfValueT{#1}{,␣\texttt{\detokenize\@xa{%
       \string#1}}}:)
  #4\scantokens{}% to provide proper line end which'll take care of \par &c.
}
```

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
```
\newcount\check@sum
```

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

CheckSum
```
\newcounter{CheckSum}
```

\step@checksum
```
\newcommand*\step@checksum{\stepcounter{CheckSum}}
```

And we'll use it in the line 4241 (\stepcounter is \global). See also the \chschange declaration, l. 7168.

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 `macrocodes`, I counted myself.)[13]

"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.

```
\def\check@checksum{\relax
  \ifnum\check@sum=\z@
    \edef\gmu@tempa{% why \edef—see line 7130
```

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

```
      \@nx\typeout{**********************************^^J%
        *␣The␣input␣file␣\gmd@inputname\space␣has␣no␣Checksum
        stated.^^J%
        *␣The␣current␣checksum␣is␣\the\c@CheckSum.^^J%
        \gmd@chschangeline% a check sum changes history entry, see below.
        *␣(package␣gmdoc␣info.)^^J%
        **********************************^^J}}
    \else
    \ifnum\check@sum=\c@CheckSum
      \edef\gmu@tempa{%
        \@nx\typeout{**********************+^^J%
          *␣The␣input␣file␣\gmd@inputname:␣Checksum␣
               passed.^^J%
          \gmd@chschangeline
          *␣(package␣gmdoc␣info.)^^J%
          **********************+^^J}}
    \else
      \edef\gmu@tempa{%
        \@nx\typeout{********\gmd@wykrzykniki^^J%
          *!␣The␣input␣file␣\gmd@inputname:^^J%
          *!␣The␣CheckSum␣stated:␣\the\check@sum\space<>␣my
          count:␣\the\c@CheckSum.^^J%
          \gmd@chschangeline
          *!␣(package␣gmdoc␣info.)^^J%
          ********\gmd@wykrzykniki^^J}}%
    \fi
  \fi
  \gmu@tempa
  \@xa\AtEndDocument\@xa{\gmu@tempa}% we print the checksum notification on the terminal immediately and at end of TEXing not to have to scroll the output far nor search the log.
  \global\check@sum\z@}

\def\gmd@wykrzykniki{!␣!␣!␣!␣!␣!␣!␣!␣!␣!␣!␣!}
```

to be able to change it when we don't want XeTeX to finish with Code 1 what usually breaks make.

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{%
  \xiipercent\space\string\chschange
  {\@ifundefined{fileversion}{v???}{\fileversion}}%
  {\the\year/\the\month/\the\day}%
  {\the\c@CheckSum}^^J%
  \xiipercent\space\string\chschange
  {\@ifundefined{fileversion}{v???}{\fileversion}}%
  {\@xa\@gobbletwo\the\year/\the\month/\the\day}%
  {% with two digit year in case you use \ChangesStart.
    \the\c@CheckSum}^^J}
```

And here the meaning of such a line is defined:

```
\outer\pdef\chschange{%
```

```
%  #1  m file version,
%  #2  m date,
% (#3) c hecksum,
% [#4] o the reason of check sum change, possibly short.
\@ifQueerEOL
{\def\EOLwasQueer{11}}{\def\EOLwasQueer{10}}%
\gmd@changes@init
\chschange@}
```

\chschange@

```
\DeclareCommand\chschange@{mmmo}{%
  \changes@{#1}{#2}{CheckSum␣#3
    \IfValueT{#4}{because␣of␣#4}%
  }% \csname... because \changes is \outer.
  \CheckSum{#3}%
  \IfValueF{#4}{%
    \if\EOLwasQueer
    \afterfi{%
      \@ifnextchar\par{%
        \@xa\gmd@textEOL\gobble}%
      {}%
    }% of \afterfi,
    \fi}% of no value of #4,
}% of \chschange@.
```

It will make a 'General' entry in the change history unless used in some `\Define`'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

```
\newcommand*\ltxPageLayout{%
```

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

```
  \setlength{\textwidth}{355pt}%
```

"Increase the marginpar width slightly, for long command names. And increase the left margin by a similar amount."

To make these settings independent from the defaults (changed e.g. in gmdocc.cls) we replace the original `\addtolengths` with `\setlengths`.

```
 \setlength\marginparwidth{95pt}%
 \setlength\oddsidemargin{82pt}%
 \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 title pages 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.

`\DocInclude` 7252
```
\DeclareCommand\DocInclude{O{}mO{}}{%
  % [#1] o path (with closing slash), will not be printed
  %  #2  m file name without extension, will be printed
  % [#3] o file extension (with dot) if not .sty, .cls, .tex, .dtx nor .fdd
```
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.

`\HLPrefix` 7264
```
\gdef\HLPrefix{\filesep}%
```
7265
```
\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 `\if`s. Therefore the definition is global. The latter is such for symmetry.

7270
```
\def\GeneralName{#2\actualchar\pk{#2}␣}%
```
for the changes' history main level entry.

Now we check whether we try to include ourselves and if so—we'll (create and) read an .auxx file instead of (the main) .aux to avoid an infinite recursion of `\input`s.

7277
```
  \edef\gmd@jobname{\jobname}%
```
7278
```
  \edef\gmd@difilename{%
```
we want the filename all 'other', just as in `\jobname`.
7280
```
    \@xa\@xa\@xa\@gobble\@xa\string\csname#2\endcsname}%
  \ifx\gmd@jobname\gmd@difilename
    \def\gmd@auxext{auxx}%
  \else
    \def\gmd@auxext{aux}%
  \fi
\relax
\clearpage
\gmd@docincludeaux␣\def\currentfile{%
```
7290
```
    gmdoc-IncludeFileNotFound.ooo}%
```
7291
```
\let\fullcurrentfile\currentfile
\@ifnonempty{#3}%
{%
  \unless\if.\@firstofmany#3\relax\@nil
    \PackageError{gmdoc}{Optional␣\xiihash3␣of
      \string\DocInclude\space
      if␣present␣has␣to␣begin␣with␣a␣dot␣(.)}{}%
  \fi
  \edef\currentfile{#2#3}%
\IfFileExists{#1\currentfile}{}%
{\PackageError{gmdoc}{\string\DocInclude\space␣file
    \currentfile\space␣not␣found}{}}%
}% of if extension given.
{% if extension not given:
```

```
7305      \IfFileExists{#1#2.fdd}{\edef\currentfile{#2.fdd}}{% it's not .fdd,
7306        \IfFileExists{#1#2.dtx}{\edef\currentfile{#2.dtx}}{% it's not
              .dtx either,
7308          \IfFileExists{#1#2.sty}{\edef\currentfile{#2.sty}}{% it's
                not .sty,
7310            \IfFileExists{#1#2.cls}{\edef\currentfile{#2.cls}}{% it's
                  not .cls,
7312              \IfFileExists{#1#2.tex}{\edef\currentfile{#2.tex}}{%
                    % it's not .tex,
7314                \IfFileExists{#1#2.fd}{\edef\currentfile{#2.fd}}{%
                      % so it must be .fd or error.
7316                  \PackageError{gmdoc}{\string\DocInclude\space␣
                        file
7317                    #1#2.fdd/dtx/sty/cls/tex/fd␣not␣found.}{}%
7318                }}}}}}%
7319    }% of if no extension given
7322    \edef\currentfile{\@xa\detokenize\@xa{\currentfile}}%
7323    \edef\fullcurrentfile{#1\currentfile}%
7324    \ifnum\@auxout=\@partaux
7325      \@latexerr{\string\DocInclude\space␣cannot␣be␣nested}\@eha
7326    \else␣\@docinclude{#1}#2#3␣\fi}% Why is #2 delimited with ␣ not braced
         as we are used to, one may ask.

7332 \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 ar-
       gument and it's delimited with a space probably in resemblance to the true
       \input (\@@input in LaTeX).
7337    \clearpage
7339    \if@filesw␣\gmd@writemauxinpaux{#2.\gmd@auxext}\fi% this strange
          macro with a long name is another spurious thing to allow _ in the filenames
          (see line 7404). which are allowed anyway unless active or 14.
7343    \@tempswatrue
7344    \if@partsw␣\@tempswafalse\edef\gmu@tempb{#2}%
7345      \@for␣\gmu@tempa:=\@partlist\do{\ifx\gmu@tempa\gmu@tempb%
            \@tempswatrue\fi}%
7346    \fi
7347    \if@tempswa% the file is on \@partlist
7348      \let\@auxout\@partaux
7349      \if@filesw
7350        \immediate\openout\@partaux␣#2.\gmd@auxext\relax% Yes, only
              #2. It's to create and process the partial .aux(x) files always in the main
              document's (driver's) directory.
7355        \immediate\write\@partaux{\relax}%
7356      \fi
```

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

```
7363      \StoringAndRelaxingDo\gmd@doIndexRelated
7364      \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.
7367      \else\let\maketitle=\InclMaketitle
```

```
    \fi% In the default setup we redefine \maketitle to typeset a common chap-
        ter 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% the file isn't on \@partlist
    \@nameuse{cp@#1#2}%
    \g@emptify\gmd@ABIOnce
  \fi
  \let\@auxout\@mainaux}% end of \@docinclude.
```

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

```
\def\xdef@filekey{{\@relaxen\narrativett% This assignment is very trick-
    ily crafted: it makes all \narrativetts present in the \filekey's expansion
    unexpandable not only the one added in this step.
    \xdef\filekey{\filekey,␣\thefilediv={\narrativett%
        \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 out to 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 localised 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{%
    \unexpanded{%
      \bgroup
      \@makeother\_% to allow underscore
      \@makeother\~% to allow paths beginning with ~/
      \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{%
    \unexpanded{%
```

```
      \bgroup
      \@makeother\_%
      \@makeother\~%
      \firstofone}\@charlb\egroup}%
  \@writeckpt{#1}%
  \immediate\write\@partaux{\@charrb}}

\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

```
\def\aalph#1{\@aalph{\csname␣c@#1\endcsname}}
\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`.

```
\def\gmd@docincludeaux{%
```

We set the things for including the files only once.

```
  \global\@relaxen\gmd@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 de-
         fined 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
```

```
\def\thefilediv{\aalph{\filedivname}}% The files will be numbered with
    letters, lowercase first.
\@xa\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.
  \@xa\ifx\csname␣ver@\currentfile\endcsname\relax
    File␣\thefilediv:␣{\narrativett\currentfile}␣%
  \else
    \GetFileInfo{\currentfile}%
    File␣\thefilediv:␣{\narrativett\filename}␣%
    Date:␣\filedate\␣%
    Version␣\fileversion
  \fi
  \hfill\thepage}%
\glet\@evenfoot\@oddfoot% see line 7506.
\@xa\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
  \let\subsubdivision=\paragraph
```

If `\filediv` is higher than `\section` we don't change the three divisions (they are `\section`, `\subsection` and `\subsubsection` 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 \gmd@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.

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

```
\def\SelfInclude{\DocInclude{\jobname}}
```

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 they will prefer what I offer, gmdocc.cls e.g.), so we put those preparations into a declaration.

`\if@ltxDocInclude`

```
\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{%
  \providecommand*\maketitle{\par
    \begingroup␣\def␣\thefootnote␣{\fnsymbol␣{footnote}}%
    \setcounter␣{footnote}\z@
    \def\@makefnmark{\rlap{\@textsuperscript{\normalfont%
        \@thefnmark}}}%
    \long\def\@makefntext##1{\parindent␣1em\noindent
      \hb@xt@1.8em{%
        \hss\@textsuperscript{\normalfont\@thefnmark}}##1}%
    \if@twocolumn␣\twocolumn␣[\@maketitle␣]%
    \else␣\newpage␣\global␣\@topnum␣\z@␣\@maketitle␣\fi
```

"For special formatting requirements (such as in TUGboat), we use page style `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@relaxen\@author\g@relaxen\@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 page style 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.

```
\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.)

```
\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 initialise
      \titlesetup if it is not yet.
\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 the `\ltxLookSetup` declaration is in force, `\Doc/SelfInclude` will make for an included file a part's title page and an article-like title.

