Grzegorz `Natror' Murzynowski

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

November 2007

Contents

a. The gmdoc.sty Package[1]

November 19, 2007

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

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

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

```
1 ⟨package⟩
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{gmdoc}
4           [2007/11/12 v0.99h a documenting package (GM)]
```

Readme

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

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

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

The package also works with XeTeX (switches automatically).

Installation

Unpack the gmdoc-tds.zip archive (this is an archive conforming the TDS standard, see CTAN/tds/tds.pdf) in a texmf directory or put the gmdoc.sty, gmdocc.cls and gmoldcomm.sty somewhere in the texmf/tex/latex branch on your own. (Creating a texmf/tex/latex/gm directory may be advisable if you consider using other packages written by me.)

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

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

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

[1] This file has version number v0.99h dated 2007/11/12.

Contents of the gmdoc.zip Archive

The distribution of the gmdoc package consists of the following six files and a TDS-compliant archive.

gmdoc.sty
gmdocc.cls
gmglo.ist
README
gmdocDoc.tex
gmdocDoc.pdf
gmdoc.tds.zip

Compiling the Documentation

The last of the above files (the .pdf, i.e., *this file*) is a documentation compiled from the .sty and .cls files by running XeLaTeX on the gmdocDoc.tex twice, then MakeIndex on the gmdocDoc.idx and gmdocDoc.glo files, and then XeLaTeX on gmdocDoc.tex once more. (Using LaTeX instead of XeLaTeX should do, too.)

MakeIndex shell commands:

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

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

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

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

Dependencies

The gmdoc bundle depends on some other packages of mine:

gmutils.sty,
gmverb.sty,
gmiflink.sty
gmeometric (for the driver of The LaTeX 2_ε Source)

and also on some standard packages:

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

that should be installed on your computer by default.

Bonus: base Drivers

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

source2e_gmdoc.tex
docstrip_gmdoc.tex
doc_gmdoc.tex

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 TEXLive2005 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 CSs in the code maybe in the future.)

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

The doc package was and still is a great inspiration for me and I would like this humble package to be considered as a sort of hommage to it[2]. If I mention copying some code or narrative but do not state the source explicitly, I mean the doc package's documentation (I have v2.1b dated 2004/02/09).

The User Interface

Used Terms

When I write of a **macro**, I mean a macro in *The TEXbook*'s meaning, i.e., a control sequence whose meaning is `\(e/g/x)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) subsituting the corresponding parameter of this macro. (These understandings are according to *The TEXbook*, I hope: TEX is a religion of Book ;-) .)

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

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

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

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

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 `\catcodes`, it's just the `%`. When the file is documented with gmdoc, such a char is re`\catcoded` and its rôle is else: it becomes the **code delimiter**.

A line containing any TEX code (not commented out) will be called a **codeline**. A line that begins with (some leading spaces and) a code delimiter will be called a **comment line** or **narration line**.

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

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

Preparing the Source File

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

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

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

Note that `^^A` behaves much like comment char although it's active in fact: it re`\catcodes` the special characters including `\`, `{` and `}` so you don't have to worry about unbalanced braces or `\ifs` in its scope. But `^^B` doesn't re`\catcode` anything (it 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 mean-
`\StraightEOL` ing of the line end with the `\StraightEOL` declaration which OCSR. As almost all the control sequences, it may be used also as an environment, i.e., `\begin{StraightEOL}` … `\end{StraightEOL}`. However, if for any reason you don't want to make an environment
`\QueerEOL` (a group), there's a `\StraightEOL`'s counterpart, the `\QueerEOL` declaration that restores again the queer[3] gmdoc's meaning of the line end. It OCSR, too. One more point to use `\StraightEOL` is where you wish some code lines to be executed both while loading the file and during the documentation pass (it's analogous to doc's not embracing some code lines in a `macrocode` environment).

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

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

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

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

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

"

The Main Input Commands

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

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

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

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

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

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

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

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

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

At the default settings, the \Doc/SelfIncluded files constitute chapters if \chapter is known and parts otherwise. The \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- \gmdocIncludes doc' ones after, so the above declaration has its counterpart, \gmdocIncludes, that may be used anywhere, too. Before the respective \DocInclude(s), of course.

Both these declarations OCSR.

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

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.

If you need to add something at the beginning of the input of file, there's the \AtBegInput \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 succeding 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 CSs 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, she may turn it off with the linesnotnum option.

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

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, she is provided the countalllines option. [441] Then the narration acquires a bit biblical look ;-), [442] as shown in this short example. This option is intended [443] for the draft versions and it is not perfect (as if anything [444] in this package was). As you see, the lines [445] are typeset continuously with the numbers printed.

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

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.

indexallmacros By default, gmdoc excludes some 300 CSs from being indexed. They are the most common CSs, LATEX internal macros and TEX primitives. To learn what CSs are excluded actually, see lines 1253–1353.

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

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

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

The Packages Required

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

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

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

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

```
\MakeShortVerb*\|
```

(as you see, the declaration has the starred version, which is for visible spaces, and non-starred for blank spaces) to get `\mylittlemacro` you may type `|\mylittlemacro|` instead of `\verb+\mylittlemacro+`. Because the char used in the last example is my favourite
`\dekclubs` and is used this way by DEK in *The TEXbook*'s format, gmverb provides a macro `\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 `\activeated` by `\MakeShortVerb`, behave as if they were 'other' in math mode, so you may type e.g., `$k|n$` to get $k|n$ etc.

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

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

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

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

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

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

Automatic marking of definitions

gmdoc implements automatic detection of a couple of definitions. 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(*)`,
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 CSs uderlying 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 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 furher 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*⟩`@`⟨*optionname*⟩. The default ⟨*KVOfamily*⟩ is the input file name. Again, `\DeclareKVOFam` 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 recognized by `\DeclareDefining` 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.

`KVpref` There are also `KVpref` and `KVfam` keys if you want to redeclare the xkeyval definitions `KVfam` 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 [type=text]).

On the other hand, if you want some of the commands listed above *not* to be detected, write \HideDefining⟨*command*⟩ in the commentary. Later you can resume detection of it with \ResumeDefining⟨*command*⟩.

\HideDefining
\ResumeDefining

If you wish to turn entire detection mechanism off, write \HideAllDefining in the narration layer. Then you can resume detection with \ResumeAllDefining.

\HideAllDefining
\ResumeAllDefining

The basic definition command, \def, seems to me a bit controversial. 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 want it to be marked at this point, in the commentary before this \def write \UnDef. That will turn off the detection just for this one occurrence of \def.

\UnDef

If you don't like \def to be detected more times, you may write \HideDefining\def of course, but there's a shorthand for this: \HideDef. To resume detection of \def you are provided also a shorthand, \ResumeDef (but \ResumeDefining\def also works).

\HideDef
\ResumeDef

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 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 exluding some macros from being indexed.)

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

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

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

\Define
\CodeUsage
\TextUsage

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

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

\MakePrivateLetters

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 dicourage 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 CSs, too. Currently, the most important difference is that the unstarred version executes `\MakePrivateLetters` while the starred does both `\MakePrivateLetters` and `\MakePrivateOthers` before reading the argument.

If you consider the marginpars a sort of sub(sub…)section marks, then you may wish to have a command that makes a marginpar of the desired CS (or whatever) at the beginning of its description, which may be fairly far from the first occurrence of its object.

\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 (`\string`ed) argument into a marginpar (if the marginpars are enabled) and create an index entry (if indexing is enabled).

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

\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

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

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 de-

\CodeUsgIndex clare 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

\CodeCommonIndex* 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`.

Similarly, to index a TEX object occurring verbatim in the narrative, you have

\TextUsgIndex `\TextUsgIndex` and `\TextCommonIndex` commands with their starless versions for a CS

\TextCommonIndex

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 `evironment`.)

Index Ex/Inclusions

\DoNotIndex

It's understandable[4] that you don't want some control sequences to be indexed in your documentation. The doc package gives a brilliant solution: the `\DoNotIndex` declaration. So do I (although here, TEXnically it's done another way). It OCSR. This declaration takes one argument consisting of a list of control sequences not to be indexed. The items of this list may be separated with commas, as in doc, but it's not obligatory. The whole list should come in curly braces (except when it's one-element), e.g.,

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

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

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

\DoIndex

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

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

\DefaultIndexExclusions
\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.

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.

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

The Changes History

The doc's documentation reads:

"To maintain a change history within the file, the \changes command may be placed amongst the description part of the changed code. It takes three arguments, thus:

\changes{⟨*version*⟩}{⟨*YYYY/MM/DD date*⟩}{⟨*text*⟩}

The changes may be used to produce an auxiliary file (LaTeX's \glossary mechanism is used for this) which may be printed after suitable formatting. The \changes [command] encloses the ⟨*date*⟩ in parentheses and appends the ⟨*text*⟩ to form the printed entry in such a change history [... obsolete remark ommitted].

\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 in and \PrintChanges print the sorted change history (in two columns), just put the \PrintChanges command as the last (commented-out, and thus executed during the documentation pass through the file) command in your package file [or in the driver]. Alternatively, this command may form one of the arguments of the \StopEventually command, although a change history is probably not required if only the description is being printed. The command assumes that MakeIndex or some other program has processed the .glo file to generate a sorted .gls file. You need a special MakeIndex style file; a suitable one is supplied with \GlossaryMin doc [and gmdoc], called [... **gmglo.ist** for gmdoc]. The \GlossaryMin, \GlossaryPrologue \GlossaryPrologue and \GlossaryParms macros are analagous to the \Index... versions [see sec. The Pa- \GlossaryParms rameters p. 19]. (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, *16*

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

```
\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 ChangesStartDate mandatory = the third argument) is later than the date stored in the counter ChangesStartDate. \ChangesStart You may set this counter with a

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

declaration.

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

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

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

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

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

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

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

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

The Parameters

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

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

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

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

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

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

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

\nostanza

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.

\CodeIndent

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.

\TextIndent

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

\EOFMark

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

\NoEOF

%⟨*some text, why not*⟩\NoEOF

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

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

\CodeDelim

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

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

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

\CodeEscapeChar

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

\MakePrivateLetters

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

\AddtoPrivateOthers

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

\LineNumFont

codelinenum

Three more parameter macros, are \IndexPrefix, \EntryPrefix and \HLPrefix. All

\IndexPrefix
\EntryPrefix
\HLPrefix

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

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

\UsgEntry
\DefEntry
The special formattings of 'usage' and 'def' index entries are determined by \UsgEntry and \DefEntry one-parameter macros (the parameter shall be substituted with the reference number) and by default are defined as \textit and \underline respectively (as in doc).

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

The index comes in a multicols environment whose columns number is determined

IndexColumns
\IndexMin
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 162. Of course, you may write your own \IndexPrologue{⟨*brand new index prologue*⟩}, but if you like the default

\AtDIPrologue
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.

By default the colour of the index entry hyperlinks is set black to let Adobe Reader

\IndexLinksBlack
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
Other index parameters are set with the \IndexParms macro defined in line 1398 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.)

\gaddtomacro

At the default gmdoc settings the .idx file is prepared for the default settings of MakeIndex (no special style). Therefore the index control chars are as usual. But if you need to use other chars as MakeIndex controls, know that they are stored in the four

\actualchar
\quotechar
\levelchar
\encapchar
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.

Frank Mittelbach in doc provides the \verbatimchar macro to (re)define the \verb's

\verbatimchar
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 inline comments gmdoc provides the \inverb⟨*charX*⟩...⟨*charX*⟩ (the name stands for 'inline 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

\cs

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)

\env

provides the \env macro, that prints its argument verbatim and without a backslash, e.g., \env{an environment} produces an environment.

\nlpercent
\+

And for line breaking at \cs and \env there is \nlpercent to ensure % if the line breaks at the beginning of a \cs or \env and \+ to use inside their 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 \+.

\pk
\file

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

Because we play a lot with the \catcodes here and want to talk about it, there are

\catletter
\catother
\catactive

\catletter, \catother and \catactive macros that print $_{11}$, $_{12}$ and $_{13}$ respectively to concisely mark the most used char categories.

I wish my self-documenting code to be able to be typeset each package separately or several in one document. Therefore I need some 'flexible' sectioning commands and

\division
\subdivision
\subsubdivision

here they are: \division, \subdivision and \subsubdivision so far, that by default are \let to be \section, \subsection and \subsubsection respectively.

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

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

\SetFileDiv

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

gmlonely
\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 TEXbook*,
\TB *The TEXbook*
\eTeX ε-TEX,
\pdfeTeX pdfε-TEX
\pdfTeX pdfTEX
\XeTeX XETEX (the first E will be reversed if the graphics package is loaded or XETEX 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 `\MacroTopsep`. Both of them take care of not cumulating the vspaces.

quotation

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

\GetFileInfo
\filedate
\fileversion
\fileinfo

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

\ProvideFileInfo
\ProvideSelfInfo
\filenote

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

A macro for the standard note is provided, `\filenote`, that expands to "This file has

[6] See gmutils for some subtle details.

version number ⟨*version number*⟩ dated ⟨*date*⟩." To place such a note in the document's
\thfileinfo title (or heading, with \DocInclude at the default settings), there's \thfileinfo macro that puts \fileinfo in \thanks.

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

If you declare the code delimiter other than % and then want % back, you may write
\CDPerc \CDPerc instead of \CodeDelim*\%.
\CDAnd If you like & as the code delimiter (as I did twice), you may write \CDAnd instead of \CodeDelim\&.

A Queerness of \label

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

doc-Compatibility

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

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

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

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

The default behaviour of macrocode(*) with the 'new' input commands is different however. Remember 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.)

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

`macrocode*` to behave the traditional way. (It's always executed by `\OldDocInput` and `\olddocIncludes`.)

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

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 CSs, possibly with special format for the 'def' and 'usage' entries.

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

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

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

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

The Package Options

```
\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.

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

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

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

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

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

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

\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}
```
withmarginpar
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).

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

The Dependencies and Preliminaries

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

```
\RequirePackage{gmutils}[2007/11/09]
```

A standard package for defining colours,

```
\RequirePackage{color}
```

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

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

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

```
\@ifXeTeX{\XeTeXdefaultencoding "cp1250"}{}
\@ifpackageloaded{hyperref}{\hypersetup{colorlinks=true,
    linkcolor=deepblue, urlcolor=blue, filecolor=blue}}{%
```

```
  \RequirePackage[colorlinks=true, linkcolor=deepblue, urlcolor=blue,
  filecolor=blue, pdfstartview=FitH, pdfview=FitBH,
  \@ifXeTeX{xetex,}{}
  pdfpagemode=UseNone]{hyperref}}
\@ifXeTeX{\XeTeXdefaultencoding "utf-8"}{}
```

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}[2007/11/09]
```

```
\if@noindex
  \AtBeginDocument{\gag@index}% for the latter macro see line 1069.
\else
   \RequirePackage{makeidx}\makeindex
\fi
```

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

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

If you want to change the verb 'hyphen', there is the `\VerbHyphen\`⟨*one char*⟩ declaration provided by gmverb.

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

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

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

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

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

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

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

And let's assume you do as we all do:

```
\CodeDelim*\%
```

We'll play with `\everypar`, a bit, and if you use such things as the `{itemize}` environment, an error would 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 inline
        comment case. \verbatimhangindent is provided by the gmverb package
        and = 3 em by default.
\@ifdefinable\@@settexcodehangi{\let\@@settexcodehangi=%
      \settexcodehangi}
```

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

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

I assume it's originally equal 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
\advance\stanzaskip by-.25\medskipamount% to preserve the stretch- and shrinkability.
```

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[7] 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 \smallskipamount'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 `\stanzaskip`, you should use the following declaration. That shall discard the unwanted setting already placed in the `\begin{document}` hook.

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

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

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

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

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

And, for a smaller part of text,

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

Since the stanza skips are inserted automatically most often (cf. lines 198, 366, 207, 325, 382), sometimes you may need to forbid them.

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

To count the lines where they have begun not before them

```
\newgif\if@newline
```

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

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

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

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

The Core

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

\DocInput
```
\newcommand*\DocInput{\bgroup\@makeother\_\Doc@Input}
\begingroup\catcode`\^^M=\active%
\firstofone{\endgroup%
```
\Doc@Input
```
  \newcommand*{\Doc@Input}[1]{\egroup\begingroup%
    \edef\gmd@inputname{#1}% we'll use it in some notifications.
    \let\gmd@currentlabel@before=\@currentlabel% we store it because we'll do \xdefs of \@currentlabel to make proper references to the line numbers so we want to restore current \@currentlabel after our group.
    \gmd@setclubpenalty% we wrapped the assignment of \clubpenalty in a macro because we'll repeat it twice more.
    \@clubpenalty\clubpenalty \widowpenalty=3333 % Most paragraphs of the code will be one-line most probably and many of the narration, too.
    \tolerance=1000 % as in doc.
```

```
    \if@codespacesblank\CodeSpacesBlank\fi% The default is that the code spaces
        are visible but here this may be cancelled due to the \codespacesblank
        option.
    \catcode`\^^M=\active%
    \@xa\@makeother\csname\code@delim\endcsname%
    \gmd@resetlinecount% due to the option uresetlinecount we reset the linenum-
        ber counter or do nothing.
    \@beginputhook% my first use of it is to redefine \maketitle just at this point not
        globally.
    \everypar=\@xa{\@xa\@codetonarrskip\the\everypar}%
^^M \let^^M=\gmd@textEOL%
\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 1927
      \@nx\EOFMark% to pretty finish the input, see line 156.
      \@nx\CodeDelim\@xa\@nx\csname\code@delim\endcsname% to ensure the code
          delimiter is the same as at the beginning of input.
      \@nx^^M\code@delim%
    }% we add guardians after \inputing a file; somehow an error occurred without
        them.
    \catcode`\%=9 % for doc-compatibility.
    \setcounter{CheckSum}{0}% we initialize the counter for the number of the es-
        cape chars (the assignment is \global).
    \@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 be-
        fore this group. In a very special case this could cause unexpected be-
        haviour of crossrefs, but anyway we acted globally and so acts hyperref.
    \endgroup%
  }% end of \Doc@Input's definition.
}% end of \firstofone's argument.
```

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

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

```
\gmd@textEOL \def\gmd@textEOL{ % a space just like in normal TEX. We put it first to cooperate
        with \^^M's \expandafter\ignorespaces. It's no problem since a space
        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 respective switches:
        the one bringing the message 'we are after narration' to True (@afternarr)
        and the 'we have put the code-narration glue' to False (@codeskipput). Since
        we are in a verbatim group and the information should be brought outside
        it, we switch the switches globally (the letter g in both).
    \@newlinegtrue% to \refstep the lines' counter at the proper point.
    \@dsdirgtrue% to handle the DocStrip directives.
    \@xa\@trimandstore\the\everypar\@trimandstore% we store the previous value
        of \everypar register to restore it at a proper point. See line 395 for the
        details.
    \begingroup%
```

```
\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.
\aftergroup\gmd@setclubpenalty%
\let\par\@@par% inside the verbatim group we wish \par to be genuine.
\ttverbatim% it does \tt and makes specials other or \active-and-breakable.
\gmd@DoTeXCodeSpace%
\@makeother\|% because \ttverbatim doesn't do that.
\MakePrivateLetters% see line 463.
\@xa\@makeother\code@delim% we are almost sure the code comment char is
      among the chars having been 12ed already. For 'almost' see the \IndexInput
      macro's definition.
```

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

```
  \@xa\@ifnextchar\@xa{\code@delim}{%
    \gmd@continuenarration}{%
    \gmd@dolspaces% it will launch \gmd@typesettexcode.
  }% end of \@ifnextchar's else.
}% end of \gmd@textEOL's definition.
```

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

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

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

Simili modo

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

For the index input hooking now declare a macro, we define it another way at line 1927.

```
\emptify\gmd@iihook
```

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

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

As you'll learn from lines 424 and 418, we use those two strange declarations to change and restore the very special meaning of the line end. Without such changes `\tableofcontents` would cause a disaster (it did indeed). And to check the catcode of `^^M` is the rôle of `\@ifQueerEOL`:

\@ifQueerEOL
```
\long\def\@ifQueerEOL#1#2{%
  \ifnum\the\catcode`\^^M=\active \afterfi{#1}\else\afterfi{#2}\fi}
```

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

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

\AtBegInputOnce
```
\long\def\AtBegInputOnce#1{%
```

```
\gaddtomacro\gmd@ABIOnce{\g@emptify\gmd@ABIOnce#1}}
```

Many tries of finishing the input cleanly led me to setting the guardians as in line 111 and to

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

Other solutions did print the last code delimiter or would require managing a special case for the macros typesetting TEX code to suppress the last line's numbering etc.
If you don't like it, see line 2106.

Due to the `codespacesblank` option in the line 99 we launch the macro defined below to change the meaning of a gmdoc-kernel macro.

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

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

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

\gmd@texcodespace
```
\def\gmd@texcodespace{\,\hskip\z@}}%
\end{obeyspaces}
```

How the continuing of the narration should look like?

\gmd@continuenarration
```
\def\gmd@continuenarration{%
  \endgroup
  \gmd@countnarrationline% see below.
  \@xa\@trimandstore\the\everypar\@trimandstore
  \everypar=\@xa{\@xa\@codetonarrskip\the\everypar}%
  \@xa\gmd@checkifEOL\@gobble}
```

Simple, isn't it? (We gobble the 'other' code delimiter. Despite of `\egroup` it's $_{12}$ because it was touched by `\futurelet` contained in `\@ifnextchar` in line 136. And in line 250 it's been read as $_{12}$. That's why it works in spite of that % is of category 'ignored'.)

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

```
\if@countalllines
```

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

And typesetting the TEX code?

```
\begingroup\catcode`\^^M=\active%
\firstofone{\endgroup%
  \def\gmd@typesettexcode{%
    \gmd@parfixclosingspace% it's to eat a space closing the paragraph, see below. It contains \par. A verbatim group has already been opened by \ttverb% atim and additional \catcode.
    \everypar={\@@settexcodehangi}% At first attempt we thought of giving the user a \toks list to insert at the beginning of every code line, but what for?
    \def^^M{%
      \@newlinegtrue% to \refstep the counter in proper place.
      \@dsdirgtrue% to handle the DocStrip directives.
      \global\gmd@closingspacewd=\z@% we don't wish to eat a closing space after a codeline, 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.
      \@ifnextchar{\gmd@texcodespace}{%
        \@dsdirgfalse\gmd@dolspaces}{\gmd@charbychar}%
    }% end of ^^M's definition.
    \let\gmd@texcodeEOL=^^M% for further checks inside \gmd@charbychar.
    \raggedright\leftskip=\CodeIndent%
    \if@aftercode\gmd@nocodeskip1{iaC}\else\if@afternarr%
        \if@codeskipput\else\gmd@codeskip1\@codeskipputgtrue%
            \@aftercodegfalse\fi%
    \else\gmd@nocodeskip1{naN}\fi\fi% if now we are switching from the narration into the code, we insert a proper vertical space.
    \@aftercodegtrue\@afternarrgfalse%
    \ifdim\gmd@ldspaceswd>\z@% and here the leading spaces.
      \leavevmode\@dsdirgfalse%
      \if@newline\grefstepcounter{codelinenum}\@newlinegfalse%
      \fi%
      \printlinenumber% if we don't want the lines to be numbered, the respective option \lets this CS to \relax.
      \hyperlabel@line%
      \mark@envir% index and/or marginize an environment if there is some to be done so, see line 1018.
      \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 lineends require extreme caution.
        \@nx\hangafter=1\space}%
    \else%
      \glet\settexcodehangi=\@@settexcodehangi%
```

\gmd@typesettexcode
^^M
\@newlinegtrue

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

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

\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 \breakablexiispace. So let us \let for future:

\gmd@texcodespace

```
\let\gmd@texcodespace=\breakablexiispace
```

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

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

\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@
      \gmd@continuenarration#1%
    \else \afterfifi{\gmd@typesettexcode#1}%
    \fi
  \fi}%
```

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

```
\newgif\ifprevhmode
```

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

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

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

\gmd@EOLorcharbychar
```
\def\gmd@EOLorcharbychar#1{%
  \ifx\gmd@texcodeEOL#1%
    \if@newline
      \if@countalllines\global\advance\c@codelinenum by\@ne
      \@newlinegfalse\fi
    \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
      \grefstepcounter{codelinenum}\@newlinegfalse
      \hyperlabel@line
    \fi
    \printlinenumber
    \mark@envir
  \else
    \if@newline
      \grefstepcounter{codelinenum}\@newlinegfalse
      \hyperlabel@line
```

```
    \fi
  \fi}
```

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

\gmd@percenthack
```
\def\gmd@percenthack{%
  \ifprevhmode\code@delim\aftergroup\space% 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@narrcheckifds@ne% remember that \gmd@precenthack is only called when we've the code delimiter and soon we'll close the verbatim group and right after \endgroup there waits \gmd@checkifEOLmixd.
  \fi}
```

\gmd@narrcheckifds@ne
```
\def\gmd@narrcheckifds@ne#1{%
  \@dsdirgfalse\@ifnextchar<{%
    \@xa\gmd@docstripdirective\@gobble}{#1}}
```

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

\gmd@checkifEOL
```
\def\gmd@checkifEOL{%
  \gmd@countnarrationline
  \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}%
  \@ifnextchar{\gmd@textEOL}{%
    \@dsdirgfalse\par\ignorespaces}{\gmd@narrcheckifds}}%
```

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

\gmd@narrcheckifds
```
\def\gmd@narrcheckifds{%
  \@dsdirgfalse\@ifnextchar<{%
    \@xa\gmd@docstripdirective\@gobble}{\ignorespaces}}
```

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

\gmd@checkifEOLmixd
```
\def\gmd@checkifEOLmixd{%
  \gmd@countnarrationline
  \everypar=\@xa{\@xa\@codetonarrskip\the\gmd@preverypar}%
  \@afternarrgfalse\@aftercodegtrue
  \ifhmode\@codeskipputgfalse\fi
  \@ifnextchar{\gmd@textEOL}{%
    {\raggedright\gmd@endpe\par}% without \raggedright this \par would be justified which is not appropriate for a long codeline that should be broken, e.g., 313.
    \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{%
      \ifhmode% (I added this \ifhmode as a result of a heavy debug.)
        \@@par
        \if@afternarr
```

```
            \if@aftercode
              \if@codeskipput\else\gmd@codeskip2\@aftercodegfalse%
                    \@codeskipputgtrue\fi
            \else\gmd@nocodeskip2{naC}%
            \fi
          \else\gmd@nocodeskip2{naN}%
          \fi
          \prevhmodegfalse\gmd@endpe% when taken out of \ifhmode, this line caused
                some codeline numbers were typeset with \leftskip = 0.
          \everypar=\@xa{%
            \@xa\@codetonarrskip\the\gmd@preverypar}%
          \let\par\@@par%
        \fi}%
      \gmd@endpe\ignorespaces}}
```

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

\gmd@endpe
```
\def\gmd@endpe{%
  \ifprevhmode
    \settexcodehangi%ndent
    \leftskip=\CodeIndent
  \else
    \leftskip=\TextIndent
    \hangindent=\z@
    \everypar=\@xa{%
      \@xa\@codetonarrskip\the\gmd@preverypar}%
  \fi}
```

Numbering (or Not) of the Lines

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

```
\if@uresetlinecount% with uresetlinecount option…
  \@relaxen\gmd@resetlinecount% … we turn resetting the counter by \DocInput
        off…
```

\resetlinecountwith, codelinenum
```
  \newcommand*\resetlinecountwith[1]{%
    \newcounter{codelinenum}[#1]}% … and provide a new declaration of the
          counter.
```

```
\else% With the option turned off…
```

DocInputsCount, codelinenum
```
  \newcounter{DocInputsCount}%
  \newcounter{codelinenum}[DocInputsCount]% … we declare the \DocInputs'
        number counter andthe codeline counter to be reset with stepping of it.
```

\gmd@resetlinecount
```
  \newcommand*\gmd@resetlinecount{\stepcounter{DocInputsCount}}% … and let
        the \DocInput increment the \DocInputs number count and thus reset the
        codeline count. It's for unique naming of the hyperref labels.
\fi
```

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

\printlinenumber
```
\newcommand*\printlinenumber{%
  \leavevmode\llap{\rlap{\LineNumFont$\phantom{999}$\llap{%
        \thecodelinenum}}%
```

```
    \hskip\leftskip}}
\LineNumFont \def\LineNumFont{\normalfont\tiny}
\if@linesnotnum\@relaxen\printlinenumber\fi
\hyperlabel@line \newcommand*\hyperlabel@line{%
  \if@pageindex% It's good to be able to switch it any time not just define it once
        according to the value of the switch set by the option.
  \else
    \raisebox{2ex}[1ex][\z@]{\gmhypertarget[clnum.%
      \HLPrefix\arabic{codelinenum}]{}}%
  \fi}
```

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

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

```
@codeskipput
\if@codeskipput \newgif\if@codeskipput
```

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

```
\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 inline comment) is followed by a homogenic 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 \long\def\@codetonarrskip{%
  \if@codeskipput\else
```

```
    \if@afternarr\gmd@nocodeskip4{iaN}\else
      \if@aftercode
```

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

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

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

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

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

`\@trimandstore`
`\@trimandstore@hash`

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

`\@trimandstore@ne`
`\trimmed@everypar`

```
\long\def\@trimandstore@ne#1#2\@trimandstore{%
  \def\trimmed@everypar{#2}%
  \ifx\@codetonarrskip#1%
    \gmd@preverypar=\@xa{\trimmed@everypar}%
  \else
    \gmd@preverypar=\@xa{\the\everypar}%
  \fi}
```

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

Life Among Queer EOLs

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

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

```
\catcode`\^^B=\active% note we re\catcode ⟨char2⟩ globally, for the entire docu-
      ment.
\foone{\catcode`\^^M=\active}%
```

```
^^B     {\def\QueerCharTwo{%
\QueerCharTwo       \def^^B##1^^M{\ignorespaces}}%
    }
\QueerCharTwo
\AtBegInput{\@ifQueerEOL{\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{\catcode`\^^M=\active}%
^^A     {\def\QueerCharOne{%
\QueerCharOne       \def^^A{%
              \bgroup\let\do\@makeother\dospecials\gmd@gobbleuntilM}}%
\gmd@gobbleuntilM     \def\gmd@gobbleuntilM#1^^M{\egroup\ignorespaces^^M}%
    }
\QueerCharOne
\AtBegInput{\@ifQueerEOL{\catcode`\^^A\active}\QueerCharOne}% see note after line 408.
```

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

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

```
\StraightEOL \def\StraightEOL{%
    \catcode`\^^M=5
    \catcode`\^^A=14
    \catcode`\^^B=14
    \def\^^M{\ }}
\foone{\catcode`\^^M=\active}%
\QueerEOL   {\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}%
}
```

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

```
\gmd@parfixclosingspace
\def\gmd@parfixclosingspace{{%
    \advance\parfillskip by-\gmd@closingspacewd\par}}
```

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

```
\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 194 to see what we do in the codeline case when no closing space is added.

