Grzegorz ‘Natror’ Murzynowski

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

March 2010

Contents

a. The gmdoc.sty Package[1]

March 4, 2010

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.

```
\ifnum\catcode`\@=11 % Why this test here—will come out in chapter The driver.
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmdoc}
          [2010/03/01 v0.991 a documenting package (GM)]
\fi
```

Readme

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

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

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

The package also works with XeTeX (switches automatically).

Installation

Unpack the 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. And you *have* to use four of them to make gmdoc work.)

1 This file has version number v0.991 dated 2010/03/01.

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.

Contents of the gmdoc.zip archive

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

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

Compiling of the documentation

The last of the above files (the .pdf, i.e., *this file*) is a documentation compiled from the .sty and .cls files by running XELATEX on the gmdoc.sty twice (`xelatex␣gmdoc.sty` in the directory you wish the documentation to be in, you don't have copy the .sty file there, TEX will find it), then MakeIndex on the gmdoc.idx and gmdoc.glo files, and then XELATEX on gmdoc.sty once more. (Using LATEX instead of XELATEX should do, too.)

MakeIndex shell commands:

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

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

Compiling the documentation requires the packages: gmdoc (gmdoc.sty and gmdocc.cls), gmutils.sty, gmverb.sty, gmiflink.sty and also some standard packages: hyperref.sty, xcolor.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

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 TEXLive2007 distribution (they only read that directory).

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

Introduction

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

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

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

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

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

The user interface

Used terms

When I write of a **macro**, I mean a macro in *The TEXbook*'s meaning, i.e., a control sequence whose meaning is `\[e|g|x]def`ined. By a **macro's parameter** I mean each of `#`⟨*digit*⟩s in its definition. When I write about a **macro's argument**, I mean the value (list of tokens) substituting 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**.

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

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

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

By a **command** I mean a certain abstract visible to the end user as a CS but consisting possibly of more than one macro. I'll talk of a **command's argument** also in the 'sense -for-the-end-user', e.g., I'll talk of the `\verb` *command's* argument although *the macro* `\verb` has no `#`⟨*digit*⟩ in its definition.

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

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

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

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

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

Preparing of the source file

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

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

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

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

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

However, it may be a bit confusing for someone acquainted with the doc conventions. If you don't fancy the `^^B` special sequence, instead you may restore the standard 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 (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).

`\QueerEOL`

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

Then they should prepare a main document file, a **driver** henceforth, to set all the required formattings such as `\documentclass`, paper size etc., and load this 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}`[4].

(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`.

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.

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

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 \DocIn¦

\SkipFilesAuthors cludes. If you wish to undeclare printing the authors names, there is \SkipFiles¦Authors 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 file-marker also precedes the index entries, if the (default) codeline index option is in force.

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

If you want to put the driver into the same .sty or .cls file (see chapter 1928 to see how), you may write \DocInput{\jobname.sty}, or \DocInclude{\jobname},

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

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

At the default settings, the \(Doc|Self)Included files constitute chapters if \chap¦ter 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)).

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

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

Both these declarations OCSR.

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

\ltxPageLayout

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

\AtBegInput

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

\AtEndInput

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

\AtBegInputOnce

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

\IndexInput

One more input command is \IndexInput (the name and idea of effect comes from doc). It takes the same argument as \DocInput, the file's (path and) name with extension. (It *has* \DocInput inside). It works properly if the input file doesn't contain explicit ⟨*char1*⟩ (^^A is OK).

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

Package options

As many good packages, this also provides some options:

linesnotnum

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

uresetlinecount

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

countalllines

Talking of line numbering further, a tradition seems to exist to number only the codelines and not to number the lines of commentary. That's the default behaviour of gmdoc but, if someone wants the comment lines to be numbered too, which may be convenient for reference purposes, they is provided the countalllines option. This option switches things to use the \inputlineno primitive for codeline numbers so you get the numbers of the source file instead of number only of the codelines. Note however, that there are no hypertargets made to the narration lines and the value of \ref is the number of the most recent codeline.

countalllines*

Moreover, if they wants to get the narration lines' number printed, there is the starred version of that option, countalllines*. I imagine someone may use it for debug. This option is not finished in details, it causes errors with \addvspace because it puts a hyperlabel at every line. When it is in force, all the index entries are referenced with the line numbers and 441 the narration acquires a bit biblical look ;-), 442 as shown in this short example. This option is intended 443 for the draft versions and it is not perfect (as if anything 444 in this package was). As you see, the lines 445 are typeset continuously with the numbers printed.

noindex

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

pageindex

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

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

By default, gmdoc excludes some 300 CSes from being indexed. They are the most common CSes, LaTeX internal macros and TeX primitives. To learn what CSes are excluded actually, see lines 5681–5807.

indexallmacros

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

If you have ambiguous feelings about whether to let the default exclusions or forbid them, see p. 16 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 alright but not all the classes support marginpars. That is the reason why this package enables marginpar-ing 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 document class.

codespacesblank
\CodeSpacesBlank

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

codespacesgrey
\CodeSpacesGrey

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

\VisSpacesGrey

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

The packages required

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

gmverb

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

\verbatimspecials

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

```
\verbatimspecials⁄«»[¿]
```

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

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

\verbeolOK

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

```
\MakeShortVerb*\|
```

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

\dekclubs

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 declare e.g., `\DeleteShortVerb\|` and the previous meaning of the char used shall be restored.

\DeleteShortVerb

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

gmutils

The gmutils package provides a couple of macros similar to some basic (LA)TEX ones, rather strictly technical and (I hope) tricky, such as `\afterfi`, `\ifnextcat`, `\addtomacro` etc. It's this package that provides the macros for formatting of names of macros and files, such as `\cs`, `\marg`, `\pk` etc. Moreover, it provides a powerful tool for defining commands with weird optional and Knuthian arguments, `\DeclareCommand`, inspired by ancient (pre-expl3) xparse's `\DeclareDocumentCommand`.

hyperref

The gmdoc package uses a lot of hyperlinking possibilities provided by hyperref which is therefore probably the most important package required. The recommended situation is that the user loads hyperref package with their favourite options *before* loading gmdoc. If they does not, gmdoc shall load it with *my* favourite options.

gmiflink

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

multicol

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

color

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

Automatic marking of definitions

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

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

of the xkeyval package:

4. `\define@key`, `\define@boolkey`, `\define@choicekey`, `\DeclareOptionX`,

and of the kvoptions package:

5. `\DeclareStringOption`, `\DeclareBoolOption`, `\DeclareComplementaryOption`, `\DeclareVoidOption`.