Let's initialise the file division macros.

```
\@relaxen\filediv
\@relaxen\filedivname
\@relaxen\thefilediv
```

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.

```
\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\@ifauthor{%
          \protect\space(\@author)}{}}% we add the author iff the 'files have different authors' and author exists (@variousauthors)
    }%
    \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 mysterious 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
        {\protect\smallerr% this macro is provided by the gmutils package after the relsize package.
          \@ifauthor
          {\protect\\[0.15em]\@nx\@author
            \ifx\relax\@date\else, \fi}% after use, \@date is let to \relax.
          {\ifx\relax\@date\else\protect\\[0.15em]\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@variousauthors` switch defined in line 7775.

If we wish to print the author's name (`\if@variousauthors`), 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.

```
          \protect\@date}}% end of \incl@filedivtitle's brace (2nd or 3rd argument).
    }% 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).

```
    }% end of the comma-\and's group.
    \@xa\filediv\incl@filedivtitle
    \@thanks
```

```
      \g@relaxen\@author␣\g@relaxen\@title␣\g@relaxen\@date
      \g@emptify\@thanks
}% 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.

\if@variousauthors
```
\newif\if@variousauthors
```

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

\PrintFilesAuthors
```
\newcommand*\PrintFilesAuthors{\@variousauthorstrue}
```

And the counterpart, if you change your mind:

\SkipFilesAuthors
```
\newcommand*\SkipFilesAuthors{\@variousauthorsfalse}
```

```
\def\@ifauthor{%
      % #1  what if true
      % #2  what if false
  \ifnum\numexpr\if@variousauthors1\else0\fi*
      \ifx\@author\relax0\else\ifx\@author\@empty0\else1\fi%
          \fi>0
    \@xa\@firstoftwo
  \else
    \@xa\@secondoftwo
  \fi
}
```

The file's date and version information

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

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

Since we may documentally input files that we don't load, as `doc` e.g., let's define a declaration to be put (in the comment layer) before the line(s) containing `\Pro¦vides`.... The `\FileInfo` command takes the stuff till the closing `]` and subsequent line end, extracts from it the info and writes it to the `.aux` and rescans the stuff. ε-TEX provides a special primitive for that action but we remain strictly TEXnical and do it with writing to a file and inputting that file.

\FileInfo
```
\newcommand*\FileInfo{%
  \bgroup
  \gmd@ctallsetup
  \bgroup% yes, we open two groups because we want to rescan tokens in 'usual'
        catcodes. We cannot put \gmd@ctallsetup into the inner macro because
        when that will be executed, the \inputlineno will be too large (the last
        not the first line).
  \let\do\@makeother
```

```
  \do\␣\do\{\do\}\do\^^M\do\\%
  \gmd@fileinfo}

\foone{%
  \catcode`!\z@
  \catcode`(\@ne
  \catcode`)\tw@
  \let\do\@makeother
  \do\␣% we make space 'other' to keep it for scanning the code where it may be
       leading.
  \do\{\do\}\do\^^M\do\\}%
(%
!def!gmd@fileinfo#1Provides#2{#3}#4[#5]#6^^M%
(!egroup% we close the group of changed catcodes, the catcodes of the arguments
      are set. And we are still in the group for \gmd@ctallsetup.
!gmd@writeFI(#2)(#3)(#5)%
!gmd@FIrescan(#1Provides#2{#3}#4[#5]#6)% this macro will close the group.

)%
)

\def\gmd@writeFI#1#2#3{%
  {\newlinechar=\endlinechar%
    \immediate\write\@auxout{%
      \global\@nx\@namedef{%
        ver@#2.\if␣P\@firstofmany#1\@nil␣sty\else␣cls\fi}{%
            #3}}}}

\foone\obeylines{%
  \def\gmd@FIrescan#1{%
 {\newlinechar=\endlinechar\scantokens{#1}}\egroup^^M}}
```

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

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

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

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

For the files generated from master, in which all the info is porvided at the beginning in macros `\⟨name⟩Version`, `\⟨name⟩Date` etc. (not to repeat that information in the body of text):

```
\def\gmd@upperDIV#1{%
  \if␣d#1D\fi
```

```
  \if␣i#1I\fi
  \if␣v#1V\fi
}
```

First we look for the info at the leaf-level, then at standalone level, then at the bundle level. If we don't find it, it'll be empty.

```
\def\edefInfo
#1% name
#2% datum
{%
  \edef\gmd@edefInfo@resa{\gmd@upperDIV␣#2}%
  \@nameedef{file#2}{%
    \ifcsname␣#1Leaf\gmd@edefInfo@resa\endcsname␣% e.g. gmbaseLeafVersion
      \xA\xA\xA\detokenize\xA\xA\xA{%
        \csname␣#1Leaf\gmd@edefInfo@resa\endcsname
      }%
    \else
      \ifcsname␣#1\gmd@edefInfo@resa\endcsname␣% e.g. gmbaseVersion
        \xA\xA\xA\detokenize\xA\xA\xA{%
          \csname␣#1\gmd@edefInfo@resa\endcsname
        }%
      \else
        \ifcsname␣\gmBundleFile␣\gmd@edefInfo@resa\endcsname␣% e.g. gmutils
          \xA\xA\xA\detokenize\xA\xA\xA{%
            \csname␣\gmBundleFile␣\gmd@edefInfo@resa\endcsname
          }%
        \fi
      \fi
    \fi
  }% of edefined macro
}% of \edefInfo
```

To get file info (the file is a leaf of a bundle or a standalone)

```
\def\FileInfoFromName#1{%
  \edefInfo{#1}{date}%
  \edefInfo{#1}{version}%
  \edefInfo{#1}{info}%
  \def\GeneralName{#1\actualchar\pk{#1}␣}% for the changes' history.
}
```

Get bundle info

```
\def\BundleInfoFromName#1{%
  \def\gmBundleFile{#1}%
  \Store@MacroSt␣{#1LeafDate}%
  \Store@MacroSt␣{#1LeafVersion}%
  \Store@MacroSt␣{#1LeafInfo}%
  \n@melet{#1LeafDate}{@undefined}%
  \n@melet{#1LeafVersion}{@undefined}%
  \n@melet{#1LeafInfo}{@undefined}%
  \FileInfoFromName{#1}%
  \Restore@MacroSt␣{#1LeafDate}%
  \Restore@MacroSt␣{#1LeafVersion}%
  \Restore@MacroSt␣{#1LeafInfo}%
```

```
}
```

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

```
\pdef\filenote{This␣file␣has␣version␣number␣\fileversion{}␣
    dated␣\filedate{}.}
```

And exactly as `\thanks`:

```
\pdef\thfileinfo{\thanks\filenote}
```

And to the footnote:

```
\pdef\fnfileinfo{%
  \gmu@ifedetokens{\@currext}{toc}%
  {}%
  {\footnote\filenote}%
}
```

Miscellanea

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

```
\foone{\obeylines}%
{%
  \pdef\IndexInput#1{%
    \Store@Macro\code@delim%
    \CodeDelim*\^^Z%
    \def\gmd@iihook{% this hook is \edefed!
      \@nx^^M%
      \code@delim\relax\@nx\let\@nx\EOFMark\relax}%
    \DocInput{#1}\Restore@Macro\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
```
\newenvironment{gmlonely}{\StraightEOL}{}
```

\skipgmlonely
```
\newcommand\skipgmlonely[1][]{%
  \def\gmu@tempa{%
    \def\gmd@skipgmltext{%
      \g@emptify\gmd@skipgmltext
      #1%
    }}% not to count the lines of the substituting text but only of the text omitted
  \gmu@tempa
  \@xa\AtBegInput\@xa{\gmu@tempa}%
```

gmlonely
```
  \renewenvironment{gmlonely}{%
    \StraightEOL
    \@fileswfalse% to forbid writing to .toc, .idx etc.
    \setboxo=\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`.

```
\foone{\catcode`\_=8␣}% we ensure the standard \catcode of _.
{%
  \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.

```
\StoreEnvironment{quotation}
\def\gmd@quotationname{quotation}
\renewenvironment{quotation}{%
```

The first non-me user complained that `abstract` comes out in quotation marks. That is because `abstract` uses `quotation` internally. So we first check whether the current environment is `quotation` or something else.

```
  \ifx\@currenvir\gmd@quotationname
  \afterfi{\par``\ignorespaces}%
  \else\afterfi{\storedcsname{quotation}}%
  \fi}
{\ifx\@currenvir\gmd@quotationname
  \afterfi{\ifhmode\unskip\fi''\par}%
  \else\afterfi{\storedcsname{endquotation}}%
  \fi}
```

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

```
\def\gmdnoindent{%
  \ifvmode\leavevmode\hskip-\parindent\ignorespaces
  \fi}%
```

`\ignorespaces` is added to eat a space inserted by `\gmd@textEOL`. Without it it also worked but it was a bug: since `\parindent` is a dimen not skip, TEX looks forward and expands macros to check whether there is a stretch or shrink part and therefore it gobbled the `\gmd@textEOL`'s space.

When a verbatim text occurs in an in-line 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. Moreover, if such a short verb breaks in its middle, it should break with the percent at the beginning of the new line. For this purpose provide `\inverb`. It breaks with a `%` at the beginning of new line. Ist starred version puts `%` also at the end of the upper line.

```
\pdef\inverb{%
```

```
  \gmu@ifstar{%
    \def\gmu@tempa{\verbhyphen}% the pre-break.
    \@emptify\gmu@tempb% the no-break.
    \gmd@inverb}%
  {\@emptify\gmu@tempa% the pre-break empty
    \def\gmu@tempb{\gmboxedspace}% the no-break boxed space.
    \gmd@inverb}}
```

\gmboxedspace

```
\newcommand*\gmboxedspace{\hbox{\normalfont{␣}}}

\pdef\gmd@nlperc{%
  \ifhmode\unskip\fi
  \begingroup\hyphenpenalty\inverbpenalty\relax
  \discretionary{\hbox{\gmu@tempa}}% (pre-break). I always put a \hbox
       here to make this discretionary score the \hyphenpenalty not \exhy¦
       phenpenalty (The TeX book p. 96) since the latter may be 10,000 in Polish
       typesetting.
  {\hbox{\narrationmark}}% (post-break)
  {\gmu@tempb}% (no-break).
  \endgroup
  \penalty10000\hskiposp\relax}

\def\inverbpenalty{-1000}

\pdef\gmd@inverb{%
  \gmd@nlperc
  \ifmmode\hbox\else\leavevmode\null\fi
  \bgroup
  \ttverbatim
  \narrativett
  \def\breakablevisspace{%
    \discretionary{\visiblespace}{\narrationmark}{%
         \visiblespace}}%
  \def\breakbslash{%
    \discretionary{}{\narrationmark\type@bslash}{%
         \type@bslash}}%
  \def\breaklbrace{%
    \discretionary
      {\xiilbrace\verbhyphen}%
      {\narrationmark}%
      {\xiilbrace}}%
  \gm@verb@eol
  \@sverb@chbsl% It's always with visible spaces.
}

\pdef\nlperc{\newline\narrationmark\ignorespaces}

\pdef\nlpercent{%
  \@emptify\gmu@tempa
  \def\gmu@tempb{\gmboxedspace}%
  \gmd@nlperc
}

\pdef\incs{% an in-line \cs
  \@emptify\gmu@tempa
  \def\gmu@tempb{\gmboxedspace}%
```

```
  \gmd@nlperc\cs
}
```

```
\def\inenv{\incs[]}% an in-line \env
```

```
\def\incmd{%
```
it has to be `\def` to let it expand to let `\cmd` convert its argument to a safe string.
```
        \nlpercent\cmd}
```

```
\def\inhash{\nlpercent\hash}
```

As you see, `\inverb` and `\nlpercent` insert 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. Then `\inverb` starts sth. like `\verb*` but the breakables of it break to `%` in the lower line.

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

An optional hyphen for CSes in the in-line comment:

`\cs`
```
\@xa\ampulexdef\csname\@dc@InnerName\cs\endcsname
[#1]␣[#1]␣{\begingroup}␣{\ifdefined}
{\begingroup␣\def\+{\discre{\gmv@hyphen}{\narrationmark}{}}%
  \ifdefined}
```

`\ds`
```
\providecommand*\ds{DocStrip}
```

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

`\CDAnd`
```
\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.

`\division`
```
\let\division=\section
```

`\subdivision`
```
\let\subdivision=\subsection
```

`\subsubdivision`
```
\let\subsubdivision=\subsubsection
```

To kill a tiny little bug in `doc.dtx` (in line 3299 `\gmu@tempb` and `\gmu@tempc` are written plain not verbatim):

`gmd@mc`
```
\newcounter{gmd@mc}
```

Note it is after the `macrocode` group

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

```
\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 `\gmu@tempb/c`).