And one more detail:

```
\bgroup\catcode`\^^M=\active%
\firstofone{\egroup%
\gmd@bslashEOL
  \def\gmd@bslashEOL{\ \@xa\ignorespaces^^M}}
```

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

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

Adjustment of `verbatim` and `\verb`

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

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

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

```
\check@percent
\AtBegInput{\long\def\check@percent#1{%
    \@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
\AtBegInput{\def\gm@verb@eol{\obeylines%
\verb@egroup
    \def^^M{\verb@egroup\@latex@error{%
        \@nx\verb ended by end of line}%
      \@ifQueerEOL{\gmd@textEOL}{\@ehc}}}}%
\endgroup
```

Macros for Marking The Macros

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

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

```
\actualchar 453 \edef\actualchar{\string @}
\quotechar  454 \edef\quotechar{\string "}
\encapchar  455 \edef\encapchar{\xiiclub}
\levelchar  456 \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 `\edefs` that process them further are postponed till `\begin{document}`.

```
\CodeEscapeChar      457 \def\CodeEscapeChar#1{%
                     458   \begingroup
                     459   \escapechar\m@ne
\code@escape@char    460   \xdef\code@escape@char{\string#1}%
                     461   \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.

```
462 \CodeEscapeChar\\
```

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

```
463 \@ifundefined{MakePrivateLetters}{%
\MakePrivateLetters 464   \def\MakePrivateLetters{\makeatletter\catcode`\*=11 }}{}
```

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

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

```
\MakePrivateOthers 465 \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 1089. So let's define the list:

```
\doprivateothers 466 \def\doprivateothers{\do\ \do\^}
```

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

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

```
\scan@macro 467 \def\scan@macro#1{% we are sure to scan at least one token and therefore we define
                    this macro as one-parameter.
```

Unlike in doc, here we have the escape char $_{12}$ so we may just have it printed during main scan char by char, i.e., in the lines 276 and 279.

So, we step the checksum counter first,

```
\step@checksum%
```
(see line 1565 for details),

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

Of course, I can imagine horrible perversities, but I don't think they should really be taken into account. Giving the letter `a` `\catcode` other than $_{11}$ surely would be one of those perversities. Therefore I feel safe to take the character `a` as a benchmark letter.

```
\ifcat a\@nx#1%
  \quote@char#1%
  \xdef\macro@iname{\gmd@maybequote#1}% global for symmetry with line 476.
  \xdef\macro@pname{\string#1}% we'll print entire name of the macro later.
```

We `\string` it here and in the lines 480 and 486 to be sure it is whole $_{12}$ for easy testing for special indexentry formats, see line 868 etc. Here we are sure the result of `\string` is $_{12}$ since its argument is $_{11}$.

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

Another reasonable `\catcodes` assumption seems to be that the digits are $_{12}$. Then we don't have to type (%)`\expandafter\@gobble\string\a`. We do the `\uccode` trick to be sure that the char we write as the macro's name is $_{12}$.

```
    {\uccode`9=`#1%
      \uppercase{\xdef\macro@iname{9}}%
    }%
    \quote@char#1%
    \xdef\macro@iname{\gmd@maybequote\macro@iname}%
    \xdef\macro@pname{\xiistring#1}%
    \afterfi \finish@macroscan
  \fi}
```

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`

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

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`

```
\def\gmd@finishifstar#1{%
  \if*\@nx#1\afterfi\finish@macroscan% note we protect #1 against expansion.
        In gmdoc verbatim scopes some chars are active (e.g. \ ).
  \else\afterfi\continue@macroscan
```

```
  \fi}
```

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

\quote@char
```
\def\quote@char#1{{\uccode`9=`#1% at first I took digit 1 for this \uccodeing but
      then #1 meant #⟨#1⟩ in \uppercase's argument, of course.
    \uppercase{%
      \gmd@ifinmeaning 9\of \indexcontrols
      {\glet\gmd@maybequote\quotechar}%
      {\g@emptify\gmd@maybequote}%
    }%
  }}
```

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

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

\gmd@ifinmeaning
```
\long\def \gmd@ifinmeaning#1\of#2#3#4{% explained in the text paragraph below.
```

\gmd@in@@
```
  \long\def\gmd@in@@##1#1##2\gmd@in@@{%
    \ifx^^A##2^^A\afterfi{#4}%
    \else\afterfi{#3}%
    \fi}%
  \@xa\gmd@in@@#2#1\gmd@in@@}%
```

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

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

The meaning of the `\gmd@ifinmeaning` parameters is as follows:

`#1` the token(s) whose presence we check,

`#2` the macro in whose meaning we search `#1` (the first token of this argument is expanded one level with `\expandafter`),

`#3` the 'if found' stuff,

`#4` the 'if not found' stuff.

In `\quote@char` the second argument for `\gmd@ifinmeaning` is `\indexcontrols` defined as the (expanded and 'other') sequence of the MakeIndex controls. `\gmd@ifinmeaning` defines its inner macro `\gmd@in@@` to take two parameters separated by the first and the second `\gmd@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 `\gmd@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 secong 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 \gmd@ifinmeaning is used also in testing whether a detector is already present in the carrier in the mechanism of automatic detection of definitions (line 555).

\ifgmd@glosscs 508 \newif\ifgmd@glosscs% we use this switch to keep the information whether a history entry is a CS or not.

\finish@macroscan 509 \newcommand*\finish@macroscan{%

First we check if the current CS is not just being defined. The switch may be set true in line 523

510 \ifgmd@adef@cshook% if so, we throw it into marginpar and index as a def entry…
511 \@ifundefined{gmd/iexcl/\macro@pname}{% … if it's not excluded from indexing.
512 \@xa\Code@MarginizeMacro\@xa{\macro@pname}%
513 \@xa\@defentryze\@xa{\macro@pname}{1}}{}% here we declare the kind of index entry and define \last@defmark used by \changes
514 \global\gmd@adef@cshookfalse% we falsify the hook that was set true just for this CS.
515 \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 countercrank way because we wish to use more general indexing macro.

516 \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 $:
\im@firstpar 517 \def\im@firstpar{[$]}%
\im@firstpar 518 \else\def\im@firstpar{}\fi
519 \@xa \index@macro\im@firstpar\macro@iname\macro@pname
520 \maybe@marginpar\macro@pname
521 \macro@pname
522 \let\next\gmd@charbychar
523 \gmd@detectors% for automatic detection of definitions. Defined and explained in the next section. It redefines \next if detects a definition command amd thus sets the switch of line 509 true.
524 \next
525 }

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 526 \def\maybe@marginpar#1{%
527 \@ifundefined{gmd/2marpar/#1}{}{%
528 \@xa\Text@Marginize\@xa{\bslash#1}% \expandafters because the \Text@Marginize command applies \string to its argument. % \macro@pname, which will be the only possible argument to \maybe@mar- % ginpar, contains the macro's name without the escapechar so we added it here.

```
    \@xa\g@relaxen\csname gmd/2marpar/#1\endcsname% we reset the switch.
  }}
```

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

Automatic detection of definitions

To begin with, let's introduce a general declaration of a defining command. `\DeclareDefining` 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.

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

```
\foone{\catcode`\*=11 }
```

`\@ifstarl`

```
  {\def\@ifstarl#1{\@ifnextchar *{\@firstoftwo{#1}}}}
```

`\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`

```
\outer\def\DeclareDefining{\begingroup
  \MakePrivateLetters
  \@ifstarl
    {\gdef\gmd@adef@defaulttype{text}\Declare@Dfng}%
    {\gdef\gmd@adef@defaulttype{cs}\Declare@Dfng}%
}
```

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

`\Declare@Dfng`

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

```
  \fi}
\Declare@Dfng@inner \def\Declare@Dfng@inner#1#2{%
  \edef\gmd@resa{%
    \@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\gmd@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\@tempa{% this \edef is to expand \gmd@adef@TYPE.
    \global\@nx\@namedef{gmd@detect@\gmd@adef@currdef}{%
      \@nx\ifx
        \@xa\@nx\csname gmd@detectname@\gmd@adef@currdef\endcsname
        \@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}}%
  \@tempa
  \SMglobal\StoreMacro*{gmd@detect@\gmd@adef@currdef}% we store the CS to allow its temporary discarding later.
}
\gmd@adef@setkeysdefault \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 671.

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

The `prefix@`⟨*command*⟩ keyvalue 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 811 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\@xa\@nx\csname gmd@adef@prefix@\gmd@adef@currdef \endcsname{%
      #1}}%
  \gmd@resa}
```

```
\gmd@KVprefdefault \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\@xa\@nx\csname gmd@adef@KVpref@\gmd@adef@currdef\endcsname{%
      \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\@xa\@nx\csname gmd@adef@KVfam@\gmd@adef@currdef\endcsname{%
      \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 gmdoc with % [KVpref=, KVfam=⟨family⟩].
    \fi}
```

\gmd@adef@settype
\gmd@adef@TYPE

```
\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}
```

\gmd@adef@setKV

```
\def\gmd@adef@setKV{%
  \edef\gmd@resa{%
    \def\@xa\@nx\csname gmd@adef@KV@\gmd@adef@currdef\endcsname{1}%
  }%
  \gmd@resa}
```

We initialize 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
```

\gmd@adef@cs

```
\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 seargants. In gmdoc code layer scopes the left brace is active so we only add a hook to its meaning (see line 23 in gmverb) and **??**nd here we switch it according to the type of detected definition.

\gmd@adef@text

```
\def\gmd@adef@text{\gdef\gmd@lbracecase{1}\gmd@charbychar}
\foone{%
  \catcode`\[\active
```

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

The detector of xkeyval `\define@(...)key`:

\gmd@adef@dk
```
  \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`:

\gmd@adef@dox
```
  \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 CSs scanning macro's `\@ifnextcat`), therefore we need a 'future' test.

\gmd@adef@checklbracket
```
\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 seargants. Therefore we have to test presence of both of them.

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

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

\gmd@adef@dfKVpref
```
\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}
\gmd@adef@scanDOXfam \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}
\gmd@adef@scanfamoth \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}
\gmd@adef@scanfamact  {\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).

```
\gm@lbracehook \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}
\gmd@lbracecase \def\gmd@lbracecase{0}% we initialize the hook caser.
```

And we define the inner left brace macros:

```
\foone{\catcode`\[1 \catcode`\]2 \catcode`\}12 }
```

```
[% Note that till line ?? 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.

\gmd@adef@scanDKfam
```
\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]
```

\gmd@adef@scanname
```
\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]
]
```

\gmd@adef@dofam
```
\def\gmd@adef@dofam{%
  \ifnum1=0\csname gmd@adef@KVfamset@\gmd@adef@currdef\endcsname
    \relax% a family declared with \DeclareDefining overrides the one currently
         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}
```

\gmd@adef@deftext
```
\def\gmd@adef@deftext#1{%
  \edef\macro@pname{\@gobble#1}% we gobble \gmd@charbychar, cf. above.
  \@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 in the changes history should appear without back-
        slash.
  \gmd@doindexingtext% redefine \do to an indexing macro.
  \@xa\do\@xa{\macro@pname}}
```

\gmd@adef@indextext

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
```

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 occurence of \def off (only the next one).

\UnDef

```
\def\UnDef{%
  \gdef\gmd@detect@def{%
    \ifx\gmd@detectname@def\macro@pname
      \def\next{\SMglobal\RestoreMacro\gmd@detect@def}%
    \fi}%
  }
\StoreMacro\UnDef% because the 'hiding' commands relax it.
```

\HideDef

```
\def\HideDef{\HideDefining\def\relaxen\UnDef}
```

\relaxen
\ResumeDef

```
\def\ResumeDef{\ResumeDefining\def\RestoreMacro\UnDef}
```

\RestoreMacro

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.

\newcount

```
\DeclareDefining[star=false]\newcount
```

\newdimen

```
\DeclareDefining[star=false]\newdimen
```

\newskip

```
\DeclareDefining[star=false]\newskip
\DeclareDefining[star=false]\newif
```

\newtoks

```
\DeclareDefining[star=false]\newtoks
```

\newbox

```
\DeclareDefining[star=false]\newbox
```

\newread

```
\DeclareDefining[star=false]\newread
```

\newwrite

```
\DeclareDefining[star=false]\newwrite
```

\newlength

```
\DeclareDefining[star=false]\newlength
\DeclareDefining\newcommand
```

\renewcommand

```
\DeclareDefining\renewcommand
\DeclareDefining\providecommand
```

\DeclareRobustCommand

```
\DeclareDefining\DeclareRobustCommand
```

\DeclareTextCommand

```
\DeclareDefining\DeclareTextCommand
```

\DeclareTextCommandDefault

```
\DeclareDefining\DeclareTextCommandDefault
```

```
\DeclareDefining*\newenvironment
\DeclareDefining*\renewenvironment
\DeclareDefining*\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

```
\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:

```
\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 occurence 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.

\HideAllDefining
```
\def\HideAllDefining{%
  \ifnum0=0\csname gmd@adef@allstored\endcsname
    \SMglobal\StoreMacro\gmd@detectors
    \global\@namedef{gmd@adef@allstored}{1}%
  \fi
  \global\emptify\gmd@detectors}% we make the carrier \empty not \relax to be
        able to declare new defining command in the scope of \HideAll...
```

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

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

Note that \ResumeAllDefining discards the effect of any \DeclareDefining that could have occured 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.

\HideDefining
```
\def\HideDefining{\begingroup
  \MakePrivateLetters
  \Hide@Dfng}
```

\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}
```

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 CSs.

\ResumeDefining
```
\def\ResumeDefining{\begingroup
  \MakePrivateLetters
  \gmd@ResumeDfng}
```

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

Indexing of CSs

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 472, 480 and 486. `#3` is used only to test if the entry should be specially formatted.

\index@macro
```
\newcommand*\index@macro[3][\verbatimchar]{{%
    \@ifundefined{gmd/iexcl/#3}%
    {% #3 is not excluded from index
      \@ifundefined{gmd/defentry/#3}%
      {% #3 is not def entry
        \@ifundefined{gmd/usgentry/#3}%
        {% #3 is not usg entry
          \edef\kind@fentry{\CommonEntryCmd}}%
        {% #3 is usg entry
```
\kind@fentry
```
          \def\kind@fentry{UsgEntry}%
          \un@usgentryze{#3}}%
      }%
      {% #3 is def entry
```
\kind@fentry
```
        \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\IndexRefCs{gmdindexpagecs{\HLPrefix}{\kind@fentry}{%
            \EntryPrefix}}%
      \else
        \edef\IndexRefCs{gmdindexrefcs{\HLPrefix}{\kind@fentry}{%
            \EntryPrefix}}%
      \fi
      \edef\@tempa{\IndexPrefix#2\actualchar%
        \quotechar\bslash verb*#1\quoted@eschar#2#1% The last macro in this line usually means the first two, but in some cases it's redefined to be empty (when we use \index@macro to index not a CS).
        \encapchar\IndexRefCs}%
      \@xa\special@index\@xa{\@tempa}% We give the indexing macro the argument expanded so that hyperref may see the explicit encapchar in order not to add its own encapsulation of |hyperpage when the (default) hyperindex=true option is in force. (After this setting the \edefs in the above may be changed to \defs.)
    }{}% closing of gmd/iexcl/ test.
    }}
```

\un@defentryze
```
\def\un@defentryze#1{%
  \@xa\g@relaxen\csname gmd/defentry/#1\endcsname
  \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.
\un@usgentryze \def\un@usgentryze#1{%
  \@xa\g@relaxen\csname gmd/usgentry/#1\endcsname}
\@emptify\EntryPrefix% this macro seems to be obsolete now (v0.98d).
```

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

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

Let's initialize `\IndexPrefix`

```
\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
\gmdindexrefcs   \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
\gmdindexrefcs   \def \gmdindexrefcs#1#2#3#4{\gmiflink[clnum.#4]{%
        \csname#2\endcsname{#4}}}%
\gmdindexpagecs  \def \gmdindexpagecs#1#2#3#4{\hyperlink{page.#4}{%
        \csname#2\endcsname{\gmd@revprefix{#3}#4}}}%
\gmd@revprefix   \def\gmd@revprefix#1{%
      \def\@tempa{#1}%
      \ifx\@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 labeled clnum.⟨number⟩. When \HLPrefix was defined as dot,
        MakeIndex rejected the entries as 'illegal page number'.
  \fi}
```

The definition is postponed till `\begin{document}` because of the `\PageIndex` declaration (added for doc-compatibility), see line 2067.

I design the index to contain hyperlinking numbers whether they are the line numbers or page numbers. In both cases the last parameter is the number, the one before

the last is the name of a formatting macro and in linenumber case the first parameter is a prefix for proper reference in multi-doc.

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

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

The third option will be just \relax by default:

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

In line 872 it's \edefed to allow an 'unmöglich' situation that the user wants to have the common index entries specially formatted. I use this to make *all* the index entries of the (\SelfIncluded) driver file to be 'usage', see codeline 14 of gmdocDoc.tex.

Now let's \def the macros declaring a CS to be indexed special way. Each declaration puts the $_{12}$ed name of the macro given it as the argument into proper macro to be \ifxed in lines 868 and 870 respectively.

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

\DefIndex
```
\outer\def\DefIndex{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Code@DefIndexStar}{\Code@DefIndex}}
```

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

\Code@DefIndexStar
```
\long\def\Code@DefIndexStar#1{%
  \endgroup
  \addto@estoindex{#1}%
  \@defentryze{#1}{0}}
```

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

\@defentryze
```
\long\def\@defentryze#1#2{%
  \@xa\glet\csname gmd/defentry/\string#1\endcsname\gmd@justadot% The
      LaTeX \@namedef macro could not be used since it's not 'long'.
```

\last@defmark
```
  \xdef\last@defmark{\string#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 \\).
  \@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.
```

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

Initialize \envirs@toindex

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

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

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

The * possibility is for marking environments etc.

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

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

For the symmetry, if we want to mark a control sequence or an environment's name to be indexed as a 'normal' entry, let's have:

\CodeCommonIndex
```
\outer\def\CodeCommonIndex{\begingroup
  \MakePrivateLetters
  \@ifstarl{\MakePrivateOthers\Code@CommonIndexStar}{%
        \Code@CommonIndex}}
```

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

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

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

\text@indexmacro
```
\long\def\text@indexmacro#1{%
  {\escapechar\m@ne \xdef\macro@pname{\xiistring#1}}%
  \@xa\quote@mname\macro@pname\relax% we process the CS's name char by char
        and quote MakeIndex controls. \relax is the iterating macro's stopper. The
        scanned CS's quoted name shall be the expansion of \macro@iname.
  \if\verbatimchar\macro@pname
```

\im@firstpar
```
    \def\im@firstpar{[$]}%
```

\im@firstpar
```
  \else\def\im@firstpar{}%
  \fi
  {\do@properindex% see line 1121.
    \@xa \index@macro\im@firstpar\macro@iname\macro@pname}}
```

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

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

\quote@charbychar
```
\def\quote@charbychar#1{%
  \if\relax#1% finish quoting when you meet \relax or:
  \else
    \quote@char#1%
    \xdef\macro@iname{\macro@iname \gmd@maybequote#1}%
    \afterfi\quote@charbychar
  \fi}
```

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

\TextUsgIndex
```
\def\TextUsgIndex{\begingroup
  \MakePrivateLetters
```

```
977   \@ifstarl{\MakePrivateOthers\Text@UsgIndexStar}{\Text@UsgIndex}}
\Text@UsgIndex
978 \long\def\Text@UsgIndex#1{%
979   \endgroup\@usgentryze#1%
980   \text@indexmacro#1}
\Text@UsgIndexStar
981 \long\def\Text@UsgIndexStar#1{\endgroup\@usgentryze{#1}%
982   \text@indexenvir{#1}}
\text@indexenvir
983 \long\def \text@indexenvir#1{%
984   \edef\macro@pname{\xiistring#1}%
985   \if\bslash\@xa\@firstofmany\macro@pname\@@nil% if \stringed #1 begins with
          a backslash, we will gobble it to make MakeIndex not see it.
986     \edef\@tempa{\@xa\@gobble\macro@pname}%
987     \@tempswatrue
988   \else
989     \let\@tempa\macro@pname
990     \@tempswafalse
991   \fi
992   \@xa\quote@mname\@tempa\relax% we process \stinged #1 char by char and quote
          MakeIndex controls. \relax is the iterating macro's stopper. The quoted
          \stringed #1 shall be the meaning of \macro@iname.
993   {\if@tempswa
\quoted@eschar
994     \def\quoted@eschar{\quotechar\bslash}%
995     \else\@emptify\quoted@eschar\fi% we won't print any backslash before an
            environment's name, but we will before a CS's name.
996     \do@properindex% see line 1121.
997     \index@macro\macro@iname\macro@pname}}
\TextCommonIndex
998 \def\TextCommonIndex{\begingroup
999   \MakePrivateLetters
1000  \@ifstarl{\MakePrivateOthers\Text@CommonIndexStar}{%
          \Text@CommonIndex}}
\Text@CommonIndex
1001 \long\def\Text@CommonIndex#1{\endgroup
1002   \text@indexmacro#1}
\Text@CommonIndexStar
1003 \long\def\Text@CommonIndexStar#1{\endgroup
1004   \text@indexenvir{#1}}
```

As you see in the lines 879 and 875, the markers of special formatting are reset after first use.

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

```
\CodeMarginize
1005 \outer\def\CodeMarginize{\begingroup
1006   \MakePrivateLetters
1007   \@ifstarl
1008     {\MakePrivateOthers\egCode@MarginizeEnvir}
1009     {\egCode@MarginizeMacro}}
```

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

```
\egCode@MarginizeMacro
1010 \long\def\egCode@MarginizeMacro#1{\endgroup
1011   \Code@MarginizeMacro#1}
\Code@MarginizeMacro
1012 \long\def\Code@MarginizeMacro#1{{\escapechar\m@ne
```

```
    \@xa\glet\csname gmd/2marpar/\string#1\endcsname\gmd@justadot
  }}
\egCode@MarginizeEnvir \long\def\egCode@MarginizeEnvir#1{\endgroup
  \Code@MarginizeEnvir{#1}}
\Code@MarginizeEnvir \long\def\Code@MarginizeEnvir#1{\addto@estomarginpar{#1}}
```

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

Here it is:

```
\mark@envir \def\mark@envir{%
  \ifx\envirs@tomarginpar\@empty
  \else
    \let\do\Text@Marginize
    \envirs@tomarginpar%
    \g@emptify\envirs@tomarginpar%
  \fi
  \ifx\envirs@toindex\@empty
  \else
    \gmd@doindexingtext
    \envirs@toindex
    \g@emptify\envirs@toindex%
  \fi}
\gmd@doindexingtext \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 781.
    \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 992 & subs. for commentary.
    {\if@tempswa
\quoted@eschar       \def\quoted@eschar{\quotechar\bslash}%
      \else\@emptify\quoted@eschar\fi
      \index@macro\macro@iname{##1}}}%
}
```

One very important thing: initialisation of the list macros:

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

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

```
\Define \outer\def\Define{\begingroup
  \MakePrivateLetters
```

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

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

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

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

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

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

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

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

And in case we don't want to index:

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

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

```
\StoreMacros{\index\codeline@wrindex}
\ungag@index \def\ungag@index{\RestoreMacros{\index\@@codeline@wrindex}}
```

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

```
\Code@DefEnvir \long\def\Code@DefEnvir#1{%
  \endgroup
  \addto@estomarginpar{#1}%
  \addto@estoindex{#1}%
  \@defentryze{#1}{0}}
\Code@UsgEnvir \long\def\Code@UsgEnvir#1{%
  \endgroup
  \addto@estomarginpar{#1}%
  \addto@estoindex{#1}%
```

```
    \@usgentryze{#1}}
\addto@estomarginpar
\long\def\addto@estomarginpar#1{%
    \edef\@tempa{\@nx\do{\xiistring#1}}% we \string the argument to allow it to be a control sequence.
    \@xa\addtomacro\@xa\envirs@tomarginpar\@xa{\@tempa}}
\addto@estoindex
\long\def\addto@estoindex#1{%
    \edef\@tempa{\@nx\do{\xiistring#1}}
    \@xa\addtomacro\@xa\envirs@toindex\@xa{\@tempa}}
```

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

```
\TextUsage
\def\TextUsage{\begingroup
    \MakePrivateLetters
    \@ifstarl{\MakePrivateOthers\Text@UsgEnvir}{\Text@UsgMacro}}
\Text@UsgMacro
\long\def\Text@UsgMacro#1{%
    \endgroup{\tt\xiistring#1}%
    \Text@Marginize#1%
    \begingroup\Code@UsgIndex#1% we declare the kind of formatting of the entry.
    \text@indexmacro#1}
\Text@UsgEnvir
\long\def\Text@UsgEnvir#1{%
    \endgroup{\tt\xiistring#1}%
    \Text@Marginize{#1}%
    \@usgentryze{#1}% we declare the 'usage' kind of formatting of the entry and index the sequence #1.
    \text@indexenvir{#1}}
```

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

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

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

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

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

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

\Text@Marginize
```
\long\def\Text@Marginize#1{%
  \gmdmarginpar{\marginpartt\xiistring#1}}
```

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

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

```
\let\marginpartt\tt
```

If we count all lines then the index entries for CSs and environments marked in the commentary should have codeline numbers not page numbers and that is `\let` in line 1067. On the other hand, if we count only the codelines, then a macro or an environment marked in the commentary should have page number not codeline number. This we declare here, among others we add the letter `p` before the page number.

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

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

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

macrocode*
```
\newenvironment*{macrocode*}{%
  \if@codeskipput\else\par\addvspace\CodeTopsep\@codeskipputgtrue\fi
  \QueerEOL}%
  {\par\addvspace\CodeTopsep\CodeEscapeChar\\}
```

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

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

macrocode
```
\newenvironment*{macrocode}{%
  \if@codeskipput\else\par\addvspace\CodeTopsep\@codeskipputgtrue\fi
  \CodeSpacesBlank\QueerEOL}%
  {\par\addvspace\CodeTopsep\CodeEscapeChar\\}
```

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

Since the first .dtx I tried to compile after the first published version of gmdoc uses a lot of commented out code in `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
```

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

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

Now we arm `oldmc` and `olmc*` with the 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.
  \glet\stored@code@delim\code@delim
  \@makeother\^^B\CodeDelim\^^B%
  \ttverbatim \gmd@DoTeXCodeSpace%
  \@makeother\|% because \ttverbatim doesn't do that.
  \MakePrivateLetters% see line 463.
  \docstrips@percent \@makeother\>%
```

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

```
  \@xa\gmd@currenvxistar\@currenvir*\relax
  \@oldmacrocode}
\bgroup\catcode`*11
\firstofone{\egroup
  \def\gm@xistar{*}}
\def\gmd@currenvxistar#1*#2\relax{%
  \edef\@currenvir{#1\if*#2\gm@xistar\fi}}
```

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

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

```
!bgroup!let =!relax& to avoid writing !noexpand  four times.
!xdef!oldmc@def[&
!def!noexpand!oldmc@end####1!noexpand%     !noexpand\end&
!noexpand{!@currenvir}[&
####1^^B!noexpand!end[!@currenvir]!noexpand!gmd@oldmcfinis]]&
!egroup& now \oldmc@edef is defined to have one parameter delimited with \end{⟨current
      env.'s name⟩}
!oldmc@def&
!oldmc@end]&
]
\def\gmd@oldmcfinis{%
  \@xa\CodeDelim\stored@code@delim
  \gmd@mchook}% see line 2000
\def\VerbMacrocodes{%
  \let\macrocode\oldmc
  \n@melet{macrocode*}{oldmc*}}
```

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

```
\bgroup\catcode`\%\active
\firstofone{\egroup
\def\docstrips@percent{\catcode`\%\active
  \let%\gmd@codecheckifds}}
```

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

`\gmd@codecheckifds`

```
\def\gmd@codecheckifds#1#2{% note that #1 is just to gobble \gmd@charbychar to-
      ken.
  \if@dsdir\@dsdirgfalse
    \if\@nx<\@nx#2\afterfifi\gmd@docstripdirective
    \else\afterfifi{\xiipercent#1#2}%
    \fi
  \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 'pri-
      vate others' to cover the case of other sequence in the argument. (We'll use
      the \macro macro also in the environment for describing and defining envi-
      ronments.)
  \gmd@ifonetoken\Hybrid@DefMacro\Hybrid@DefEnvir}%
  {\par\addvspace\MacroTopsep\@codeskipputgtrue}
```

It came out that the doc's author(s) give the `macro` environment also starred versions of commands as argument. It's OK since (the default version of) `\MakePrivateLetters` makes * a letter and therefore such a starred version is just one CS. However, in doc.dtx 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}
\@xa\let\csname endmacro*\endcsname\endmacro
```

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

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

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

\gmd@ifsingle

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

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

\gmd@ifonetoken

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

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

\Hybrid@DefMacro

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

\Hybrid@DefEnvir

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

\Text@MarginizeNext

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

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

\gmd@evpaddonce

```
\long\def\gmd@evpaddonce#1{%
```

```
  \stepnummacro\gmd@oncenum
  \@xa\long\@xa\edef%
    \csname gmd/evp/NeuroOncer\gmd@oncenum\endcsname{%
      \@nx\g@relaxen
      \csname gmd/evp/NeuroOncer\gmd@oncenum\endcsname}% Why does it work
          despite it shouldn't? Because when the CS got with \csname...\endcsname
          is undefined, it's equivalent \relax and therefore unexpandable. That's
          why it passes \edef and is able to be assigned.
  \@xa\addtomacro\csname gmd/evp/NeuroOncer\gmd@oncenum\endcsname{#1}%
  \@xa\addto@hook\@xa\everypar\@xa{%
    \csname gmd/evp/NeuroOncer\gmd@oncenum\endcsname}%
}
\nummacro\gmd@oncenum% We store the number uniquifying the auxiliary macro in
    a macro to save count registers (cf. gmutils sec. To Save Precious Count Registers).
```

environment

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

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

Index Exclude List

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

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

\DoNotIndex

```
\def\DoNotIndex{\bgroup\MakePrivateLetters\DoNot@Index}
```

\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.}
```

\dont@index

```
\long\def\dont@index#1{%
  \def\@tempa{\@nx#1}% My TEX Guru's trick to deal with \fi and such, i.e., to hide
      from TEX when it is processing a test's branch without expanding.
  \if\@tempa.% a dot finishes expelling
  \else
    \if\@tempa,% The list this macro is put before may contain commas and that's
        O.K., we just continue the work.
      \afterfifi\dont@index
    \else% what is else shall off the Index be expelled.
      {\escapechar\m@ne
        \xdef\@tempa{\string#1}}%
      \@xa\let%
      \csname gmd/iexcl/\@tempa\endcsname=\gmd@iedir% In the default case explained e.g. by the macro's name, the last macro's meaning is such that
          the test in line 866 will turn false and the subject CS shall not be indexed.
          We \let not \def to spare TEX's memory.
```

[9] This idea comes from Marcin Woliński.