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

\DeclareDefining

If you want to add detection of a defining command not listed above, use the `\DeclareDefining` declaration. It comes in two flavours: 'sauté' and with star. The 'sauté' version (without star and without an optional argument) declares a defining command of the kind of `\def` and `\newcommand`: its main argument, whether wrapped in braces or not, is a CS. The starred version (without the optional argument) declares a defining command of the kind of `\newenvironment` and `\DeclareOption`: whose main mandatory argument is text. Both versions provide an optional argument in which you can set the keys.

type

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

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

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

\DeclareDOXHead

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

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

\DeclareKVOFam

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

star

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

KVpref
KVfam

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

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

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

declare

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

or

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

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

\HideDefining

On the other hand, if you want some of the commands listed above *not* to be detected, write `\HideDefining\`⟨*command*⟩ in the commentary. If both ⟨*command*⟩ and ⟨*command**⟩ are detected, then both will be hidden. `\HideDefining` is always `\global`. Later you can resume detection of ⟨*command*⟩ and ⟨*command**⟩ with `\ResumeDefining`⟨*command*⟩ which is always `\global` too. Moreover, if you wish to suspend automatic detection of the defining ⟨*command*⟩ only once (the next occurrence), there is `\HideDefining*` which suspends detection of the next occurrence of ⟨*command*⟩. So, if you wish to 'hide' `\providecommand*` once, write

\ResumeDefining

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

\HideAllDefining
\ResumeAllDefining

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

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

\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` which has the starred version `\HideDef*` equivalent to `\UnDef`. To resume detection of `\def` you are provided also a shorthand, `\ResumeDef` (but `\ResumeDefining\def` also works).

\HideDef
\HideDef*
\ResumeDef

Since I use `\pdef` most often, I provide also `\UnPdef`, analogous to `\UnDef`.

\UnPdef

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

Manual Marking of the Macros and Environments

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

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

This package provides two kinds of special formatting of the index entries: 'usage', with the reference number italic by default, and 'def' (in doc called 'main'), with the reference number roman (upright) and underlined by default. All the reference numbers, also those with no special formatting, are made hyperlinks to the page or the codeline according to the respective indexing option (see p. 11).

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

The arguments to all three are as follows:

`#1` [`*`] to indicate whether we mark a single CS or more than one token(s): without star for a single CS, with star for environment names etc., the starred version executes `\@sanitize`,

`[#2]` `o` version to be marginized and printed here,

`#3` `m` version to be put to the index, and also (printed here and) marginized if the previous argument is missing.

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

`\MakePrivateLetters`

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

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

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

right before the code line

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

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

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

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

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

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

`\Describe`

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

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

`\CodeMarginize`
`\TextMarginize`

But what if you want just to make a marginpar with macro's or environment's name? Then you have `\CodeMarginize` to declare what to put into a marginpar in the TEX code (it's `\outer`) and `\TextMarginize` to do so in the commentary. According to

the spirit of this part of the interface, these commands also take one argument and have their starred versions for strings other than control sequences.

\marginpartt

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

\gmdmarginpar

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

\DefIndex
\CodeUsgIndex

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

\CodeCommonIndex*

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

\TextUsgIndex
\TextCommonIndex

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

macro
environment

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

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

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

Index ex/inclusions

\DoNotIndex

It's understandable[6] that you don't want some control sequences to be indexed in your documentation. The doc package gives a brilliant solution: the `\DoNotIndex` declaration. So do I (although here, TEXnically it's done another way). It OCSR. This declaration takes one argument consisting of a list of control sequences not to be indexed. The items

6 After reading doc's documentation ;-) .

of this list may be separated with commas, as in doc, but it's not obligatory. The whole list should come in curly braces (except when it's one-element), e.g.,

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

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

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

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

\DoIndex

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

DefaultIndexExclusions
DefaultIndexExclusions

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

The DocStrip directives

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

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

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

The changes history

The doc's documentation reads:

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

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

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

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

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

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

\RecordChanges

To cause the change information to be written out, include `\RecordChanges` in the driver['s preamble or just in the source file (gmdocc.cls does it for you)]. To read in and

\PrintChanges

print the sorted change history (in two columns), just put the `\PrintChanges` com-

mand as the last (commented-out, and thus executed during the documentation pass through the file) command in your package file [or in the driver]. Alternatively, this command may form one of the arguments of the `\StopEventually` command, although a change history is probably not required if only the description is being printed. The command assumes that MakeIndex or some other program has processed the `.glo` file to generate a sorted `.gls` file. You need a special MakeIndex style file; a suitable one is supplied with `doc` [and `gmdoc`], called [… **`gmglo.ist`** for `gmdoc`]. The `\GlossaryMin`, `\GlossaryPrologue` and `\GlossaryParms` macros are analogous to the `\Index`… versions [see sec. The parameters p. 21]. (The LaTeX 'glossary' mechanism is used for the change entries.)"

`\GlossaryMin`
`\GlossaryPrologue`
`\GlossaryParms`

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`, *18*

Such a setting is in force till the next definition and *every* detected definition resets it. In `gmdoc` `\changes` is `\outer`.

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

`\MakeGlossaryControls`

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

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

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

`ChangesStartDate`
`\ChangesStart`

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

declaration.

If the `ChangesStartDate` is set to a date contemporary to TeX i.e., not earlier than September 1982[7], then a note shall appear at the beginning of the changes history that informs the reader of omitting the earlier changes entries.

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

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

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

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

\CheckSum

While scanning the CSes in the code, gmdoc counts them and prints the information about their number on the terminal and in .log. Moreover, you may declare \Check¦Sum{⟨*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*⟩.

\toCTAN

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

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

The parameters

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

\stanzaskip

\stanzaskip is a vertical space inserted when a blank (code) line is met. It's equal \medskipamount by default. Subsequent blank code lines do not increase this space.

\CodeTopsep

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

\UniformSkips
\NonUniformSkips

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

\stanza
\chunkskip

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

\nostanza

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

\CodeIndent

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

\TextIndent

There's also a parameter for the indent of the narration, \TextIndent, but you

should use it only in emergency (otherwise what would be the margins for?). It's 0 sp by default.

By default, the end of a `\DocInput` file is marked with

`\EOFMark` given by the `\EOFMark` macro.

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

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

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

The unstarred version of `\CodeDelim` changes also the verb 'hyphen', the char ap- `\narrationmark` pearing at the verbatim line breaks that is and affects the `\narrationmark` which by default typesets `%` followed by an en space.

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

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

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

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

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

`\IndexPrefix` Three more parameter macros, are `\IndexPrefix`, `\EntryPrefix` and `\HLPre|` `\EntryPrefix` `fix`. All three are provided with the account of including multiple files in one docu- `\HLPrefix` ment. They are equal (almost) `\@empty` by default. The first may store main level index entry of which all indexed macros and environments would be sub-entries, e.g., the name of the package. The third may or even should store a text to distinguish equal codeline numbers of distinct source files. It may be the file name too, of course. The second

macro is intended for another concept, namely the one from ltxdoc class, to distinguish the codeline numbers from different files *in the index* by the file marker. Anyway, if you document just one file per document, there's no need of redefining those macros, nor when you input multiple files with `\DocInclude`.

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

\UsgEntry
\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 the source of chapter 1928, where `\def\CommonEntryCmd{UsgEntry}` makes all the index entries of the driver formatted as 'usage'.

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

IndexColumns
\IndexMin

\IndexPrologue

The text put at the beginning of the index is declared with a one-argument `\IndexPrologue`. Its default text at current index option you may admire on page 263. Of course, you may write your own `\IndexPrologue{`*⟨brand new index prologue⟩*`}`, but if you like the default and want only to add something to it, you are provided `\AtDIPrologue` one-argument declaration that adds the stuff after the default text. For instance, I used it to add a label and hypertarget that is referred to two sentences earlier.

\AtDIPrologue

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

\IndexLinksBlack

\IndexParms
\gaddtomacro

Other index parameters are set with the `\IndexParms` macro defined in line 5926 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.)