One more detail to explain and define: the `\gmd@mcdiag` macro may be defined to type out a diagnostic message (the `macrocode[*]`'s number, code line number and input line number).

```
\@emptify\gmd@mcdiag

\def\mcdiagOn{\def\gmd@mcdiag{%
    \typeout{^^J\bslash␣end{\gmd@lastenvir}␣No.\the\c@gmd@mc
    \space\on@line,␣cln.\the\c@codelinenum.}}}

\def\mcdiagOff{\@emptify\gmd@mcdiag}
```

An environment to display the meaning of macro parameters: its items are automatically numbered as `#1`, `#2` etc.

`\DeclareEnvironment{enumargs}{o}%` the optional argument specifies number of `#`'s; it's of the `o` type to inform if it was not given by the user to handle a possible active char touched by argument's catcher; can be `1` (the default), `2` or `4`; any else produces one `#`.

```
{%
  \StraightEOL
  \if@aftercode
    \edef\gmu@tempa{\the\leftskip}%
    \edef\gmu@tempb{\the\hangindent}%
  \fi
  \enumerate
  \if@aftercode
    \leftskip=\glueexpr\gmu@tempa+\gmu@tempb\relax
  \fi
  \edef\gmd@ea@hashes{%
    \#\ifcase\IfValueTF{#1}{#1}{1}\relax
    \or\or\#\or\or\#\#\#\fi}%

  \@namedef{label\@enumctr}{%
    \env{\if@aftercode\narrationmark\fi
      \relax% to stop \ignorespaces
      \gmd@ea@bwrap
      \gmd@ea@hashes
      \csname␣the\@enumctr\endcsname
      \gmd@ea@ewrap}}% of \label⟨@enumctr⟩.
  \let\mand\item
\gmd@ea@wraps
  \provide\gmd@ea@wraps{%
    \emptify\gmd@ea@ewrap
    \emptify\gmd@ea@bwrap}%
  \gmd@ea@wraps
  \def\opt{%
    \def\gmd@ea@bwrap{[}\def\gmd@ea@ewrap{]}%
  \item
    \gmd@ea@wraps}%

  \settowidth{\@tempdima}{\narrativett␣x\gmd@ea@hashes7x}%
  \edef\gmd@ea@xxxwd{\the\@tempdima}%
```

```
\dc 8307  \DeclareCommand\dc␣!{%
8308    Q{*>}␣% (1) we check whether there's a sergeant right of the prefix or a star to
              suppress parentheses,
8310    Q{P!lL\long␣iI}␣% (2) an optional 'bare' prefix for a 'long' argument or for
              ignored
8312    b␣% (3) prefix(es) in curly braces ( This way we allow the prefix(es) to be braced
              or not at the author's option),
8315    >\@xa␣T{\@dc@argtypes}␣% (4) (optional) argument type specifier,
8317    b␣% (5) (optional) default value of the specified argument or (for K and G)
              mandatory.
8319    b␣% (6) default of K and G.
8320  }{%
8321     \gmu@ifxany␣*{##1}%
8322    {% a * suppresses bracket/brace/parentheses decoration.
8323      \def\gmd@ea@bwrap{\hbox␣to␣\gmd@ea@xxxwd\bgroup\hss}%
8324      \def\gmd@ea@ewrap{\hss\egroup}%
8325    }%
8326    {% if there's no * in #1, be wrap the item label in braces/brackets/parentheses.
8328      \gmu@ifxany␣##4{bB}{% I decide not to print m type arguments in braces
                because the braces are not mandatory for this type.
8331        \def\gmd@ea@bwrap{\{}%
8332        \def\gmd@ea@ewrap{\}}%
8333      }{}%
8334      \gmu@ifxany␣##4{cC}{%
8335        \def\gmd@ea@bwrap{(}%
8336        \def\gmd@ea@ewrap{)}%
8337      }{}%
8338      \gmu@ifxany␣##4{oO}{%
8339        \def\gmd@ea@bwrap{[}%
8340        \def\gmd@ea@ewrap{]}%
8341      }{}%
8342      \gmu@ifxany␣##4{G}{%
8343        \def\gmd@ea@bwrap{\detokenize\@xa{\@firstoftwo##5}}%
8344        \def\gmd@ea@ewrap{\detokenize\@xa{\@secondoftwo##5}}%
8345      }{}%
8346      \gmu@ifxany␣##4{A}{%
8347        \def\gmd@ea@bwrap{<}%
8348        \def\gmd@ea@ewrap{>}%
8349      }{}%
8350    }% of if no * in #1.
8351    \gmu@ifxany␣##4{mQsSTK\afterassignment}{%
8352      \def\gmd@ea@bwrap{\hbox␣to␣\gmd@ea@xxxwd\bgroup\hss}%
8353      \def\gmd@ea@ewrap{\hss\egroup}%
8354    }{}%
```

we add a normal space

```
8356    \addtomacro\gmd@ea@ewrap{{\normalfont\␣}}%
8357    \IfValueT{##2}{%
8358      \addtomacro\gmd@ea@ewrap{>\{\string##2\}}}%
8359    \IfValueT{##3}{%
8360      \addtomacro\gmd@ea@ewrap{>\{##3\}}}%
8361    \IfValueT{##4}{%
8362      \ifx␣s##4%
```

```
        \addtomacro\gmd@ea@ewrap{%
          \llap{\metachar[}\scanverb{*}\metachar]}%
      \else\addtomacro\gmd@ea@ewrap{##4}%
      \fi}%
    \IfValueT{##5}{%
      \addtomacro\gmd@ea@ewrap{\{%
    %\ttverbatim breakable chars won't work because we are in the item's label's
    % \hbox.
        \scanverb*{##5}%
        \}}}%
    \IfValueT{##6}{%
      \addtomacro\gmd@ea@ewrap{\{%
    %\ttverbatim breakable chars won't work because we are in the item's label's
    % \hbox.
        \scanverb*{##6}%
        \}}}%
    \def\gmd@blubra{%
      \addtomacro\gmd@ea@bwrap{%
        \begingroup
        \relaxen\gmd@ea@hashes
        \@namedef{the\@enumctr}{\<ign.>}%
      }%
      \prependtomacro\gmd@ea@ewrap{%
        \endgroup}%
      \addtomacro\gmd@ea@ewrap{%
        \global␣\advance␣\csname␣c@\@enumctr\endcsname␣\m@ne
      }%
      \emptify\gmd@blubra
    }%
    \gmu@ifsbintersect␣{##2}{Ii}{\gmd@blubra}{}%
    \gmu@ifsbintersect␣{##3}{Ii}{\gmd@blubra}{}%
    \gmu@ifxany␣##4{\afterassignment}{\gmd@blubra}{}%
  \item\relax}%
```

`\IfNoValueT{#1}{\@ifnextac\@gobble{}}%` to gobble a possible active line end or active `^^A` or `^^B` that might have occurred because of `\futurelet` of the optional argument checker.

```
  }% of begin definition
{\endenumerate}
```

The starred version is intended for lists of arguments some of which are optional: to align them in line.

enumargs*
\gmd@ea@wraps

```
\newenvironment*{enumargs*}{%
  \def\gmd@ea@wraps{%
    \def\gmd@ea@bwrap{␣}\def\gmd@ea@ewrap{␣}}%
  \enumargs}{\endenumargs}
```

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 2877.

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`
```
\def\CharacterTable{\begingroup
  \@makeother\{\@makeother\}%
  \Character@Table}

\foone{%
  \catcode`\[=1 \catcode`\]=2 %
  \@makeother\{\@makeother\}}%
[
```
`\Character@Table`
```
  \def\Character@Table#1{#2}[\endgroup
     \message[^^J^^J gmdoc.sty package:^^J
     ==== The input file contains the \bslash
          CharacterTable.^^J
     ==== If you really need to check the correctness of the
          chars,^^J
     ==== please notify the author of gmdoc.sty at the email
          address^^J
     ==== 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 8819.

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

`\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{%
  \@bsphack
  \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 `\MakePriv|ateOthers` re`\catcode`ing shall not be done to it.

`\DescribeEnv`
```
\outer\def\DescribeEnv{%
  \@bsphack
  \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{%
```
It doesn't typeset its argument in the point of occur-

rence.

```
\leavevmode
\@bsphack
\begingroup\MakePrivateLetters
\gmu@ifstar{\MakePrivateOthers\Describe@Env}{%
    \Describe@Macro}}
```

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

`\Describe@Macro`

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

`\Describe@Env`

```
\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}%
  \@esphack}
```

Note that here the environments' names are typeset in `\narrativett` font just like the macros', *unlike* in doc.

`\MacroFont`

My understanding of 'minimality' includes avoiding too much freedom as causing chaos not beauty. That's the philosophical and æ sthetic reason why I don't provide `\MacroFont`. In my opinion there's a noble tradition of typesetting the TEX code in `\tt` font and this tradition sustained should be. If one wants to change the tradition, let them 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 2691. 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.

`\SpecialEscapechar`

```
\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 line number among others, so it's not appropriate to put two things into one macro. Thus let's give the user a notice if they 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 (æsthetic) 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.

\CodelineNumbered
```
\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 they'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

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

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

The next two declarations I find useful and smart:

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

\EnableCrossrefs
\DisableCrossrefs
```
\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
\StopEventually

```
\newcommand*\AlsoImplementation{\@bsphack
  \long\def\StopEventually##1{\gdef\Finale{##1}}%
  %\init@checksum
  \@esphack}
```

we define \Fin¦ale 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 probably numerous) input(s).

```
\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
\StopEventually

```
\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 TEX book* 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:

```
\let\SpecialMainIndex=\DefIndex
```

*

\SpecialMainEnvIndex

```
\def\SpecialMainEnvIndex{\csname␣CodeDefIndex\endcsname*}%
```

we don't type \DefIndex explicitly here because 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 occurs it takes `$` as the `\verb*`'s delimiter. So strange delimiters are chosen deliberately to allow any 'other' chars in the environments' names. If this would cause problems, please notify me and we'll think of adjustments.

\verbatimchar 8692 `\def\verbatimchar{&}`

\IndexPrologue `\IndexPrologue` is defined in line 6388. And other doc index commands too.

8708 `\@ifundefined{main}{}{\let\DefEntry=\main}`

8710 `\@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 don't want an error while gmdocumenting doc-prepared files, better let's define doc's counter for the modules' depths.

StandardModuleDepth 8718 `\newcounter{StandardModuleDepth}`

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

\ 8723 `\noeffect@info{DocstyleParms}`

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

\DontCheckModules 8728 `\@emptify\DontCheckModules␣\noeffect@info{DontCheckModules}`
\CheckModules 8729 `\@emptify\CheckModules␣\noeffect@info{CheckModules}`

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

\AltMacroFont 8733 `\@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 8739 `\def\MakePercentIgnore{\catcode`\%9\relax}`
\MakePercentComment 8740 `\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[14] 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 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 old-style input commands change the inner macros so that also `\MakeShortVerb` works as in doc (cf. line 8781).

14 What a *false* modesty! ;-)

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
```
\def\OldDocInput{%
  \AtBegInputOnce{\StraightEOL
    \let\@MakeShortVerb=\old@MakeShortVerb
    \OldMacrocodes}%
  \bgroup\@makeother\_% it's to allow _ in the filenames. The next macro will
      close the group.
  \Doc@Input}
```

We don't switch the `@codeskipput` switch neither we check it because in 'old' world there's nothing to switch this switch in the narration layer.