```
      \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 CSs would be rejected
      by MakeIndex anyway.
\begin{MakePrivateLetters}% Yes, \DoNotIndex does \MakePrivateLetters on its
      own but No, it won't have any effect if it's given in another macro's \def.
\DefaultIndexExclusions  \gdef\DefaultIndexExclusions{%
    \DoNotIndex{\@ \@@par \@beginparpenalty \@empty}%
    \DoNotIndex{\@flushglue \@gobble \@input}%
    \DoNotIndex{\@makefnmark \@makeother \@maketitle}%
    \DoNotIndex{\@namedef \@ne \@spaces \@tempa}%
    \DoNotIndex{\@tempb \@tempswafalse \@tempswatrue}%
    \DoNotIndex{\@thanks \@thefnmark \@topnum}%
    \DoNotIndex{\@@ \@elt \@forloop \@fortmp \@gtempa \@totalleftmargin}%
    \DoNotIndex{\" \/ \@ifundefined \@nil \@verbatim \@vobeyspaces}%
    \DoNotIndex{\| \~ \  \active \advance \aftergroup \begingroup \bgroup}%
    \DoNotIndex{\mathcal \csname \def \documentstyle \dospecials \edef}%
    \DoNotIndex{\egroup}%
    \DoNotIndex{\else \endcsname \endgroup \endinput \endtrivlist}%
    \DoNotIndex{\expandafter \fi \fnsymbol \futurelet \gdef \global}%
    \DoNotIndex{\hbox \hss \if \if@inlabel \if@tempswa \if@twocolumn}%
    \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 1250.
    \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/providecommand` and `\(e/g/x)defs`, as the most common, in my opinion excluded should be.

And some my exclusions:

```
\DoNotIndex{\@@input \@auxout \@currentlabel \@dblarg}%
\DoNotIndex{\@ifdefinable \@ifnextchar \@ifpackageloaded}%
\DoNotIndex{\@indexfile \@let@token \@sptoken \^}% the latter comes from
      CSs like \^^M, see sec. 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 IMO.
\DoNotIndex{\@outerparskip \@partaux \@partlist \@plus}%
\DoNotIndex{\@sverb \@sxverbatim}%
\DoNotIndex{\@tempcnta \@tempcntb \@tempskipa \@tempskipb}%
      I think the layout parameters even the kernel, should not be excluded:
      % \@topsep \@topsepadd \abovedisplayskip \clubpenalty etc.
```

```
    \DoNotIndex{\@writeckpt}%
    \DoNotIndex{\bfseries \chapter \part \section \subsection}%
    \DoNotIndex{\subsubsection}%
    \DoNotIndex{\char \check@mathfonts \closeout}%
    \DoNotIndex{\fontsize \footnotemark \footnotetext \footnotesize}%
    \DoNotIndex{\g@addto@macro \hfilneg \Huge \huge}%
    \DoNotIndex{\hyphenchar \if@partsw \IfFileExists }%
    \DoNotIndex{\include \includeonly \indexspace}%
    \DoNotIndex{\itshape \language \LARGE \Large \large}%
    \DoNotIndex{\lastbox \lastskip \m@th \makeglossary}%
    \DoNotIndex{\maketitle \math@fontsfalse \math@fontstrue \mathsf}%
    \DoNotIndex{\MessageBreak \noindent \normalfont \normalsize}%
    \DoNotIndex{\on@line \openout \outer}%
    \DoNotIndex{\parbox \part \rmfamily \rule \sbox}%
    \DoNotIndex{\sf@size \sffamily \skip}%
    \DoNotIndex{\textsc \textup \toks@ \ttfamily \vbox}%
  %%\DoNotIndex{\begin*} maybe in the future, if the idea gets popular…
    \DoNotIndex{\hspace* \newcommand* \newenvironment* \providecommand*}%
    \DoNotIndex{\renewenvironment* \section* \chapter*}%
  }% of \DefaultIndexExclusions.
```

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

```
\end{MakePrivateLetters}
```

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

```
\if@indexallmacros\else
  \DefaultIndexExclusions
\fi
```

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

\DoIndex
```
\def\DoIndex{\bgroup\MakePrivateLetters\Do@Index}
```

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

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

\UndoDefaultIndexExclusions
```
\def\UndoDefaultIndexExclusions{%
  \StoreMacro\DoNotIndex
  \let\DoNotIndex\DoIndex
  \DefaultIndexExclusions
  \RestoreMacro\DoNotIndex}
```

Index Parameters

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

\IndexPrologue
```
\long\def\IndexPrologue#1{\@bsphack\def\index@prologue{#1}\@esphack}
```

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

\index@prologue
```
\@ifundefined{index@prologue} {\def\index@prologue{\indexdiv{Index}%
```

```
    \markboth{Index}{Index}%
    Numbers written in italic refer to the \if@pageindex pages \else
    code lines \fi where the
    corresponding entry is described; numbers underlined refer to the
    \if@pageindex\else code line of the \fi definition; numbers in
    roman refer to the \if@pageindex pages\else code lines \fi where
    the entry is used.
    \if@pageindex\else
      \ifx\HLPrefix\@empty
        The numbers preceded with `p.' are page numbers.
      \else The numbers with no prefix are page numbers.
    \fi\fi
    \ifx\IndexLinksBlack\relax\else
      All the numbers are hyperlinks.
    \fi
    \gmd@dip@hook% this hook is intended to let a user add something without re-
                 defining the entire prologue, see below.
  }}{}
```

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

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

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

```
\RequirePackage{multicol}
```

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

```
\newdimen\IndexMin \IndexMin = 133pt\relax% originally it was set 80 pt, but
                 with my default prologue there's at least 4.7 cm needed to place the prologue
                 and some index entries on the same page.
\newcount\c@IndexColumns \c@IndexColumns = 3
\renewenvironment{theindex}
  {\begin{multicols}\c@IndexColumns[\index@prologue][\IndexMin]%
     \IndexLinksBlack
     \IndexParms \let\item\@idxitem \ignorespaces}%
   {\end{multicols}}
\def\IndexLinksBlack{\hypersetup{linkcolor=black}}% To make Adobe Reader
                 work faster.
\@ifundefined{IndexParms}
  {\def\IndexParms{%
     \parindent \z@
     \columnsep 15pt
     \parskip 0pt plus 1pt
     \rightskip 15pt
     \mathsurround \z@
     \parfillskip=-15pt plus 1 fil %doc defines this parameter rigid but that's
                 because of the stretchable space (more precisely, a \dotfill) between
```

the item and the entries. But in gmdoc we define no such special delimiters, so we add an ifinite stretch.

```
      \small
      \def\@idxitem{\par\hangindent 30pt}%
\subitem      \def\subitem{\@idxitem\hspace*{15pt}}%
\subsubitem      \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.

```
    }}{}
\PrintIndex \def\PrintIndex{%
```

we ensure the standard meaning of the line end character not to cause a disaster.

```
  \@ifQueerEOL{\StraightEOL\printindex\QueerEOL}{\printindex}}
```

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

The DocStrip Directives

```
\foone{\@makeother\<\@makeother\>
  \glet\sgtleftxii=<}
{
\gmd@docstripdirective   \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}}%
\gmd@docstripinner   \def\gmd@docstripinner#1>{%
    \endgroup
\gmd@modulehashone   \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 `\catcodes`; the directive's text has its `\catcodes` fixed. Then we put the directive's text wrapped with the formatting macro into one macro in order to give just one token the gmdoc's TeX code scanner. Then launch this big TeX code scanning machinery by calling `\gmd@textEOL` which is an alias for the 'narrative' meaning of the line end. This macro opens the verbatim group and launches the char-by-char scanner. That is this scanner because of what we encapsulated the directive's text with the formatting into one macro: to let it pass the scanner. That's why in the 'old' macrocodes case the active % closes the group before launching `\gmd@docstripdirective`.

The 'verbatim' directive macro works very similarly.

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

(~Verbatim ;-) from doc:)

```
\Module  \providecommand*\Module[1]{{\mod@math@codes$\langle\mathsf{#1}%
           \rangle$}}
\ModuleVerb  \providecommand*\ModuleVerb[1]{{\mod@math@codes$\langle\langle\mathsf{%
           #1}$}}
\mod@math@codes  \def\mod@math@codes{\mathcode`\|="226A \mathcode`\&="2026 }
```

The Changes History

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

"To provide a change history log, the `\changes` command has been introduced. This takes [one optional and] three [mandatory] arguments, respectively, [the macro that'll become the entry's second level,] the version number of the file, the date of the change, and some detail regarding what change has been made [i.e., the description of the change]. The [second] of these arguments is otherwise ignored, but the others are written out and may be used to generate a history of changes, to be printed at the end of the document. [… I ommit an obsolete remark about then-older MakeIndex's versions.]

The output of the `\changes` command goes into the ⟨*Glossary_File*⟩ and therefore uses the normal `\glossaryentry` commands. Thus MakeIndex or a similar program can be used to process the output into a sorted "glossary". The `\changes` command commences by taking the usual measures to hide its spacing, and then redefines `\protect` for use within the argument of the generated `\indexentry` command. We re-code nearly all chars found in `\@sanitize` to letter since the use of special package which make some characters active might upset the `\changes` command when writing its entries to the file. However we have to leave % as comment and as ⟨*space*⟩ otherwise chaos will happen. And, of course the \ should be available as escape character."

We put the definition inside a macro that will be executed by (the first use of) `\RecordChanges`. And we provide the default definition of `\changes` as a macro just gobbling its arguments. We do this to provide no changes' writing out if `\RecordChanges` is not used.

```
\gmd@DefineChanges  \def\gmd@DefineChanges{%
\changes    \outer\long\def\changes{\@bsphack\begingroup\@sanitize
      \catcode`\\\z@ \catcode`\ 10 \MakePercentIgnore
      \MakePrivateLetters \StraightEOL
      \MakeGlossaryControls
      \changes@}}
\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][]{}}
\MakeGlossaryControls \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 'en-
        cap' char remains untouched. I decided to preserve the doc's settings for the
        compatibility.
\changes@ \newcommand\changes@[4][\generalname]{%
    \if@RecentChange{#3}% if the date is later than the one stored in \c@Changes-
        % StartDate,
      \@tempswafalse
      \ifx\generalname#1% then we check whether a CS-entry is given in the optional
          first argument or is it unchanged.
        \ifx\last@defmark\relax\else% if no particular CS is specified in #1, we
            check whether \last@defmark contains something and if so, we put it
            into \@tempb scratch macro.
          \@tempswatrue
          \edef\@tempb{% it's a bug fix: while typesetting traditional .dtxes, \last@defmark
              came out with \ at the beginning (which resulted with \\⟨name⟩ in the
              change log) but while typesetting the 'new' way, it occurred without
              the bslash. So we gobble the bslash if it's present and two lines be-
              low we handle the exception of \last@defmark = {\} (what would
              happen if a definition of \\ was marked in new way gmdocing).
            \if\bslash\last@defmark\else\last@defmark\fi}%
          \ifx\last@defmark\bslash\let\@tempb\last@defmark\fi%
          \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, he should type
          \changes[generalname]...)
        \@tempswatrue
        {\escapechar\m@ne
          \xdef\@tempb{\string#1}}%
        \if\bslash\@xa\@firstofmany\string#1\relax\@@nil% we check whether
            #1 is a CS...
\gmd@glossCStest    \def\gmd@glossCStest{1}% ... and tell the glossary if so.
        \fi
      \fi
\gmd@glossCStest    \@ifundefined{gmd@glossCStest}{\def\gmd@glossCStest{0}}{}%
      \protected@edef\@tempa{\@nx\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 1614.
          \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.
            \@tempb
```

```
          \actualchar\bslash verb*%
          \if\verbatimchar\@tempb$\else\verbatimchar\fi
          \if1\gmd@glossCStest\quotechar\bslash\fi \@tempb
          \if\verbatimchar\@tempb$\else\verbatimchar\fi
        \else
          \space\actualchar\generalname
        \fi
        :\levelchar#4\encapchar hyperpage}}%
    \@tempa
    \grelaxen\gmd@glossCStest
  \fi\endgroup\@esphack}
```

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

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

\ChangesGeneral
```
\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
```

\if@RecentChange
```
\def\if@RecentChange#1{%
  \gmd@setChDate#1\@@nil\@tempcnta
  \ifnum\@tempcnta>\c@ChangesStartDate}
```

\gmd@setChDate
```
\def\gmd@setChDate#1/#2/#3\@@nil#4{%
```
the last parameter will be a `\count` register.
```
  #4=#1\relax
  \multiply#4 by\@M
  \count8=#2\relax%
```
I know it's a bit messy not to check whether the `#4` `\count` is `\count8` but I know this macro will only be used with `\count0` (`\@tempcnt`) and some higher (not a scratch) one.
```
  \multiply\count8 by100 %
  \advance#4 by\count8 \count8=\z@
  \advance#4 by#3\relax}
```

Having the test defined, let's define the command setting the date counter. `#1` is to be the version and `#2` the date {⟨*year*⟩/⟨*month*⟩/⟨*day*⟩}.

\ChangesStart
```
\def\ChangesStart#1#2{%
  \gmd@setChDate#2\@@nil\c@ChangesStartDate
  \typeout{^^JPackage gmdoc info: ^^JChanges' start date #1 memorized
    as \string<\the\c@ChangesStartDate\string> \on@line.^^J}
  \advance\c@ChangesStartDate\m@ne%
```
we shall show the changes *at the specified day* and later.
```
  \ifnum\c@ChangesStartDate>19820900 %
```
[10] see below.

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

```
    \edef\@tempa{%
      \@nx\g@addto@macro\@nx\glossary@prologue{%
        The changes
        \if\relax\GeneralName\relax\else of \GeneralName\space\fi
        earlier than
        #1 \if\relax#1\relax #2\else(#2)\fi\space are not shown.}}%
    \@tempa
  \fi}
```

(Explanation to line 1519.) My TEX Guru has remarked that the change history tool should be used for documenting the changes that may be significant for the users not only for the author and talking of what may be significant to the user, no changes should be hidden since the first published version. However, the changes' start date may be used to provide hiding the author's 'personal' notes: she should only date the 'public' changes with the four digit year and the 'personal' ones with two digit year and set `\ChangesStart{}{1000/0/0}` or so.

In line 1519 I establish a test value that corresponds to a date earlier than any TEX stuff and is not too small (early) to ensure that hiding the two digit year changes shall not be mentioned in the changes prologue.

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

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

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

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

`\RecordChanges`
```
\def\RecordChanges{\makeglossary\gmd@DefineChanges
  \@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`
`\c@GlossaryColumns`
```
\newdimen\GlossaryMin          \GlossaryMin       = 80pt
\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 \let\item\@idxitem \ignorespaces}%
  {\end{multicols}}
```

Here is the MakeIndex style definition:

```
⟨/package⟩
⟨+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 '>'
⟨*package⟩
```

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

`makeindex -r -s gmglo.ist -o` ⟨*myfile*⟩`.gls` ⟨*myfile*⟩`.glo`

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

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

`\GlossaryPrologue` `\glossary@prologue`

```
\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."

`\glossary@prologue`

```
\@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."

`\GlossaryParms`

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

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

`\PrintChanges`

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

The Checksum

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

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

```
\check@sum 1563 \newcount\check@sum
\CheckSum 1564 \def\CheckSum#1{\@bsphack\global\check@sum#1\relax\@esphack}
CheckSum 1565 \newcounter{CheckSum}
\step@checksum 1566 \newcommand*\step@checksum{\stepcounter{CheckSum}}
```

And we'll use it in the line 468 (`\stepcounter` is `\global`). See also the `\chschange` declaration, l. 1601.

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.)[11]

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

In gmdoc we have the `\AtEndInput` hook.

```
1567 \AtEndInput{\check@checksum}
```

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

```
\check@checksum 1568 \def\check@checksum{\relax
1569   \ifnum\check@sum=\z@
1570     \typeout{**********************************}%
1571     \typeout{* The input file \gmd@inputname\space has no Checksum
1572       stated.}%
1573     \typeout{* The current checksum is \the\c@CheckSum.}%
1574     \gmd@chschangeline% a check sum changes history entry, see below.
1575     \typeout{* (package gmdoc info.)}
1576     \typeout{**********************************}%
1577   \else
1578     \ifnum\check@sum=\c@CheckSum
1579       \typeout{************************}%
1580       \typeout{* The input file \gmd@inputname: Checksum passed.}%
1581       \gmd@chschangeline
1582       \typeout{* (package gmdoc info.)}
1583       \typeout{************************}%
1584     \else
1585       \typeout{********!*!*!*!*!*!*!*!*!*!*!}%
1586       \typeout{*! The input file \gmd@inputname:}
1587       \typeout{*! The CheckSum stated: \the\check@sum\space<> my
1588         count: \the\c@CheckSum.}
1589       \gmd@chschangeline
1590       \typeout{*! (package gmdoc info.)}
1591       \typeout{********!*!*!*!*!*!*!*!*!*!*!}%
```

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

```
    \fi
  \fi
  \global\check@sum\z@}
```

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

\gmd@chschangeline
```
\def\gmd@chschangeline{%
  \typeout{\xiipercent\space\string\chschange{%
      \fileversion}{\the\year/\the\month/\the\day}{\the\c@CheckSum}}
  \typeout{\xiipercent\space\string\chschange{\fileversion}{%
      \@xa\@gobbletwo\the\year/\the\month/\the\day}{% with two digit year in
            case you use \ChangesStart.
      \the\c@CheckSum}}}
```

And here the meaning of such a line is defined:

\chschange
```
\newcommand*\chschange[3]{%
  \csname changes\endcsname{#1}{#2}{CheckSum #3}%\csname... because \cha-
       % nges is \outer.
  \CheckSum{#3}}
```

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 titlepages with only the filename and article-like titles made by `\maketitle`.

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

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

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

\Doc@Include
```
\newcommand*{\Doc@Include}[2][]{% originally it took just one argument. Here we make it take two, first of which is intended to be the path (with the closing /). This is intended not to print the path in the page footers only the filename.
  \egroup% having the arguments read, we close the group opened by the previous macro for _12.
```

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

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

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

\@docinclude
```
\def\@docinclude#1#2 {% To match the macro's parameter string, is an answer. But why is \@docinclude defined so? Originally, in ltxdoc it takes one argument and it's delimited with a space probably in resemblance to the true \input % (\@@input in LaTeX).
  \clearpage
  \if@filesw \gmd@writemauxinpaux{#2.aux}\fi% this strange macro with a long name is another thing to allow _ in the filenames (see line 1661).
  \@tempswatrue
  \if@partsw \@tempswafalse\edef\@tempb{#2}%
    \@for \@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswatrue\fi}%
  \fi
  \if@tempswa \let\@auxout\@partaux
    \if@filesw
```

```
      \immediate\openout\@partaux #2.aux\relax% Yes, only #2. It's to create
          and process the partial .aux files always in the main document's (driver's)
          directory.
      \immediate\write\@partaux{\relax}%
    \fi
```

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

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

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

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

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

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

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

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

We wrap `\@input{`⟨*partial* .aux⟩`}` in a $_{12}$ hacked scope. This hack is especially recommended here since the .aux file may contain a non-`\global` stuff that should not be localized by a group that we would have to establish if we didn't use the hack. (Hope you understand it. If not, notify me and for now I'll only give a hint: "Look at it with the TEX's eyes". More uses of this hack are to be seen in gmutils where they are a bit more explained.)

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

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

`\gmd@writeckpt`
```
\def\gmd@writeckpt#1{%
  \immediate\write\@partaux{%
    \string\bgroup\string\@makeother\string\_%
    \string\firstofone\@charlb\string\egroup}
  \@writeckpt{#1}%
  \immediate\write\@partaux{\@charrb}}
```

`\gmd@doIndexRelated`
```
\def\gmd@doIndexRelated{%
  \do\tableofcontents \do\makeindex \do\EnableCrossrefs
  \do\PrintIndex \do\printindex \do\RecordChanges \do\PrintChanges
  \do\theglossary \do\endtheglossary}
\@emptify\filesep
```

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

`\aalph`
```
\def\aalph#1{\@aalph{\csname c@#1\endcsname}}
```
`\@aalph`
```
\def\@aalph#1{%
  \ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or
        j\or k\or l\or m\or n\or o\or p\or q\or r\or s\or
        t\or u\or v\or w\or x\or y\or z\or A\or B\or C\or
        D\or E\or F\or G\or H\or I\or J\or K\or L\or M\or
        N\or O\or P\or Q\or R\or S\or T\or U\or V\or W\or
        X\or Y\or Z\else\@ctrerr\fi}
```

A macro that initialises things for `\DocInclude`.

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

We set the things for including the files only once.

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

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

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

```
\@xa\let\csname the\filedivname\endcsname=\thefilediv% This line lets \the⟨chapter⟩ etc. equal \thefilediv.
\filesep \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: {\ttfamily\currentfile} %
  \else
    \GetFileInfo{\currentfile}%
    File \thefilediv: {\ttfamily\filename} %
    Date: \filedate\ %
    Version \fileversion
  \fi
  \hfill\thepage}%
\glet\@evenfoot\@oddfoot% see line 1711.
\@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 \docincludeaux.
```

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

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

`\SelfInclude`

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

```
\SelfInclude \newcommand*{\SelfInclude}[2][.tex]{% As you guess, the optional #1 is the jobname's extension. The second parameter is not for the filename (note it's known: as \jobname!), but for the stuff to be put at begin input.
```

```
\AtBegInputOnce{#2}%
\gdef\HLPrefix{\filesep}%
\gdef\EntryPrefix{\filesep}% we define two rather kernel parameters etc. as in \DocInclude.
\relax
\clearpage
\docincludeaux
\edef\currentfile{\jobname#1}%
\let\fullcurrentfile\currentfile
\def\GeneralName{\jobname\actualchar\pk{\jobname} }% for the changes history main level entry.
\ifnum\@auxout=\@partaux
  \@latexerr{\string\DocInclude\space cannot be nested}\@eha
\else
  \if@filesw
    \gmd@writemauxinpaux{\jobname.auxx}% this queer macro allows _ in the file names. In this particular case \string\jobname would do, but anyway we provide a more general solution. Note the .auxx extension used instead of .aux. This is done to avoid an infinite recurrence of \inputs.
  \fi
  \@tempswatrue
  \if@partsw \@tempswafalse\edef\@tempb{\jobname}%
    \@for
    \@tempa:=\@partlist\do{\ifx\@tempa\@tempb\@tempswatrue\fi}%
  \fi
  \if@tempswa \let\@auxout\@partaux
    \if@filesw
      \immediate\openout\@partaux \jobname.auxx\relax
      \immediate\write\@partaux{\relax}
    \fi
```

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

```
    \StoringAndRelaxingDo% provided by gmutils
    \gmd@doIndexRelated
    \if@ltxDocInclude\part{\currentfile}%
    \else\let\maketitle=\InclMaketitle
    \fi
    \if@ltxDocInclude\xdef@filekey\fi
    \GetFileInfo{\currentfile}% it's my (GM) addition with the account of using file info in the included files' title etc.
    \incl@DocInput{\fullcurrentfile}% originally just \currentfile, no difference in \SelfInclude.
    \if@ltxDocInclude\else\xdef@filekey\fi% in the default case we add new file to the file key after the input because in this case it's files own \maketitle what launches the sectioning command that increases the counter.
```

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

```
    \RestoringDo
    \gmd@doIndexRelated
    \clearpage% among others, causes the \writes to be executed which is crucial for proper toc-ing e.g.
    \gmd@writeckpt{\jobname.x}% note the .x in the checkpoint used to distinguish this instance (input) of the driver file from its main instance.
```

```
      \if@filesw \immediate\closeout\@partaux \fi
    \else\@nameuse{cp@\jobname.x}% note .x: it's used for the same reason as
        above.
    \fi
    \let\@auxout\@mainaux
  \fi}% end of \SelfInclude.
```

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

\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 be-
    gin input by \olddocIncludes.
```

Redefinition of \maketitle

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

"The macro to generate titles is easily altered in order that it can be used more than once (an article with many titles). In the original, diverse macros were concealed after use with \relax. We must cancel anything that may have been put into \@thanks, etc., otherwise all titles will carry forward any earlier such setting!"

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

```
\AtBegInput{%
```

\maketitle
```
  \providecommand*\maketitle{\par
    \begingroup \def \thefootnote {\fnsymbol {footnote}}%
    \setcounter {footnote}\z@
    \def\@makefnmark{\hbox to\z@{$\m@th^{\@thefnmark}$\hss}}%
```

\@makefntext
```
    \long\def\@makefntext##1{\parindent 1em\noindent
      \hbox to1.8em{\hss$\m@th^{\@thefnmark}$}##1}%
    \if@twocolumn \twocolumn [\@maketitle ]%
    \else \newpage \global \@topnum \z@ \@maketitle \fi
```

"For special formatting requirements (such as in TUGboat), we use pagestyle `titlepage` for this; this is later defined to be `plain`, unless already defined, as, for example, by ltugboat.sty."

```
    \thispagestyle{titlepage}\@thanks \endgroup
```

"If the driver file documents many files, we don't want parts of a title of one to propagate to the next, so we have to cancel these:"

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

"When a number of articles are concatenated into a journal, for example, it is not usual for the title pages of such documents to be formatted differently. Therefore, a class such as ltugboat can define this macro in advance. However, if no such definition exists, we use pagestyle plain for title pages."