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

\actualchar
\quotechar
\levelchar
\encapchar

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

\verbatimchar

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

\StopEventually
\Finale
\AlsoImplementation
\OnlyDescription

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

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

The narration macros

\verb

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

\inverb

For short verbatim texts in the in-line comments gmdoc provides the `\inverb`⟨*a char*⟩…⟨*a char*⟩ (the name stands for 'in-line verbatim') command that redefines the gmverb breakables to break with `%` at the beginning of the lower line to avoid mistaking such a broken verbatim commentary text for the code.

But nor `\verb[*]` neither `\inverb` will work if you put them in an argument of another macro. For such a situation, or if you just prefer, gmdoc (gmutils) provides a robust command `\cs`, which takes one obligatory argument, the macro's name without the backslash, e.g., `\cs{mymacro}` produces `\mymacro`. I take account of a need of printing some other text verbatim, too, and therefore `\cs` has the first argument optional, which is the text to be typeset before the mandatory argument. It's the backslash by default, but if you wish to typeset something without the \, you may write `\cs[]{%not␣␣a~macro}`. Moreover, for typesetting the environments' names, gmdoc (gmutils) provides the `\env` macro, that prints its argument verbatim and without a backslash, e.g., `\env{an␣environment}` produces `an environment`.

\cs

\env

\incs
\inenv
\incmd

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

\nlperc
\+

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

\nlpercent

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

ilrr

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

\ilrr
\ilju

\pk
\file

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

\catletter
\catother
\catactive

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

\division
\subdivision
\subsubdivision

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

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

It's resolved in gmutils so you are free at this point, and even more free than in the standard classes: if you give a sectioning command just one optional argument, it will be the title to toc and to the running head (that's standard in scls[8]). If you give *two* 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 XEHTEX (the first `E` will be reversed if the graphics package is loaded or XEHTEX 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.

8 See gmutils for some subtle details.

\stanza
\chunkskip

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

quotation

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

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

\GetFileInfo
\filedate
\fileversion
\fileinfo

The `\GetFileInfo{`⟨*file name with extension*⟩`}` command defines `\filedate`, `\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).

If the input file doesn't contain `\Provides`… in the code layer, there are commands

\ProvideFileInfo

`\ProvideFileInfo{`⟨*file name with extension*⟩`}[`⟨*info*⟩`]`. (⟨*info*⟩ should consist of: ⟨*year*⟩/⟨*month*⟩/⟨*day*⟩␣⟨*version number*⟩␣⟨*a short note*⟩.)

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

\FileInfo
\filenote

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

\thfileinfo

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

\gmdnoindent

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

\CDPerc
\CDAnd

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

\CS

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

enumargs
\mand
\opt
enumargs*

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

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

For an example driver file see chapter The driver.

A queerness of `\label`

You should be loyally informed that `\label` in gmdoc behaves slightly non-standard in the `\DocInput/Included` files: the automatic redefinitions of `\ref` at each code line

are *global* (since the code is typeset in groups and the `\ref`s will be out of those groups), so a `\ref`erence in the narrative will point at the last code line not the last section, *unlike* in the standard LaTeX.

doc-compatibility

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

\AfterMacrocode

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

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

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

\OldDocInput
\DocInclude
\olddocIncludes
macrocode

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

Within their scope/argument the `macrocode` environments behave as in doc, i.e. they are a kind of verbatim and require to be ended with `%␣␣␣␣␣\end{macrocode[*]}`.

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

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

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

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

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

oldmc

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

\OldMacrocodes

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

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

1928 ⟨*package⟩

The driver part

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

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

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

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

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

\skiplines
\endskiplines

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

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

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

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

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

\errorcontextlines=100

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

\verbLongDashes

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

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

%\includeonly{gmoldcomm}

\begin{document}
```

```
\maketitle

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

\SelfInclude
\DocInclude{gmdocc}
```

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

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

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

MakeIndex shell commands:

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

(gmglo.ist should be put into some texmf/makeindex directory.)

And “That’s all, folks” ;-) .

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

The code

For debug

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

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

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

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

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

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

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

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

The package options

```
\RequirePackage{gmutils}[2008/08/30]% includes redefinition of \newif
    to make the switches \protected.
\RequirePackage{xkeyval}% we need key-vals later, but maybe we'll make the
    option key-val as well.
```

Maybe someone wants the code lines not to be numbered.

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

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

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

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

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

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

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

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

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

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

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

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

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

pageindex
```
\DeclareOption{pageindex}{\@pageindextrue}
```

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

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

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

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

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

This switch is copied from mwbk.cls for compatibility with it. Thanks to it loading an mwcls with `[withmarginpar]` option shall switch marginpars on in this package, too. To be compatible with the standard classes, let's `\let`:

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

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

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

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

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

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

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

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

The dependencies and preliminaries

We require another package of mine that provides some tricky macros analogous to the LaTeX standard ones, such as `\newgif` and `\@ifnextcat`. Since 2008/08/08 it also makes `\if...` switches `\protected` (redefines `\newif`)

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

A standard package for defining colours,

```
\RequirePackage{xcolor}
```

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

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

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

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

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

```
\RequirePackage{gmiflink}
```

And a slight redefinition of `verbatim`, `\verb[*]` and providing of `\MakeShortVerb[*]`.

```
\RequirePackage{gmverb}[2008/08/20]
```

```
\StoreMacros{\@verbatim\verb}
```

```
\if@noindex
  \AtBeginDocument{\gag@index}%  for the latter macro see line 5169.
\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{\@bsphack\gm@ifstar\Code@Delim@St\Code@Delim}
```

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

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

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

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

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

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

```
\CodeDelim\%
```

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

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

We'll play with `\everypar`, a bit, and if you use such things as the `{itemize}` environment, an error would occur if we didn't store the previous value of `\everypar`

and didn't restore it at return to the narration. So let's assign a `\toks` list to store the original `\everypar`:

```
\gmd@preverypar 2327 \newtoks\gmd@preverypar
```

```
\settexcodehangi 2329 \newcommand*\settexcodehangi{%
2330   \hangindent=\verbatimhangindent␣\hangafter=\@ne}%
```
we'll use it in the in-line comment case. `\verbatimhangindent` is provided by the gmverb package and = 3 em by default.

```
2334 \@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 2340 \newlength\TextIndent
```

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

```
2344 \newlength\CodeIndent
\CodeIndent 2347 \CodeIndent=1,5em\relax
```

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

```
2350 \@ifundefined{stanzaskip}{\newlength\stanzaskip}{}
```

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

```
\stanzaskip 2355 \stanzaskip=\medskipamount
```

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

```
2362 \newskip\CodeTopsep
2363 \newskip\MacroTopsep
```

And let's set them. For æsthetic minimality[9] let's unify them and the other most important vertical spaces used in gmdoc. I think a macro that gathers all these assignments may be handy.

```
\UniformSkips 2379 \def\UniformSkips{%
 \CodeTopsep 2381   \CodeTopsep=\stanzaskip
\MacroTopsep 2382   \MacroTopsep=\stanzaskip
2383   \abovedisplayskip=\stanzaskip
%%␣\abovedisplayshortskip remains untouched as it is 0.0 pt plus 3.0 pt by default.
2388   \belowdisplayskip=\stanzaskip
2389   \belowdisplayshortskip=.5\stanzaskip%
```
due to DEK's idea of making the short below display skip half of the normal.