I had a hot and wild TEX all the night and what a bliss when the 'Successfully 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

```
% \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.

`\OCRInclude`

I realised that I want to print all my TEX source files verbatim just in case my computers and electronic memories break so that I can reconstruct them via OCR . For this purpose I provide `\OCRInclude`. It takes the same arguments as `\DocInclude` only typesets a file with no index nor line numbers.

\OCRInclude
```
\DeclareCommand\OCRInclude{O{}mO{}}{%
```

```
\Store@Macro\incl@DocInput
\def\incl@DocInput##1{%
  \begingroup
  \CodeSpacesBlank
  \@beginputhook
  \title{\currentfile}\maketitle
  \noverbatimspecials
  \relaxen\@xverbatim
  \relaxen\check@percent
  \Restore@Macro\@verbatim
  \verbatimleftskip\z@skip
  \verbatim
  \@makeother\|% because \ttverbatim doesn't do that.
  \texcode@hook% we add some special stuff, e.g. in gmdocc.cls we
  \@input{##1}%
  \endgroup}%
\csname\@dc@InnerName\DocInclude\endcsname{#1}{#2}{#3}%
\Restore@Macro\incl@DocInput
}
```

\incl@DocInput

Polishing, development and bugs

• `\MakePrivateLetters` theoretically may interfere with `\active`ating some chars to allow line breaks. 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 2527–2532. 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?[15]

• Do you use `\listoftables` and/or `\listoffigures` in your documentations? If so, I should 'EOL-straighten' them like `\tableofcontents`, I suppose (cf. line 2984).

• Some lines of non-printing stuff such as `\Define`... 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.

• Specifying `codespacesgrey,␣codespacesblank` results in typesetting all the spaces grey including the leading ones.

• About the DocStrip verbatim mode directive see above.

15 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.

[No] ⟨*eof*⟩

Until version 0.99i a file that is `\DocInput` had to be ended with a comment line with an `\EOF` or `\NoEOF` CS that suppressed the end-of-file character to make input end properly. Since version 0.99i however the proper ending of input is achieved with `\everyeof` and therefore `\EOF` and `\NoEOF` become a bit obsolete.

If the user doesn't wish the documentation to be ended by '⟨*eof*⟩', they should redefine the `\EOFMark` CS or end the file with a comment ending with `\NoEOF` macro defined below[16]:

```
8941 \foone{\catcode`\^^M\active␣}{%
\@NoEOF 8942   \def\@NoEOF#1^^M{%
8943     \@relaxen\EOFMark endinput}%
 \@EOF  8944   \def\@EOF#1^^M{ endinput}}

\NoEOF  8946 \def\NoEOF{\QueerEOL\@NoEOF}
 \EOF   8947 \def\EOF{\QueerEOL\@EOF}
```

As you probably see, `\[No]EOF` have the 'immediate' `\endinput` effect: the file ends even in the middle of a line, the stuff after `\(No)EOF` will be gobbled unlike with a bare `\endinput`.

```
9055 ⟨/ doc⟩
9056 ⟨*docc⟩
```

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 T1 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 **??**). 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.,
noindex the default. An exception is made for the `noindex` option, which is provided by this

16 Thanks to Bernd Raichle at BachoTEX 2006 Pearl Session where he presented `\input`ing a file inside `\edef`.

class and passed to gmdoc.sty. This is intended for the case you don't want to make an index.

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

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 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 homonymic with the class's name: `mwrep` for mwrep and `mwbk` for mwbk. For the symmetry there's also `mwart` option (equivalent to the default setting).

mwrep
mwbk
mwart

The existence test is done for any MW class option as it is in the default case.

Since version 0.99g (November 2007) the bundle goes XeTeX and that means you can use the system fonts if you wish, just specify the `sysfonts` option and the three basic XeTeX-related packages (fontspec, xunicode and xltxtra) will be loaded and then you can specify fonts with the fontspec declarations. For use of them check the driver of this documentation where the TeX Gyre Pagella font is specified as the default Roman.

sysfonts

There are also some options for mono and sans fonts, see the changes history for details.

minion The `minion` option sets Adobe Minion Pro as the main font, the `pagella` sets TeX Gyre Pagella as the main font.

pagella

cronos The `cronos` option sets Adobe Cronos Pro as the sans serif font, the `trebuchet` option sets MS Trebuchet as sans serif.

trebuchet

cursor The `cursor` (working only with XeTeX & fontspec) option sets TeX Gyre Cursor as the typewriter font. It emboldens it to the optical weight of Computer/Latin Modern Mono in the code (`embolden=2.5`) and leaves light (`embolden=1`) for verbatims in the narrative. Moreover, this option also prepares a condensed version (`extend=0.87`) for verbatims in the marginpars.

Note that with no option for the monospaced font the default (with XeTeX) will be Latin Modern Mono and then Latin Modern Mono Light Condensed is set for verbatims in marginpars (if available).

\verbatimspecials This class sets `\verbatimspecials/«»[¿]` if the engine is XeTeX, see the gmverb documentation to learn about this declaration. Remember that `\verbatimspecials` whatever would they be, have no effect on the code layer.

\EOFMark The `\EOFMark` in this class typesets like this (of course, you can redefine it as you wish):

⟨*eof*⟩

The Code

```
\PassOptionsToPackage{rgb}{xcolor}
\RequirePackage{xkeyval}
```

A shorthands for options processing (I know xkeyval to little to redefine the default prefix and family).

\gm@DOX
\gm@EOX

```
\newcommand*\gm@DOX{\DeclareOptionX[gmcc]<>}
\newcommand*\gm@EOX{\ExecuteOptionsX[gmcc]<>}
```

We define the `class` option. I prefer the mwcls, but you can choose anything else, then the standard article is loaded. Therefore 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.

\ifgmcc@mwcls

```
\newif\ifgmcc@mwcls
```

Note that the following option defines `\gmcc@class#1`.

class

```
\gm@DOX{class}{% the default will be Marcin Woliński class (mwcls) analogous to
```

article, see line 9388.

\gmcc@CLASS

```
  \def\gmcc@CLASS{#1}%
  \@for\gmcc@resa:=mwart,mwrep,mwbk\do␣{%
    \ifx\gmcc@CLASS\gmcc@resa\gmcc@mwclstrue\fi}%
}
```

mwart

```
\gm@DOX{mwart}{\gmcc@class{mwart}}% The mwart class may also be declared
```

explicitly.

mwrep

```
\gm@DOX{mwrep}{\gmcc@class{mwrep}}% If you need chapters, this option chooses
```

an MW class that corresponds to report,

mwbk

```
\gm@DOX{mwbk}{\gmcc@class{mwbk}}% and this MW class corresponds to book.
```

article

```
\gm@DOX{article}{\gmcc@class{article}}% you can also choose article. A meta-
```

remark: When I tried to do the most natural thing, to `\ExecuteOptionsX` inside such declared option, an error occurred: 'undefined control sequence `% \XKV@resa␣->␣\@nil`'.

outeroff

```
\gm@DOX{outeroff}{\let\outer\relax}% This option allows \outer-prefixed
```

macros to be gmdoc-processed with all the bells and whistles.

\if@debug

```
\newif\if@debug
```

debug

```
\gm@DOX{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

```
\gm@DOX{noindex}{%
  \PassOptionsToPackage{noindex}{gmdoc}}% This option turns the writ-
```

ing out to .idx file off.

\if@gmccnochanges

```
\newif\if@gmccnochanges
```

nochanges

```
\gm@DOX{nochanges}{\@gmccnochangestrue}% This option turns the writing
```

out to .glo file off.

Since version 0.99g the gmdoc bundle goes XeTeX. That means that if XeTeX is detected, we may load the fontspec package and the other two of basic three XeTeX-related, and then we `\fontspec` the fonts. But the default remains the old way and the new way is given as the option below.

```
\ifgmcc@oldfonts 9297 \newif\ifgmcc@oldfonts
      sysfonts 9299 \gm@DOX{sysfonts}{\gmcc@oldfontsfalse}
         mptt 9308 \gm@DOX{mptt}[17]{\relax}% now a no-op, left only for backwards compatibility. It was an option for setting the marginpar typewriter font.
    \gmcc@tout 9318 \def\gmcc@tout#1{\typeout{^^J@@@@␣gmdocc␣class:␣#1^^J}}
  \gmcc@setfont 9320 \def\gmcc@setfont#1{%
              9321   \gmcc@oldfontsfalse% note that if we are not in XeTeX, this switch will be turned true in line 9455
              9323   \AtEndOfClass{%
              9324     \ifdefined\zf@init\afterfi{%
              9325       \gmcc@tout{Main␣font␣set␣to␣#1}%
    \gmcc@dff 9326       \def\gmcc@dff{Numbers={OldStyle,␣Proportional}}
              9327       \@xa\setmainfont\@xa[\gmcc@dff,␣Mapping=tex-text]{#1}%
              9337       \@xa\defaultfontfeatures\@xa{\gmcc@dff,␣Scale=MatchLowercase}% when put before \setmainfont,
              9339       \gmath
 \LineNumFont 9340       \def\LineNumFont{%
              9341         \normalfont\scriptsize\addfontfeature{%
                             Numbers=Monospaced}}%
              9342     }%
              9343     \else\afterfi{\gmcc@tout{I~can␣set␣main␣font␣to␣#1␣only␣in
              9344         XeTeX/fontspec}}%
              9345     \fi
              9346   }}
        minion 9348 \gm@DOX{minion}{\gmcc@setfont{Minion␣Pro}}
       pagella 9349 \gm@DOX{pagella}{\gmcc@setfont{TeX␣Gyre␣Pagella}
              9351 }
       cronos 9352 \gm@DOX{cronos}{%
              9353   \AtEndOfClass{\setsansfont[Mapping=tex-text]{Cronos␣Pro}}}
    trebuchet 9354 \gm@DOX{trebuchet}{%
              9356   \AtEndOfClass{\setsansfont[Mapping=tex-text]{Trebuchet␣MS}}}
       myriad 9357 \gm@DOX{myriad}{%
              9359   \AtEndOfClass{\setsansfont[Mapping=text-text]{Myriad␣Web␣
                       Pro}}}
          lsu 9360 \gm@DOX{lsu}{%
              9362   \AtEndOfClass{\setsansfont[Mapping=tex-text]{Lucida␣Sans␣
                       Unicode}}}
       cursor 9364 \gm@DOX{cursor}{%
              9370   \AtEndOfClass{%
              9371     \setmonofont[FakeBold=2.5,␣BoldFeatures={FakeBold=0},
              9372     FakeStretch=0.87,␣Ligatures=NoCommon
              9373     ]{TeX␣Gyre␣Cursor}%
 \marginpartt 9374     \def\marginpartt{\tt\addfontfeature{FakeBold=2,
              9375         FakeStretch=0.609}%
              9376       \color{black}}% to provide proper color when marginpar occurs between lines that break a coloured text.
\narrativett 9378     \def\narrativett{\ttfamily\addfontfeature{FakeBold=1}}%
              9379     \let\UrlFont\narrativett
              9380   }% of \AtEndOfClass.
              9381 }% of the cursor option.
```

```
fontspec 9384 \gm@DOX{fontspec}{\PassOptionsToPackage{#1}{fontspec}}
         9388 \gm@EOX{class=mwart}% We set the default basic class to be mwart.
\if@gmcc@tikz@ 9394 \newif\if@gmcc@tikz@
      tikz 9395 \gm@DOX{tikz}{\@gmcc@tikz@true}
         9397 \PassOptionsToPackage{countalllines}{gmdoc}%
         9401 \DeclareOptionX*{\PassOptionsToPackage{\CurrentOption}{gmdoc}}
         9404 \ProcessOptionsX[gmcc]<>
         9407 \long\def\@gobble#1{}
\@firstofone 9408 \long\def\@firstofone#1{#1}
         9410 \if@gmcc@tikz@\expandafter\@firstofone\else\expandafter%
                  \@gobble\fi
         9411 {\RequirePackage{tikz}}
         9426 \ifgmcc@mwcls
         9427   \IfFileExists{\gmcc@CLASS.cls}{}{\gmcc@mwclsfalse}% As announced,
                    we do the ontological test to any mwcls.
         9429 \fi
         9430 \ifgmcc@mwcls
         9434   \LoadClass[fleqn,␣oneside,␣noindentfirst,␣11pt,␣
                    withmarginpar,
         9435   sfheadings]{\gmcc@CLASS}%
         9438 \else
         9439   \LoadClass[fleqn,␣11pt]{article}% Otherwise the standard article is loaded.
         9441 \fi
         9448 \RequirePackage[mw=on]{gmutils}[2008/10/08]% we load it early to provide
                  % \@ifXeTeX, but after loading the base class since this package redefines
                  some environments.
         9452 \ifgmcc@mwcls\afterfi\ParanoidPostsec\fi
         9455 \@ifXeTeX{}{\gmcc@oldfontstrue}
         9458 \AtBeginDocument{\mathindent=\CodeIndent}
```