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

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

```
  \providecommand*\@maketitle{%
    \newpage\null \vskip 2em\relax%
    \begin{center}%
      \titlesetup
      \let \footnote \thanks
      {\LARGE \@title \par}%
      \vskip 1.5em%
      {\large \lineskip .5em%
        \begin{tabular}[t]{c}%
          \strut \@author
        \end{tabular}\par}%
      \vskip 1em%
      {\large \@date}%
    \end{center}%
    \par \vskip 1.5em\relax}%
```

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

```
\title  \providecommand*\title[1]{\gdef\@title{#1}}
\author \providecommand*\author[1]{\gdef\@author{#1}}
  \date \providecommand*\date[1]{\gdef\@date{#1}}
\thanks \providecommand*\thanks[1]{\footnotemark
    \protected@xdef\@thanks{\@thanks
      \protect\footnotetext[\the\c@footnote]{#1}}%
  }%
   \and \providecommand*\and{% % \begin{tabular}
    \end{tabular}%
    \hskip 1em \@plus.17fil%
    \begin{tabular}[t]{c}}% % \end{tabular} And finally, let's initialize \tit-
        % lesetup if it is not yet.
\titlesetup \providecommand*\titlesetup{}%
}% end of \AtBegInput.
```

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

Let's initialize the file division macros.

```
\@relaxen\filediv
\@relaxen\filedivname
```

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

Now, we redefine \maketitle and its relatives.

\InclMaketitle
\and
\thanks

```
\def\InclMaketitle{%
  {\def\and{, }% we make \and just a comma.
    {\let\thanks=\@gobble% for the toc version of the heading we discard \thanks.
      \protected@xdef\incl@titletotoc{\@title\if@fshda\protect\space
        (\@author)\fi}% we add the author iff the 'files have different authors'
             % (@fshda)
    }%
    \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
        {\smallerr% this macro is provided by the gmutils package after the relsize
             package.
          \if@fshda\\[0.15em]\protect\@author
            \if\relax\@date\relax\else, \fi
          \else
            \if\relax\@date\relax\else\\[0.15em]\fi
          \fi
```

The default is that all the included files have the same author(s). In this case we won't print the author(s) in the headings. Otherwise we wish to print them. The information which case are we in is brought by the \if@fshda switch defined in line 1865.

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

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

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

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

And the counterpart, if you change your mind:

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

The File's Date and Version Information

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

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

Since we may documentally input files that we don't load, as doc e.g., let's define a declaration to be put (in the comment layer) before the line(s) containing \Provides.... The \FileInfo command takes the stuff till the closing] and subsequent line end, extracts from it the info and writes it to the .aux and rescans the stuff. ε-TEX provides a special primitive for that action but we remain strictly TEXnical and do it with writing to a file and inputting that file.

\FileInfo
```
\newcommand*\FileInfo{%
  \bgroup
  \let\do\@makeother
  \do\ \do\{\do\}\do\^^M\do\\%
  \gmd@fileinfo}
\bgroup
\catcode`!\z@
\catcode`<\@ne
\catcode`>\tw@
\let\do\@makeother
\do\ \do\{\do\}\do\^^M\do\\%
!firstofone<!egroup%
```

\gmd@fileinfo
```
!def!gmd@fileinfo#1Provides#2{#3}#4[#5]#6
<%
!egroup%
!gmd@writeFI<#2><#3><#5>%
!gmd@docrescan<#1Provides#2{#3}#4[#5]#6
>%
>%
>
```

```
\gmd@writeFI
\def\gmd@writeFI#1#2#3{%
  \immediate\write\@auxout{%
    \global\@nx\@namedef{%
      ver@#2.\if P\@firstofmany#1\@@nil sty\else cls\fi}{#3}}}
\gmd@docrescan
\gmd@docrescanfile
\def\gmd@docrescan#1{%
  \newwrite\gmd@docrescanfile
  \immediate\openout\gmd@docrescanfile=\jobname.docrescan\relax
  {\newlinechar=`\^^M%
    \immediate\write\gmd@docrescanfile{%
      \xiipercent#1\xiipercent\@nx\NoEOF}%
  }%
  \immediate\closeout\gmd@docrescanfile
  \@@input\jobname.docrescan
}
```

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

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

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

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

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

And exactly as `\thanks`:

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

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}%
{%
\IndexInput
  \def\IndexInput#1{%
    \StoreMacro\code@delim%
    \CodeDelim\^^Z%
\gmd@iihook
    \def\gmd@iihook{% this hook is \edefed!
      \@nx^^M%
      \code@delim\relax\@nx\let\@nx\EOFMark\relax}%
```

```
    \DocInput{#1}\RestoreMacro\code@delim}%
}
```

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

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

gmlonely
```
\newenvironment{gmlonely}{\StraightEOL}{}
```
\skipgmlonely
```
\newcommand\skipgmlonely[1][]{%
  \def\@tempa{%
```
\gmd@skipgmltext
```
    \def\gmd@skipgmltext{\g@emptify\gmd@skipgmltext#1}}%
  \@tempa
  \@xa\AtBegInput\@xa{\@tempa}%
```
gmlonely
```
  \renewenvironment{gmlonely}{%
    \StraightEOL
    \@fileswfalse% to forbid writing to .toc, .idx etc.
    \setbox0=\vbox\bgroup}{\egroup\gmd@skipgmltext}}
```

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

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

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

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

and a letter, $_{11}$

\catletter
```
  \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

copyrnote
```
\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

quotation
```
\renewenvironment{quotation}{\par``\ignorespaces}{\unskip''\par}
```

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

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

When a verbatim text occurs in an inline comment, it's advisable to precede it with % if it begins a not first line of such a comment not to mistake it for a part of code. 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
```
\newcommand*\inverb{%
```

```
    \@ifstar{%
      \def\@tempa{{\tt\xiipercent}}%
      \@emptify\@tempb% here and in the paralell points of the other case and \nlpercent
                      I considered an \ifhmode test but it's not possible to be in vertical mode
                      while in an inline comment. If there happens vertical mode, the commen-
                      tary begins to be 'outline' (main text).
      \gmd@inverb}%
    {\@emptify\@tempa
      \def\@tempb{\gmboxedspace}%
      \gmd@inverb}}
\gmboxedspace \newcommand*\gmboxedspace{\hbox{\normalfont{ }}}
  \gmd@nlperc \newcommand*\gmd@nlperc[1][]{%
    \unskip
    \discretionary{\@tempa}{{\tt\xiipercent\gmboxedspace}}{\@tempb}%
    \penalty10000\hskip0sp\relax}
 \gmd@inverb \newcommand*\gmd@inverb[1][]{%
    \gmd@nlperc
    \ifmmode\hbox\else\leavevmode\null\fi
    \bgroup
    \ttverbatim
\breakablexiispace  \def\breakablexiispace{%
      \discretionary{\xiispace}{\xiipercent\gmboxedspace}{\xiispace}}%
  \breakbslash  \def\breakbslash{%
      \discretionary{}{\xiipercent\gmboxedspace\bslash}{\bslash}}%
  \breaklbrace  \def\breaklbrace{%
      \discretionary
        {\xiilbrace\verbhyphen}%
        {\xiipercent\gmboxedspace}%
        {\xiilbrace}}%
    \gm@verb@eol
    \@sverb@chbsl% It's always with visible spaces.
}
   \nlpercent \newcommand*\nlpercent{%
    \@ifstar{\def\@tempa{{\tt\xiipercent}}%
      \@emptify\@tempb
      \gmd@nlperc}%
    {\@emptify\@tempa
      \def\@tempb{\gmboxedspace}%
      \gmd@nlperc}}
```

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.

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

An optional hyphen for CSs in the inline comment:

```
\@ifundefined{+}{}{\typeout{^^Jgmdoc.sty: redefining \bslash+.}}
    \+ \def\+{\discre{{\normalfont-}}{{\tt\xiipercent\gmboxedspace}}{}}
   \ds \@ifundefined{ds}{\def\ds{DocStrip}}{}
```

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

\CDAnd
```
\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 \@tempb and \@tempc are written plain not verbatim):

gmd@mc
```
\newcounter{gmd@mc}
```
\gmd@mchook
```
\def\gmd@mchook{\stepcounter{gmd@mc}%
  \gmd@mcdiag
  \csname gmd@mchook\the\c@gmd@mc\endcsname}
```
\AfterMacrocode
```
\long\def\AfterMacrocode#1#2{\@namedef{gmd@mchook#1}{#2}}
```

What have I done? I declare a new counter and employ it to count the macrocode(*)s (and oldmc(*)s too, in fact) and attach a hook to (after) the end of every such environment. That lets us to put some stuff pretty far inside the compiled file (for the buggie in doc.dtx, to redefine \@tempb/c).

One more detail to expalin and define: the \gmd@mcdiag macro may be defined to type out a diagnostic message (the macrocode(*)'s number, code line number and input line number).

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

An environment to display the meaning of macro parameters: its items are automatically numbered as #1, #2 etc.

enumargs
```
\newenvironment*{enumargs}
  {\enumerate
    \@namedef{label\@enumctr}{%
      \cs[]{\#\csname the\@enumctr\endcsname }}}
  {\endenumerate}
```

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 \lets to make gmdoc able to typeset sources written for doc than to write a new package that adds hyperlinking to doc. So…

The doc package makes % an ignored char. Here the % delimits the code and therefore has to be 'other'. But only the first one after the code. The others we may re\catcode to be ignored and we do it indeed in line 113.

At the very beginning of a doc-prepared file we meet a nice command `\CharacterTable`. 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}

\begingroup
\catcode`\<=1 \catcode`\>=2 %
\@makeother\{\@makeother\}%
\firstofone<\endgroup
```
\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 2102.

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{%
  \begingroup\MakePrivateLetters
  \gmd@ifonetoken\Describe@Macro\Describe@Env}
```

Note that if the argument to `\DescribeMacro` is not a (possibly starred) control sequence, then as an environment's name shall it be processed *except* the `\MakePrivateOthers` re\catcodeing shall not be done to it.

\DescribeEnv
```
\outer\def\DescribeEnv{%
  \begingroup\MakePrivateOthers\Describe@Env}
```

Actually, I've used the `\Describe...` commands myself a few times, so let's `\def` a common command with a starred version:

\Describe
```
\outer\def\Describe{% It doesn't typeset its argument in the point of occurrence.
  \begingroup\MakePrivateLetters
  \@ifstarl{\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\ignorespaces}
```

\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}\ignorespaces}
```

Note that here the environments' names are typeset in `\tt` font just like the macros', *unlike* in doc.

My understanding of 'minimality' includes avoiding too much freedom as causing chaos not beauty. That's the philosophical and æsthetic reason why I don't provide `\MacroFont`. In my opinion there's a noble tradition of typesetting the TEX code in `\tt` font nad this tradition sustained should be. If one wants to change the tradition, let her redefine `\tt`, in TEX it's no problem. I suppose `\MacroFont` is not used explicitly, and that it's (re)defined at most, but just in case let's `\let`:

\MacroFont

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

We have provided `\CodeIndent` in line 57. And it corresponds with doc's `\MacroIndent` so

\CodeIndent
\MacroIndent

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

\MacroIndent

And similarly the other skips:

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

\MacrocodeTopsep

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

\MacroTopsep

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

\SpecialEscapechar

`\theCodelineNo` is not used in gmdoc. Instead of it there is `\LineNumFont` declaration and a possibility to redefine `\thecodelinenum` as for all the counters. Here the `\LineNumFont` is used two different ways, to set the benchmark width for a linenumber among others, so it's not appropriate to put two things into one macro. Thus let's give the user a notice if he defined this macro:

\theCodelineNo
\LineNumFont

Because of possible localness of the definitions it seems to be better to add a check at the end of each `\DocInput` or `\IndexInput`.

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

I hope this lack will not cause big trouble.

For further notifications let's define a shorthand:

\noeffect@info

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

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

\PrintDescribeMacro
\PrintMacroName
\PrintDescribeEnv
\PrintEnvName

Let us just notify the user.

```
\AtEndInput{%
  \noeffect@info{PrintDescribeMacro}%
```

```
    \noeffect@info{PrintMacroName}%
    \noeffect@info{PrintDescribeEnv}%
    \noeffect@info{PrintEnvName}}
```

\CodelineNumbered

The \CodelineNumbered declaration of doc seems to be equivalent to our noindex option with the linesnotnum option set off so let's define it such a way.

\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 she'll not use gmdoc's options but just the default.

The \CodelineIndex and \PageIndex declarations correspond with the gmdoc's default and the pageindex option respectively. Therefore let's \let

```
\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}}% we define \Finale just to
        expand to the argument of \StopEventually not to to add anything to the
        end input hook because \Finale should only be executed if entire document
        is typeset.
  %\init@checksum is obsolete in gmdoc at this point: the CheckSum counter is reset just
        at the beginning of (each of virtually numerous) input(s).
    \@esphack}
\AlsoImplementation
```

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

\OnlyDescription \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 TEXbook* format's) disguise \<⟨*the argument*⟩>.

The checksum mechanism is provided and developed for a slightly different purpose.

Most of doc's indexing commands have already been 'almost defined' in gmdoc:

```
\let\SpecialMainIndex=\DefIndex
\def\SpecialMainEnvIndex{\csname CodeDefIndex\endcsname*}% we don't type
    \DefIndex explicitly here because it's \outer, remember?
\let\SpecialIndex=\CodeCommonIndex
\let\SpecialUsageIndex=\TextUsgIndex
\def\SpecialEnvIndex{\csname TextUsgIndex\endcsname*}
\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.

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.

```
\def\verbatimchar{&}
```

One more a very neat macro provided by doc. I copy it verbatim and put into gmutils, too. (`\DeclareRobustCommand` doesn't issue an error if its argument has been defined, it only informs about redefining.)

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

`\IndexPrologue` is defined in line 1366. And other doc index commands too.

```
\@ifundefined{main}{}{\let\DefEntry=\main}
\@ifundefined{usage}{}{\let\UsgEntry=\usage}
```

About how the DocStrip directives are supported by gmdoc, see section The DocStrip…. This support is not *that* sophisticated as in doc, among others, it doesn't count the modules' nesting. Therefore if we dont want an error while gmdocumenting doc-prepared files, better let's define doc's counter for the modules' depths.

```
\newcounter{StandardModuleDepth}
```

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

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

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

```
\@emptify\DontCheckModules \noeffect@info{DontCheckModules}
\@emptify\CheckModules \noeffect@info{CheckModules}
```

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

```
\@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
\MakePercentComment
```
\def\MakePercentIgnore{\catcode`\%9\relax}
\def\MakePercentComment{\catcode`\%14\relax}
```

gmdocing doc.dtx

The author(s) of doc suggest(s):

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

Therefore I hope that after doc.dtx has been gmdoc-ed, one can say gmdoc is doc-compatible "at most—if not at all".

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

One of minor errors was caused by my understanding of a 'shortverb' char: due to gmverb, in the math mode an active 'shortverb' char expands to itself's 'other' version thanks to `\string` (It's done with | in mind). doc's concept is different, there a 'shortverb' char should in the math mode work as shortverb. So let it be as they wish: gmverb provides `\OldMakeShortVerb` and the oldstyle input commands change the inner macros so that also `\MakeShortVerb` works as in doc (cf. line 2098).

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
    \VerbMacrocodes}%
  \bgroup\@makeother\_% it's to allow _ in the filenames. The next macro will close
        the group.
  \Doc@Input}
```

We don't swith the `@codeskipput` switch neither we check it because in 'old' world there's nothing to switch this switch in the narration layer.

I had a hot and wild TEX all the night nad what a bliss when the 'Succesfully formated 67 page(s)' message appeared.

My package needed fixing some bugs and adding some compatibility adjustments (listed in the previous section) and the original doc.dtx source file needed a few adjustments too because some crucial differences came out. I'd like to write a word about them now.

The first but not least is that the author(s) of doc give the CS marking commands non-macro arguments sometimes, e.g., `\DescribeMacro{StandardModuleDepth}`. Therefore we should launch the *starred* versions of corresponding gmdoc commands. This means the doc-like commands will not look for the CS's occurrence in the code but will mark the first codeline met.

Another crucial difference is that in gmdoc the narrative and the code layers are separated with only the code delimiter and therefore may be much more mixed than in doc. among others, the `macro` environment is *not* a typical `verbatim` like: the texts commented out within `macrocode` are considered a normal commentary i.e., not verbatim. Therefore some macros 'commented out' to be shown verbatim as an example source must have been 'additionally' verbatimized for gmdoc with the shortverb chars e.g. You may also change the code delimiter for a while, e.g., the line

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

```
% \AVerySpecialMacro % delete the first % when...
```

was got with

```
\CodeDelim\.
% \AVerySpecialMacro % delete the first % when.\unskip|..|\CDPerc
```

One more difference is that my shortverb chars expand to their $_{12}$ versions in the math mode while in doc remain shortverb, so I added a declaration `\OldMakeShortVerb` etc.

Moreover, it's TEXing doc what inspired adding the `\StraightEOL` and `\QueerEOL` declarations.

Polishing, Development and Bugs

• `\MakePrivateLetters` theoretically may interfere with `\active`ating some chars to allow linebreaks. But making a space or an opening brace a letter seems so perverse that we may feel safe not to take account of such a possibility.

• When `countalllines` option is enabled, the comment lines that don't produce any printed output result with a (blank) line too because there's put a hypertarget at the beginning of them. But for now let's assume this option is for draft versions so hasn't be perfect.

• Marcin Woliński suggests to add the marginpar clauses for the AMS classes as we did for the standard ones in the lines 20–22. Most probably I can do it on request when I only know the classes' names and their 'marginpar status'.

• When the `countalllines` option is in force, some `\list` environments shall raise the 'missing `\item`' error if you don't put the first `\item` in the same line as `\begin{%` ⟨*environment*⟩`}` because the (comment-) line number is printed.

• I'm prone to make the control sequences hyperlinks to the(ir) 'definition' occurrences. It doesn't seem to be a big work compared with what has been done so far.

• Is `\RecordChanges` really necessary these days? Shouldn't be the `\makeglossary` command rather executed by default?[13]

• Do you use `\listoftables` and/or `\listoffigures` in your documentations? If so, I should 'EOL-straighten' them like `\tableofcontents`, I suppose (cf. line 148).

• 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.

• My TEX Guru remarked that `\jobname` expands to the main file name without extension iff that extension is .tex. Should I replace `\jobname` with `\jobnamewoe` then? (The latter always expands to the file name without extension.)

• About the DocStrip verbatim mode directive see above.

(No) ⟨*eof*⟩

If the user doesn't wish the documentation to be ended by ⟨*eof*⟩, he should end the file with a comment ending with `\NoEOF` macro defined below[14]:

```
\@NoEOF
```

```
\bgroup\catcode`\^^M\active \firstofone{\egroup%
  \def\@NoEOF#1^^M{%
```

[13] It's understandable that ten years earlier writing things out to the files remarkably decelerated TEX, but nowadays it does not in most cases. That's why `\makeindex` is launched by default in gmdoc.

[14] Thanks to Bernd Raichle at BachoTEX 2006 Pearl Session where he presented `\input`ing a file inside `\edef`.

```
      \@relaxen\EOFMark\@xa\@nx\endinput^^M}}
\NoEOF \def\NoEOF{\QueerEOL\@NoEOF}
```

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

```
\endinput
⟨/package⟩
```

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

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


```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{gmdocc}
            [2007/11/19 v0.78 a class for gmdoc driver files (GM)]
```

Intro

This file is a part of gmdoc bundle and provides a document class for the driver files documenting (LA)TEX packages &a. with my gmdoc.sty package. It's not necessary, of course: most probably you may use another document class you like.

By default this class loads mwart class with a4paper (default) option and lmodern package with 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 a). The rôle of the gmdocc document class is rather auxiliary and exemplary. Most probably, you may use your favourite document class with the settings you wish. This class I wrote to meet my needs of fine formatting, such as not numbered sections and sans serif demi bold headings.

However, with the users other than myself in mind, I added some conditional clauses that make this class works also if an mwcls class or the lmodern package are unknown.

Of rather many options supported by gmdoc.sty, this class chooses my favourite, i.e., the default. An exception is made for the `noindex` option, which is provided by this class and passed to gmdoc.sty. This is intended for the case you don't want to make an index.

noindex

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

nochanges

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.)

[1] This file has version number v0.78 dated 2007/11/19.

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 homonimic with the class'es 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

\EOFMark The `\EOFMark` in this class typesets like this (of course, you can redefine it as you wish):

□

The Code

```
4 \RequirePackage{xkeyval}
```

A shorthands for options processing (I know xkeyval to little to redefine the default prefix and family).

```
5 \newcommand*\gm@DOX{\DeclareOptionX[gmcc]<>}
6 \newcommand*\gm@EOX{\ExecuteOptionsX[gmcc]<>}
```

\gm@DOX
\gm@EOX

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.

```
7 \newif\ifgmcc@mwcls
```

\ifgmcc@mwcls

Note that the following option defines `\gmcc@class#1`.

```
8 \gm@DOX{class}{% the default will be Marcin Woliński class (mwcls) analogous to
      article, see line 39.
9   \def\gmcc@CLASS{#1}%
10  \@for\gmcc@resa:=mwart,mwrep,mwbk\do {%
11    \ifx\gmcc@CLASS\gmcc@resa\gmcc@mwclstrue\fi}%
12 }
13 \gm@DOX{mwart}{\gmcc@class{mwart}}% The mwart class may also be declared ex-
      plicitly.
14 \gm@DOX{mwrep}{\gmcc@class{mwrep}}% If you need chapters, this option chooses
      an MW class that corresponds to report,
15 \gm@DOX{mwbk}{\gmcc@class{mwbk}}% and this MW class corresponds to book.
```

class
\gmcc@CLASS
mwart
mwrep
mwbk

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 occured: '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 writing outto .idx file off.

\if@gmccnochanges
```
\newif\if@gmccnochanges
```

nochanges
```
\gm@DOX{nochanges}{\@gmccnochangestrue}%
```
This option turns the writing outto .glo file off.

gmeometric
```
\gm@DOX{gmeometric}{}%
```
The gmeometric package causes the `\geometry` macro provided by geometry package is not restricted to the preamble.

Since version 0.99g of gmdoc the bundle goes XeTeX and that means geometry should be loaded with `dvipdfm` option and the `\pdfoutput` counter has to be declared and that's what gmeometric does by default if with XeTeX. And gmeometric has passed enough practical test. Therefore the `gmeometric` option becomes obsolete and the package is loaded always instead of original geometry.

As already mentioned, 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
```
\newif\ifgmcc@oldfonts
\gmcc@oldfontstrue
```

sysfonts
```
\gm@DOX{sysfonts}{\gmcc@oldfontsfalse}
```

Now we define a key-val option that sets the version of marginpar typewriter font definition (relevant only with the `sysfonts` option). 0 for OpenType LMTT LC visible for the system (not on my computer), 1 for LMTT LC specially on my computer, any else number to avoid an error if you don't have OpenType LMTT LC installed (and leave the default gmdoc's definition of `\marginpartt`; all the versions allow the user to define marginpar typewriter herself).

mptt
\mpttversion
```
\gm@DOX{mptt}[17]{\def\mpttversion{#1}}%
```
the default value (17) works if the user puts the `mptt` option with no value. In that case leaving the default gmdoc's definition of marginpar typewriter and letting the user to redefine it himself seemed to me most natural.

\gmcc@setfont
```
\def\gmcc@setfont#1{%
  \gmcc@oldfontsfalse% note that if we are not in XeTeX, this switch will be turned true in line 57
  \AtBeginDocument{%
    \@ifXeTeX{%
      \setromanfont[Mapping=tex-text]{#1}%
      \let\sl\it \let\textsl\textit
```

```
    }{}}%
}
\gm@DOX{minion}{\gmcc@setfont{Minion Pro}}
\gm@DOX{pagella}{\gmcc@setfont{TeX Gyre Pagella}}
\gm@EOX{class=mwart}% We set the default basic class to be mwart.
\gm@EOX{mptt=0}% We default to set the marginpar typewriter font to OpenType
      LMTT LC.
\ProcessOptionsX[gmcc]<>
\ifgmcc@mwcls
  \IfFileExists{\gmcc@CLASS.cls}{}{\gmcc@mwclsfalse}% As announced, we do
      the ontological test to any mwcls.
\fi
\ifgmcc@mwcls
  \XKV@ifundefined{XeTeXdefaultencoding}{}{%
    \XeTeXdefaultencoding "cp1250"}% mwcls are encoding-sensitive because MW
      uses Polish diacritics in the commentaries.
  \LoadClass[fleqn, oneside, noindentfirst, 11pt, withmarginpar,
  sfheadings]{\gmcc@CLASS}%
  \XKV@ifundefined{XeTeXdefaultencoding}{}{%
    \XeTeXdefaultencoding "utf-8"}%
\else
  \LoadClass[fleqn, 11pt]{article}% Otherwise the standard article is loaded.
\fi
\RequirePackage{gmutils}% earlier to provide \@ifXeTeX.
\ifgmcc@mwcls\afterfi\ParanoidPostsec\fi
\@ifXeTeX{}{\gmcc@oldfontstrue}
\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 `\leftskip` in `verbatim`. Thanks to that and the `\edverbs` declaration below you may display single verbatim lines with `\[...\]`:

```
\[|\verbatim\stuff|\] .
```

```
\ifgmcc@oldfonts
  \IfFileExists{lmodern.sty}{% We also examine the ontological status of this
      package
    \RequirePackage{lmodern}% and if it shows to be satisfactory (the package
      shows to be), we load it and set the proper font encoding.
    \RequirePackage[T1]{fontenc}%
  }{}%
```

A couple of diacritics I met while gmdocing these files and The Source etc. Somewhy the accents didn't want to work at my XeTeX settings so below I define them for XeTeX as respective chars.

```
  \def\agrave  {\`a}%
  \def\cacute  {\'c}%
  \def\eacute  {\'e}%
  \def\idiaeres{\"\i}%
  \def\nacute  {\'n}%
  \def\ocircum {\^o}%
```

```
\oumlaut   \def\oumlaut {\"o}%
\uumlaut   \def\uumlaut {\"u}%
\else% this case hapens only with XeTeX.
  \let\do\relaxen
  \do\Finv\do\Game\do\beth\do\gimel\do\daleth% these five caused the 'already defined' error.
  \XeTeXthree
  \let\fontencoding\@gobble
\agrave    \def\agrave  {\char"00E0 }%
\cacute    \def\cacute  {\char"0107 }% Note the space to be sure the number ends here.
\eacute    \def\eacute  {\char"00E9 }%
\idiaeres  \def\idiaeres{\char"00EF }%
\nacute    \def\nacute  {\char"0144 }%
\oumlaut   \def\oumlaut {\char"00F6 }%
\uumlaut   \def\uumlaut {\char"00FC }%
\ocircum   \def\ocircum {\char"00F4 }%
  \AtBeginDocument{%
\ae          \def\ae{\char"00E6 }%
             \def\l {\char"0142 }%
\oe          \def\oe{\char"0153 }%
  }%
\fi
```

Now we set the page layout.