```
2391   \advance\belowdisplayshortskip␣by\smallskipamount
2392   \advance\belowdisplayshortskip␣by-1\smallskipamount%
```
We advance `% \belowdisplayshortskip` forth and back to give it the `\small¦% skipamount`'s shrink- and stretchability components.

```
2396   \topsep=\stanzaskip
2397   \partopsep=\z@
```

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

```
}
```

We make it the default,

```
\UniformSkips
```

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

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

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

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

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

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

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

And, for a smaller part of text,

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

Since the stanza skips are inserted automatically most often (cf. lines 2891, 3318, 2911, 3199, 3371), sometimes you may need to forbid them.

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

To count the lines where they have begun not before them

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

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

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

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

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

The core

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

```
2474 \begingroup\catcode`\^^M=\active%
2475 \firstofone{\endgroup%
\DocInput 2476   \newcommand*{\DocInput}[1]{\begingroup%
2479     \edef\gmd@inputname{#1}%
```
we'll use it in some notifications.
```
2481     \let\gmd@currentlabel@before=\@currentlabel%
```
we store it because we'll do `\xdef`s of `\@currentlabel` to make proper references to the line numbers so we want to restore current `\@currentlabel` after our group.
```
2486     \gmd@setclubpenalty%
```
we wrapped the assignment of `\clubpenalty` in a macro because we'll repeat it twice more.
```
2488     \@clubpenalty\clubpenalty␣\widowpenalty=3333␣%
```
Most paragraphs of the code will be one-line most probably and many of the narration, too.
```
2493     \tolerance=1000␣%
```
as in doc.
```
2496     \@xa\@makeother\csname\code@delim\endcsname%
2498     \gmd@resetlinecount%
```
due to the option `uresetlinecount` we reset the line number counter or do nothing.
```
^^M 2501     \QueerEOL%
```
It has to be before the begin-input-hook to allow change by that hook.
```
2506     \@beginputhook%
```
my first use of it is to redefine `\maketitle` just at this point not globally.
```
2508     \everypar=\@xa{\@xa\@codetonarrskip\the\everypar}%
\gmd@guardedinput 2510     \edef\gmd@guardedinput{%
2511       \@nx\@@input␣#1\relax%
```
`\@nx` is `\noexpand`, see gmutils. `\@@input` is the true TEX's `\input`.
```
2515       \gmd@iihook%
```
cf. line 7340
```
2516       \@nx\EOFMark%
```
to pretty finish the input, see line 2718.
```
2518       \@nx\CodeDelim\@xa\@nx\csname\code@delim\endcsname%
```
to ensure the code delimiter is the same as at the beginning of input.
```
2523       \@nx^^M\code@delim%
2525     }%
```
we add guardians after `\input`ing a file; somehow an error occurred without them.
```
2527     \catcode`\%=9␣%
```
for doc-compatibility.
```
2528     \setcounter{CheckSum}{0}%
```
we initialise the counter for the number of the escape chars (the assignment is `\global`).
```
2530     \everyeof{\relax}%
```
`\@nx` moved not to spoil input of toc e.g.
```
2531     \@xa\@xa\@xa^^M\gmd@guardedinput%
2532     \par%
2534     \@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.
```
2537     \glet\@currentlabel=\gmd@currentlabel@before%
```
we restore value from before this group. In a very special case this could cause unex-

pected behaviour of cross-refs, but anyway we acted globally and so acts hyperref.

```
2541        \endgroup%
2542    }% end of \Doc@Input's definition.
2543 }% 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`

```
2553 \pdef\gmd@textEOL{␣%
```
a space just like in normal TEX. We put it first to cooperate with `\^^M`'s `\expandafter\ignorespaces`. It's no problem since a space ␣10 doesn't drive TEX out of the vmode.

```
2557     \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).

```
2564     \@newlinegtrue%
```
to `\refstep` the lines' counter at the proper point.

```
2566     \@dsdirgtrue%
```
to handle the DocStrip directives.

```
2567     \@xa\@trimandstore\the\everypar\@trimandstore%
```
we store the previous value of `\everypar` register to restore it at a proper point. See line 3407 for the details.

```
2570     \begingroup%
2576     \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.

```
2580     \aftergroup\gmd@setclubpenalty%
2581     \let\par\@@par%
```
inside the verbatim group we wish `\par` to be genuine.

```
2583     \ttverbatim%
```
it does `\tt` and makes specials other or `\active`-and-breakable. to turn verbatim specials off in `\scanverbs`.

```
2587     \gmd@DoTeXCodeSpace%
2588     \@makeother\|%
```
because `\ttverbatim` doesn't do that.

```
2589     \MakePrivateLetters%
```
see line 3681.

```
2590     \@xa\@makeother\code@delim%
```
we are almost sure the code comment char is among the chars having been ₁₂ed already. For 'almost' see the `\Index¦Input` 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.

```
2599     \texcode@hook%
```
we add some special stuff, e.g. in gmdocc.cls we make star low.

```
2601     \@xa\@ifnextchar\@xa{\code@delim}{%
2603         \gmd@continuenarration}{%
2604         \gmd@dolspaces%
```
it will launch `\gmd@typesettexcode`.

```
2605     }% end of \@ifnextchar's else.
2606 }% end of \gmd@textEOL's definition.

2609 \emptify\texcode@hook
```

`\gmd@setclubpenalty`

```
2611 \def\gmd@setclubpenalty{\clubpenalty=3333␣}
```

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

```
\AtEndInput 2615 \def\AtEndInput{\g@addto@macro\@endinputhook}
\@endinputhook 2616 \def\@endinputhook{}
```

Simili modo

```
\AtBegInput 2619 \def\AtBegInput{\g@addto@macro\@beginputhook}
\@beginputhook 2620 \def\@beginputhook{}
```

For the index input hooking now declare a macro, we define it another way at line 7340.

```
2624 \emptify\gmd@iihook
```

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

```
2629 \AtBegInput{\let\gmd@@toc\tableofcontents
\tableofcontents 2630   \def\tableofcontents{%
2631     \@ifQueerEOL{\StraightEOL\gmd@@toc\QueerEOL}%
2632     {\gmd@@toc}}}
```

As you'll learn from lines 3503 and 3490, we use those two strange declarations to change and restore the very special meaning of the line end. Without such changes `\tableofcontents` would cause a disaster (it did indeed). And to check the catcode of `^^M` is the rôle of `\@ifEOLactive`:

```
\@ifEOLactive 2644 \def\@ifEOLactive{%
                     % #1 what if end of line is active,
                     % #2 what if not.
2649   \ifnum\catcode`\^^M=\active␣\@xa\@firstoftwo\else\@xa%
           \@secondoftwo\fi}

2651 \foone\obeylines{%
\@ifQueerEOL 2652   \def\@ifQueerEOL{%
                       % #1 what if line end is 'queer',
                       % #2 what if not 'queer'.
2658     \@ifEOLactive{%
2659       \ifx^^M\gmd@textEOL\@xa\@firstoftwo\else\@xa%
               \@secondoftwo\fi}%
2660     {\@secondoftwo}}% of \@ifQueerEOL
2661 }% of \foone
```

A footnote for the 'queer' line ends scope.