The `fleqn` option makes displayed formulæ be flushed left and `\mathindent` is their indentation. Therefore we ensure it is always equal `\CodeIndent` just like `\left|skip` in `verbatim`. Thanks to that and the `\edverbs` declaration below you may display single verbatim lines with `\[...\]`:

```
\[|\verbatim\stuff|\].
```

```
         9466 \ifgmcc@oldfonts
         9467   \IfFileExists{lmodern.sty}{% We also examine the ontological status of
                    this package
         9469     \RequirePackage{lmodern}% and if it shows to be satisfactory (the package
                      shows to be), we load it and set the proper font encoding.
         9472     \RequirePackage[T1]{fontenc}%
         9473   }{}%
```

A couple of diacritics I met while gmdocing these files and The Source etc. Some why the accents didn't want to work at my XeTeX settings so below I define them for XeTeX as respective chars.

```
\agrave  9477   \def\agrave␣␣{\`a}%
```

```
\cacute 9478   \def\cacute␣␣{\'c}%
\eacute 9479   \def\eacute␣␣{\'e}%
\idiaeres 9480   \def\idiaeres{\"\i}%
\nacute 9481   \def\nacute␣␣{\'n}%
\ocircum 9482   \def\ocircum␣{\^o}%
\oumlaut 9483   \def\oumlaut␣{\"o}%
\uumlaut 9484   \def\uumlaut␣{\"u}%
9485 \else% this case happens only with XeTeX.
9486   \let\do\relaxen
9487   \do\Finv\do\Game\do\beth\do\gimel\do\daleth% these five caused the
          'already defined' error.
9489   \let\@zf@euenctrue\zf@euencfalse
9490    \XeTeXthree%
\agrave 9491   \def\agrave␣␣{\char"00E0␣}%
\cacute 9492   \def\cacute␣␣{\char"0107␣}% Note the space to be sure the number ends
          here.
\eacute 9494   \def\eacute␣␣{\char"00E9␣}%
\idiaeres 9495   \def\idiaeres{\char"00EF␣}%
\nacute 9496   \def\nacute␣␣{\char"0144␣}%
\oumlaut 9497   \def\oumlaut␣{\char"00F6␣}%
\uumlaut 9498   \def\uumlaut␣{\char"00FC␣}%
\ocircum 9499   \def\ocircum␣{\char"00F4␣}%
9500   \AtBeginDocument{%
\ae 9501     \def\ae{\char"00E6␣}%
9502     \def\l␣{\char"0142␣}%
\oe 9503     \def\oe{\char"0153␣}%
9504   }%
9505 \fi
```

Now we set the page layout.

```
9508 \RequirePackage{geometry}
\gmdoccMargins@params 9509 \def\gmdoccMargins@params{{top=77pt,␣height=687pt,␣% =53 lines but
          the lines option seems not to work 2007/11/15 with TeX Live 2007 and
          XeTeX0.996-patch1
9512     left=4cm,␣right=2.2cm}}
\gmdoccMargins 9513 \def\gmdoccMargins{%
9514   \@xa␣␣\newgeometry\gmdoccMargins@params}
9516 \@xa\geometry\gmdoccMargins@params
9519 \if@debug% For debugging we load also the trace package that was very helpful to
          me.
9521   \RequirePackage{trace}%
9522   \errorcontextlines=100␣% And we set an error info parameter.
9523 \fi
\ifdtraceon 9525 \newcommand*\ifdtraceon{\if@debug\afterfi\traceon\fi}
\ifdtraceoff 9526 \newcommand*\ifdtraceoff{\if@debug\traceoff\fi}
```

We load the core package:

```
9529 \RequirePackage{gmdoc}
9531 \ifgmcc@oldfonts
9532   \@ifpackageloaded{lmodern}{% The Latin Modern font family provides a light
          condensed typewriter font that seems to be the most suitable for the margin-
          par CS marking.
```

```
\marginpartt 9535     \def\marginpartt{\normalfont\fontseries{lc}\ttfamily}}{}%
             9536 \else
\marginpartt 9537   \def\marginpartt{\fontspec{LMTypewriter10␣LightCondensed}}%
             9538 \fi
             9544 \raggedbottom
             9546 \setcounter{secnumdepth}{0}% We wish only the parts and chapters to be
                      numbered.
 \thesection 9549 \renewcommand*\thesection{\arabic{section}}% isn't it redundant at the
                      above setting?
             9552 \@ifnotmw{}
             9553 {% if MW class
             9554   \@ifclassloaded{mwart}
             9555   {% We set the indentation of Contents:
             9556     \SetTOCIndents{{}{\quad}{\quad}{\quad}{\quad}{\quad}{%
                          \quad}}%
             9557   }
             9558   {% for mwart ...
             9559     \SetTOCIndents{{}{\bf9.\enspace}{\quad}{\quad}{\quad}{%
                          \quad}{\quad}}%
             9560   }% and for the two other mwcls.
             9561   \pagestyle{outer}}% We set the page numbers to be printed in the outer and
                        bottom corner of the page.
 \titlesetup 9564 \def\titlesetup{\bfseries\sffamily}% We set the title(s) to be boldface
                      and sans serif.
             9567 \if@gmccnochanges\let\RecordChanges\relax\fi% If the nochanges op-
                      tion is on, we discard writing out to the .glo file.
             9570 \RecordChanges% We turn the writing the \changes out to the .glo file if not the
                      above.
```

Necessarily before recatcode'ing of L|Land \[\].

```
             9574 \RequirePackage{amsfonts}
             9575 \RequirePackage[intlimits]{amsmath}
             9576 \RequirePackage{amssymb}
             9580 \dekclubs*% We declare the club sign | to be a shorthand for \verb*.
             9584 \edverbs% to redefine \[ so that it puts a shortverb in a \hbox.
             9585 \smartunder% and we declare the _ char to behave as usual in the math mode and
                      outside math to be just an underscore.
             9588 \exhyphenpenalty\hyphenpenalty% 'cause mwcls set it =10000 due to Polish
                      customs.
  \EOFMark 9591 \def\EOFMark{\rightline{\ensuremath{\square}}}
             9593 \DoNotIndex{\@nx␣\@xa␣%
             9594 }
       \ac   9596 \provide\ac{\acro}
        \+   9599 \def\+{\-\penalty\@M\hskip\z@}␣% a discretionary hyphen that allows fur-
                      ther hyphenation
             9602 \Xedekfracc
```

```
\let\mch\metachar

\ATfootnotes
\AtBegInput{\ATfootnotes}

\UrlFix

\GMverbatimspecials

\texcode@hook \def\texcode@hook{\makestarlow}

\let\lv\leavevmode
\CommandLet\ac\acro

\OK \def\OK{\acro{OK}\spifletter}

(

\pdef\oczko{;-)\spifletter}

</docc>
```

The gmoldcomm package

```
<*oldcomm>
```

Scan CSs and put them in tt. If at beginning of line, precede them with `%`. Obey lines in the commentary.

```
oldcomments \newenvironment{oldcomments}{%
  \catcode`\\=\active
  \let\do\@makeother
  \do\$% Not only CSs but also special chars occur in the old comments.
  \do\|\do\#\do\{\do\}\do\^\do\_\do\&%
  \gmoc@defbslash
  \obeylines
  \Store@Macro\finish@macroscan
\finish@macroscan \def\finish@macroscan{%
    \@xa\gmd@ifinmeaning\macro@pname\of\gmoc@notprinted%
    {}{{\tt\ifvmode\%\fi\bslash\macro@pname}}%
    \gmoc@checkenv
  }%
}{}

{\escapechar\m@ne
\xdef\gmoc@notprinted{\string\begin,\string\end}}

\gmoc@maccname \def\gmoc@maccname{macrocode}
 \gmoc@ocname \def\gmoc@ocname{oldcomments}

\foone{%
  \catcode`\[=1␣\catcode`\]=2
  \catcode`\{=12␣\catcode`\}=12␣}
\gmoc@checkenv [\def\gmoc@checkenv[%
  \@ifnextchar{%
    [\gmoc@checkenvinn][]]%
\gmoc@checkenvinn \def\gmoc@checkenvinn{#1}[%
      \gmoc@resa \def\gmoc@resa[#1]%
```

```
  \ifx\gmoc@resa\gmoc@maccname
    \def\next[%
      \begingroup
\@currenvir     \def\@currenvir[macrocode]%
      \Restore@Macro\finish@macroscan
      \catcode`\\=\z@
      \catcode`\{=1 \catcode`\}=2
      \macrocode]%
  \else
    \ifx\gmoc@resa\gmoc@ocname
      \def\next[\end[oldcomments]]%
    \else
      \def\next[%
        \{#1\}%
      ]%
    \fi
  \fi
  \next]%
]

\foone{%
  \catcode`\/=\z@
  \catcode`\\=\active}
\gmoc@defbslash {/def/gmoc@defbslash{%
    /let\/scan@macro}}

\task \def\task#1#2{}

</ oldcomm>
<*docstrip>
```

A driver file to typeset dostrip.dtx with the gmdoc package.
GM 2006/12/1

```
\PassOptionsToPackage{%
     countalllines,codespacesgrey,indexallmacros}{gmdoc}

\if11
  \documentclass[debug, pagella, fontspec=quiet]{gmdocc}%
  \mcdiagOn
\else
  \documentclass[pagella]{gmdocc}%
\fi

\ltxLookSetup
\gmdoccMargins
\twocoltoc% For towocolumn table of contents.

\DeleteShortVerb\|
\OldMakeShortVerb*\|% To define shortverb | such that it remains shortverb in
      math mode (by default I define it to be | in math mode.

\relaxen\ds
\emptify\EOFMark

\fooatletter{%
  \@ifXeTeX{%
    \let\gm@TrueAcute\'
```

```
    \def\'#1{%
      \ifx\f@family\rmdefault
        \if␣n#1\nacute
        \else\typeout{******␣\cs{'}␣with␣argument␣}\show#1
        \fi
      \else
        \gm@TrueAcute#1%
      \fi
    }}{}}

\HideAllDefining

\begin{document}

\BasePath
\def\BasePath{/home/natror/texmf/source/latex/base/}
  \addtomacro\endabstract{\aftergroup\tableofcontents}
  \AtBegInputOnce{\date{Printed␣\today\\␣with␣\pk{gmdoc}␣
       package␣by
      Natror}\let\date\gobble
    \let\renewenvironment\gobbletwo}% the only renewed env. in docstrip.
         dtx is theglossary. I prefer it to be twocolumn.

\BasePathdocstrip.dtx
  \OldDocInput{\BasePath␣docstrip.dtx}

  \typeout{%
    ^^JProduce␣change␣log␣with^^J%
    makeindex␣-r␣-s␣gmglo.ist␣-o␣\jobname.gls␣\jobname.glo^^J}

  \typeout{%
    ^^JProduce␣index␣with^^J%
    makeindex␣-r␣\jobname.idx^^J}

\end{document}

⟨/ docstrip⟩
⟨*LaTeXsource⟩
```

Some Typesetting Remarks

This driver typesets The Source 2_ε included in the TEXLive 2005 distribution. Some tricks here are done just for fixing typos in the Source Files. The Source Files themselves are intact.

Most probably you should redefine the `\BasePath` macro so that it was the path of the `\dots/source/latex/base` directory on your system. The path levels should be separated with slashes (even on Windows) and should also end with a slash (to concatenate well with the file name).

While TEXing The Source again after a fatally erroneous pass there happened the 'TEX capacity exceded error' sometimes. TEXing once again was the right thing to do.

The hyperref package usually issues some warnings about non existence of some hypertargets. I consider it rather a feature of hyperref (a bug?) than a bug in the typeset file(s).