```
\RequirePackage{gmeometric}
\gmdoccMargins  \def\gmdoccMargins{%
  \geometry{top=77pt, height=687pt, % =53 lines but the lines option seems not to work 2007/11/15 with TeX Live 2007 and XeTeX 0.996-patch1
    left=4cm, right=2.2cm}}
\gmdoccMargins
\if@debug% For debugging we load also the trace package that was very helpful to me.
  \RequirePackage{trace}%
  \errorcontextlines=100 % And we set an error info parameter.
\fi
\ifdtraceon   \newcommand*\ifdtraceon{\if@debug\afterfi\traceon\fi}
\ifdtraceoff  \newcommand*\ifdtraceoff{\if@debug\traceoff\fi}
```

We load the core package:

```
\RequirePackage{gmdoc}
\ifgmcc@oldfonts
  \@ifpackageloaded{lmodern}{% The Latin Modern font family provides a light condensed typewriter font that seems to be the most suitable for the marginpar CS marking.
\marginpartt    \def\marginpartt{\normalfont\fontseries{lc}\ttfamily}}{}%
\else
  \IfFileExists{/media/BleuDeParis/texmf/tex/xelatex/crop.cfg}%
\mpttversion      {\def\mpttversion{1}}{}% my system doesn't see LMTT LC, so on my computer I have to tell XeTeXexplicitly where the font is.
  \ifcase\mpttversion
    \font\marginpartt= lmtypewriter10-lightcondensed.otf at 10.95pt
  \or% (when =1, on my computer that is)
```

```
    \font\marginpartt= %
    [/home/natror/.fonts/lmtypewriter10-lightcondensed.otf] at 10.95pt
  \else % other values allow you to define \marginpartt yourself.
  \fi
\fi
\raggedbottom
\setcounter{secnumdepth}{0}% We wish only the parts and chapters to be numbered.
\renewcommand*\thesection{\arabic{section}}% isn't it redundant at the above setting?
\@ifnotmw{}{%
  \@ifclassloaded{mwart}{% We set the indentation of Contents:
    \SetTOCIndents{{}{\quad}{\quad}{\quad}{\quad}{\quad}{\quad}}}{% for mwart
    \SetTOCIndents{{}{\bf9.\enspace}{\quad}{\quad}{\quad}{\quad}{\quad}}}%
        and for the two other mwclss.
  \pagestyle{outer}}% We set the page numbers to be printed in the outer and bottom corner of the page.
\def\titlesetup{\bfseries\sffamily}% We set the title(s) to be boldface and sans serif.
\if@gmccnochanges\let\RecordChanges\relax\fi% If the nochanges option is on, we discard writing outto the .glo file.
\RecordChanges% We turn the writing the \changes outto the .glo file if not the above.
\dekclubs% We declare the club sign | to be a shorthand for \verb*.
\edverbs% to redefine \[ so that it puts a shortverb in a \hbox.
\smartunder% and we declare the _ char to behave as usual in the math mode and outside math to be just an uderscore.
\exhyphenpenalty\hyphenpenalty% 'cause mwcls set it =10000 due to Polish customs.
\VerbT1% to ensure T1 font encoding in verbatims (in code).
\RequirePackage{amssymb}
\def\EOFMark{\rightline{\ensuremath{\square}}}
\endinput
```

c. gmdocDoc.tex, The Driver File

```
\documentclass[outeroff,mwrep,pagella]{gmdocc}
\twocoltoc
\title{The \pk{gmdoc} Package\\ i.e., \pk{gmdoc.sty} and
  \pk{gmdocc.cls}}
\author{Grzegorz `Natror' Murzynowski}
\date{November 2007}
\begin{document}
\maketitle
\setcounter{page}{2}% hyperref cries if it sees two pages numbered 1.
\tableofcontents
\DoIndex\maketitle
\DocInclude{gmdoc}
\DocInclude{gmdocc}
\SelfInclude{\def\CommonEntryCmd{UsgEntry}%
  \filediv[\file{gmdocDoc.tex}, The Driver
  File]{\file{gmdocDoc.tex}, \gmhypertarget{The Driver} File}%
  \label{Driver}}
```

\CommonEntryCmd

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

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

MakeIndex shell commands:

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

(gmglo.ist should be put into some texmf/makeindex directory.)

And “That’s all, folks” ;-) .

d. The gmutils Package[1]

Written by Grzegorz Murzynowski,
natror at o2 dot pl

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmutils}
     [2007/11/16 v0.85 some rather TeXnical macros, some of them
          tricky (GM)]
```

Intro

The gmutils.sty package provides some macros that are analogous to the standard LaTeX ones but extend their functionality, such as `\@ifnextcat`, `\addtomacro` or `\begin(*)`. The others are just conveniences I like to use in all my TeX works, such as `\afterfi`, `\pk` or `\cs`.

I wouldn't say they are only for the package writers but I assume some nonzero (La)TeX-awareness of the user.

For details just read the code part.

The remarks about installation and compiling of the documentation are analogous to those in the chapter gmdoc.sty and therefore ommitted.

Contents of the gmutils.zip Archive

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

gmutils.sty
README
gmutilsDoc.tex
gmutilsDoc.pdf
gmutils.tds.zip

A couple of abbreviations

\@xa
\@nx

```
\let\@xa\expandafter
\let\@nx\noexpand
```

The `\newgif` declaration's effect is used even in the LaTeX 2_ε source by redefining some particular user defined ifs (UD-ifs henceforth) step by step. The goal is to make the

[1] This file has version number v0.85 dated 2007/11/16.

UD-if's assignment global. I needed it at least twice during gmdoc writing so I make it a macro. It's an almost verbatim copy of LaTeX's `\newif` modulo the letter *g* and the `\global` prefix. (File d: ltdefns.dtx Date: 2004/02/20 Version v1.3g, lines 139–150)

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

'Almost' is also in the detail that in this case, which deals with `\global` assignments, we don't have to bother with storing and restoring the value of `\escapechar`: we can do all the work inside a group.

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

After `\newgif\iffoo` you may type `{\foogtrue}` and the `\iffoo` switch becomes globally equal `\iftrue`. Simili modo `\foogfalse`. Note the letter *g* added to underline globalness of the assignment.

If for any reason, no matter how queer ;-) may it be, you need *both* global and local switchers of your `\if...`, declare it both with `\newif` and `\newgif`.

Note that it's just a shorthand. `\global\if`⟨*switch*⟩`true/false` *does* work as expected.

There's a trouble with `\refstepcounter`: defining `\@currentlabel` is local. So let's `\def` a `\global` version of `\refstepcounter`.

Warning. I use it because of very special reasons in gmdoc and in general it is probably not a good idea to make `\refstepcounter` global since it is contrary to the original LaTeX approach.

\grefstepcounter
```
\newcommand*\grefstepcounter[1]{%
  {\let\protected@edef=\protected@xdef\refstepcounter{#1}}}
```

Naïve first try `\globaldefs=\tw@` raised an error `unknown command \reserved@e`. The matter was to globalize `\protected@edef` of `\@currentlabel`.

Thanks to using the true `\refstepcounter` inside, it observes the change made to `\refstepcounter` by hyperref.

Another shorthand. It may decrease a number of `\expandafter`s e.g.

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

LaTeX provides a very useful `\g@addto@macro` macro that adds its second argument to the current definition of its first argument (works iff the first argument is a no argument macro). But I needed it some times in a document, where `@` is not a letter. So:

\gaddtomacro
```
\let\gaddtomacro=\g@addto@macro
```

The redefining of the first argument of the above macro(s) is `\global`. What if we want it local? Here we are:

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

And for use in the very document,

```
\addtomacro 25 \let\addtomacro=\addto@macro
\addtotoks  26 \long\def\addtotoks#1#2{%
            27   #1=\@xa{\the#1#2}}
\@emptify   28 \newcommand*\@emptify[1]{\let#1=\@empty}
 \emptify   29 \@ifdefinable\emptify{\let\emptify\@emptify}
```

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

```
\g@emptify  30 \newcommand*\g@emptify{\global\@emptify}
 \gemptify  31 \@ifdefinable\gemptify{\let\gemptify\g@emptify}
\@relaxen   32 \newcommand*\@relaxen[1]{\let#1=\relax}
 \relaxen   33 \@ifdefinable\relaxen{\let\relaxen\@relaxen}
```

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

```
\g@relaxen  34 \newcommand*\g@relaxen{\global\@relaxen}
 \grelaxen  35 \@ifdefinable\grelaxen{\let\grelaxen\g@relaxen}
```

For the heavy debugs I was doing while preparing gmdoc, as a last resort I used `\showlists`. But this command alone was usually too little: usually it needed setting `\showboxdepth` and `\showboxbreadth` to some positive values. So,

```
\gmshowlists 36 \def\gmshowlists{\showboxdepth=1000 \showboxbreadth=1000 \showlists}
\nameshow    37 \newcommand*\nameshow[1]{\@xa\show\csname#1\endcsname}
```

Standard `\string` command returns a string of 'other' chars except for the space, for which it returns $_{10}$. In gmdoc I needed the spaces in macros' and environments' names to be always $_{12}$, so I define

```
\xiistring  38 \def\xiistring#1{%
            39   \if\@nx#1\xiispace
            40     \xiispace
            41   \else
            42     \string#1%
            43   \fi}
```

`\@ifnextcat, \@ifnextac`

As you guess, we `\def \@ifnextcat` à la `\@ifnextchar`, see LaTeX 2_ε source dated 2003/12/01, file d, lines 253–271. The difference is in the kind of test used: while `\@ifnextchar` does `\ifx`, `\@ifnextcat` does `\ifcat` which means it looks not at the meaning of a token(s) but at their `\catcode`(s). As you (should) remember from *The TEXbook*, the former test doesn't expand macros while the latter does. But in `\@ifnextcat` the peeked token is protected against expanding by `\noexpand`. Note that the first parameter is not protected and therefore it shall be expanded if it's a macro. Because an assignment is involved, you can't test whether the next token is an active char.

```
\@ifnextcat 44 \long\def\@ifnextcat#1#2#3{%
            45   \def\reserved@d{#1}%
            46   \def\reserved@a{#2}%
            47   \def\reserved@b{#3}%
            48   \futurelet\@let@token\@ifncat}
  \@ifncat  49 \def\@ifncat{%
            50   \ifx\@let@token\@sptoken
            51     \let\reserved@c\@xifncat
            52   \else
```

```
    \ifcat\reserved@d\@nx\@let@token
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
{\def\:{\let\@sptoken= } \: % this makes \@sptoken a space token.
\def\:{\@xifncat} \@xa\gdef\: {\futurelet\@let@token\@ifncat}}
```

Note the trick to get a macro with no parameter and requiring a space after it. We do it inside a group not to spoil the general meaning of `\:` (which we extend later).

The next command provides the real `\if` test for the next token. *It* should be called `\@ifnextchar` but that name is assigned for the future `\ifx` text, as we know. Therefore we call it `\@ifnextif`.

\@ifnextif
```
\long\def\@ifnextif#1#2#3{%
  \def\reserved@d{#1}%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
  \futurelet\@let@token\@ifnif}
```

\@ifnif
```
\def\@ifnif{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifnif
  \else
    \if\reserved@d\@nx\@let@token
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
{\def\:{\let\@sptoken= } \: % this makes |\@sptoken| a space token.
\def\:{\@xifnif} \@xa\gdef\: {\futurelet\@let@token\@ifnif}}
```

But how to peek at the next token to check whether it's an active char? First, we look with `\@ifnextcat` whether there stands a group opener. We do that to avoid taking a whole `{...}` as the argument of the next macro, that doesn't use `\futurelet` but takes the next token as an argument, tests it and puts back intact.

\@ifnextac
```
\long\def\@ifnextac#1#2{%
  \@ifnextcat\bgroup{#2}{\gm@ifnac{#1}{#2}}}
```

\gm@ifnac
```
\long\def\gm@ifnac#1#2#3{%
  \ifcat\@nx~\@nx#3\afterfi{#1#3}\else\afterfi{#2#3}\fi}
```

Yes, it won't work for an active char `\let` to `{`$_1$, but it *will* work for an active char `\let` to a char of catcode $\neq 1$. (Is there anybody on Earth who'd make an active char working as `\bgroup`?)

Now, define a test that checks whether the next token is a genuine space, $_{10}$ that is. First define a CS let such a space. The assignment needs a little trick (*The TEXbook* appendix D) since `\let`'s syntax includes one optional space after `=`.

```
\let\gmu@reserveda\*%
```

*
```
\def\*{%
```

```
  \let\*\gmu@reserveda
  \let\gm@letspace= }%
\* %
\@ifnextspace \def\@ifnextspace#1#2{%
  \let\@reserveda\*%
\*  \def\*{%
    \let\*\@reserveda
    \ifx\@let@token\gm@letspace\afterfi{#1}%
    \else\afterfi{#2}%
    \fi}%
  \futurelet\@let@token\*}
```

First use of this macro is for an active - that expands to --- if followed by a space. Another to make dot checking whether is followed by ~ without gobbling the space if it occurs instead.

`\afterfi` and Pals

It happens from time to time that you have some sequence of macros in an `\if...` and you would like to expand `\fi` before expanding them (e.g., when the macros should take some tokens next to `\fi...` as their arguments. If you know how many macros are there, you may type a couple of `\expandafters` and not to care how terrible it looks. But if you don't know how many tokens will there be, you seem to be in a real trouble. There's the Knuthian trick with `\next`. And here another, revealed to me by my TEX Guru.

I think the situations when the Knuthian (the former) trick is not available are rather seldom, but they are imaginable at least: the `\next` trick involves an assignment so it won't work e.g. in `\edef`. But in general it's only a matter of taste which one to use.

One warning: those macros peel the braces off, i.e.,

```
\if..\afterfi{\@makeother\^^M}\fi
```

causes a leakage of `^^M`$_{12}$. To avoid pollution write

```
\if..\afterfi{\bgroup\@makeother\^^M\egroup}\fi.
```

```
\afterfi 97 \long\def\afterfi#1#2\fi{\fi#1}
```

And two more of that family:

```
\afterfifi 98 \long\def\afterfifi#1#2\fi#3\fi{\fi\fi#1}
\afteriffifi 99 \long\def\afteriffifi#1#2\if#3\fi#4\fi{\fi#1}
```

Notice the refined elegance of those macros, that cover both 'then' and 'else' cases thanks to `#2` that is discarded.

```
\afterififfifi 100 \long\def\afterififfifi#1#2\fi#3\fi#4\fi{\fi#1}
\afteriffifi 101 \long\def\afteriffifi#1#2\fi#3\fi#4\fi{\fi\fi#1}
\afterfifi 102 \long\def\afterfifi#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}
```

Almost an Environment or Redefinition of `\begin`

We'll extend the functionality of `\begin`: the non-starred instances shall act as usual and we'll add the starred version. The difference of the latter will be that it won't check whether the 'environment' has been defined so any name will be allowed.

This is intended to structure the source with named groups that don't have to be especially defined and probably don't take any particular action except the scoping.

(If the \begin*'s argument is a (defined) environment's name, \begin* will act just like \begin.)

Original LaTeX's \begin:

```
\def\begin#1{%
  \@ifundefined{#1}%
    {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
    {\def\reserved@a{\def\@currenvir{#1}%
        \edef\@currenvline{\on@line}%
        \csname #1\endcsname}}%
    \@ignorefalse
    \begingroup\@endpefalse\reserved@a}
```

\@begnamedgroup
\@begnamedgroup
\@currenvir

```
\@ifdefinable\@begnamedgroup{\relax}
\def\@begnamedgroup#1{%
  \@ignorefalse% not to ignore blanks after group
  \begingroup\@endpefalse
  \def\@currenvir{#1}%
  \edef\@currenvline{\on@line}%
  \csname #1\endcsname}% if the argument is a command's name (an environment's
```

e.g.), this command will now be executed. (If the corresponding control sequence hasn't been known to TeX, this line will act as \relax.)

For back compatibility with my earlier works

\bnamegroup

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

And for the ending

\enamegroup

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

And we make it the starred version of \begin.

\old@begin
\begin
\begin*
\begin

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

Improvement of \end

It's very clever and useful that \end checks whether its argument is ifx-equivalent @currenvir. However, it works not quite as I would expect: Since the idea of environment is to open a group and launch the cs named in the \begin's argument. That last thing is done with \csname . . . \endcsname so the char catcodes are equivalent. Thus should be also in the \end's test and therefore we ensure the compared texts are both expanded and made all 'other'.

\@checkend

```
\def\@checkend#1{%
  \edef\reserved@a{\@xa\string\csname#1\endcsname}%
  \edef\exii@currenvir{\@xa\string\csname\@currenvir\endcsname}%
  \ifx\reserved@a\exii@currenvir\else\@badend{#1}\fi}
```

Thanks to it you may write \begin{macrocode*} with $*_{12}$ and end it with \end{% macrocode*} with $*_{11}$ (that was the problem that led me to this solution). The error messages looked really funny:

```
! LaTeX Error: \begin{macrocode*} on input line 1844 ended by \end{macrocode*}.
```

Of course, you might write also \end{macrocode\star} where \star is defined as 'other' star or letter star.

From relsize

As file relsize.sty, v3.1 dated July 4, 2003 states, LaTeX 2_ε version of these macros was written by Donald Arseneau asnd@triumf.ca and Matt Swift swift@bu.edu after the LaTeX 2.09 smaller.sty style file written by Bernie Cosell cosell@WILMA.BBN.COM.

I take only the basic, non-math mode commands with the assumption that there are the predefined font sizes.

\relsize You declare the font size with `\relsize{⟨n⟩}` where $\langle n \rangle$ gives the number of steps ("mag-step" = factor of 1.2) to change the size by. E.g., $n = 3$ changes from `\normalsize` to `\LARGE` size. Negative n selects smaller fonts. `\smaller` == `\relsize{-1}`; `\larger` == `\relsize{1}`. `\smallerr`(my addition) == `\relsize{-2}`; `\largerr` guess yourself.

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

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

Change the number by the given increment:

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

watch out for size underflow:

```
      \ifnum\@tempcnta<\z@ \rs@size@warning{small}{\string\tiny}%
           \@tempcnta\z@ \fi
      \@xa\endgroup
      \ifcase\@tempcnta   % set new size based on altered number
         \tiny \or \scriptsize \or \footnotesize \or \small \or %
              \normalsize \or
         \large \or \Large \or \LARGE \or \huge \or \Huge \else
          \rs@size@warning{large}{\string\Huge}\Huge
\fi\fi}% end of \relsize.
```

\rs@size@warning
```
\providecommand*\rs@size@warning[2]{\PackageWarning{gmutils
      (relsize)}{%
 Size requested is too #1.\MessageBreak Using #2 instead}}
```

\rs@unknown@warning
```
\providecommand*\rs@unknown@warning{\PackageWarning{gmutils
      (relsize)}{Current font size
 is unknown! (Why?!?)\MessageBreak Assuming \string\normalsize}}
```

And a handful of shorthands:

\larger
```
\DeclareRobustCommand*\larger[1][\@ne]{\relsize{+#1}}
```

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

`\firstofone` and the Queer `\catcodes`

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

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

The next command, `\foone`, is intended as two-argument for shortening of the `\bgroup...\firstofone{\egroup...}` hack.

```
\foone       \long\def\foone#1{\bgroup#1\egroupfirstofone}
             \long\def\egroupfirstofone#1{\egroup#1}
\fooatletter \long\def\fooatletter{\foone\makeatletter}
```

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

```
\gobble      \long\def\gobble#1{}
\gobbletwo   \let\gobbletwo\@gobbletwo
             \foone{\catcode`\_=8 }%
\subs        {\let\subs=_}
             \foone{\@makeother\_}%
\xiiunder    {\def\xiiunder{_}}
             \@ifundefined{XeTeXversion}{}{%
\xiiunder      \def\xiiunder{\char"005F }%
               \let\_\xiiunder}
```

Now, let's define such a smart _ (underscore) which will be usual $_8$ in the math mode and $_{12}$ ('other') outside math.

```
             \foone{\catcode`\_=\active}
             {%
\smartunder    \newcommand*\smartunder{%
                 \catcode`\_=\active
                 \def_{\ifmmode\subs\else\_\fi}}}% We define it as \_ not just as \xiiunder
                       because some font encodings don't have _ at the \char`\_ position.
             \foone{\catcode`\!=0
               \@makeother\\}
\xiibackslash {!newcommand*!xiibackslash{\}}
\bslash      \@ifundefined{bslash}{\let\bslash=\xiibackslash}{}
             \foone{\@makeother\%}
\xiipercent  {\def\xiipercent{%}}
             \foone{\@makeother\&}%
\xiiand      {\def\xiiand{&}}
             \foone{\@makeother\ }%
\xiispace    {\def\xiispace{ }}
```

Metasymbols

I fancy also another Knuthian trick for typesetting ⟨*metasymbols*⟩ in *The TEXbook*. So I repeat it here. The inner `\meta` macro is copied verbatim from doc's v2.1b documentation dated 2004/02/09 because it's so beautifully crafted I couldn't resist. I only don't make it `\long`.

"The new implementation fixes this problem by defining `\meta` in a radically different way: we prevent hypenation by defining a `\language` which has no patterns associated with it and use this to typeset the words within the angle brackets."

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

`\meta`

"Since the old implementation of `\meta` could be used in math we better ensure that this is possible with the new one as well. So we use `\ensuremath` around `\langle` and `\rangle`. However this is not enough: if `\meta@font@select` below expands to `\itshape` it will fail if used in math mode. For this reason we hide the whole thing inside an `\nfss@text` box in that case."

```
  \ensuremath\langle
  \ifmmode \@xa \nfss@text \fi
  {%
    \meta@font@select
```

Need to keep track of what we changed just in case the user changes font inside the argument so we store the font explicitly.

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

But I define `\meta@font@select` as the brutal and explicit `\it` instead of the original `\itshape` to make it usable e.g. in the gmdoc's `\cs` macro's argument.

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

`\meta@font@select`

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

```
\def\<#1>{\meta{#1}}
```

`\<...>`

Macros for Printing Macros and Filenames

First let's define three auxiliary macros analogous to `\dywiz` from polski.sty: a shorthands for `\discretionary` that'll stick to the word not spoiling its hyphenability and that'll won't allow a linebreak just before nor just after themselves. The `\discretionary` TEX primitive has three arguments: `#1` 'before break', `#2` 'after break', `#3` 'without break', remember?

```
\def\discre#1#2#3{\kern0sp\discretionary{#1}{#2}{#3}\penalty10000%
```

`\discre`

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

```
      \hskip0sp\relax}
\discret 199 \def\discret#1{\kern0sp\discretionary{#1}{#1}{#1}\penalty10000%
      \hskip0sp\relax}
```

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

```
200 \def\:{\ifmmode\afterfi{\mskip\medmuskip}\else\afterfi{\discret{}}\fi}
\vs 201 \newcommand*{\vs}{\discre{\textvisiblespace}{}{\textvisiblespace}}
```

Then we define a macro that makes the spaces visible even if used in an argument (i.e., in a situation where re`\catcode`ing has no effect).

```
\printspaces 202 \def\printspaces#1{{\let~=\vs \let\ =\vs \gm@pswords#1 \@@nil}}
\gm@pswords 203 \def\gm@pswords#1 #2\@@nil{%
204   \ifx\relax#1\relax\else#1\fi
205   \ifx\relax#2\relax\else\vs\penalty\hyphenpenalty\gm@pswords#2\@@nil\fi}%
```

note that in the recursive call of `\gm@pswords` the argument string is not extended with a guardian space: it has been already by `\printspaces`.

```
\sfname 206 \DeclareRobustCommand*\sfname[1]{\textsf{\printspaces{#1}}}
\file 207 \let\file\sfname% it allows the spaces in the filenames (and prints them as ␣).
```

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

```
\pk 208 \DeclareRobustCommand*{\pk}[1]{\textsf{\textup{#1}}}
```

Some (if not all) of the below macros are copied from doc and/or ltxdoc.

A macro for printing control sequences in arguments of a macro. Robust to avoid writing an explicit \ into a file. It calls `\ttfamily` not `\tt` to be usable in headings which are boldface sometimes.

```
\cs 209 \DeclareRobustCommand*{\cs}[2][\bslash]{{%
\- 210       \def\-{\discretionary{{\rmfamily-}}{}{}}%
211       \def\{{\char`\{}\def\}{\char`\}}\ttfamily #1#2}}
\env 212 \DeclareRobustCommand*{\env}[1]{\cs[]{#1}}
```

And for the special sequences like `^^A`:

```
213 \foone{\@makeother\^}
\hathat 214   {\DeclareRobustCommand*\hathat[1]{\cs[^^]{#1}}}
```

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

```
\possfil 215 \newcommand*\possfil{\hfil\penalty1000\hfilneg}
```

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

"`\cmd{\foo}` Prints `\foo` verbatim. It may be used inside moving arguments. `\cs{%foo}` also prints `\foo`, for those who prefer that syntax. (This second form may even be used when `\foo` is `\outer`)."

```
\cmd 216 \def\cmd#1{\cs{\@xa\cmd@to@cs\string#1}}
\cmd@to@cs 217 \def\cmd@to@cs#1#2{\char\number`#2\relax}
```

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

```
\marg 218 \def\marg#1{{\ttfamily\char`\{}\meta{#1}{\ttfamily\char`\}}}
```

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

```
\oarg 219 \def\oarg{\@ifnextchar[\@oargsq\@oarg}
\@oarg 220 \def\@oarg#1{{\ttfamily[}\meta{#1}{\ttfamily]}}
\@oargsq 221 \def\@oargsq[#1]{\@oarg{#1}}
```

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

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

But we can have all three in one command.

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

Storing and Restoring the Meanings of CSs

First a Boolean switch af globalness of assignments and its verifier.

```
\ifgmu@SMglobal \newif\ifgmu@SMglobal
\SMglobal       \def\SMglobal{\gmu@SMglobaltrue}
```

The subsequent commands are defined in such a way that you can 'prefix' them with `\SMglobal` to get global (re)storing.

A command to store the current meaning of a CS in another macro to temporarily redefine the CS and be able to set its original meanig back (when grouping is not recommended):

```
\StoreMacro     \def\StoreMacro{%
                  \bgroup\makeatletter\@ifstar\egStore@MacroSt\egStore@Macro}
\egStore@Macro  \long\def\egStore@Macro#1{\egroup\Store@Macro{#1}}
\egStore@MacroSt \long\def\egStore@MacroSt#1{\egroup\Store@MacroSt{#1}}
\Store@Macro    \long\def\Store@Macro#1{%
                  \escapechar92
                  \ifgmu@SMglobal\afterfi\global\fi
                  \@xa\let\csname /gmu/store\string#1\endcsname#1%
                  \global\gmu@SMglobalfalse}
\Store@MacroSt  \long\def\Store@MacroSt#1{%
                  \edef\gmu@smtempa{%
                    \ifgmu@SMglobal\global\fi
                    \@nx\let\@xa\@nx\csname/gmu/store\bslash#1\endcsname%
```

we add backslash because to ensure compatibility between `\(Re)StoreMacro` and `\(Re)StoreMacro*`, that is. to allow writing e.g. `\StoreMacro\kitten` and then `\RestoreMacro*{%kitten}` to restore the meaning of `\kitten`.

```
                    \@xa\@nx\csname#1\endcsname}
                  \gmu@smtempa
                  \global\gmu@SMglobalfalse}% we wish the globality to be just once.
```

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

The starred version, `\StoreMacro*` works with csnames (without the backslash). It's first used to store the meanings of robust commands, when you may need to store not only `\foo`, but also `\csname foo \endcsname`.

The next command iterates over a list of CSs and stores each of them. The CS may be separated with commas but they don't have to.

```
\StoreMacros 250 \long\def\StoreMacros{\bgroup\makeatletter\Store@Macros}
\Store@Macros 251 \long\def\Store@Macros#1{\egroup
252   \gmu@setsetSMglobal
253   \let\gml@StoreCS\Store@Macro
254   \gml@storemacros#1.}
\gmu@setsetSMglobal 255 \def\gmu@setsetSMglobal{%
256   \ifgmu@SMglobal
257     \let\gmu@setSMglobal\gmu@SMglobaltrue
258   \else
259     \let\gmu@setSMglobal\gmu@SMglobalfalse
260   \fi}
```

And the inner iterating macro:

```
\gml@storemacros 261 \long\def\gml@storemacros#1{%
\gmu@reserveda 262   \def\gmu@reserveda{\@nx#1}% My TEX Guru's trick to deal with \fi and such, i.e., to hide #1 from TEX when it is processing a test's branch without expanding.
263   \if\gmu@reserveda.% a dot finishes storing.
264     \global\gmu@SMglobalfalse
265   \else
266     \if\gmu@reserveda,% The list this macro is put before may contain commas and that's O.K., we just continue the work.
267       \afterfifi\gml@storemacros
268     \else% what is else this shall be stored.
269       \gml@StoreCS{#1}% we use a particular CS to may \let it both to the storing macro as above and to the restoring one as below.
270       \afterfifi{\gmu@setSMglobal\gml@storemacros}%
271     \fi
272   \fi}
```

And for the restoring

```
\RestoreMacro 273 \def\RestoreMacro{%
274   \bgroup\makeatletter\@ifstar\egRestore@MacroSt\egRestore@Macro}
\egRestore@Macro 275 \long\def\egRestore@Macro#1{\egroup\Restore@Macro{#1}}
\egRestore@MacroSt 276 \long\def\egRestore@MacroSt#1{\egroup\Restore@MacroSt{#1}}
\Restore@Macro 277 \long\def\Restore@Macro#1{%
278   \escapechar92
279   \ifgmu@SMglobal\afterfi\global\fi
280   \@xa\let\@xa#1\csname /gmu/store\string#1\endcsname
281   \global\gmu@SMglobalfalse}
\Restore@MacroSt 282 \long\def\Restore@MacroSt#1{%
283   \edef\gmu@smtempa{%
284     \ifgmu@SMglobal\global\fi
285     \@nx\let\@xa\@nx\csname#1\endcsname
286     \@xa\@nx\csname/gmu/store\bslash#1\endcsname}% cf. the commentary in line 246.
287   \gmu@smtempa
288   \global\gmu@SMglobalfalse}
\RestoreMacros 289 \long\def\RestoreMacros{\bgroup\makeatletter\Restore@Macros}
\Restore@Macros 290 \long\def\Restore@Macros#1{\egroup
291   \gmu@setsetSMglobal
```

```
  \let\gml@StoreCS\Restore@Macro% we direct the core CS towards restoring and
        call the same iterating macro as in line 254.
  \gml@storemacros#1.}
```

As you see, the `\RestoreMacros` command uses the same iterating macro inside, it only changes the meaning of the core macro.

And to restore *and* use immediately:

`\StoredMacro`
`\Stored@Macro`

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

It happended (see the definition of `\@docinclude` in gmdoc.sty) that I needed to `\relax` a bunch of macros and restore them after some time. Because the macros were rather numerous and I wanted the code more readable, I wanted to `\do` them. After a proper defining of `\do` of course. So here is this proper definition of `\do`, provided as a macro (a declaration).