```
\qfootnote 2664 \pdef\qfootnote{%
2665   \@ifQueerEOL
2666   {\begingroup\StraightEOL\qfootnote@}%
2667   {\footnote}}

\qfootnote@ 2669 \DeclareCommand\qfootnote@{o>Lm}{%
2670   \endgroup␣% yes, we close the group: the arguments are already parsed and
                 passed to this macro.
2672   \edef\gmu@tempa{%
2673     \@nx\footnote␣\IfValueT{#1}{[#1]}}%
2674   \gmu@tempa{#2}%
2675 }
```

An emphasis command for 'queer' line ends.

```
\qemph 2678 \pdef\qemph{%
2679   \@ifQueerEOL
2680   {\begingroup\StraightEOL\qemph@}%
```

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

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

```
\gmd@ABIOnce \@emptify\gmd@ABIOnce
\AtEndOfPackage{\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 2523 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 8312.

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

```
\begin{obeyspaces}%
\gdef\CodeSpacesVisible{%
\gmd@DoTeXCodeSpace \def\gmd@DoTeXCodeSpace{%
\obeyspaces\let␣=\breakablevisspace}}%
\CodeSpacesBlank \gdef\CodeSpacesBlank{%
\let\gmd@DoTeXCodeSpace\gmobeyspaces%
\let\gmd@texcodespace=\␣}% the latter \let is for the \if...s.
\CodeSpacesSmall \gdef\CodeSpacesSmall{%
\gmd@DoTeXCodeSpace \def\gmd@DoTeXCodeSpace{%
\obeyspaces\def␣{\,\hskip\z@}}%
\gmd@texcodespace \def\gmd@texcodespace{\,\hskip\z@}}%

\end{obeyspaces}
\CodeSpacesGrey \def\CodeSpacesGrey{%
    \CodeSpacesVisible
    \VisSpacesGrey% defined in gmverb
}%
```

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

```
\CodeSpacesVisible
```

How the continuing of the narration should look like?

```
\gmd@continuenarration \def\gmd@continuenarration{%
    \endgroup
    \gmd@cpnarrline% 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 2601. And

in line 3019 it's been read as $_{12}$. That's why it works in spite of that `%` is of category 'ignored'.)

```
\if@countalllines
```

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

`\gmd@countnarrline@`

```
\def\gmd@countnarrline@{%
    \gmd@grefstep{codelinenum}\@newlinegfalse
    \everypar=\@xa{%
      \@xa\@codetonarrskip\the\gmd@preverypar}% the \hyperlab
          % el@line macro puts a hypertarget in a \raise i.e., drives TeX
          into the horizontal mode so \everypar shall be issued. There-
          fore we should restore it.
  }% of \gmd@countnarrline@
```

`\gmd@grefstep`

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

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

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

`\gmd@cpnarrline`

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

`\gmd@ctallsetup`

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

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

\skiplines
```
\def\skiplines{\bgroup
  \let\do\@makeother␣\dospecials␣% not \@sanitize because the latter
    doesn't recatcode braces and we want all to be quieten.
  \gmd@skiplines}
\edef\gmu@tempa{%
  \long\def\@nx\gmd@skiplines##1\bslash␣endskiplines{%
    \egroup}}
\gmu@tempa
```

And typesetting the TEX code?

\gmd@typesettexcode
```
\foone\obeylines{%
  \def\gmd@typesettexcode{%
    \gmd@parfixclosingspace% it's to eat a space closing the paragraph, see
      below. It contains \par.
```

A verbatim group has already been opened by `\ttverbatim` and additional `\catcode`.

```
    \everypar={\@@settexcodehangi}% At first attempt we thought of giv-
      ing the user a \toks list to insert at the beginning of every code line, but
      what for?
```

^^M \@newlinegtrue
```
    \def^^M{% TEX code EOL
      \@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\@aftercodegfalse%
        \fi%
      \else\gmd@nocodeskip1{naN}%
      \fi%
    \fi% if now we are switching from the narration into the code, we insert
         a proper vertical space.
    \@aftercodegtrue\@afternarrgfalse%
    \ifdim\gmd@ldspaceswd>\z@% and here the leading spaces.
      \leavevmode\@dsdirgfalse%
      \if@newline\gmd@grefstep{codelinenum}\@newlinegfalse%
      \fi%
      \printlinenumber% if we don't want the lines to be numbered, the re-
           spective 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 5043.
      \hskip\gmd@ldspaceswd%
      \advance\hangindent␣by\gmd@ldspaceswd%
      \xdef\settexcodehangi{%
        \@nx\hangindent=\the\hangindent% and also set the hanging in-
             dent setting for the same line comment case. BTW., this % or rather
             lack of it costed me five hours of debugging and rewriting. Active
             line ends require extreme caution.
        \@nx\hangafter=1\space}%
    \else%
      \glet\settexcodehangi=\@@settexcodehangi%
           % \printlinenumber here produced line numbers for blank lines
           which is what we don't want.
    \fi% of \ifdim
    \gmd@ldspaceswd=\z@%
    \prevhmodegfalse% we have done \par so we are not in the hmode.
    \@aftercodegtrue% we want to know later that now we are typesetting
         a codeline.
    \if@ilgroup\aftergroup\egroup\@ilgroupfalse\fi% when we are
         in the in-line comment group (for ragged right or justified), we want to
         close it. But if we did it here, we would close the verbatim group for the
         code. But we set the switch false not to repeat \aftergroup\egroup.
    \gmd@charbychar% we'll eat the code char by char to scan all the macros and
         thus to deal properly with the case \% in which the % will be scanned and
         won't launch closing of the verbatim group.
  }% of \gmd@typesettexcode.
}% of \foone\obeylines.
```

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

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

\gmd@ldspaceswd
```
\newlength\gmd@ldspaceswd% to store total length of gobbled leading spaces.
```

It costed me some time to reach that in my verbatim scope a space isn't $_{12}$ but $_{13}$, namely `\let` to `\breakablevisspace`. So let us `\let` for future:

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

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

`\gmd@dolspaces`
```
\def\gmd@dolspaces{%
  \ifx\gmd@texcodespace\@let@token
    \@dsdirgfalse
    \afterfi{\settowidth{\gmd@spacewd}{\visiblespace}%
    \gmd@ldspaceswd=\z@
    \gmd@eatlspace}%
  \else\afterfi{% about this smart macro and other of its family see gmutils sec. 3.
    \if@afternarr\if@aftercode
        \ifilrr\bgroup␣\gmd@setilrr\fi
    \fi\fi
    \par% possibly after narration
    \if@afternarr\if@aftercode
        \ifilrr\egroup\fi
    \fi\fi
    \gmd@typesettexcode}%
  \fi}
```

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

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

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

```
\newgif\ifprevhmode
```

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

`\gmd@charbychar`
```
\newcommand*\gmd@charbychar[1]{%
  \ifhmode\prevhmodegtrue
  \else\prevhmodegfalse
  \fi
  \if\code@delim\@nx#1%
    \def\next{% occurs when next a \hskip4.875pt is to be put
      \gmd@percenthack% to typeset % if a comment continues the codeline.
    \endgroup%
```