One more thing you shouldn't bother of is the differences of the checksums, I mean the usual gmdoc message that the checksum stated in the file differs from gmdoc's own count. That is O.K. since the checksum stated in a traditional .dtx is the number of

backslashes in the macrocodes while the checksum handled and expected by gmdoc is the number of *the escape chars*. Don't get the difference? Assume the declared code escape char is \ (as usual) and consider \\ in the code. Due to the traditional counting this CS increases the checksum by 2 while due to mine by 1: the second bslash is *not* escape char: it's the CS name.

Moreover, when you declare \CodeEscapeChar\! e.g., the code

```
!Alice␣!\!has␣!an␣!aligator
```

increases the 'new way' checksum by 5 not by 1 as it would do the traditional one.

This driver uses an unofficial little package gmeometric to allow the \geometry command also inside document. This package is included in the drivers' directory.

The Body

"This document will typeset the LaTeX sources as a single document. This will produce quite a large file (roughly 555 pages) and may take a long time.

Some notes on processing this document are contained at the end of this document's source file, after \end{document} (not typeset)."

First a special index style for makeindex.

```
\begin{filecontents}{gmsource2e.ist}
preamble
"\n␣\\begin{theindex}␣\n"
postamble
"\n\n␣\\end{theindex}\n"
```

file. May they be cursed!

```
heading_prefix␣␣␣"{\\bfseries\\hfill␣"
heading_suffix␣␣␣"\\hfill}\\nopagebreak\n"
headings_flag␣␣␣␣␣␣␣1
```

and just for source2e:
Remove R so I is treated in sequence I J K not I II III

```
page_precedence␣"rnaA"
\end{filecontents}
```

```
\PassOptionsToPackage{codespacesgrey,␣indexallmacros}{gmdoc}
```

```
\if11
  \documentclass[debug,␣minion,␣cronos,␣cursor,␣
       fontspec=quiet]{gmdocc}%
```

```
  \mcdiagOn
```

```
\else
  \documentclass[fontspec=quiet]{gmdocc}%
\fi
```

```
\foone{\catcode`_=12␣}
{\if1␣1\includeonly{source2e_by_gmdoc}\fi}
```

```
\usepackage{gmoldcomm}% Definitions of oldcomments and \task.
```

```
\listfiles
```

```
\ltxLookSetup
```

```
\gmdoccMargins
\olddocIncludes% This is the crucial declaration to drive gmdoc into the traditional settings.
\twocoltoc% For towocolumn table of contents.

\DeleteShortVerb\|
\OldMakeShortVerb*\|% To define shortverb | such that it remains shortverb in math mode (by default I define it to be | in math mode.
```

Do not index some TeX primitives, and some common plain TeX commands.

```
\DoNotIndex{\def,\long,\edef,\xdef,\gdef,\let,\global}
\DoNotIndex{\if,\ifnum,\ifdim,\ifcat,\ifmmode,\ifvmode,%
    \ifhmode,%
        \iftrue,\iffalse,\ifvoid,\ifx,\ifeof,\ifcase,%
            \else,\or,\fi}
\DoNotIndex{\box,\copy,\setbox,\unvbox,\unhbox,\hbox,%
        \vbox,\vtop,\vcenter}
\DoNotIndex{\@empty,\immediate,\write}
\DoNotIndex{\egroup,\bgroup,\expandafter,\begingroup,%
    \endgroup}
\DoNotIndex{\divide,\advance,\multiply,\count,\dimen}
\DoNotIndex{\relax,\space,\string}
\DoNotIndex{\csname,\endcsname,\@spaces,\openin,\openout,%
        \closein,\closeout}
\DoNotIndex{\catcode, endinput}
\DoNotIndex{\jobname,\message,\read,\the,\m@ne,\noexpand}
\DoNotIndex{\hsize,\vsize,\hskip,\vskip,\kern,\hfil,\hfill,%
    \hss}
\DoNotIndex{\m@ne,\z@,\z@skip,\@ne,\tw@,\p@}
\DoNotIndex{\dp,\wd,\ht,\vss,\unskip}
```

Set up the Index and Change History to use `\part`.

```
\makeatletter
\def\indexdiv{\part*}
\AtDIPrologue{\@ifnotmw{%
    \markboth{Index}{Index}%
    \addcontentsline{toc}{part}{Index}}{}%
}
\GlossaryPrologue{\part*{Change␣History}%
```

Allow control names to be hyphenated here…

```
  {\GlossaryParms\ttfamily\hyphenchar\font=`\-}%
  \@ifnotmw{%
    \markboth{Change␣History}{Change␣History}%
    \addcontentsline{toc}{part}{Change␣History}}{}%
}
```

"The standard `\changes` command modified slightly to better cope with this multiple file document."— Not quite:

```
\makeatletter
 % \def\changes@#1#2#3{%
 %   \let\protect\@unexpandable@protect
```

```
%   \edef\@tempa{\noexpand\glossary{#2\space\currentfile%
    \space#1\levelchar
%                                       \ifx\saved@macroname\@empty
%                                         \space
%                                         \actualchar
%                                         \generalname
%                                       \else
%                                         \expandafter\@gobble
%                                         \saved@macroname
%                                         \actualchar
%                                         \string\verb\quotechar*%
%                                     \verbatimchar\saved@macroname
%                                         \verbatimchar
%                                       \fi
%                                       :\levelchar #3}}%
%   \@tempa\endgroup\@esphack}
```

```
\makeatother
```

Produce a Change Log and (2 column) Index.

```
\RecordChanges
\CodelineIndex
\EnableCrossrefs
\setcounter{IndexColumns}{2}
```

Needed for documentation in ltoutenc.dtx.

```
\usepackage{textcomp}
```

```
\olddocIncludes
\HideAllDefining
```

```
\fooatletter{%
  \@ifXeTeX{%
    \def\"#1{%
      \if␣o#1\oumlaut\fi
      \if␣u#1\uumlaut\fi
    }}{}}
```

```
\foone{\makeatletter\catcode`\#=12␣}{%
```

\gmd@wykrzykniki
```
  \def\gmd@wykrzykniki{#␣#␣#␣#␣#␣#␣#␣#}}
```

```
\begin{document}
```

```
 \title{The␣\LaTeXe\␣Sources\thanks{Typeset␣with␣\pk{gmdoc}␣
      by␣Natror
     on␣\today.}}
 \author{%
  Johannes␣Braams\\
  David␣Carlisle\\
  Alan␣Jeffrey\\
  Leslie␣Lamport\\
  Frank␣Mittelbach\\
  Chris␣Rowley\\
  Rainer␣Sch\"opf}
```

\BasePath
```
\def\BasePath{/home/natror/texmf/source/latex/base/}
```

This command will be used to input the patch file if that file exists.

```
\includeltpatch 10036 \newcommand{\includeltpatch}{%
  \currentfile 10037   \def\currentfile{ltpatch.ltx}
          10038   \part{ltpatch}
          10039   {\let\ttfamily\relax
          10040     \xdef\filekey{\filekey,␣\thepart={\ttfamily%
                         \currentfile}}}%
          10041   Things␣we␣did␣wrong\ldots
          10042   \IndexInput{ltpatch.ltx}}
```

Get the date from ltvers.dtx

```
          10047 \makeatletter
          10048 \let\patchdate=\@empty
          10049 \begingroup
\ProvidesFile 10050    \def\ProvidesFile#1\fmtversion#2{\date{#2} endinput}
          10051    \input{\BasePath␣ltvers.dtx}
          10052 \global\let\X@date=\@date
```

Add the patch version if available.

```
      \Xdef 10055    \long\def\Xdef#1#2#3\def#4#5{%
          10056     \xdef\X@date{#2}%
          10057     \xdef\patchdate{#5}%
          10058     endinput}%
          10059    \InputIfFileExists{ltpatch.ltx}
      \Xdef 10060     {\let\def\Xdef}{\global\let\includeltpatch\relax}
          10061 \endgroup

          10063 \ifx\@date\X@date
     \Xpatch 10064    \def\Xpatch{0}
          10065    \ifx\patchdate\Xpatch\else
          10066      \edef\@date{\@date\space␣Patch␣level␣\patchdate}
          10067    \fi
          10068 \else
          10069    \@warning{ltpatch.ltx␣does␣not␣match␣ltvers.dtx!}
          10070    \let\includeltpatch\relax
          10071 \fi
          10072 \makeatother

          10074 \pagenumbering{roman}
          10075 \thispagestyle{empty}

          10078 \maketitle
          10079 \relax
          10081 \emptify\maketitle

          10083 \tableofcontents

          10085 \clearpage

          10087 \pagenumbering{arabic}
```

"Each of the following `\DocInclude` lines includes a file with extension `.dtx`. Each of these files may be typeset separately. For instance

```
latex␣ltboxes.dtx
```

will typeset the source of the LaTeX box commands."

(Well, I (Natror) prepared only this common driver.)

If this file is processed, each of these separate .dtx files will be contained as a part of a single document. Using ltxdoc.cfg you can then optionally produce a combined index and/or change history for the entire source of the format file. Note that such a document will be quite large (about 555 pages).

```
\DocInclude[\BasePath]{ltdirchk}␣% System dependent initialisation
\AfterMacrocode{53}{\def\do{\cs{do}}}% A bare \do in narration on line 161.
\DocInclude[\BasePath]{ltplain}␣␣% LaTeX version of Knuth's plain.tex.
\DocInclude[\BasePath]{ltvers}␣␣␣% Current version date.
\DocInclude[\BasePath]{ltdefns}␣␣% Initial definitions.
\DocInclude[\BasePath]{ltalloc}␣␣% Allocation of counters and others.
\DocInclude[\BasePath]{ltcntrl}␣␣% Program control macros.
\DocInclude[\BasePath]{lterror}␣␣% Error handling.
\DocInclude[\BasePath]{ltpar}␣␣␣␣% Paragraphs.
\DocInclude[\BasePath]{ltspace}␣␣% Spacing, line and page breaking.
\DocInclude[\BasePath]{ltlogos}␣␣% Logos.
\DocInclude[\BasePath]{ltfiles}␣␣% \input files and related commands.
\AtBegInputOnce{\let\task\gobble}% In general \task gobbles two, but in this file it's used with one argument and next to it is \changes (which in gmdoc is \outer so gobbling it raises an error).
\DocInclude[\BasePath]{ltoutenc}␣% Output encoding interface.
\DocInclude[\BasePath]{ltcounts}␣% Counters.
\DocInclude[\BasePath]{ltlength}␣% Lengths.
\DocInclude[\BasePath]{ltfssbas}␣% NFSS Base macros.
\DocInclude[\BasePath]{ltfsstrc}␣% NFSS Tracing (and tracefnt.sty).
\DocInclude[\BasePath]{ltfsscmp}␣% NFSS1 Compatibility.
\DocInclude[\BasePath]{ltfssdcl}␣% NFSS Declarative interface.
\DocInclude[\BasePath]{ltfssini}␣% NFSS Initialisation.
\DocInclude[\BasePath]{fontdef}␣␣% fonttext.ltx/fontmath.ltx
\DocInclude[\BasePath]{preload}␣␣% preload.ltx
\DocInclude[\BasePath]{ltfntcmd}␣% \textrm etc.
\DocInclude[\BasePath]{ltpageno}␣% Page numbering.
\DocInclude[\BasePath]{ltxref}␣␣␣% Cross referencing.
\AfterMacrocode{1137}{\let\GMDebugCS\cs
\cs  \def\cs##1{\expandafter\GMDebugCS\expandafter{\string##1}}}
```