`\StoringAndRelaxingDo`

```
\long\def\StoringAndRelaxingDo{%
  \gmu@SMdo@setscope
  \long\def\do##1{%
    \gmu@SMdo@scope
    \@xa\let\csname /gmu/store\string##1\endcsname##1%
    \gmu@SMdo@scope\let##1\relax}}
```

`\gmu@SMdo@setscope`

```
\def\gmu@SMdo@setscope{%
  \ifgmu@SMglobal\let\gmu@SMdo@scope\global
  \else\let\gmu@SMdo@scope\relax
  \fi
  \global\gmu@SMglobalfalse}
```

And here is the counter-definition for restore.

`\RestoringDo`

```
\long\def\RestoringDo{%
  \gmu@SMdo@setscope
  \long\def\do##1{%
    \gmu@SMdo@scope
    \@xa\let\@xa##1\csname /gmu/store\string##1\endcsname}}
```

Note that both `\StoringAndRelaxingDo` and `\RestoringDo` are sensitive to the `\SMglobal` 'prefix'.

And to store a cs as explicitly named cs, i.e. to `\let` one csname another (`\n@melet` not `\@namelet` becasuse the latter is defined in Till Tantau's beamer class another way) (both arguments should be text):

`\n@melet`

```
\def\n@melet#1#2{%
  \edef\gmu@nl@reserveda{%
    \let\@xa\@nx\csname#1\endcsname
    \@xa\@nx\csname#2\endcsname}%
  \gmu@nl@reserveda}
```

The `\global` prefix doesn't work with `\n@melet` so we define the alternative.

`\gn@melet`

```
\def\gn@melet#1#2{%
  \edef\gmu@nl@reserveda{%
    \global\let\@xa\@nx\csname#1\endcsname
    \@xa\@nx\csname#2\endcsname}%
  \gmu@nl@reserveda}
```

Not only preamble!

Let's remove some commands from the list to erase at begin document! Primarily that list was intended to save memory not to forbid anything. Nowadays, when memory is cheap, the list of only-preamble commands should be rethought IMO.

\not@onlypreamble
```
\newcommand\not@onlypreamble[1]{{%
  \def\do##1{\ifx#1##1\else\@nx\do\@nx##1\fi}%
  \xdef\@preamblecmds{\@preamblecmds}}}
\not@onlypreamble\@preamblecmds
\not@onlypreamble\@ifpackageloaded
\not@onlypreamble\@ifclassloaded
\not@onlypreamble\@ifl@aded
\not@onlypreamble\@pkgextension
```

And let's make the message of only preamble command's forbidden use informative a bit:

\gm@notprerr
```
\def\gm@notprerr{ can be used only in preamble (\on@line)}
\AtBeginDocument{%
  \def\do#1{\@nx\do\@nx#1}%
  \edef\@preamblecmds{%
    \def\@nx\do##1{%
      \def##1{! \@nx\string##1 \@nx\gm@notprerr}}%
    \@preamblecmds}}
```

Third Person Pronouns

Is a reader of my documentations 'she' or 'he' and does it make a difference?

Not to favour any gender in the personal pronouns, define commands that'll print alternately masculine and feminine pronoun of third person. By 'any' I mean not only typically masculine and typically feminine but the entire amazingly rich variety of people's genders, *including* those who do not describe themselves as 'man' or 'woman'.

One may say two pronouns is far too little to cover this variety but I could point Ursula's K. LeGuin's *The Left Hand Of Darkness* as another acceptable answer. In that moody and moderate SF novel the androgynous persons are usually referred to as 'mister', 'sir' or 'he': the meaning of reference is extended. Such an extension also my automatic pronouns do suggest. It's *not* political correctness, it's just respect to people's diversity.

gm@PronounGender
```
\newcounter{gm@PronounGender}
```

\gm@atppron
```
\newcommand*\gm@atppron[2]{%
  \stepcounter{gm@PronounGender}% remember \stepcounter is global.
  \ifodd\value{gm@PronounGender}#1\else#2\fi}
```

\heshe
```
\newcommand*\heshe{\gm@atppron{he}{she}}
```
\hisher
```
\newcommand*\hisher{\gm@atppron{his}{her}}
```
\himher
```
\newcommand*\himher{\gm@atppron{him}{her}}
```
\hishers
```
\newcommand*\hishers{\gm@atppron{his}{hers}}
```

\HeShe
```
\newcommand*\HeShe{\gm@atppron{He}{She}}
```
\HisHer
```
\newcommand*\HisHer{\gm@atppron{His}{Her}}
```
\HimHer
```
\newcommand*\HimHer{\gm@atppron{Him}{Her}}
```
\HisHers
```
\newcommand*\HisHers{\gm@atppron{His}{Hers}}
```

To Save Precious Count Registers

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

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

Need an explanation? The `\nummacro` declaration defines its argument (that should be a CS) as `{0}` which is analogous to `\newcount` declaration but doesn't use up any count register.

Then you may use this numeric macro as something between TEX's count CS and LATEX's counter. The macros `\stepnummacro` and `\addtonummacro` are analogous to LATEX's `\stepcounter` and `\addtocounter` respectively: `\stepnummacro` advances the number stored in its argument by 1 and `\addtonummacro` advances it by the second argument. As the LATEX's analogoi, they have the global effect (the effect of global warming ;-)).

So far I've used only `\nummacro` and `\stepnummacro`. Notify me if you use them and whether you need sth. more, `\multiplynummacro` e.g.

Improvements to mwcls Sectioning Commands

That is, 'Expe-ri-mente'[3] mit MW sectioning & `\refstepcounter` to improve mwcls's cooperation with hyperref. They shouldn't make any harm if another class (non-mwcls) is loaded.

We `\refstep` sectioning counters even if the sectionings are not numbered, because otherwise

1. pdfTEX cried of multiply defined `\labels`,
2. e.g. in a table of contents the hyperlink `<rozdzia\l\ Kwiaty polskie>` linked not to the chapter's heading but to the last-before-it change of `\ref`.

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

Auxiliary macros for the kernel sectioning macro:

[3] A. Berg, *Wozzeck*.

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

To avoid \defing of \mw@sectionxx if it's undefined, we redefine \def to gobble the definition and restore the original meaning of itself.

Why shouldn't we change the ontological status of \mw@sectionxx (not define if undefined)? Because some macros (in gmdocc e.g.) check it to learn whether they are in an mwcls or not.

But let's make a shorthand for this test since we'll use it three times in this package and maybe also somewhere else.

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

I know it may be of bad taste (to write such a way *here*) but I feel so lonely and am in an alien state of mind after 3 hour sleep last night and, worst of all, listening to sir Edward Elgar's flamboyant Symphonies d'Art Nouveau.

A *decent* person would just wrap the following definition in \@ifundefined's Else. But look, the definition is so long and I feel so lonely etc. So, I define \def (for some people there's nothing sacred) to be a macro with two parameters, first of which is delimited by digit 4 (the last token of \mw@sectionxx's parameter string) and the latter is undelimited which means it'll be the body of the definition. Such defined \def does nothing else but restores its primitive meaning by the way sending its arguments to the Gobbled Tokens' Paradise. Luckily, \RestoreMacro contains \let not \def.

The kernel of MW's sectioning commands:

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

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

```
      \@ifundefined{if@mainmatter}{}{%
           \gm@dontnumbersectionsoutofmainmatter}
  \fi
%   \ifHeadingNumbered
%     \refstepcounter{#1}%
%     \protected@edef\HeadingNumber{\csname the#1\endcsname\relax}%
%   \else
%     \let\HeadingNumber\@empty
%   \fi
```

\HeadingRHeadText
```
  \def\HeadingRHeadText{#2}%
```
\HeadingTOCText
```
  \def\HeadingTOCText{#3}%
```
\HeadingText
```
  \def\HeadingText{#4}%
```
\mw@HeadingType
```
  \def\mw@HeadingType{#1}%
  \if\mw@HeadingBreakBefore
    \if@specialpage\else\thispagestyle{closing}\fi
    \@ifundefined{if@openright}{}{\gm@clearpagesduetoopenright}%
    \if\mw@HeadingBreakAfter
```

```
      \thispagestyle{blank}\else
      \thispagestyle{opening}\fi
       \global\@topnum\z@
  \fi% of \if\mw@HeadingBreakBefore
```

placement of \refstep suggested by me (GM)

```
  \ifHeadingNumbered
    \refstepcounter{#1}%
    \protected@edef\HeadingNumber{\csname the#1\endcsname\relax}%
  \else
    \let\HeadingNumber\@empty
    \gm@hyperrefstepcounter
  \fi% of \ifHeadingNumbered
  \if\mw@HeadingRunIn
    \mw@runinheading
  \else
    \if\mw@HeadingWholeWidth
      \if@twocolumn
        \if\mw@HeadingBreakAfter
        \onecolumn
        \mw@normalheading
        \pagebreak\relax
              \if@twoside
                \null
                \thispagestyle{blank}%
                \newpage
              \fi% of \if@twoside
        \twocolumn
        \else
          \@topnewpage[\mw@normalheading]%
        \fi% of \if\mw@HeadingBreakAfter
      \else
        \mw@normalheading
        \if\mw@HeadingBreakAfter\pagebreak\relax\fi
      \fi% of \if@twocolumn
    \else
      \mw@normalheading
      \if\mw@HeadingBreakAfter\pagebreak\relax\fi
    \fi% of \if\mw@HeadingWholeWidth
  \fi% of \if\mw@HeadingRunIn
  }
```

An improvement of MW's \SetSectionFormatting

A version of MW's \SetSectionFormatting that lets to leave some settings unchanged by leaving the respective argument empty ({} or []).

Notice: If we adjust this command for new version of MWCLS, we should name it \SetSectionFormatting and add issuing errors if the inner macros are undefined.

```
\relaxen\SetSectionFormatting
\newcommand*\SetSectionFormatting[5][\empty]{%
  \ifx\empty#1\relax\else% empty (not \empty!) #1 also launches \else.
    \def\mw@HeadingRunIn{10}\def\mw@HeadingBreakBefore{10}%
    \def\mw@HeadingBreakAfter{10}\def\mw@HeadingWholeWidth{10}%
```

\SetSectionFormatting
\mw@HeadingRunIn
\mw@HeadingBreakBefore
\mw@HeadingBreakAfter
\mw@HeadingWholeWidth

```
    \@ifempty{#1}{}{\mw@processflags#1,\relax}% If #1 is omitted, the flags are
          left unchanged. If #1 is given, even as [], the flags are first cleared and then
          processed again.
  \fi
  \@ifundefined{#2}{\@namedef{#2}{\mw@section{#2}}}{}%
  \mw@secdef{#2}{@preskip} {#3}{2 oblig.}%
  \mw@secdef{#2}{@head}    {#4}{3 oblig.}%
  \mw@secdef{#2}{@postskip}{#5}{4 oblig.}%
  \ifx\empty#1\relax
    \mw@secundef{#2@flags}{1 (optional)}%
  \else\mw@setflags{#2}%
  \fi}
```

\mw@secdef

```
\def\mw@secdef#1#2#3#4{% #1 the heading name,
        % #2 the command distinctor,
        % #3 the meaning,
        % #4 the number of argument to error message.
  \@ifempty{#3}
    {\mw@secundef{#1#2}{#4}}
    {\@namedef{#1#2}{#3}}}
```

\mw@secundef

```
\def\mw@secundef#1#2{%
  \@ifundefined{#1}{%
    \ClassError{mwcls/gm}{%
      command \bslash#1  undefined \MessageBreak
      after \bslash SetSectionFormatting!!!\MessageBreak}{%
      Provide the #2 argument of \bslash SetSectionFormatting.}}{}}
```

First argument is a sectioning command (wo. \) and second the stuff to be added at the beginning of the heading declarations.

\addtoheading

```
\def\addtoheading#1#2{%
    \n@melet{gmu@reserveda}{#1@head}%
    \toks\z@=\@xa{\gmu@reserveda}%
    \toks\tw@={#2}%
    \edef\gmu@reserveda{\the\toks\tw@\the\toks\z@}%
    \n@melet{#1@head}{gmu@reserveda}%
}
```

Negative `\addvspace`

When two sectioning commands appear one after another (we may assume that this occurs only when a lower section appears immediately after higher), we prefer to put the *smaller* vertical space not the larger, that is, the preskip of the lower sectioning not the postskip of the higher.

For that purpose we modify the very inner macros of MWCLS to introduce a check whether the previous vertical space equals the postskip of the section one level higher.

```
\@ifnotmw{}{% We proceed only in MWCLS
```

The information that we are just after a heading will be stored in the `\gmu@prevsec` macro: any heading will define it as the section name and `\everypar` (any normal text) will clear it.

\@afterheading

```
  \def\@afterheading{%
    \@nobreaktrue
    \xdef\gmu@prevsec{\mw@HeadingType}% added now
```

```
    \everypar{%
      \grelaxen\gmu@prevsec% added now. All the rest is original LaTeX.
      \if@nobreak
        \@nobreakfalse
        \clubpenalty \@M
        \if@afterindent \else
          {\setbox\z@\lastbox}%
        \fi
      \else
        \clubpenalty \@clubpenalty
        \everypar{}%
      \fi}}
```

If we are (with the current heading) just after another heading (one level lower I suppose), then we add the less of the higher header's post-skip and the lower header pre-skip or, if defined, the two-header-skip. (We put the macro defined below just before `\addvspace` in MWCLS inner macros.

\gmu@checkaftersec

```
  \def\gmu@checkaftersec{%
    \@ifundefined{gmu@prevsec}{}{%
      \ifgmu@postsec% an additional switch that is true by default but may be
            turned into an \ifdim in special cases, see line 507.
        {\@xa\mw@getflags\@xa{\gmu@prevsec}%
          \glet\gmu@reserveda\mw@HeadingBreakAfter}%
```

\gmu@reserveda

```
        \if\mw@HeadingBreakBefore\def\gmu@reserveda{11}\fi% if the current
              heading inserts page break before itself, all the play with vskips is
              irrelevant.
        \if\gmu@reserveda\else
          \penalty10000\relax
          \skip\z@=\csname\gmu@prevsec @postskip\endcsname\relax
          \skip\tw@=\csname\mw@HeadingType @preskip\endcsname\relax
          \@ifundefined{\mw@HeadingType @twoheadskip}{
            \ifdim\skip\z@>\skip\tw@
              \vskip-\skip\z@% we strip off the post-skip of previous header if
                    it's bigger than current pre-skip
            \else
              \vskip-\skip\tw@% we strip off the current pre-skip otherwise
            \fi}{% But if the two-header-skip is defined, we put it
            \penalty10000
            \vskip-\skip\z@
            \penalty10000
            \vskip-\skip\tw@
            \penalty10000
            \vskip\csname\mw@HeadingType @twoheadskip\endcsname
            \relax}%
          \penalty10000
          \hrule height\z@\relax% to hide the last (un)skip before subsequent
                \addvspaces.
          \penalty10000
        \fi
      \fi
    }% of \@ifundefined{gmu@prevsec} 'else'
  }% of \def\gmu@checkaftersec
```

\ParanoidPostsec

```
  \def\ParanoidPostsec{% this version of \ifgmu@postsec is intended for the spe-
```

cial case of sections may contain no normal text, as while gmdocing.

\ifgmu@postsec
```
    \def\ifgmu@postsec{% note this macro expands to an open \if.
      \skip\z@=\csname\gmu@prevsec @postskip\endcsname\relax
      \ifdim\lastskip=\skip\z@\relax% we play with the vskips only if the last
            skip is the previous heading's postskip (a counter-example I met while
            gmdocing).
    }}
  \let\ifgmu@postsec\iftrue
```

\gmu@getaddvs
```
  \def\gmu@getaddvs#1\addvspace#2\gmu@getaddvs{%
    \toks\z@={#1}
    \toks\tw@={#2}}
```

And the modification of the inner macros at last:

\gmu@setheading
```
  \def\gmu@setheading#1{%
    \@xa\gmu@getaddvs#1\gmu@getaddvs
    \edef#1{%
      \the\toks\z@\@nx\gmu@checkaftersec
      \@nx\addvspace\the\toks\tw@}}
  \gmu@setheading\mw@normalheading
  \gmu@setheading\mw@runinheading
```

\SetTwoheadSkip
```
  \def\SetTwoheadSkip#1#2{\@namedef{#1@twoheadskip}{#2}}
}% of \@ifnotmw
```

My heading setup for mwcls

The setup of heading skips was tested in 'real' typesetting, for money that is. The skips are designed for 11/13 pt leading and together with my version of mw11.clo option file for mwcls make the headings (except paragraph and subparagraph) consist of an integer number of lines. The name of the declaration comes from my employer, "Wiedza Powszechna" Editions.

```
\@ifnotmw{}{% We define this declaration only when in mwcls.
```

\WPheadings
```
\def\WPheadings{%
  \SetSectionFormatting[breakbefore,wholewidth]
      {part}{\z@\@plus1fill}{}{\z@\@plus3fill}%
  \@ifundefined{chapter}{}{%
    \SetSectionFormatting[breakbefore,wholewidth]
      {chapter}
      {66\p@}% {67\p@} for Adventor/Schola 0,95.
      {\FormatHangHeading{\LARGE}}
      {27\p@\@plus0,2\p@\@minus1\p@}%
  }%
  \SetTwoheadSkip{section}{27\p@\@plus0,5\p@}%
  \SetSectionFormatting{section}
      {24\p@\@plus0,5\p@\@minus5\p@}%
      {\FormatHangHeading {\Large}}
      {10\p@\@plus0,5\p@}% ed. Krajewska of "Wiedza Powszechna", as we understand her, wants the skip between a heading and text to be rigid.
  \SetTwoheadSkip{subsection}{11\p@\@plus0,5\p@\@minus1\p@}%
  \SetSectionFormatting{subsection}
      {19\p@\@plus0,4\p@\@minus6\p@}
      {\FormatHangHeading {\large}}% 12/14 pt
```

```
      {6\p@\@plus0,3\p@}% after-skip 6 pt due to p.12, not to squeeze the before-skip too much.
  \SetTwoheadSkip{subsubsection}{10\p@\@plus1,75\p@\@minus1\p@}%
  \SetSectionFormatting{subsubsection}
      {10\p@\@plus0,2\p@\@minus1\p@}
      {\FormatHangHeading {\normalsize}}
      {3\p@\@plus0,1\p@}% those little skips should be smaller than you calculate out of a geometric progression, because the interline skip enlarges them.
  \SetSectionFormatting[runin]{paragraph}
      {7\p@\@plus0,15\p@\@minus1\p@}
      {\FormatRunInHeading{\normalsize}}
      {2\p@}%
  \SetSectionFormatting[runin]{subparagraph}
      {4\p@\@plus1\p@\@minus0,5\p@}
      {\FormatRunInHeading{\normalsize}}
      {\z@}%
}% of \WPheadings
}% of \@ifnotmw
```

Compatibilising Standard and mwcls Sectionings

If you use Marcin Woliński's document classes (mwcls), you might have met their little queerness: the sectioning commands take two optional arguments instead of standard one. It's reasonable since one may wish one text to be put into the running head, another to the toc and yet else to the page. But the order of optionalities causes an incompatibility with the standard classes: MW section's first optional argument goes to the running head not to toc and if you've got a source file written with the standard classes in mind and use the first (and only) optional argument, the effect with mwcls would be different if not error.

Therefore I counter-assign the commands and arguments to reverse the order of optional arguments for sectioning commands when mwcls are in use and reverse, to make mwcls-like sectioning optionals usable in the standard classes.

With the following in force, you may both in the standard classes and in mwcls give a sectioning command one or two optional arguments (and mandatory the last, of course). If you give just one optional, it goes to the running head and to toc as in scls (which is unlike in mwcls). If you give two optionals, the first goes to the running head and the other to toc (like in mwcls and unlike in scls).

(In both cases the mandatory last argument goes only to the page.)

What more is unlike in scls, it's that even with them the starred versions of sectioning commands allow optionals (but they still send them to the Gobbled Tokens' Paradise).

(In mwcls, the only difference between starred and non-starred sec commands is (not) numbering the titles, both versions make a contents line and a mark and that's not changed with my redefinitions.)

```
\@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those written with two optionals.
\gm@secini  \def\gm@secini{gm@la}%
 \gm@secxx  \def\gm@secxx#1#2[#3]#4{%
              \ifx\gm@secstar\@empty
                \n@melet{gm@true@#1mark}{#1mark}% a little trick to allow a special version of the heading just to the running head.
```

```
566         \@namedef{#1mark}##1{% we redefine \⟨sec⟩mark to gobble its argument and
                to launch the stored true marking command on the appropriate argument.
567           \csname gm@true@#1mark\endcsname{#2}%
568           \n@melet{#1mark}{gm@true@#1mark}% after we've done what we wanted
                we restore original \#1mark.
569         }%
\gm@secstar 570         \def\gm@secstar{[#3]}% if \gm@secstar is empty, which means the sectioning
                command was written starless, we pass the 'true' sectioning command #3 as the optional argument. Otherwise the sectioning command was written with star so the 'true' s.c. takes no optional.
571       \fi
572       \@xa\@xa\csname\gm@secini#1\endcsname
573       \gm@secstar{#4}}%
574 }{% we are in mwcls and want to reverse MW's optionals order i.e., if there's just one
        optional, it should go both to toc and to running head.
\gm@secini 575   \def\gm@secini{gm@mw}%
576   \let\gm@secmarkh\@gobble% in mwcls there's no need to make tricks for special
        version to running headings.
\gm@secxx 577   \def\gm@secxx#1#2[#3]#4{%
578     \@xa\@xa\csname\gm@secini#1\endcsname
579     \gm@secstar[#2][#3]{#4}}%
580 }
\gm@sec  581 \def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
\gm@secx 582 \def\gm@secx#1[#2]{%
583   \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}% if there's only
        one optional, we double it not the mandatory argument.
\gm@straightensec 584 \def\gm@straightensec#1{% the parameter is for the command's name.
585   \@ifundefined{#1}{}{% we don't change the ontological status of the command
        because someone may test it.
586     \n@melet{\gm@secini#1}{#1}%
587     \@namedef{#1}{%
\gm@secstar 588       \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
\gm@secstar 589         \def\gm@secstar{}\gm@sec{#1}}}%
590 }%
591 \let\do\gm@straightensec
592 \do{part}\do{chapter}\do{section}\do{subsection}\do{subsubsection}
593 \@ifnotmw{}{\do{paragraph}}% this 'straightening' of \paragraph with the standard article caused the 'TEX capacity exceeded' error. Anyway, who on Earth wants paragraph titles in toc or running head?
```

`enumerate*` and `itemize*`

We wish the starred version of `enumerate` to be just numbered paragraphs. But hyperref redefines `\item` so we should do it a smart way, to set the LaTeX's `list` parameters that is.

(Marcin Woliński in mwcls defines those environments slightly different: his item labels are indented, mine are not; his subsequent paragraphs of an item are not indented, mine are.)

```
enumerate* 594 \@namedef{enumerate*}{%
```

```
  \ifnum\@enumdepth>\thr@@
    \@toodeep
  \else
    \advance\@enumdepth\@ne
    \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
    \@xa\list\csname label\@enumctr\endcsname{%
      \partopsep\topsep \topsep\z@ \leftmargin\z@
      \itemindent\@parindent %%\advance\itemindent\labelsep
      \labelwidth\@parindent
      \advance\labelwidth-\labelsep
      \listparindent\@parindent
      \usecounter \@enumctr
      \def\makelabel##1{##1\hfil}}%
  \fi}
\@namedef{endenumerate*}{\endlist}
```

itemize*

```
\@namedef{itemize*}{%
  \ifnum\@itemdepth>\thr@@
    \@toodeep
  \else
    \advance\@itemdepth\@ne
    \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
    \@xa\list\csname\@itemitem\endcsname{%
      \partopsep\topsep \topsep\z@ \leftmargin\z@
      \itemindent\@parindent
      \labelwidth\@parindent
      \advance\labelwidth-\labelsep
      \listparindent\@parindent
      \def\makelabel##1{##1\hfil }}%
  \fi}
\@namedef{enditemize*}{\endlist}
```

The Logos

We'll modify The LaTeX logo now to make it fit better to various fonts.

```
\let\oldLaTeX\LaTeX
\let\oldLaTeXe\LaTeXe
\def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
```

\DeclareLogo

```
\newcommand*\DeclareLogo[3][\relax]{%
```

#1 is for non-LaTeX spelling and will be used in the PD1 encoding (to make pdf bookmarks);
#2 is the command, its name will be the PD1 spelling by default,
#3 is the definition for all the font encodings except PD1.

\gmu@reserveda

```
  \ifx\relax#1\def\gmu@reserveda{\@xa\@gobble\string#2}%
  \else
```

\gmu@reserveda

```
    \def\gmu@reserveda{#1}%
  \fi
  \edef\gmu@reserveda{%
```

\@nx

```
    \@nx\DeclareTextCommand\@nx#2{PD1}{\gmu@reserveda}}
  \gmu@reserveda
  \DeclareTextCommandDefault#2{#3}%
```

\DeclareRobustCommand*

```
  \DeclareRobustCommand*#2{#3}}% added for XeTeX
```

\DeclareLogo

```
\DeclareLogo\LaTeX{%
  {%
    L%
    \setbox\z@\hbox{\check@mathfonts
      \fontsize\sf@size\z@
      \math@fontsfalse\selectfont
      A}%
    \kern-.57\wd\z@
    \sbox\tw@ T%
    \vbox to\ht\tw@{\copy\z@ \vss}%
    \kern-.2\wd\z@}% originally -, 15 em for T.
  {%
    \ifdim\fontdimen1\font=\z@
    \else
      \count\z@=\fontdimen5\font
      \multiply\count\z@ by 64\relax
      \divide\count\z@ by\p@
      \count\tw@=\fontdimen1\font
      \multiply\count\tw@ by\count\z@
      \divide\count\tw@ by 64\relax
      \divide\count\tw@ by\tw@
      \kern-\the\count\tw@ sp\relax
    \fi}%
  \TeX}
```

\LaTeXe

```
\DeclareLogo\LaTeXe{\mbox{\m@th \if
    b\expandafter\@car\f@series\@nil\boldmath\fi
    \LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}
```

```
\StoreMacro\LaTeX
\StoreMacro*{LaTeX }
```

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

\LaTeXpar

```
\DeclareLogo[(La)TeX]{\LaTeXpar}{%
  {%
    \setbox\z@\hbox{(}% )
    \copy\z@
    \kern-.2\wd\z@ L%
    \setbox\z@\hbox{\check@mathfonts
      \fontsize\sf@size\z@
      \math@fontsfalse\selectfont
      A}%
    \kern-.57\wd\z@
    \sbox\tw@ T%
    \vbox to\ht\tw@{\box\z@%
      \vss}%
  }%
  \kern-.07em% originally -, 15 em for T.
  {% (
    \sbox\z@)%
    \kern-.2\wd\z@\copy\z@
    \kern-.2\wd\z@}\TeX
}
```

"Here are a few definitions which can usefully be employed when documenting package files: now we can readily refer to $\mathcal{AMS}$-TEX, BIBTEX and SLITEX, as well as the usual TEX and LATEX. There's even a PLAIN TEX and a WEB."

```
\@ifundefined{AmSTeX}
  {\def\AmSTeX{\leavevmode\hbox{$\mathcal A\kern-.2em\lower.376ex%
        \hbox{$\mathcal M$}\kern-.2em\mathcal S$-\TeX}}}{}
\DeclareLogo\BibTeX{{\rmfamily B\kern-.05em%
    \textsc{i{\kern-.025em}b}\kern-.08em% the kern is wrapped in braces for my
          \fakescaps' sake.
    \TeX}}
\DeclareLogo\SliTeX{{\rmfamily S\kern-.06emL\kern-.18em\raise.32ex\hbox
        {\scshape i}\kern -.03em\TeX}}
\DeclareLogo\PlainTeX{\textsc{Plain}\kern2pt\TeX}
\DeclareLogo\Web{\textsc{Web}}
```

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

```
\DeclareLogo[The TeX book]\TeXbook{\textsl{The \TeX book}}
\let\TB\TeXbook% TUG Boat uses this.
\DeclareLogo[e-TeX]\eTeX{%
  \ensuremath{\varepsilon}-\kern-.125em\TeX}% definition sent by Karl Berry
        from TUG Boat itself.
\DeclareLogo[pdfe-TeX]\pdfeTeX{pdf\eTeX}
\DeclareLogo\pdfTeX{pdf\TeX}
\@ifundefined{XeTeX}{%
  \DeclareLogo\XeTeX{X\kern-.125em\relax
    \@ifundefined{reflectbox}{%
      \lower.5ex\hbox{E}\kern-.1667em\relax}{%
      \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
    \TeX}}{}
\@ifundefined{XeLaTeX}{%
  \DeclareLogo\XeLaTeX{X\kern-.125em\relax
    \@ifundefined{reflectbox}{%
      \lower.5ex\hbox{E}\kern-.1667em\relax}{%
      \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
    \LaTeX}}
```

As you see, if TEX doesn't recognize `\reflectbox` (graphics isn't loaded), the first `E` will not be reversed. This version of the command is intended for non-XETEX usage. With XETEX, you can load the xltxtra package (e.g. with the gmutils `\XeTeXthree` declaration) and then the reversed `E` you get as the Unicode Latin Letter Reversed E.