```
      \gmd@checkifEOLmixd}% to see if next is ^^M and then do \par.
    \else% i.e., we've not met the code delimiter
      \ifx\relax#1\def\next{%
        \endgroup}% special case of end of file thanks to \everyeof.
      \else
        \if\code@escape@char\@nx#1%
          \@dsdirgfalse% yes, just here not before the whole \if because then
                we would discard checking for DocStrip directives doable by the
                active % at the 'old macrocode' setting.
          \def\next{%
            \gmd@countthelineNUM#1\scan@macro}%
        \else
          \def\next{%
            \gmd@EOLorcharbychar#1}%
        \fi
      \fi
    \fi\next}
\debug@special \def\debug@special#1{%
    \ifhmode\special{color␣push␣gray␣0.#1}%
    \else\special{color␣push␣gray␣0.#1000}\fi}
```

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

```
\gmd@EOLorcharbychar \def\gmd@EOLorcharbychar#1{%
    \ifx\gmd@texcodeEOL#1%
      \if@newline
        \@newlinegfalse
      \fi
      \afterfi{#1}% here we print #1.
    \else% i.e., #1 is not a (very active) line end,
      \afterfi
      {%
\gmd@countthelineNUM#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@countthelineNUM \def\gmd@countthelineNUM{%
    \ifvmode
      \if@newline
        \leavevmode
        \gmd@grefstep{codelinenum}\@newlinegfalse
        \hyperlabel@line
      \fi
      \printlinenumber
      \mark@envir
    \else% not vmode
      \if@newline
        \gmd@grefstep{codelinenum}\@newlinegfalse
```

```
        \hyperlabel@line
      \fi
    \fi}
```

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

\gmd@percenthack
```
\def\gmd@percenthack{%
  \ifprevhmode\aftergroup\narrationmark% We add a space after %, be-
      cause I think it looks better. It's done \aftergroup to make the spaces
      possible after the % not to be typeset.
  \else\aftergroup\gmd@narrcheckifds@ne% remember that \gmd@pre¦
      centhack 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@cpnarrline
  \everypar=\@xa{\@xa\@codetonarrskip% we add the macro that'll insert
      a vertical space if we leave the code and enter the narration.
    \the\gmd@preverypar}%
  \@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@cpnarrline
  \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., 3165.
    \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 3185      \def\par{% the narration \par.
     3186        \ifhmode% (I added this \ifhmode as a result of a heavy debug.)
     3188          \if@afternarr\if@aftercode
     3189               \unless\if@ilgroup\bgroup\@ilgrouptrue\fi
     3190               \ifilrr\gmd@setilrr\fi
     3191          \fi\fi
     3192          \@@par
     3193          \if@afternarr
     3194            \if@aftercode
     3195            \if@ilgroup\egroup\fi% if we are both after code and after narration it means we are after an in-line comment. Then we probably end a group opened in line 3238
     3199              \if@codeskipput\else\gmd@codeskip2%
                            \@aftercodegfalse\fi
     3201            \else\gmd@nocodeskip2{naC}%
     3202            \fi
     3203          \else\gmd@nocodeskip2{naN}%
     3204          \fi
     3205          \prevhmodegfalse\gmd@endpe% when taken out of \ifhmode, this line caused some codeline numbers were typeset with \leftskip = 0.
     3208          \everypar=\@xa{%
     3209            \@xa\@codetonarrskip\the\gmd@preverypar}%
     3210          \let\par\@@par%
     3211        \fi}% of \par.
     3212      \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 3219 \def\gmd@endpe{%
     3220   \ifprevhmode
     3221     \settexcodehangi% ndent
     3222     \leftskip=\CodeIndent
     3224   \else
     3225     \leftskip=\TextIndent
     3226     \hangindent=\z@
     3227     \everypar=\@xa{%
     3228       \@xa\@codetonarrskip\the\gmd@preverypar}%
     3230   \fi}
```

Now a special treatment for an in-line comment:

```
\ifilrr 3234 \newif\ifilrr
```

```
\ilrr 3236 \def\ilrr{%
     3237   \if@aftercode
     3238     \unless\if@ilgroup\bgroup\@ilgrouptrue\fi% If we are 'aftercode', then we are in an in-line comment. Then we open a group to be able to declare e.g. \raggedright for that comment only. This group is closed in line 3195 or 2948.
     3243     \ilrrtrue
     3244   \fi}
```

```
\if@ilgroup 3246 \newif\if@ilgroup
\gmd@setilrr 3248 \def\gmd@setilrr{\rightskipoptplus\textwidth}
\ilju 3250 \def\ilju{% when in-line comments are ragged right in general but we want just
          this one to be justified.
3252   \if@aftercode
3253     \unless\if@ilgroup\bgroup\@ilgrouptrue\fi
3254     \ilrrfalse
3255   \fi}
\verbcodecorr 3257 \def\verbcodecorr{% a correction of vertical spaces between a verbatim and
          code. We put also a \par to allow parindent in the next commentary.
3261   \vskip-\lastskip\vskip-4\CodeTopsep\vskip3\CodeTopsep\par}
```

Numbering (or not) of the lines

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

```
3269 \if@uresetlinecount% with uresetlinecount option…
3270   \@relaxen\gmd@resetlinecount% … we turn resetting the counter by \Doc¦
          % Input off…
\resetlinecountwith 3272   \newcommand*\resetlinecountwith[1]{%
codelinenum 3273     \newcounter{codelinenum}[#1]}% … and provide a new declaration of
          the counter.
3275 \else% With the option turned off…
DocInputsCount 3276   \newcounter{DocInputsCount}%
codelinenum 3277   \newcounter{codelinenum}[DocInputsCount]% … we declare the \DocIn¦
          puts′ number counter and the codeline counter to be reset with stepping
          of it.
\gmd@resetlinecount 3283   \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.
3287 \fi
```

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

```
\printlinenumber 3291 \newcommand*\printlinenumber{%
3292   \leavevmode\llap{\rlap{\LineNumFont$\phantom{999}$\llap{%
          \thecodelinenum}}%
3293     \hskip\leftskip}}
\LineNumFont 3295 \def\LineNumFont{\normalfont\tiny}
3297 \if@linesnotnum\@relaxen\printlinenumber\fi
\hyperlabel@line 3299 \newcommand*\hyperlabel@line{%
3300   \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.
3303   \else
3304     \raisebox{2ex}[1ex][\z@]{\gmhypertarget[clnum.%
3305       \HLPrefix\arabic{codelinenum}]{}}%
3306   \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\@nostanzagfalse}
```

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

@codeskipput

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

A switch to control \nostanzas:
```
\newgif\if@nostanza
```

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

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

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

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

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

\if@aftercode
```
\newgif\if@aftercode
```

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

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

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

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

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

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

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

```
            \gmd@codeskip4%
          \else\gmd@nocodeskip4{naC}%
          \fi
        \fi
    \fi
    \leftskip\TextIndent% this line is a patch against a bug-or-feature that in
        certain cases the narration \leftskip is left equal the code leftskip. (It
        happens when there are subsequent code lines after an in-line comment
        not ended with an explicit \par.) Before v0.99n it was just after line 3371.
    \@aftercodegfalse\@nostanzagtrue
}
```

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

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

```
\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 `\if`s and we even define an auxiliary macro because `\everypar` may contain some `\if`s or `\fi`s.