\cs{\@defaultsubs} on line 257, \cs{\@refundefined} on line 263. It's the first step. The next is done before \PrintChanges.

```
\AfterMacrocode{1139}{\let\cs\GMDebugCS}
\AfterMacrocode{1183}{% The last \changes have second argument {1994/05/26/16}.
```

```
    \csname␣changes\endcsname{v0.9i}{1993/12/16}{\cs{literal}␣
        added}%
    \csname␣changes\endcsname{v1.0r}{1994/05/26}{\cs{literal}␣
        removed}%
    \gdef\GMdebugChanges{\expandafter\def\csname
      changes\endcsname####1####2####3{}}%
    \aftergroup\GMdebugChanges}% A trick with \aftergroup 'cause that
        macrocode is inside macro.
\DocInclude[\BasePath]{ltmiscen}␣% Miscellaneous environment defini-
    tions.
\DocInclude[\BasePath]{ltmath}␣␣␣% Mathematics set up.
\DocInclude[\BasePath]{ltlists}␣␣% List and related environments.
\DocInclude[\BasePath]{ltboxes}␣␣% Parbox and friends.
\DocInclude[\BasePath]{lttab}␣␣␣␣% tabbing, tabular and array.
\DocInclude[\BasePath]{ltpictur}␣% Picture mode.
\DocInclude[\BasePath]{ltthm}␣␣␣␣% Theorem environments.
\DocInclude[\BasePath]{ltsect}␣␣␣% Sectioning.
\DocInclude[\BasePath]{ltfloat}␣␣% Floats.
\DocInclude[\BasePath]{ltidxglo}␣% Index and Glossary.
\DocInclude[\BasePath]{ltbibl}␣␣␣% Bibliography.
\DocInclude[\BasePath]{ltpage}␣␣␣% \pagestyle, \raggedbottom, \sloppy.
\DocInclude[\BasePath]{ltoutput}␣% Output routine.
\DocInclude[\BasePath]{ltclass}␣␣% Package & Class interface.
\DocInclude[\BasePath]{lthyphen}␣% Hyphenation (hyphen.ltx).
\DocInclude[\BasePath]{ltfinal}␣␣% Last minute initialisations and dump.
\includeltpatch␣␣␣␣␣␣␣␣% Corrections distributed after the full release.
```

Stop here if ltxdoc.cfg says `\AtEndOfClass{\OnlyDescription}`

```
\StopEventually{\end{document}}
```

```
\clearpage
\pagestyle{headings}
```

Make TEX shut up.

```
\hbadness=10000
\newcount\hbadness
\hfuzz=\maxdimen
```

```
\typeout{%
  ^^JProduce␣change␣log␣with^^J%
  makeindex␣-r␣-s␣gmglo.ist␣-o␣\jobname.gls␣\jobname.glo^^J}
```

```
{% The next step of debug of ltmiscen.dtx's \changes...{...\cs{\@default|
      subs...}} etc.
   How does it work? Remember \cs is robust. The typo lies in giving it a CS
   argument instead of expected CS name without backslash. So, in the step
```

1 we only \string the argument CS to let it be written outto the .glo file. Then, in step 2, we redefine \cs to first \string its argument inside an \if. Remember that \if expands two tokens next to it until it finds sth. unexpandable, so it'll execute \string. Then, if the first char of the \stringed argument is \₁₂, the condition is satisfied and \if...\fi expands to what follows that backslash and precedes \else. So if the argument was a CS, its backslash will be gobbled by \if. Otherwise \if...\fi expands to what is between \else and \fi, to the unchanged argument that is. Then to that list of tokens the original \cs is applied.

```
10239  \let\GMDebugCS\cs
\cs 10240  \def\cs#1{\GMDebugCS{\if\bslash\string#1\else#1\fi}}%
10241  \PrintChanges}

10243 \typeout{%
10244   ^^JProduce␣index␣with^^J%
10245   makeindex␣-r␣-s␣gmsource2e.ist␣\jobname.idx^^J}
```

"Makeindex needs a symbol between the parts of composite page numbers but we dont want one, so:"—I skip that.

```
% \begingroup
% \def\endash{--}
% \catcode`\-\active
% \def-{\futurelet\temp\indexdash}
% \def\indexdash{\ifx\temp-\endash\fi}

10257 \geometry{bottom=3.6cm}
10258 \clearpage

10263 \PrintIndex
```

Make sure that the index is not printed twice (ltxdoc.cfg might have a second

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 preceded with 'p.' are page numbers. All the numbers are hyperlinks.

command)

```
\let\PrintChanges\relax
\let\PrintIndex\relax

\gmdoccMargins
\clearpage
```

```
\csname␣@ifnotmw\endcsname{\pagestyle{headings}}{\pagestyle{%
      outer}}

\gmdocIncludes
\SelfInclude{%
  \csname␣gag@index\endcsname% we turn writing outto the .idx out for the
       driver since it's not a part of The Source.
}

\end{document}

To␣use␣this␣file␣to␣produce␣a␣fully␣indexed␣source␣code
you␣need␣to␣execute␣the␣following␣(or␣equivalent)␣commands:

   latex␣source2e_by_gmdoc.tex

   makeindex␣-s␣gmsource2e.ist␣source2e_by_gmdoc.idx
   makeindex␣-s␣gmglo.ist␣-o␣source2e_by_gmdoc.gls␣
        source2e_by_gmdoc.glo

   latex␣source2e_by_gmdoc.tex
   latex␣source2e_by_gmdoc.tex

The␣makeindex␣style␣gmsource2e.ist␣is␣used␣in␣place␣of␣the␣
      usual
doc␣gind.ist␣to␣ensure␣that␣I␣is␣used␣in␣the␣sequence␣I␣J␣K
not␣I␣II␣II,␣which␣would␣be␣the␣default␣makeindex␣behaviour.

The␣third␣run␣with␣latex␣is␣only␣required␣to␣get␣the␣table␣of
contents␣entries␣for␣the␣change␣log␣and␣index.␣You␣may␣speed␣
      things␣up
by␣using␣the␣\includeonly␣mechanism␣so␣as␣not␣to␣typeset␣the␣
      source
files␣on␣the␣second␣run.␣This␣involves␣changing␣the␣file
ltxdoc.cfg
between␣the␣latex␣runs.

The␣following␣unix␣script␣automates␣this.
  (It␣could␣easily␣be␣ported␣to␣scripts␣for␣DOS␣or␣VMS,
   rm␣is␣ReMove␣a␣file,␣and␣echo␣"..."␣>␣file␣writes␣...␣to␣
        "file".)

After␣this␣script␣(after␣the␣second␣==========)␣is␣a␣
      similar␣script
that␣will␣produce␣the␣documentation␣for␣all␣the␣files␣in␣the␣
      base
distribution␣that␣are␣*not*␣included␣in␣source2e.dvi.␣This␣
      second␣script
was␣requested,␣but␣before␣using␣it,␣beware␣it␣will␣take␣a␣
      long␣time!
It␣may␣however␣be␣modified␣as␣required,␣eg␣to␣not␣typeset␣the␣
      fdd␣files
or␣whatever...

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Natror␣(GM):␣I~didn't␣touch␣the␣following␣so␣it's␣probably␣
      not␣quite␣suitable
for␣gmdoc-ing.
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

==============
#!/bin/sh

rm␣␣-f␣source2e_by_gmdoc.gls␣source2e_by_gmdoc.ind␣source2e_by_gmdoc.toc

#␣First␣run:
#␣Create␣new␣standard␣ltxdoc.cfg␣file
#␣Pass␣the␣(possibly␣empty)␣list␣of␣arguments␣supplied␣on␣the
#␣command␣line␣to␣article␣class.
#
#␣If␣you␣use␣A4␣paper,␣running␣this␣script␣with␣argument
#␣␣␣␣␣a4paper
#␣may␣save␣about␣30␣pages.
#
echo␣"\PassOptionsToClass{$*}{article}"␣>␣ltxdoc.cfg

#␣Now␣LaTeX␣the␣file␣with␣this␣cfg␣file.
#
latex␣source2e.tex

#␣Make␣the␣Change␣log␣and␣Glossary.
#
makeindex␣-s␣source2e.ist␣source2e.idx
makeindex␣-s␣gglo.ist␣-o␣source2e.gls␣source2e.glo

#␣Second␣run:␣append␣\includeonly{}␣to␣ltxdoc.cfg␣to␣speed␣up␣things
#␣(this␣run␣needed␣only␣to␣get␣changes␣and␣index␣listed␣in␣.toc␣file)
#
#␣Note␣that␣the␣index␣will␣not␣be␣made␣incorrect␣by␣the␣insertion
#␣of␣the␣table␣of␣contents␣as␣the␣front␣matter␣uses␣a␣diferent␣page
#␣numbering␣scheme.
#
echo␣"\includeonly{}"␣>>␣ltxdoc.cfg

latex␣source2e.tex

#␣Third␣and␣final␣run,␣to␣put␣everything␣together.
#␣First␣restore␣the␣cfg␣file:
#
echo␣"\PassOptionsToClass{$*}{article}"␣>␣ltxdoc.cfg
latex␣source2e.tex

==============
#!/bin/sh

#␣Running␣this␣script␣will␣process␣all␣the␣dtx␣fdd␣and␣*guide.tex
#␣and␣ltnews*.tex␣files␣in␣the␣LaTeX␣distribution,␣except␣the␣dtx
#␣files␣included␣in␣source2e.tex.
#␣(The␣shell␣first␣script␣in␣the␣comments␣of␣source2e.tex␣will
```

```
#␣␣process␣those.)

#␣Any␣command␣line␣arguments␣(eg␣a4paper)␣are␣taken␣as␣options␣to␣the
#␣article␣class.

#␣This␣script␣is␣likely␣to␣take␣ages!

echo␣"\PassOptionsToClass{$*}{article}"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>␣ltxdoc.cfg
echo␣"\batchmode"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>>␣ltxdoc.cfg

#␣The␣next␣four␣lines␣produce␣full␣indexes␣and␣change␣logs
#␣you␣may␣not␣want␣those.
echo␣"\AtBeginDocument{\RecordChanges}"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>>␣ltxdoc.cfg
echo␣"\AtEndDocument{\PrintChanges}"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>>␣ltxdoc.cfg
echo␣"\AtBeginDocument{\CodelineIndex\EnableCrossrefs}"␣>>␣ltxdoc.cfg
echo␣"\AtEndDocument{\PrintIndex}"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>>␣ltxdoc.cfg

#␣If␣you␣do␣not␣want␣any␣code␣listings,␣just␣documentation,␣then␣instead
#␣of␣the␣above␣four␣lines,␣uncomment␣the␣following:
#␣echo␣"\AtBeginDocument{\OnlyDescription}"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>>␣ltxdoc.cfg

echo␣"\PassOptionsToClass{$*}{article}"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>␣ltxguide.cfg
echo␣"\batchmode"␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣>>␣ltxguide.cfg

cp␣ltxguide.cfg␣ltnews.cfg

for␣i␣in␣*dtx␣*fdd␣*guide.tex␣ltnews*.tex
do
B=`basename␣$i␣.dtx`

if␣(grep␣"Include{$B}"␣source2e.tex␣>/dev/null␣;␣)
then
echo␣In␣source2e:␣$i
else
echo␣latex␣$i
  if␣(latex␣$i␣>␣/dev/null)
  then
    echo␣latex␣$i
    latex␣$i␣>␣/dev/null
    echo␣makeindex␣-s␣gind.ist␣$B.idx
    makeindex␣-s␣gind.ist␣$B.idx␣>␣/dev/null␣2>␣/dev/null
    echo␣makeindex␣-s␣gglo.ist␣-o␣$B.gls␣$B.glo
    makeindex␣-s␣gglo.ist␣-o␣$B.gls␣$B.glo␣>␣/dev/null␣2>␣/dev/null
    echo␣latex␣$i
    latex␣$i␣>␣/dev/null
```

```
  else
    echo␣"!!!␣LaTeX␣ERROR:␣$i.␣(See␣$B.log.)"
  fi
fi

done
⟨/ LaTeXsource⟩
⟨*docbygmdoc⟩
\PassOptionsToPackage{hyperindex=false}{hyperref}%
```
Because FM writes some almost explicit indexing commands where he uses 'encapsulating' i.e., a command to encapsulate the page number, which would interfere with hyperref's default |hyperpage.
```
\documentclass[countalllines,
codespacesblank,␣outeroff,␣pagella,␣cronos,␣cursor,
fontspec=quiet]{gmdocc}
\usepackage{array}

\VisSpacesGrey

\def\BasePath{/home/natror/texmf/source/latex/base/}%
```
\BasePath Of course, you should change it to the respective path on your computer.
```
\ltxLookSetup
\gmdoccMargins
\olddocIncludes%
```
This is the crucial declaration.
```
\twocoltoc

\DeleteShortVerb\|
\OldMakeShortVerb*\|
\HideAllDefining

\makeatletter

\edef\gmd@wykrzykniki{\xiihash\space\xiihash\space}
\edef\gmd@wykrzykniki{\gmd@wykrzykniki\gmd@wykrzykniki}
\edef\gmd@wykrzykniki{\gmd@wykrzykniki\gmd@wykrzykniki}
\edef\gmd@wykrzykniki{\gmd@wykrzykniki\gmd@wykrzykniki}

\author{Frank␣Mittelbach␣\and␣David␣Carlisle}
\title{The␣\pk{doc}␣and␣\pk{shortvrb}␣Packages\\␣and\\
  the␣\pk{ltxdoc}␣Class}
\date{Typeset␣with␣the␣\pk{gmdoc}␣package␣by␣Natror\\\today}

\errorcontextlines=1000
\fooatletter{%
  \typeout{@@@@@␣\meaning\@begindocumenthook}}
\begin{document}

\smartunder

\typeout{@@@@␣in␣document}

\maketitle
\typeout{@@@@␣after␣title}

\addtocontents{toc}{%
```
to discard \begin{multicols}{2} of one included document. (Table of contents is declared twocolumn with \twocoltoc above.)
```
    \let\protect\begin\protect\@gobbletwo
```

```
        \protect\Store@Macro\protect\end
        \def\protect\end{\protect\Restore@Macro\protect\end%
            \protect\@gobble}%
}% Because one document has a multicols twocolumn table of contents and the
    other has usual one column, this will put entire toc in(to) multicols.

\tableofcontents

\makeatletter
\AfterMacrocode{161}{% it's for a tiny little typo in line 3299: They forgot to
    wrap \@tempb and \@tempc in shortverbs.
  \def\@tempb{\cs{@tempb}␣}\def\@tempc{\cs{@tempc}␣}}

\AtBegInputOnce{%
  \chschange{v2.1b}{2006/10/20}{2126}%
  \let\CheckSum\gobble
```