Expanding turning stuff all into `other'

While typesetting a unicode file contents with inputenc package I got a trouble with some Unicode sequences that expanded to unexpandable CSs: they could'nt be used within `\csname...\endcsname`. My TEXGuru advised to use `\meanig` to make all the name 'other'. So—here we are.

Don't use them in `\edefs`, they would expand not quite.

The next macro turns its #2 all into 'other' chars and assigns them to #1 which has to be a CS or an active char.

\def@other
\gm@def@other@tempa
```
\long\def\def@other#1#2{%
  \long\def\gm@def@other@tempa{#2}%
  \all@other#1{#2}}
```

The next macro is intended to be put in \edefs with a macro argument. The meaning of the macro will be made all 'other' and the words '(long) macro:->' gobbled.

\all@other
```
\def\all@other#1{\@xa\gm@gobmacro\meaning#1}
```

The \gm@gobmacro macro above is applied to gobble the \meaning's beginnig, `long macro:->` all 'other' that is.

\@nx
\gm@gobmacro
```
\edef\gmu@reserveda{%
  \def\@nx\gm@gobmacro##1\@xa\@gobble\string\macro:->{}}
\gmu@reserveda
```

In the next two macros' names, 'unex' stands both for not expanding the argument(s) and for disastrously partial unexpandability of the macros themselves.

\unex@namedef
```
\long\def\unex@namedef#1#2{%
  \edef@other\gmu@reserveda{#1}%
  \@xa\long\@xa\def\csname\gmu@reserveda\endcsname{#2}}
```

\unex@nameuse
```
\long\def\unex@nameuse#1{%
  \edef@other\gmu@reserveda{#1}%
  \csname\gmu@reserveda\endcsname}
```

Brave New World of XƎTEX

\@ifXeTeX
```
\newcommand\@ifXeTeX[2]{%
  \@xa\ifx\csname XeTeXversion\endcsname\relax
  \afterfi{#2}\else\afterfi{#1}\fi}
```

\XeTeXthree
```
\def\XeTeXthree{%
  \@ifXeTeX{%
    \RequirePackage{fontspec}%
    \RequirePackage{xunicode}%
    \RequirePackage{xltxtra}%
    \AtBeginDocument{%
      \RestoreMacro\LaTeX\RestoreMacro*{LaTeX }}% my version of the LaTeX logo has been stored just after defining, in line 666.
}{}}
```

The \udigits declaration causes the digits to be typeset uppercase. I provide it since by default I prefer the lowercase (nautical) digits.

```
\AtBeginDocument{%
  \@ifpackageloaded{fontspec}{%
```
\udigits
```
    \DeclareRobustCommand*\udigits{%
      \addfontfeature{Numbers=Uppercase}}%
  }{%
    \emptify\udigits}}
```

Fractions

\Xedekfracc
```
\def\Xedekfracc{\@ifstar{%
```

Minion GM doesn't feature the `frac` font feature properly so, with the starred version of the declaration we use the characters from the font where available (see the `\@namedefs` below) and the `numr` and `dnom` features with the fractional slash otherwise (via `\gmu@dekfracc`).

\gmu@dekfracc
```
    \def\gmu@dekfracc########1/########2{%
      {\addfontfeature{VerticalPosition=Numerator}########1}%
           \kern-0.05em
      \char"2044\kern-.05em
      {\addfontfeature{VerticalPosition=Denominator}########2}}%
```

We define the fractional macros. Since Adobe Minion Pro doesn't contain $\frac{n}{5}$ nor $\frac{n}{6}$, we don't provide them here.

```
    \@namedef{gmu@xefracc1/4}{\char"BC\relax}%
    \@namedef{gmu@xefracc1/2}{\char"BD\relax}%
    \@namedef{gmu@xefracc3/4}{\char"BE\relax}%
    \@namedef{gmu@xefracc1/3}{\char"2153\relax}%
    \@namedef{gmu@xefracc2/3}{\char"2154\relax}%
    \@namedef{gmu@xefracc1/8}{\char"215B\relax}%
    \@namedef{gmu@xefracc3/8}{\char"215C\relax}%
    \@namedef{gmu@xefracc5/8}{\char"215D\relax}%
    \@namedef{gmu@xefracc7/8}{\char"215E\relax}%
```

\dekfracc
```
    \def\dekfracc########1/########2{%
      \@ifundefined{gmu@xefracc########1/########2}{%
        \gmu@dekfracc{########1}/{########2}}{%
        \csname gmu@xefracc########1/########2\endcsname}}%
  }{%
```

\dekfracc
```
    \def\dekfracc####1/####2{{%
        \addfontfeature{Fractions=On}%
        ####1/####2}}%
    \let\fnfracc\dekfracc
    }}
```

What have we just done? We defined two versions of the `\Xefractions` declaration. The starred version is intended to make use only of the built-in fractions such as $1/2$ or $7/8$. To achieve that, a handful of macros is defined that expand to the Unicodes of built-in fractions and `\dekfracc` command is defined to use them.

The unstarred version makes use of the Fraction font feature and therefore is much simpler.

Note that in the first argument of `\@ifstar` we wrote 8 (eight) `#`s to get the correct definition and in the second argument 'only' 4. (The LaTeX 2_ε Source claims that that is changed in the 'new implementation' of `\@ifstar` so maybe it's subject to change.)

`\resizegraphics`

```
\@ifXeTeX{%
```

\resizegraphics
```
  \def\resizegraphics#1#2#3{{%
      \setbox0=\hbox{\XeTeXpicfile #3 }%
      \ifx!#1\else
        \dimen0=#1\relax
        \count2=\wd0
        \divide\count2 by1000\relax
```

```
        \count0=\dimen0\relax
        \divide\count0\count2
      \fi
      \ifx!#2\else
        \dimen0=#1\relax
        \count6=\ht0
        \divide\count6 by1000\relax
        \count4=\dimen0\relax
        \divide\count4\count6
      \fi
      \ifx!#1\count0=\count4\fi
      \ifx!#2\count4=\count0\fi
      \XeTeXpicfile #3 xscaled \count0 yscaled \count4
    }}}{%
\resizegraphics  \def\resizegraphics#1#2#3{%
      \resizebox{#1}{#2}{%
        \includegraphics{#3}}}}%
```

The [options] in the `\XeTeXpicfile` command use the following keywords:
`width` ⟨*dimen*⟩
`height` ⟨*dimen*⟩
`scaled` ⟨*scalefactor*⟩
`xscaled` ⟨*scalefactor*⟩
`yscaled` ⟨*scalefactor*⟩
`rotated` ⟨*degrees*⟩

```
\GMtextsuperscript \def\GMtextsuperscript{%
  \@ifXeTeX{%
\textsuperscript    \def\textsuperscript##1{{%
        \addfontfeature{VerticalPosition=Numerator}##1}}%
  }{\truetextsuperscript}}
\truetextsuperscript \def\truetextsuperscript{%
\textsuperscript  \DeclareRobustCommand*\textsuperscript[1]{%
    \@textsuperscript{\selectfont##1}}
\@textsuperscript  \def\@textsuperscript##1{%
    {\m@th\ensuremath{^{\mbox{\fontsize\sf@size\z@##1}}}}}}
```

Varia

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

```
\gmu@tilde \def\gmu@tilde{%
  \leavevmode\lower.8ex\hbox{$\,\widetilde{\mbox{ }}\,$}}
```

Originally there was just `\ ` instead of `\mbox{ }` but some commands of ours do redefine `\ `.

```
\*  \DeclareRobustCommand*\*{\gmu@tilde}
\AtBeginDocument{% to bypass redefinition of \~ as a text command with various
      encodings
\~  \DeclareRobustCommand*\~{%
    \@ifnextchar/{\gmu@tilde\kern-0,1667em\relax}\gmu@tilde}}
```

We prepare the proper kerning for "~/".

The standard \obeyspaces declaration just changes the space's \catcode to 13 ('active'). Usually it is fairly enough because no one 'normal' redefines the active space. But we are *not* normal and we do *not* do usual things and therefore we want a declaration that not only will \activeate the space but also will (re)define it as the \ primitive. So define \gmobeyspaces that obeys this requirement.

(This definition is repeated in gmverb.)

```
\foone{\catcode`\ \active}%
\gmobeyspaces
{\def\gmobeyspaces{\catcode`\ \active\let \ }}
```

While typesetting poetry, I was surprised that sth. didn't work. The reason was that original \obeylines does \let not \def, so I give the latter possibility.

```
\foone{\catcode`\^^M\active}% the comment signs here are crucial.
\defobeylines
{\def\defobeylines{\catcode`\^^M=13 \def^^M{\par}}}
```

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

```
\deksmallskip \def\deksmallskip{\vskip\smallskipamount}
\undeksmallskip \def\undeksmallskip{\vskip-\smallskipamount}
\dekmedskip \def\dekmedskip{\vskip\medskipamount}
\dekbigskip \def\dekbigskip{\vskip\bigskipamount}
\hfillneg \def\hfillneg{\hskip 0pt plus -1fill\relax}
```

In some \if(cat?) test I needed to look only at the first token of a tokens' string (first letter of a word usually) and to drop the rest of it. So I define a macro that expands to the first token (or {⟨*text*⟩}) of its argument.

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

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

```
\TODO \newcommand*{\TODO}[1][]{{%
    \sffamily\bfseries\huge TODO!\if\relax#1\relax\else\space\fi#1}}
```

I like twocolumn tables of contents. First I tried to provide them by writing \begin{%multicols}{2} and \end{multicols} outto the .toc file but it worked wrong in some cases. So I redefine the internal LaTeX macro instead.

```
\twocoltoc \newcommand*\twocoltoc{%
  \RequirePackage{multicol}%
\@starttoc  \def\@starttoc##1{%
    \begin{multicols}{2}\makeatletter\@input {\jobname .##1}%
      \if@filesw \@xa \newwrite \csname tf@##1\endcsname
        \immediate \openout \csname tf@##1\endcsname \jobname .##1%
             \relax
      \fi
      \@nobreakfalse\end{multicols}}}
\@onlypreamble\twocoltoc
```

The macro given below is taken from the multicol package (where its name is \enough@room). I put it in this package since I needed it in two totally different works.

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

Two shorthands for debugging:

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

An equality sign properly spaced:

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

And for the LaTeX's pseudo-code statements:

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

The job name without extension.

```
\def\gm@jobn#1.#2\@@nil{#1}
\def\jobnamewoe{\@xa\gm@jobn\jobname.\@@nil}% We add the dot to be sure there is one although I'm not sure whether you can TeX a file that has no extrension.
```

While typesetting a UTF-8 ls-R result I found a difficulty that follows: UTF-8 encoding is handled by the inputenc package. It's O.K. so far. The UTF-8 sequences are managed using active chars. That's O.K. so far. While writing such sequences to a file, the active chars expand. You feel the blues? When the result of expansion is read again, it sometimes is again an active char, but now it doesn't star a correct UTF-8 sequence.

Because of that I wanted to 'freeze' the active chars so that they would be `\writen` to a file unexpanded. A very brutal operation is done: we look at all 256 chars' catcodes and if we find an active one, we `\let` it `\relax`. As the macro does lots and lots of assignments, it shouldn't be used in `\edefs`.

```
\def\freeze@actives{%
  \count\z@\z@
  \@whilenum\count\z@<\@cclvi\do{%
    \ifnum\catcode\count\z@=\active
      \uccode`\~=\count\z@
      \uppercase{\let~\relax}%
    \fi
    \advance\count\z@\@ne}}
```

A macro that typesets all 256 chars of given font. It makes use of `\@whilenum`.

```
\newcommand*\ShowFont[1][6]{%
  \begin{multicols}{#1}[The current font (the \f@encoding\ encoding):]
    \parindent\z@
    \count\z@\m@ne
    \@whilenum\count\z@<\@cclv\do{
      \advance\count\z@\@ne
      \ \the\count\z@:~\char\count\z@\par}
  \end{multicols}}
```

A couple of macros for typesetting liturgic texts such as psalmody of Liturgia Horarum. I wrap them into a declaration since they'll be needed not every time.

```
\newcommand*\liturgiques[1][red]{% Requires the color package.
  \gmu@RPif{color}{color}%
  \newcommand*\czerwo{\small\color{#1}}% environment
  \newcommand{\czer}[1]{\leavevmode{\czerwo##1}}% we leave vmode because if we don't, then verse's \everypar would be executed in a group and thus its effect lost.
  \def\*{\czer{$*$}}
  \def\+{\czer{$\dag$}}
  \newcommand*\nieczer[1]{\textcolor{black}{##1}}}
```

After the next definition you can write `\gmu@RP[`⟨*options*⟩`]{`⟨*package*⟩`}{`⟨*csname*⟩`}` to get the package `#2` loaded with options `#1` if the csname `#3` is undefined.

```
\gmu@RPif
\gmu@resa
\newcommand*\gmu@RPif[3][]{%
  \ifx\relax#1\relax
  \else \def\gmu@resa{[#1]}%
  \fi
  \@xa\RequirePackage\gmu@resa{#2}}
```

Since inside `document` we cannot load a package, we'll redefine `\gmu@RPif` to issue a request before the error issued by undefined CS.

```
\AtBeginDocument{%
\gmu@RPif
  \renewcommand*\gmu@RPif[3][]{%
    \@ifundefined{#3}{%
      \@ifpackageloaded{#2}{}{%
        \typeout{^^J! Package `#2' not loaded!!! (\on@line)^^J}}}{}}}
```

It's very strange to me but it seems that `c` is not defined in the basic math packages. It is missing at least in the *Symbols* book.

```
\continuum
\providecommand*\continuum{\gmu@RPif{eufrak}{mathfrak}\mathfrak{c}}
```

And this macro I saw in the ltugproc document class nad I liked it.

```
\acro
\def\acro#1{\gmu@acroinner#1\gmu@acroinner}
\gmu@acroinner
\def\gmu@acroinner#1{%
  \ifx\gmu@acroinner#1\else
    \ifcat a\@nx#1%
      \ifnum`#1=\uccode`#1%
        {\scshape\lowercase{#1}}%
      \else{\smallerr#1}%
      \fi
    \else#1%
    \fi
    \afterfi\gmu@acroinner
  \fi}
```

Since the fonts I am currently using do not support required font feature, I skip the following definition.

```
\iffalse
\acroff
\acro
\newcommand*\acroff{%
  \def\acro##1{{\addfontfeature{Letters=UppercaseSmallCaps}##1}}}
\fi
\IMO
\AKA
\newcommand*\IMO{\acro{IMO}}
\newcommand*\AKA{\acro{AKA}}
\qxenc
\newcommand*\qxenc{\fontencoding{QX}\selectfont}
```

The `\copyright` command is unavailable in T1 and U (unknown) encodings so provide

```
\qxcopyright
\qxcopyrights
\newcommand*\qxcopyright{{\qxenc\copyright}}
\newcommand*\qxcopyrights{%
  \let\gmu@copyright\copyright
  \def\copyright{{\qxenc\gmu@copyright}}}
\fixcopyright
\newcommand*\fixcopyright{%
  \@ifXeTeX{\def\copyright{\char"00A9 }}{\qxcopyrights}}
```

Probably the only use of it is loading gmdocc.cls 'as second class'. This command takes first argument optional, options of the class, and second mandatory, the class name. I use it in an article about gmdoc.

\secondclass
\ifSecondClass
```
\def\secondclass{%
  \newif\ifSecondClass
  \SecondClasstrue
  \@fileswithoptions\@clsextension}% [outeroff,gmeometric]{gmdocc} it's loading gmdocc.cls with all the bells and whistles except the error message.
```

Cf. *The TEXbook* exc. 11.6.
A line from LaTeX:

```
% \check@mathfonts\fontsize\sf@size\z@\math@fontsfalse\selectfont
```

didn't work as I would wish: in a `\footnotesize`'s scope it still was `\scriptsize`, so too large.

\dekfracc
```
\def\dekfracc#1/#2{\leavevmode\kern.1em
  \raise.5ex\hbox{\udigits\scriptsize#1}\kern-.1em
  /\kern-.15em\lower.25ex\hbox{\udigits\scriptsize#2}}
```

\fnfracc
```
\def\fnfracc#1/#2{\leavevmode\kern.1em
  \raise.6ex\hbox{\udigits\tiny#1}\kern-.1em
  /\kern-.1em\lower.25ex\hbox{\udigits\tiny#2}}
```

A macro that acts like `\,` (thin and unbreakable space) except it allows hyphenation afterwards:

\ikern
```
\newcommand*\ikern{\,\penalty10000\hskip0sp\relax}
```

And a macro to forbid hyphenation of the next word:

\nohy
```
\newcommand*\nohy{\kern\z@}
```

`\@ifempty`

\@ifempty
\gmu@reserveda
```
\long\def\@ifempty#1#2#3{%
  \def\gmu@reserveda{#1}%
  \ifx\gmu@reserveda\@empty\afterfi{#2}%
  \else\afterfi{#3}\fi
}
```

`\include` not only .tex's

`\include` modified by me below lets you to include files of any extension provided that extension in the argument.

If you want to `\include` a non-.tex file and deal with it with `\includeonly`, give the latter command full file name, with the extension that is.

\gmu@getext
\gmu@filename
\gmu@fileext
```
\def\gmu@getext#1.#2\@@nil{%
  \def\gmu@filename{#1}%
  \def\gmu@fileext{#2}}
```

```
\def\include#1{\relax
  \ifnum\@auxout=\@partaux
  \@latex@error{\string\include\space cannot be nested}\@eha
  \else \@include#1 \fi}
```

\@include
\gmu@fileext
```
\def\@include#1 {%
  \gmu@getext#1.\@@nil
  \ifx\gmu@fileext\empty\def\gmu@fileext{tex}\fi
```

```
  \clearpage
  \if@filesw
    \immediate\write\@mainaux{\string\@input{\gmu@filename.aux}}%
  \fi
  \@tempswatrue
  \if@partsw
    \@tempswafalse
    \edef\reserved@b{#1}%
    \@for\reserved@a:=\@partlist\do{%
      \ifx\reserved@a\reserved@b\@tempswatrue\fi}%
  \fi
  \if@tempswa
    \let\@auxout\@partaux
    \if@filesw
      \immediate\openout\@partaux \gmu@filename.aux
      \immediate\write\@partaux{\relax}%
    \fi
    \@input@{\gmu@filename.\gmu@fileext}%
    \inclasthook
    \clearpage
    \@writeckpt{\gmu@filename}%
    \if@filesw
      \immediate\closeout\@partaux
    \fi
  \else
```

If the file is not included, reset `\@include` `\deadcycles`, so that a long list of non-included files does not generate an 'Output loop' error.

```
    \deadcycles\z@
    \@nameuse{cp@\gmu@filename}%
  \fi
  \let\@auxout\@mainaux}
```

`\whenonly` `\gmu@whonly`

```
\newcommand\whenonly[3]{%
  \def\gmu@whonly{#1,}%
  \ifx\gmu@whonly\@partlist\afterfi{#2}\else\afterfi{#3}\fi}
```

I assume one usually includes chapters or so so the last page style should be closing.

`\inclasthook`

```
\def\inclasthook{\thispagestyle{closing}}
```

Faked small caps

`\gmu@scapLetters`

```
\def\gmu@scapLetters#1{%
  \ifx#1\relax\relax\else% two \relaxes to cover the case of empty #1.
    \ifcat a#1\relax
      \ifnum\the\lccode`#1=`#1\relax
        {\gmu@scsetup\MakeUppercase{#1}}% not Plain \uppercase because that
              works bad with inputenc.
      \else#1%
      \fi
    \else#1%
    \fi%
    \@xa\gmu@scapLetters
```

```
  \fi}%
\def\gmu@scapSpaces#1 #2\@@nil{%
  \ifx#1\relax\relax
  \else\gmu@scapLetters#1\relax
  \fi
  \ifx#2\relax\relax
  \else\afterfi{\ \gmu@scapSpaces#2\@@nil}%
  \fi}
\def\gmu@scapss#1\@@nil{{\def~{{\nobreakspace}}%
    \gmu@scapSpaces#1 \@@nil}}% % \def\\{{\newline}}\relax adding redefinition of \\ caused stack overflow Note it disallows hyphenation except at \-.
\DeclareRobustCommand\fakescaps[2][\gmu@scalematchX]{{%
    \let\gmu@scsetup=#1\gmu@scapss#2\@@nil}}
```

Experimente z akcentami patrz no3.tex.

```
\def\tinycae{{\tiny\AE}}% to use in \fakescaps[\tiny]{...}
\RequirePackage{calc}
```

wg \zf@calc@scale pakietu fontspec.

```
\@ifXeTeX{%
  \def\gmu@scalematchX{%
    \begingroup
      \ifx\zf@scale\empty\def\gmu@scalar{1.0}%
      \else\let\gmu@scalar\zf@scale\fi
      \setlength\@tempdima{\fontdimen5\font}% 5—ex height
      \setlength\@tempdimb{\fontdimen8\font}% 8—XeTeX synthesized uppercase height.
      \divide\@tempdimb by1000\relax
      \divide\@tempdima by\@tempdimb
      \setlength{\@tempdima}{\@tempdima*\real{\gmu@scalar}}%
      \@ifundefined{fakesc@extrascale}{}{%
        \setlength{\@tempdima}{\@tempdima*\real{\fakesc@extrascale}}}%
      \@tempcnta=\@tempdima
      \divide\@tempcnta by 1000\relax
      \@tempcntb=-1000\relax
      \multiply\@tempcntb by\@tempcnta
      \advance\@tempcntb by\@tempdima
      \xdef\gmu@scscale{\the\@tempcnta.%
        \ifnum\@tempcntb<100 0\fi
        \ifnum\@tempcntb<10 0\fi
        \the\@tempcntb}%
    \endgroup
    \addfontfeature{Scale=\gmu@scscale}%
  }}{\let\gmu@scalematchX\smallerr}
\def\fakescextrascale#1{\def\fakesc@extrascale{#1}}
```

See above/see below

To generate a phrase as in the header depending of whether the respective label is before of after.

```
\newcommand*\wyzejnizej[1]{%
```

```
  \edef\gmu@tempa{\@ifundefined{r@#1}{\arabic{page}}{%
      \@xa\@xa\@xa\@secondoftwo\csname r@#1\endcsname}}%
  \ifnum\gmu@tempa<\arabic{page}\relax wy\.zej\fi
  \ifnum\gmu@tempa>\arabic{page}\relax ni\.zej\fi
  \ifnum\gmu@tempa=\arabic{page}\relax \@xa\ignorespaces\fi
}
```

luzniej and napapierki---environments used in page breaking for money

The name of first of them comes from Polish typesetters' phrase "rozbijać [skład] na papierki"—'to broaden [leading] with paper scratches'.

\napapierkistretch
```
\def\napapierkistretch{0,3pt}% It's quite much for 11/13pt typesetting
```

napapierki
```
\newenvironment*{napapierki}{%
  \par\global\advance\baselineskip%
  by 0ptplus\napapierkistretch\relax}{%
  \par\dimen\z@=\baselineskip
  \global\baselineskip=\dimen\z@}% so that you can use \endnapapierki in interlacing environments
```

luzniej
```
\newenvironment*{luzniej}[1][1]{%
  \multiply\tolerance by 2\relax
  \looseness=#1\relax}{\par}
```

The original \pauza of polski has the skips rigid (one is even a kern). It begins with \ifhmode to be usable also at the beginning of a line as the mark of a dialogue.

```
\AtBeginDocument{% to be independent of moment of loading of polski.
```

\pauza \pauzacore
```
  \DeclareRobustCommand*\pauza{%
    \ifhmode\unskip\penalty10000\hskip0.2em plus0.1em\relax\fi
    \pauzacore\hskip.2em plus0.1em\ignorespaces}}
```

\pauzacore
```
\def\pauzacore{---}
```

\shortpauza \pauzacore
```
\def\shortpauza{%
  \def\pauzacore{--\kern,23em\relax\llap{--}}}
```

```
\endinput
```

e. The gmiflink Package[1]

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


```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmiflink}
    [2006/08/16 v0.97 Conditionally hyperlinking package (GM)]
```

Introduction, usage

This package protects you against an error when a link is dangling and typesets some plain text instead of a hyperlink then. It is intended for use with the hyperref package. Needs *two* LaTeX runs.

I used it for typesetting the names of the objects in a documentation of a computer program. If the object had been defined a `\hyperlink` to its definition was made, otherwise a plain object's name was typeset. I also use this package in authomatic making of hyperlinking indexes.

The package provides the macros `\gmiflink`, `\gmifref` and `\gmhypertarget` for conditional making of hyperlinks in your document.

\gmhypertarget `\gmhypertarget[⟨name⟩]{⟨text⟩}` makes a `\hypertarget{⟨@name⟩}{⟨text⟩}` and a `\label{⟨@name⟩}`.

\gmiflink `\gmiflink[⟨name⟩]{⟨text⟩}` makes a `\hyperlink{⟨@name⟩}{⟨text⟩}` to a proper hypertarget if the corresponding *label* exists, otherwise it typesets ⟨*text*⟩.

\gmifref `\gmifref[⟨name⟩]{⟨text⟩}` makes a (hyper-) `\ref{⟨@name⟩}` to the given label if the label exists, otherwise it typesets ⟨*text*⟩.

The ⟨@*name*⟩ argument is just ⟨*name*⟩ if the ⟨*name*⟩ is given, otherwise it's ⟨*text*⟩ in all three macros.

For the example(s) of use, examine the gmiflink.sty file, lines 45–58.

The remarks about installation and compiling of the documentation are analogous to those in the chapter gmdoc.sty and therefore ommitted.

Contents of the gmiflink.zip archive

The distribution of the gmiflink package consists of the following four files and a TDS-compliant archive.

gmiflink.sty
README

[1] This file has version number v0.97 dated 2006/08/16.

gmiflinkDoc.tex
gmiflinkDoc.pdf
gmiflink.tds.zip

The Code

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

A new counter, just in case

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

The macro given below creates both hypertarget and hyperlabel, so that you may reference both ways: via `\hyperlink` and via `\ref`. It's pattern is the `\label` macro, see LaTeX Source2e, file x, line 32.

But we don't want to gobble spaces before and after. First argument will be a name of the hypertarget, by default the same as typeset text, i.e., argument #2.