Life among queer EOLs

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

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

```
\catcode`\^^B=\active% note we re\catcode ⟨char2⟩ globally, for the entire
    document.
\foone{\obeylines}%
  {\def\QueerCharTwo{%
      \protected\def^^B##1^^M{%
```

`^^B` `\QueerCharTwo`

```
3434            \ifhmode\unskip\space\ignorespaces\fi}}% It shouldn't be \ not
                    to drive TeX into hmode.
3436    }
3438 \QueerCharTwo
3440 \AtBegInput{\@ifEOLactive{\catcode`\^^B\active}{}\QueerCharTwo}%
                We repeat redefinition of ⟨char2⟩ at begin of the documenting input, be-
                cause doc.dtx suggests that some packages (namely inputenc) may re\cat¦
                code 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.

```
3462 \catcode`\^^A=\active% note we re\catcode ⟨char1⟩ globally, for the entire
                document.
3464 \foone\obeylines{%
^^A 3465   \def\QueerCharOne{%
\QueerCharOne 3466     \def^^A{%
3468       \bgroup\let\do\@makeother\dospecials%
                \gmd@gobbleuntilM}}%
\gmd@gobbleuntilM 3469   \def\gmd@gobbleuntilM#1^^M{\egroup\ignorespaces^^M}%
3470 }
3472 \QueerCharOne
3474 \AtBegInput{\@ifEOLactive{\catcode`\^^A%
                \active}\QueerCharOne}% see note after line
                3440.
```

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

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

```
\StraightEOL 3490 \def\StraightEOL{%
3491   \catcode`\^^M=5
3492   \catcode`\^^A=14
3493   \catcode`\^^B=14
3494   \def\^^M{\␣}}
3502 \foone\obeylines{%
\QueerEOL 3503   \def\QueerEOL{%
3504     \catcode`\^^M=\active%
3505     \let^^M\gmd@textEOL%
3506     \catcode`\^^A=\active%
3507     \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' line end I command it to add a ␣10 at first. It works but has one unwelcome feature: if the line has nearly `\textwidth`, this closing space may cause line breaking and setting a blank line. To fix this I `\advance` the `\parfillskip`:

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

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 2884 to see what we do in the codeline case when no closing space is added.

And one more detail:

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

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

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

Adjustments of `verbatim` and `\verb`

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

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

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

\check@percent
```
\AtBegInput{\long\def\check@percent#1{%
```

```
3593        \gmd@cpnarrline% to count the verbatim lines and possibly print their
                numbers. This macro is used only by the verbatim end of line.
3595        \@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
```
3598 \def\AddtoPrivateOthers#1{%
3599   \@xa\def\@xa\doprivateothers\@xa{%
3600     \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.

```
3611 \begingroup
3612 \obeylines%
```
\gm@verb@eol \verb@egroup
```
3613 \AtBegInput{\def\gm@verb@eol{\obeylines%
3614      \def^^M{\verb@egroup\@latex@error{%
3615          \@nx\verb␣ended␣by␣end␣of␣line}%
3616        \@ifEOLactive{^^M}{\@ehc}}}}%
3617 \endgroup
```

To distinguish the code typewriter from the narrative typewriter:

```
3620 \ampulexdef\@verbatim\ttverbatim\verbatim@specials
3621 {\ttverbatim\narrativett\verbatim@specials}

3623 \ampulexdef\verb\ttverbatim\verbatim@specials
3624 {\ttverbatim\narrativett\verbatim@specials}%
```

\texttt
```
3626 \pdef\texttt#1{{\narrativett#1}}
```

We rollback the `\ampulexdef` made to `\verb` in the index, see line 5895

Macros for marking of the macros

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

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

\actualchar \quotechar \encapchar \levelchar
```
3653 \edef\actualchar{\string␣@}
3654 \edef\quotechar{\string␣"}
3655 \edef\encapchar{\xiiclub}
3656 \edef\levelchar{\string␣!}
```

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

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

\CodeEscapeChar
```
3668 \def\CodeEscapeChar#1{%
```

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

\code@escape@char

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.

```
\CodeEscapeChar\\
```

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

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

\MakePrivateLetters

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

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

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

\MakePrivateOthers

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 5233. So let's define the list:

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

\doprivateothers

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

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

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

\scan@macro

```
\def\scan@macro@#1{%
```

\scan@macro@ we are sure to scan at least one token which is not the line end and therefore we define this macro as one-parameter.

Unlike in doc, here we have the escape char $_{12}$ so we may just have it printed during main scan char by char, i.e., in the lines 3087 and 3091.

So, we step the checksum counter first,

```
\step@checksum%
```

(see line 6489 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 3751.

```
3735      \xdef\macro@pname{\string#1}%
```
we'll print entire name of the macro later.

We `\string` it here and in the lines 3755 and 3767 to be sure it is whole $_{12}$ for easy testing for special index entry formats, see line 4655 etc. Here we are sure the result of `\string` is $_{12}$ since its argument is $_{11}$.

```
3742      \afterfi{\@ifnextcat{a}{\gmd@finishifstar#1}{%
              \finish@macroscan}}%
3743    \else% #1 is not a letter, so we have just scanned a one-char CS.
```

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

```
3750      {\uccode`9=`#1%
3751        \uppercase{\xdef\macro@iname{9}}%
3752      }%
3753      \quote@char#1%
3754      \xdef\macro@iname{\gmd@maybequote\macro@iname}%
3755      \xdef\macro@pname{\xiistring#1}%
3756      \afterfi␣\finish@macroscan
3757    \fi}%
```
of `\scan@macro@`. The `\xiistring` macro, provided by gmutils, is used instead of original `\string` because we wish to get $␣_{12}$('other' space).

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

`\continue@macroscan`

```
3764 \def\continue@macroscan#1{%
3765   \quote@char#1%
3766   \xdef\macro@iname{\macro@iname␣\gmd@maybequote#1}%
3767   \xdef\macro@pname{\macro@pname␣\string#1}%
```
we know `#1` to be $_{11}$, so we don't need `\xiistring`.

```
3770   \@ifnextcat{a}{\gmd@finishifstar#1}{\finish@macroscan}%
3771 }
```

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`

```
3780 \def\gmd@finishifstar#1{%
3781   \if*\@nx#1\afterfi\finish@macroscan%
```
note we protect `#1` against expansion. In gmdoc verbatim scopes some chars are active (e.g. `\`).

```
3784   \else\afterfi\continue@macroscan
3785   \fi}
```

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

`\quote@char`

```
3789 \def\quote@char#1{{\uccode`9=`#1%
```
at first I took digit 1 for this `\uc¦ode`ing but then `#1` meant `#`⟨*#1*⟩ in `\uppercase`'s argument, of course.

```
3792     \uppercase{%
3793       \@ifinmeaning␣9\of␣\indexcontrols
3794       {\glet\gmd@maybequote\quotechar}%
3795       {\g@emptify\gmd@maybequote}%
3796     }%
```

```
}}
```

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.