Of course, none of the documents is not loaded, so we give the fileinfo explicitly.

```
  \def\filedate{2004/02/09}\def\fileversion{v2.1b}%
  \let\GetFileInfo\relax
  \addtomacro\IndexParms{\arraybackslash}}% because \IndexParms use
      \raggedright and FM executes \IndexParms inside a tabular.

\DocInclude[\BasePath]{doc}

\AtBegInputOnce{%
  \chschange{v2.0u}{2006/10/20}{410}%
  \let\CheckSum\gobble
  \def\filedate{1999/08/08}␣\def\fileversion{v2.0u}% see line 10514.
  \let\GetFileInfo\relax
```

The rest of this \AtBegInputOnce's contents is necessary since DC wrote it not commented out, which with doc results with printing it both to the package (class) and the documentation, but with gmdoc it puts this stuff in the code layer that'll be only printed verbatim.

```
  \providecommand\dst{\expandafter{\normalfont\scshape␣
      docstrip}}
  \title{The␣file␣\texttt{ltxdoc.dtx}␣for␣use␣with
    \LaTeXe.\thanks{This␣file␣has␣version
      number␣\fileversion,␣dated␣\filedate.}\\[2pt]
    It␣contains␣the␣code␣for␣\texttt{ltxdoc.cls}}
  \date{\filedate}
  \author{David␣Carlisle}
  \maketitle}

\DocInclude[\BasePath]{ltxdoc}%

\gmdocIncludes

\AtBegInputOnce{%
  \title{\pk{doc_by_gmdoc.tex}␣The␣Driver\thanks{As␣mentioned␣
      in␣the
    title,␣I~typeset␣these␣package␣and␣class␣with␣the␣\pk{%
      gmdoc}
    package,␣for␣which␣are␣they␣a~great␣inspiration␣and␣the␣
      base.
```

```
10550       The␣typesetting␣needed␣only␣a~few␣tricks,␣so␣here␣
                i~give␣the
10552       code␣of␣the␣`driver':␣a~snake␣eats␣its␣tail␣;-)␣.}}
10553     \author{Grzegorz␣`Natror'␣Murzynowski}%
10554     \date{\today}%
10555     \maketitle}
10556 \SelfInclude

10558 \typeout{%
10559   Produce␣change␣log␣with^^J%
10560   makeindex␣-r␣-s␣gmglo.ist␣-o␣\jobname.gls␣\jobname.glo^^J
10561   (gmglo.ist␣should␣be␣put␣into␣some␣texmf/makeindex␣
          directory.)^^J}
10562 \PrintChanges
10563 \typeout{%
10564   Produce␣index␣with^^J%
10565   makeindex␣-r␣\jobname^^J}
10566  \PrintIndex

10568 \end{document}
```

MakeIndex shell commands:

```
10572 makeindex␣-r␣doc_gmdoc
10573 makeindex␣-r␣-s␣gmglo.ist␣-o␣doc_gmdoc.gls␣doc_gmdoc.glo
```

_ bf: _ bfseries _

```
10578 ⟨/ docbygmdoc⟩
10579 \endinput
```

End of file ‘gmdoc.gmd’.

⟨*eof*⟩

Change History

gmdoc changed
  \c@ChangesStartDate:
    from TEX’s arithmetic to \numexpr, 6812
gmdoc v0.74
  \edverbs:
    used to simplify displaying shortverbs, 9580
gmdoc v0.75
  General:
    CheckSum 130 , 9056
gmdoc v0.76
  General:
    CheckSum 257 , 9056
  \OK:
    The gmeometric option made obsolete and the gmeometric package is loaded always, for XETEX-compatibility. And the class options go xkeyval., 9623
gmdoc v0.77
  General:
    CheckSum 262 , 9056
  \OK:
    Bug fix of sectioning commands in mwcls and the default font encoding for TEXing old way changed from QX to T1 because of the ‘corrupted NTFS tables’ error, 9623
gmdoc v0.78
  General:
    CheckSum 267 , 9056
  \OK:
    Added the pagella option not to use Adobe Minion Pro that is not freely licensed, 9623
gmdoc v0.79
  General:
    CheckSum 271 , 9056

gmdoc v0.80
General:
CheckSum 275 , 9056
CheckSum 276 , 9056
gmcc@fontspec:
added, 9384
gmdoc v0.81
General:
put to CTAN on 2008/11/22, 9056
gmdoc v0.82
General:
CheckSum 303 , 9056
CheckSum 316 because of \verbatimspecials, hyphenation in verbatims etc., 9056
CheckSum 320 , 9056
\ac:
added, 9596
countalllines:
gmdoc option here executed by default, 9397
gmcc@cronos:
added, for Iwona sans font, 9352
gmcc@cursor:
added, for TEX Gyre Cursor mono font, which I embolden a little and shrink horizontally a little, 9364
subtly distinguished weights of the TEX Gyre Cyursor typewriter font in the code and in verbatims in the commentary, 9364
\gmcc@dff:
I commented out setting of Latin Modern fonts for sans serif and monospaced: XETEX/fontspec does that by default., 9327
gmcc@lsu:
added, for Lucida Sans Unicode sans font, 9360
gmcc@myriad:
added, for Myriad Web Pro sans font, 9357
gmcc@trebuchet:
added, for Trebuchet MS sans font, 9354
\LineNumFont:
added, 9340
gmdoc v0.83
General:
CheckSum 332 because of abandoning gmeometric since geometry v.5.2 provides \newgeometry, 9056
gmdoc v0.96
\gmFileKind:
CheckSum 2395 , 409
gmdoc v0.98d
\c@ChangesStartDate:
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. 94. 6840
gmdoc v0.991
\gmFileKind:
CheckSum 6134 because of compatibilising the enumargs environment with \DeclareCommand of gmutils v.0.991; abandoning gmeometric, 409
put to CTAN on 2010/03/04, 409
gmdoc v0.992
\ds:
\CS etc. definitions moved to gmmeta (part of gmutils), 8200
gmdoc v0.993
General:
CheckSum 7785 , 9056
CheckSum 7786 , 9056
CheckSum 7818 , 9056
\verb@egroup:
due to troubles with bad fontification in the narration layer I implement the counterpart to \narrativett: \codett, which is \tt by default so it even may be transparent to the users., 4101
gmdoc v0.99a
\gmFileKind:
CheckSum 4479 , 409
gmdoc v0.99b
General:
Thanks to the \edverbs declaration in the class, displayed shortverbs simplified; Emacs mode changed to doctex. Author's true name more exposed, 8947
gmdoc v0.99c
General:
A bug fixed in \DocInput and all \expandafters changed to \@xa and \noexpands to \@nx, 8947
The TEX-related logos now are declared with \DeclareLogo provided in gmutils, 8947
\DocInput:
added ensuring the code delimiter to be the same at the end as at the beginning, 2868
\gmd@bslashEOL:
a bug fix: redefinition of it left solely to \QueerEOL, 4049
gmdoc v0.99d
General:
\@namelet renamed to \n@melet to solve a conflict with the beamer class (in gmutils at first), 8947

\afterfi & pals made two-argument, 8947
\FileInfo:
added, 7828
gmdoc v0.99e
General:
a bug fixed in \DocInput and \IndexInput, 8947
\gmFileKind:
CheckSum 4574 , 409
gmdoc v0.99g
General:
The bundle goes XeTeX. The TeX-related logos now are moved to gmutils. ^^A becomes more comment-like thanks to re\catcode'ing. Automatic detection of definitions implemented, 8947
\gmFileKind:
CheckSum 5229 , 409
hyperref:
added bypass of encoding for loading url, 2573
\OldDocInput:
obsolete redefinition of the macro environment removed, 8778
quotation:
added, 8084
gmdoc v0.99h
General:
Fixed behaviour of sectioning commands (optional two heading skip check) of mwcls/gmutils and respective macro added in gmdocc. I made a tds archive, 8947
gmdoc v0.99i
General:
A "feature not bug" fix: thanks to \everyeof the \[No]EOF is now not necessary at the end of \DocInput file., 8947
\gmFileKind:
CheckSum 5247 , 409
gmdoc v0.99j
\gmFileKind:
CheckSum 5266 , 409
quotation:
Improved behaviour of redefined quotation to be the original if used by another environment., 8049
gmdoc v0.99k
\gmFileKind:
CheckSum 5261 , 409
hyperref:
removed some lines testing if XeTeX colliding with tikz and most probably obsolete, 2591
gmdoc v0.99l
\CodeSpacesGrey:
added due to Will Robertson's suggestion, 3109
codespacesgrey:
added due to Will Robertson's suggestion, 2552
\FileInfo:
\scantokens used instead of \write and \@@input which simplified the macro, 7862
\gmd@writeckpt:
Made a shorthand for \Docinclude\jobname instead of repeating 99% of \DocInclude's code, 7551
\gmFileKind:
CheckSum 5225 , 409
macrocode:
removed \CodeSpacesBlank, 5895
gmdoc v0.99m
\@oldmacrocode@launch:
renamed from \VerbMacrocodes, 5989
^^M:
there was \let^^M but \QueerEOL is better: it also redefines \^^M, 2851
General:
Counting of all lines developed (the countalllines package option), now it uses \inputlineno, 8947
\changes:
changed to write the line number instead of page number by default and with codelineindex option which seems to be more reasonable especially with the countalllines option, 5674
\DocInclude:
resetting of codeline number with every \filedivname commented out because with the countalllines option it caused that reset at \maketitle after some lines of file, 7487
\FileInfo:
\egroup of the inner macro moved to the end to allow \gmd@ctallsetup. From the material passed to \gmd@FIrescan ending ^^M stripped not to cause double labels., 7845
\gmd@bslashEOL:
also \StraightEOL with countalllines package option lets \^^M to it, 4049

added and introduced—\code@delim forked to what delimits the code (\code@delim) and what is typeset at the boundary of code: \narrationmark, 2663

\narrativett:

introduced in gmutils and employed in the narrative verbatims, including \ampulexdef of the gmverb macros, 5770

\PrintChanges:

added, 7006

made a shorthand for \chgs not \changes, 6984

\step@checksum:

!*!*!… sequence changed to ! ! !… for better distinction, 7122

added, 7135

\Text@UsgEnvir:

added \@sanitize in the starred version, 5796

\titlesetup:

a bug fixed: \if\relax\@date changed to \ifx, 7712

gmdoc v0.99t

General:

Since geometry v.5.2 gmeometric is obsolete so was removed, 8947

gmdoc v1.0

\gmd@chgs:

made \long (consider it a bug fix), 7012, 7021