\gmhypertarget
```
\DeclareRobustCommand*\gmhypertarget{%
  \@ifnextchar[{\gm@hypertarget}{\@dblarg{\gm@hypertarget}}}
```

\gm@hypertarget
```
\def\gm@hypertarget[#1]#2{% If argument #1 = \empty, then we'll use #2, i.e., the
      same as name of hypertarget.
  \refstepcounter{GMhlabel}% we \label{\gmht@firstpar}
  \hypertarget{#1}{#2}%
  \protected@write\@auxout{}{%
    \string\newlabel{#1}{{#2}{\thepage}{\relax}{GMhlabel.\arabic{%
         GMhlabel}}{}}}%
}% end of \gm@hypertartget.
```

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

\gmifref
```
\DeclareRobustCommand*\gmifref{\@ifnextchar[{\gm@ifref}{% ]
    \@dblarg{\gm@ifref}}}
```

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

\gmiflink
```
\DeclareRobustCommand*\gmiflink{\@ifnextchar[{\gm@iflink}{%
    \@dblarg{\gm@iflink}}}
```

\gm@iflink
```
\def\gm@iflink[#1]#2{%
  \expandafter\ifx\csname r@#1\endcsname\relax\relax%
  #2\else\hyperlink{#1}{#2}\fi%
}% end of \gm@iflink
```

It's robust because when just `\newcommand*`ed, use of `\gmiflink` in an indexing macro resulted in errors: `\@ifnextchar` has to be `\noexpanded` in `\edefs`.

```
\endinput
```

The old version — all three were this way primarily.

```
\newcommand*\gmiflink[2][\empty]{{%
```

```
  \def\gmht@test{\empty}\def\gmht@firstpar{#1}%
  \ifx\gmht@test\gmht@firstpar\def\gmht@firstpar{#2}\fi%
  \expandafter\ifx\csname r@\gmht@firstpar\endcsname\relax\relax%
  #2\else\hyperlink{\gmht@firstpar}{#2}\fi%
}}
```

f. The gmverb Package[1]

November 19, 2007

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

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

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmverb}
    [2007/11/13 v0.84 After shortvrb (FM) but my way (GM)]
```

Intro, Usage

This package redefines the `\verb` command and the `verbatim` environment so that the verbatim text can break into lines, with `%` (or another character chosen to be the comment char) as a 'hyphen'. Moreover, it allows the user to define her own `verbatim`-like environments provided their contents would be not *horribly* long (as long as a macro's argument may be at most).

This package also allows the user to declare a chosen char(s) as a 'short verb' e.g., to write `|\a\verbatim\example|` instead of `\verb|\a\verbatim\example|`.

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

`\fixbslash` (If you don't like linebreaking at backslash, there's the `\fixbslash` declaration (observing the common scoping rules, hence OCSR) and an analogous declaration for the `\fixlbrace` left brace: `\fixlbrace`.)

The default 'hyphen' is `%` since it's the default comment char. If you wish another `\VerbHyphen` char to appear at the linebreak, use the `\VerbHyphen` declaration that takes `\`⟨*char*⟩ as the only argument. This declaration is always global.

`\verbeolOK` Another difference is the `\verbeolOK` declaration (OCSR). Within its scope, `\verb` allows an end of a line in its argument and typesets it just as a space.

[1] This file has version number v0.84 dated 2007/11/13.

As in the standard version(s), the plain `\verb` typesets the spaces blank and `\verb*` makes them visible.

\MakeShortVerb Moreover, gmverb provides the `\MakeShortVerb` macro that takes a one-char control sequence as the only argument and turns the char used into a short verbatim delimiter, e.g., after `\MakeShortVerb*\|` (as you guess, the declaration has its starred version, which is for visible spaces, and the non-starred for the spaces blank) you may type `|% \mymacro|` to get `\mymacro` instead of typing `\verb+\mymacro+`. Because the char used in this example is my favourite and used just this way by DEK in the *The TEXbook*'s format,

\dekclubs gmverb provides a macro `\dekclubs` as a shorthand for `\MakeShortVerb*\|`.

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

\DeleteShortVerb e.g., `\DeleteShortVerb\|` and the previous meaning of the char used shall be restored.

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

\OldMakeShortVerb However, if you don't like such a conditional behaviour, you may use `\OldMakeShortVerb` instead, what I do when I like to display short verbatims in displaymath.

\dekclubs There's one more declaration provided by gmverb: `\dekclubs`, which is a shorthand

\dekclubs* for `\MakeShortVerb*\|` and `\dekclubs*` for `\OldMakeShortVerb*\|`.

So that, after the latter declaration, you can write

`\[|`⟨*verbatim stuff*⟩`|\]`

instead of

`\[\hbox{|`⟨*the stuff*⟩`|}\]`

to get a displayed shortverb.

Both versions of `\dekclubs` OCSR.

As many good packages, this also does not support any options.

The remarks about installation and compiling of the documentation are analogous to those in the chapter gmdoc.sty and therefore ommitted.

Contents of the gmverb.zip Archive

The distribution of the gmverb package consists of the following four files and a TDS-compliant archive.

gmverb.sty
README
gmverbDoc.tex
gmverbDoc.pdf
gmverb.tds.zip

The Code

Preliminaries

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

For `\firstofone`, `\afterfi`, `\gmobeyspaces`, `\@ifnextcat`, `\foone` and `\noexpand`'s and `\expandafter`'s shorthands `\@nx` and `\@xa` resp.

```
5 \foone{\@makeother\%}
```

\xiipercent

```
6 {\def\xiipercent{%}}
```

Someone may want to use another char for comment, but we assume here 'orthodoxy'. Other assumptions in gmdoc are made. The 'knowledge' what char is the comment char is used to put proper 'hyphen' when a `verbatim` line is broken.

\verbhyphen
```
\let\verbhyphen\xiipercent
```

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

\VerbHyphen
```
\def\VerbHyphen#1{%
  {\escapechar\m@ne
    \@xa\gdef\@xa\verbhyphen\@xa{\string#1}}}
```

As you see, it's always global.

The Breakables

Let's define a `\discretionary` left brace such that if it breaks, it turns {% at the end of line. We'll use it in almost Knuthian `\ttverbatim`—it's part of this 'almost'.

```
\foone{\catcode`\[=1 \@makeother\{ \catcode`\]=2 }%
[%
```
\breaklbrace
```
   \def\breaklbrace[\discretionary[{%
     \verbhyphen][][{%
       ]]%
```
\xiilbrace
```
   \def\xiilbrace[{%
     ]%
]% of \firstofone
\foone{\catcode`\[=1 \catcode`\{=\active \catcode`\]=2 }%
[%
```
\dobreaklbrace
```
   \def\dobreaklbrace[\catcode`\{=\active
   \def{%
```
\breaklbrace
```
     [\breaklbrace\gm@lbracehook]]%
]
```

Now we only initialize the hook. Real use of it will be made in gmdoc.

```
\relaxen\gm@lbracehook
```

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

```
\foone{\catcode`\!=0 \@makeother\\}%
{%
```
\bslash
```
  !def!bslash{\}%
```
\breakbslash
```
  !def!breakbslash{!discretionary{!verbhyphen}{\}{\}}%
}
```

Sometimes linebreaking at a backslash may be unwelcome. The basic case, when the first CS in a verbatim breaks at the lineend leaving there %, is covered by line 186. For the others let's give the user a countercrank:

\fixbslash
```
\newcommand*\fixbslash{\let\breakbslash=\bslash}%
```
to use due to the common scoping rules. But for the special case of a backslash opening a verbatim scope, we deal specially in the line 186.

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

\fixlbrace
```
\newcommand*\fixlbrace{\let\breaklbrace=\xiilbrace}
\foone{\catcode`\!=0 \catcode`\\=\active}%
{%
```

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

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

```
\foone{\catcode`\ =12 }% note this space is 10 and is gobbled by parsing the number.
\xiispace
{\def\xiispace{ }%
\breakablexiispace
\def\breakablexiispace{\discretionary{ }{}{ }}}
\foone\obeyspaces% it's just re\catcode'ing.
{%
\activespace
\newcommand*\activespace{ }%
\dobreakvisiblespace
\breakablexiispace
\newcommand*\dobreakvisiblespace{\def {\breakablexiispace}\obeyspaces}%%
      \defing it caused a stack overflow disaster with gmdoc.
\dobreakblankspace
\newcommand*\dobreakblankspace{\let =\space\obeyspaces}%
}
\bgroup\@makeother\|
\xiiclub
\firstofone{\egroup\def\xiiclub{|}}
```

Almost-Knuthian `\ttverbatim`

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

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

While typesetting stuff in the QX fontencoding I noticed there were no spaces in verbatims. That was because the QX encoding doesn't have any reasonable char at position 32. So we provide a hook in the very core of the verbatim making macros to set proper fontencoding for instance.

```
\@emptify\ttverbatim@hook
\VerbT1
\def\VerbT1{\def\ttverbatim@hook{\fontencoding{T1}\selectfont}}
\VerbT
```

We wish the visible spaces to be the default.

```
\ttverbatim@hook
\let\dobreakspace=\dobreakvisiblespace
```

The Core: From shortvrb

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

```
\MakeShortVerb*
\MakeShortVerb
\@shortvrbdef
\@shortvrbdef
\def\MakeShortVerb{%
  \@ifstar
  {\def\@shortvrbdef{\verb*}\@MakeShortVerb}%
  {\def\@shortvrbdef{\verb}\@MakeShortVerb}}
\@MakeShortVerb
\def\@MakeShortVerb#1{%
  \@xa\ifx\csname cc\string#1\endcsname\relax
  \@shortvrbinfo{Made }{#1}\@shortvrbdef
  \add@special{#1}%
```

```
  \AddtoPrivateOthers#1% a macro to be really defined in gmdoc.
  \@xa
  \xdef\csname cc\string#1\endcsname{\the\catcode`#1}%
  \begingroup
  \catcode`\~\active \lccode`\~`#1%
  \lowercase{%
    \global\@xa\let
    \csname ac\string#1\endcsname~%
    \@xa\gdef\@xa~\@xa{%
      \@xa\ifmmode\@xa\string\@xa~%
      \@xa\else\@xa\afterfi{\@shortvrbdef~}\fi}}% This terrible number of \expandafters
                                                 is to make the shortverb char just other in the math mode (my addition).
  \endgroup
  \global\catcode`#1\active
 \else
 \@shortvrbinfo\@empty{#1 already}{\@empty\verb(*)}%
 \fi}
\DeleteShortVb  \def\DeleteShortVerb#1{%
  \@xa\ifx\csname cc\string#1\endcsname\relax
  \@shortvrbinfo\@empty{#1 not}{\@empty\verb(*)}%
  \else
  \@shortvrbinfo{Deleted }{#1 as}{\@empty\verb(*)}%
  \rem@special{#1}%
  \global\catcode`#1\csname cc\string#1\endcsname
  \global \@xa\let \csname cc\string#1\endcsname \relax
  \ifnum\catcode`#1=\active
  \begingroup
  \catcode`\~\active \lccode`\~`#1%
  \lowercase{%
    \global\@xa\let\@xa~%
    \csname ac\string#1\endcsname}%
  \endgroup \fi \fi}
```

My little addition

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

For the commentary on the below macro see the doc package's documentation. Here let's only say it's just amazing: so tricky and wicked use of `\do`. The internal macro `\rem@special` defines `\do` to expand to nothing if the `\do`'s argument is the one to be removed and to unexpandable CSs `\do` and ⟨*`\do`'s argument*⟩ otherwise. With `\do` defined

this way the entire list is just globally expanded itself. Analogous hack is done to the \@sanitize list.

\rem@special
```
\def\rem@special#1{%
  \def\do##1{%
    \ifnum`#1=`##1 \else \@nx\do\@nx##1\fi}%
  \xdef\dospecials{\dospecials}%
  \begingroup
  \def\@makeother##1{%
    \ifnum`#1=`##1 \else \@nx\@makeother\@nx##1\fi}%
  \xdef\@sanitize{\@sanitize}%
  \endgroup}
```

And now the definition of verbatim itself. As you'll see (I hope), the internal macros of it look for the name of the current environment (i.e., \@currenvir's meaning) to set their expectation of the environment's \end properly. This is done to allow the user to define his/her own environments with \verbatim inside them. I.e., as with the verbatim package, you may write \verbatim in the begdef of your environment and then necessarily \endverbatim in its enddef. Of course (or maybe surprisingly), the commands written in the begdef after \verbatim will also be executed at \begin{⟨*environment*⟩}.

verbatim \verbatim
```
\def\verbatim{\@beginparpenalty \predisplaypenalty \@verbatim
  \frenchspacing \gmobeyspaces \@xverbatim}% in the LaTeX version there's %%
       \@vobeyspaces instead of \gmobeyspaces.
```
verbatim*
```
\@namedef{verbatim*}{\@beginparpenalty \predisplaypenalty \@verbatim
  \@sxverbatim}
```

\endverbatim
```
\def\endverbatim{\@@par
  \ifdim\lastskip >\z@
    \@tempskipa\lastskip \vskip -\lastskip
    \advance\@tempskipa\parskip \advance\@tempskipa -\@outerparskip
    \vskip\@tempskipa
  \fi
  \addvspace\@topsepadd
  \@endparenv}
```

```
\n@melet{endverbatim*}{endverbatim}
```

```
\begingroup \catcode `!=0 %
\catcode `[= 1 \catcode`]=2 %
\catcode`\{=\active
\@makeother\}%
\catcode`\\=\active%
```

\@xverbatim
```
!gdef!@xverbatim[%
  !edef!verbatim@edef[%
    !def!noexpand!verbatim@end%
      ####1!noexpand\end!noexpand{!@currenvir}[%
      ####1!noexpand!end[!@currenvir]]]%
  !verbatim@edef
  !verbatim@end]%
!endgroup
```

\@sxverbatim
```
\let\@sxverbatim=\@xverbatim
```

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

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

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

```
  \parsep\parskip
```

From `\@trivlist`:

```
  \if@noskipsec \leavevmode \fi
  \@topsepadd \topsep
  \ifvmode
    \advance\@topsepadd \partopsep
  \else
    \unskip \par
  \fi
  \@topsep \@topsepadd
  \advance\@topsep \parskip
  \@outerparskip \parskip
```

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

```
  \@@par\addvspace\@topsep
  \if@minipage\else\vskip\parskip\fi
  \leftmargin\parindent% please notify me if it's a bad idea.
  \advance\@totalleftmargin\leftmargin
  \raggedright
  \leftskip\@totalleftmargin% so many assignments to preserve the list thinking for possible future changes. However, we may be sure no internal list shall use \@totalleftmargin as far as no inner environments are possible in verbatim(*).
  \@@par% most probably redundant.
  \@tempswafalse
  \def\par{% but I don't want the terribly ugly empty lines when a blank line is met. Let's make them gmdoc-like i.e., let a vertical space be added as in between stanzas of poetry. Originally \if@tempswa\hbox{}\fi, in my version will be
    \ifvmode\if@tempswa\addvspace\stanzaskip\@tempswafalse\fi\fi
    \@@par
    \penalty\interlinepenalty \check@percent}%
  \everypar{\@tempswatrue\hangindent\verbatimhangindent\hangafter\@ne}% since several chars are breakable, there's a possibility of breaking some lines. We wish them to be hanging indented.
  \obeylines
  \ttverbatim}
```

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

\verbatimhangindent
```
\newlength\verbatimhangindent
\verbatimhangindent=3em
```

\check@percent
```
\providecommand*\check@percent{}
```

In the gmdoc package shall it be defined to check if the next line begins with a comment char.

Similarly, the next macro shall in gmdoc be defined to update a list useful to that package. For now let it just gobble its argument.

\AddtoPrivateOthers
```
\providecommand*\AddtoPrivateOthers[1]{}
```

Both of the above are `\provided` to allow the user to load gmverb after gmdoc (which would be redundant since gmdoc loads this package on its own, but anyway should be harmless).

Let's define the 'short' verbatim command.

```
\verb*  \def\verb{\relax\ifmmode\hbox\else\leavevmode\null\fi
\verb    \bgroup
         \ttverbatim
         \gm@verb@eol
         \@ifstar{\@sverb@chbsl}{\gmobeyspaces\frenchspacing\@sverb@chbsl}}% in the LaTeX version there's \@vobeyspaces instead of \gmobeyspaces.
\@sverb@chbsl  \def\@sverb@chbsl#1{\@sverb#1\check@bslash}
\@def@breakbslash  \def\@def@breakbslash{\breakbslash}% because \ is \defined as \breakbslash not \let.
```

For the special case of a backslash opening a (short) verbatim, in which it shouldn't be breakable, we define the checking macro.

```
\check@bslash  \def\check@bslash{\@ifnextchar{\@def@breakbslash}{\bslash\@gobble}{}}
               \let\verb@balance@group\@empty
\verb@egroup   \def\verb@egroup{\global\let\verb@balance@group\@empty\egroup}
\gm@verb@eol   \let\gm@verb@eol\verb@eol@error
```

The latter is a LaTeX 2_ε kernel macro that `\activeate`s line end and defines it to close the verb group and to issue an error message. We use a separate CS 'cause we are not quite positive to the forbidden line ends idea. (Although the allowed line ends with a forgotten closing shortverb char caused funny disasters at my work a few times.) Another reason is that gmdoc wishes to redefine it for its own queer purpose.

However, let's leave my former 'permissive' definition under the `\verb@eol` name.

```
\begingroup
\obeylines\obeyspaces%
\gdef\verb@eolOK{\obeylines%
\check@percent  \def^^M{ \check@percent}%
}%
\endgroup
```

The `\check@percent` macro here is `\provided` to be `\@empty` but in gmdoc employed shall it be.

Let us leave (give?) a user freedom of choice:

```
\verbeolOK  \def\verbeolOK{\let\gm@verb@eol\verb@eolOK}
```

And back to the main matter,

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

And finally, what I thought to be so smart and clever, now is just one of many possible uses of a general almost Rainer Schöpf's macro:

\dekclubs
```
\def\dekclubs{\@ifstar{\OldMakeShortVerb*\|}{\MakeShortVerb*\|}}
```

But even if a shortverb is unconditional, the spaces in the math mode are not printed. So,

\edverbs
```
\newcommand*\edverbs{%
  \let\gmv@dismath\[%
  \let\gmv@edismath\]%
  \def\[{%
    \@ifnextac\gmv@disverb\gmv@dismath}%
  \let\edverbs\relax}%
```

\gmv@disverb
```
\def\gmv@disverb{%
  \gmv@dismath
  \hbox\bgroup\def\]{\egroup\gmv@edismath}}
```

doc- And shortvrb-Compatibility

One of minor errors while TEXing doc.dtx was caused by my understanding of a 'shortverb' char: at my settings, in the math mode an active 'shortverb' char expands to itself's 'other' version thanks to `\string`. doc/shortvrb's concept is different, there a 'shortverb' char should work as usual in the math mode. So let it may be as they wish:

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

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

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

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

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

g. The gmeometric Package[1]

Written by Grzegorz Murzynowski,
natror at o2 dot pl


```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmeometric}
     [2007/11/17 v0.72 to allow the `geometry' macro in the document
          (GM)]
```

Introduction, usage

This package allows you to use the `\geometry` macro, provided by the geometry v3.2 by Hideo Umeki, anywhere in a document: originally it's claused `\@onlypreamble` and the main work of gmeometric is to change that.

Note it's rather queer to change the page layout *inside* a document and it should be considered as drugs or alcohol: it's O.K. only if you *really* know what you're doing.

In order to work properly, the macro should launch the `\clearpage` or the `\cleardoublepage` to 'commit' the changes. So, the unstarred version trigges the first while the starred the latter. If that doesn't work quite as expected, try to precede or succede it with `\onecolumn` or `\twocolumn`.

It's important that `\clear(double)page` launched by `\geometry` not to be a no-op, i.e., `\clear(double)page` immediately preceding `\geometry` (nothing is printed in between) discards the 'commitment'.

You may use gmeometric just like geometry i.e., to specify the layout as the package options: they shall be passed to geometry.

This package also checks if the engine is XeTeX and sets the proper driver if so. Probably it's redundant since decent XeTeX packages provide their geometry.cfg file that does that.

The remarks about installation and compiling of the documentation are analogous to those in the chapter gmdoc.sty and therefore ommitted.

Contents of the gmeometric.zip archive

The distribution of the gmeometric package consists of the following four files.

gmeometric.sty
README

[1] This file has version number v0.72 dated 2007/11/17.

gmeometricDoc.tex
gmeometricDoc.pdf
gmeometric.tds.zip

Usage

The main use of this package is to allow the `\geometry` command also inside the `document` (originally it's `\@onlypreamble`). To make `\geometry` work properly is quite a different business. It may be advisable to 'commit' the layout changes with `\newpage`, `\clearpage`, or `\cleardoublepage` and maybe `\one/twocolumn`.

Some layout commands should be put before `\one/twocolumn` and other after it. An example:

```
\thispagestyle{empty}
\advance\textheight 3.4cm\relax
\onecolumn
\newpage
\advance\footskip-1.7cm
\geometry{hmargin=1.2cm,vmargin=1cm}
\clearpage
```

And another:

```
\newpage
\geometry{bottom=3.6cm}
```

In some cases it doesn't work perfectly anyway. Well, the (LPPL) license warns about it.

The Code

```
\RequirePackage{gmutils}[2007/04/23]% this package defines the storing and restoring commands.
```

redefine `\@onlypreamble`, add storing to BeginDocument.

\gme@tobestored
```
\newcommand*\gme@tobestored{{%
    \Gm@cnth \Gm@cntv \c@Gm@tempcnt \Gm@bindingoffset \Gm@wd@mp
    \Gm@odd@mp \Gm@even@mp \Gm@orgw \Gm@orgh \Gm@dimlist}}
\@xa\AtBeginDocument\@xa{\@xa\StoreMacros\gme@tobestored}
\StoreMacro\@onlypreamble
\let\@onlypreamble\@gobble
```

To make it work properly in XeTeX:

\pdfoutput
```
\@ifXeTeX{%
  \@ifundefined{pdfoutput}{\newcount\pdfoutput}{}%
  \PassOptionsToPackage{dvipdfm}{geometry}%
}{}
\RequirePackageWithOptions{geometry}
```

Restore `\@onlypreamble`:

```
\RestoreMacro\@onlypreamble
```

Hypothesis: `\ifx...\@undefined` fails in the document because something made `\csname Gm@lines\endcsname`. So we change the test to decent. And i think I've found

the guilty: \@ifundefined in \Gm@showparams. So I change it to the more elegant \ifx% \@undefined.

\Gm@showparams

```
\def\Gm@showparams{%
  -------------------- Geometry parameters^^J%
  \ifGm@pass
  'pass' is specified!! (disables the geometry layouter)^^J%
  \else
  paper: \ifx\Gm@paper\@undefined class default\else\Gm@paper\fi^^J%
  \Gm@checkbool{landscape}%
  twocolumn: \if@twocolumn\Gm@true\else--\fi^^J%
  twoside: \if@twoside\Gm@true\else--\fi^^J%
  asymmetric: \if@mparswitch --\else\if@twoside\Gm@true\else --\fi%
       \fi^^J%
  h-parts: \Gm@lmargin, \Gm@width, \Gm@rmargin%
  \ifnum\Gm@cnth=\z@\space(default)\fi^^J%
  v-parts: \Gm@tmargin, \Gm@height, \Gm@bmargin%
  \ifnum\Gm@cntv=\z@\space(default)\fi^^J%
  hmarginratio: \ifnum\Gm@cnth<5 \ifnum\Gm@cnth=3--\else%
    \Gm@hmarginratio\fi\else--\fi^^J%
  vmarginratio: \ifnum\Gm@cntv<5 \ifnum\Gm@cntv=3--\else%
    \Gm@vmarginratio\fi\else--\fi^^J%
  lines: \ifx\Gm@lines\@undefined--\else\Gm@lines\fi^^J% here I (natror) fix
       the bug: it was \@ifundefined that of course was assigning \relax to
       % \Gm@lines and that resulted in an error when \geometry was used inside
       document.
  \Gm@checkbool{heightrounded}%
  bindingoffset: \the\Gm@bindingoffset^^J%
  truedimen: \ifx\Gm@truedimen\@empty --\else\Gm@true\fi^^J%
  \Gm@checkbool{includehead}%
  \Gm@checkbool{includefoot}%
  \Gm@checkbool{includemp}%
  driver: \Gm@driver^^J%
  \fi
  -------------------- Page layout dimensions and switches^^J%
  \string\paperwidth\space\space\the\paperwidth^^J%
  \string\paperheight\space\the\paperheight^^J%
  \string\textwidth\space\space\the\textwidth^^J%
  \string\textheight\space\the\textheight^^J%
  \string\oddsidemargin\space\space\the\oddsidemargin^^J%
  \string\evensidemargin\space\the\evensidemargin^^J%
  \string\topmargin\space\space\the\topmargin^^J%
  \string\headheight\space\the\headheight^^J%
  \string\headsep\@spaces\the\headsep^^J%
  \string\footskip\space\space\space\the\footskip^^J%
  \string\marginparwidth\space\the\marginparwidth^^J%
  \string\marginparsep\space\space\space\the\marginparsep^^J%
  \string\columnsep\space\space\the\columnsep^^J%
  \string\skip\string\footins\space\space\the\skip\footins^^J%
  \string\hoffset\space\the\hoffset^^J%
  \string\voffset\space\the\voffset^^J%
  \string\mag\space\the\mag^^J%
  \if@twocolumn\string\@twocolumntrue\space\fi%
```

```
  \if@twoside\string\@twosidetrue\space\fi%
  \if@mparswitch\string\@mparswitchtrue\space\fi%
  \if@reversemargin\string\@reversemargintrue\space\fi^^J%
  (1in=72.27pt, 1cm=28.45pt)^^J%
  ----------------------}
```

Add restore to BeginDocument:

```
\@xa\AtBeginDocument\@xa{\@xa\RestoreMacros\gme@tobestored}
\endinput
```

h. The gmoldcomm Package[1]

November 19, 2007

This is a package for handling the old comments in LaTeX 2$_\varepsilon$ Source Files when LaTeXing them with the gmdoc package.

Written by Natror (Grzegorz Murzynowski) 2007/11/10.

It's a part of the gmdoc bundle and as such a subject to the LaTeX Project Public License.

Scan CSs and put them in tt. If at beginning of line, precede them with %. Obey lines in the commentary.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmoldcomm}
          [2007/11/10 v0.99 LaTeX old comments handling (GM)]
```

oldcomments

```
\newenvironment{oldcomments}{%
  \catcode`\\=\active
  \let\do\@makeother
  \do\$% Not only CSs but also special chars happen in the old comments.
  \do\|\do\#\do\{\do\}\do\^\do\_\do\&%
  \gmoc@defbslash
  \obeylines
  \StoreMacro\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[%
```

[1] This file has version number ? dated ?.

```
      \begingroup
\@currenvir
      \def\@currenvir[macrocode]%
      \RestoreMacro\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{}
\endinput
```

Change History

gmdoc v0.96
  General:
    CheckSum 2395, 4
gmdoc v0.98d
  \ChangesStart:
    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. 75.
gmdoc v0.99a
  General:
    CheckSum 4479, 4
gmdoc v0.99b
  General:
    Thanks to the \edverbs declaration in the class, displayed shortverbs simplified; Emacs mode changed to doctex. Author's true name more exposed, 98
gmdoc v0.99c
  ^^M:
    a bug fix: redefinition of it left solely to \QueerEOL, 39
  General:
    A bug fixed in \DocInput and all \expandafters changed to \@xa and \noexpands to \@nx, 98
    The TEX-related logos now are declared with \DeclareLogo provided in gmutils, 98
  \DocInput:
    added ensuring the code delimiter to be the same at the end as at the beginning, 28
gmdoc v0.99d
  General:
    \@namelet renamed to \n@melet to solve a conflict with the beamer class (in gmutils at first), 98
    \afterfi & pals made two-argument, 98
  \FileInfo:
    added, 87
gmdoc v0.99e
  General:
    a bug fixed in \DocInput and \IndexInput, 98
    CheckSum 4574, 4
gmdoc v0.99g
  General:
    CheckSum 5229, 4
    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, 98
  \gmd@ifinmeaning:
    made more elegant: \if changed to \ifx made four parameters and not expanding to an open \iftrue/false. Also renamed from \@ifismember, 42
  hyperref:
    added bypass of encoding for loading url, 24
  \inverb:
    added, 89
  \OldDocInput:
    obsolete redefinition of the macro environment removed, 96
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, 98
gmdocc v0.74
  \edverbs:
    used to simplify displaying shortverbs, 104
gmdocc v0.75
  General:
    CheckSum 130, 99
gmdocc v0.76
  General:
    CheckSum 257, 99
  \EOFMark:
    The gmeometric option made obsolete and the gmeometric package is loaded always, for XETEX-compatibility. And the class options go xkeyval., 104
gmdocc v0.77

General:
CheckSum 262, 99
\EOFMark:
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, 104
gmdocc v0.78
General:
CheckSum 267, 99
\EOFMark:
Added the pagella option not to use Adobe Minion Pro that is not freely licensed, 104
gmdocDoc vNo
General:
CheckSum 41, 105
CheckSum 42, 105
CheckSum 53, 105
gmeometric v0.69
General:
CheckSum 40, 153
gmeometric v0.70
General:
Back to the v0.68 settings because \not@onlypreamble was far too little. Well, in this version the redefinition of \geometry is given up since the 'committing' commands depend on the particular situation so defining only two options doesn't seem advisable, 156
CheckSum 36, 153
gmeometric v0.71
General:
a TDS-compliant zip archive made, 156
CheckSum 41, 153
gmeometric v0.72
General:
CheckSum 239, 153
\Gm@showparams:
a bug fix: \@ifundefined{Gm@lines} raised an error when \geometry used inside the document, I change it to \ifx\@undefined, 155
gmutils v0.74
General:
Added macros to make sectioning commands of mwcls and standard classes compatible. Now my sectionings allow two optionals in both worlds and with mwcls if there's only one optional, it's the title to toc and running head not just to the latter, 140
\begin:
The catcodes of \begin and \end argument(s) don't have to agree strictly anymore: an environment is properly closed if the \begin's and \end's arguments result in the same \csname, 111
gmutils v0.75
\@ifnextac:
added, 109
\@ifnextcat:
\let for #1 changed to \def to allow things like \noexpand~ , 108
\@ifnextif:
\let for #1 changed to \def to allow things like \noexpand~ , 109
gmutils v0.76
General:
A 'fixing' of \dots was rolled back since it came out they were O.K. and that was the QX encoding that prints them very tight, 140
\freeze@actives:
added, 135
gmutils v0.77
General:
\afterfi & pals made two-argument as the Marcin Woliński's analogoi are. At this occasion some redundant macros of that family are deleted, 140
gmutils v0.78
General:
\@namelet renamed to \n@melet to solve a conflict with the beamer class. The package contents regrouped, 140
gmutils v0.79
\not@onlypreamble:
All the actions are done in a group and therefore \xdef used instead of \edef because this command has to use \do (which is contained in the \@preamblecmds list) and \not@onlypreamble itself should be able to be let to \do, 119
gmutils v0.80
General:
CheckSum 1689, 106
\hfillneg:
added, 134
gmutils v0.81
\fnfracc:
moved here from pmlectionis.cls, 137
\ifSecondClass:
moved here from pmlectionis.cls, 137
gmutils v0.82
\ikern:
added, 137
gmutils v0.83

Index

Numbers written in italic refer to the code lines where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used. The numbers with no prefix are page numbers. All the numbers are hyperlinks.

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

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

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

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

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