In `\quote@char` the second argument for gmutils `\@ifinmeaning` is `\index¦controls` defined as the (expanded and 'other') sequence of the MakeIndex controls. `\@ifinmeaning` defines its inner macro `\gmd@in@@` to take two parameters separated by the first and the second `\@ifinmeaning`'s parameter, which are here the char investigated by `\quote@char` and the `\indexcontrols` list. The inner macro's parameter string is delimited by the macro itself, why not. `\gmd@in@@` is put before a string consisting of `\@ifinmeaning`'s second and first parameters (in such a reversed order) and `\gmd@in@@` itself. In such a sequence it looks for something fitting its parameter pattern. `\gmd@in@@` is sure to find the parameters delimiter (`\gmd@in@@` itself) and the separator, `\ifismember`'s `#1` i.e., the investigated char, because they are just there. But the investigated char may be found not near the end, where we put it, but among the MakeIndex controls' list. Then the rest of this list and `\ifismember`'s `#1` put by us become the second argument of `\gmd@in@@`. What `\gmd@in@@` does with its arguments, is just a check whether the second one is empty. This may happen *iff* the investigated char hasn't been found among the MakeIndex controls' list and then `\gmd@in@@` shall expand to `\iffalse`, otherwise it'll expand to `\iftrue`. (The `\after`... macros are employed not to (mis)match just got `\if`... with the test's `\fi`.) "(Deep breath.) You got that?" If not, try doc's explanation of `\ifnot@excluded`, pp. 36–37 of the v2.1b dated 2004/02/09 documentation, where a similar construction is attributed to Michael Spivak.

Since version 0.99g `\@ifinmeaning` is used also in testing whether a detector is already present in the carrier in the mechanism of automatic detection of definitions (line 4002).

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}}
```

`\ifgmd@glosscs`

```
\newif\ifgmd@glosscs%
```

we use this switch to keep the information whether a history entry is a CS or not.

`\finish@macroscan`

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

First we check if the current CS is not just being defined. The switch may be set true in line 3899

```
  \ifgmd@adef@cshook%
```

if so, we throw it into marginpar and index as a def entry…

```
    \gm@ifundefined{gmd/iexcl/\macro@pname\space}{%
```

… if it's not excluded from indexing.

```
      \@xa\Code@MarginizeMacro\@xa{\macro@pname}%
      \@xa\@defentryze\@xa{\macro@pname}{1}}{}%
```

here we declare the kind of index entry and define `\last@defmark` used by `\changes`

```
      \global\gmd@adef@cshookfalse%
```
we falsify the hook that was set true just for this CS.
```
    \fi
```

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

```
    \if\verbatimchar\macro@pname%
```
it's important that `\verbatimchar` comes before the macro's name: when it was reverse, the `\tt` CS turned this test true and left the `\verbatimchar` what resulted with '`\+tt`' typeset. Note that this test should turn true iff the scanned macro name shows to be the default `\verb`'s delimiter. In such a case we give `\verb` another delimiter, namely `$`:

`\im@firstpar`
```
      \def\im@firstpar{[$%
        ]}%
```
`\im@firstpar`
```
    \else\def\im@firstpar{}%
    \fi
    \@xa␣\index@macro\im@firstpar\macro@iname\macro@pname
    \maybe@marginpar\macro@pname
    \if\xiispace\macro@pname\relax\gmd@texcodespace
    \else
      {\noverbatimspecials\RestoreMacro\verb
        \@xa\scanverb\@xa{\macro@pname}}%
```
we typeset scanned CS.
```
    \fi
    \let\next\gmd@charbychar
    \gmd@detectors%
```
for automatic detection of definitions. Defined and explained in the next section. It redefines `\next` if detects a definition command and thus sets the switch of line 3860 true.
```
    \next
}
```

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

`\maybe@marginpar`
```
\def\maybe@marginpar#1{%
  \gm@ifundefined{gmd/2marpar/\@xa\detokenize\@xa{#1}}{}{%
    \edef\gmu@tempa{%
      \unexpanded{\Text@Marginize*}%
      {\bslash\@xa\unexpanded\@xa{#1}}%
    }\gmu@tempa
```
`\macro@pname`, which will be the only possible argument to `\maybe@marg¦ % inpar`, contains the macro's name without the escape char so we added it here.
```
    \@xa\g@relaxen
    \csname␣gmd/2marpar/\@xa\detokenize\@xa{#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 CSes are implemented in the section after it.

Automatic detection of definitions

To begin with, let's introduce a general declaration of a defining command. `\Declare¦Defining` comes in two flavours: 'sauté', and with star. The 'sauté' version without an optional argument declares a defining command of the kind of `\def` and `\newcom¦mand`: 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 `\newen¦vironment` and `\DeclareOption`: whose main mandatory argument is text. Both versions provide an optional argument in which you can set the keys. Probably the most important key is `star`. It determines whether the starred version of a defining command should be taken into account. For example, `\newcommand` should be declared with `[star=true]` while `\def` with `[star=false]`. You can also write just `[star]` instead of `[star=true]`. It's the default if the `star` key is omitted.

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

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

`\DeclareDefining` and the detectors

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

`\DeclareDefining`

```
\outer\def\DeclareDefining{\begingroup
  \MakePrivateLetters
  \gm@ifstar
    {\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 switch 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\@ifinmeaning
```

```
      \csname␣gmd@detect@\gmd@adef@currdef\endcsname
      \of\gmd@detectors{}{%
        \@xa\gaddtomacro\@xa\gmd@detectors\@xa{%
          \csname␣gmd@detect@\gmd@adef@currdef\endcsname}}% we add a CS
                % \gmd@detect@⟨def name⟩ (a detector) to the meaning of the
                detectors' carrier. And we define it to detect the #2 command.
    \@xa\xdef\csname␣gmd@detectname@\gmd@adef@currdef%
        \endcsname{%
      \gmd@adef@currdef}%
    \edef\gmu@tempa{% this \edef is to expand \gmd@adef@TYPE.
      \global\@nx\@namedef{gmd@detect@\gmd@adef@currdef}{%
        \@nx\ifx
          \@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}}%
    \gmu@tempa
    \SMglobal\StoreMacro*{gmd@detect@\gmd@adef@currdef}% we store the
        CS to allow its temporary discarding later.
}
```

md@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 4200.

star

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

The `prefix@`⟨*command*⟩ key-value will be used to create additional index entry for detected definiendum (a **definiendum** is the thing defined, e.g. in `\newenvironment{%foo}` the env. `foo`). For instance, `\newcounter` is declared with `[prefix=\bslash␣c@]` in line 4463 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 KVprefix 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 key-
              val key whose macro's name begins with the prefix/family, either de-
              fault 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 initialise the carrier of detectors:

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

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

`\ifgmd@adef@cshook`

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

`\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 sergeants. In gmdoc code layer scopes the left brace is active so we only add a hook to its meaning (see line 371 in gmverb) and 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 CSes 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 sergeants. 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 initialise the hook caser.

And we define the inner left brace macros:

```
\foone{\catcode`\[1␣\catcode`\]2␣\catcode`\}12␣}
  [% Note that till line 4330 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{%
  \@xa\def\@xa\macro@pname\@xa{\@gobble#1}% we gobble \gmd@charbychar, cf. above.
  \edef\macro@pname{\@xa\detokenize\@xa{\macro@pname}␣}% note the space at the end.
  \edef\macro@pname{\@xa\@xiispaces\macro@pname\@@nil}%
  \@xa\Text@Marginize\@xa{\macro@pname}%
  \gmd@adef@indextext
  \edef\gmd@adef@altindex{%
```

```
      \csname␣gmd@adef@prefix@\gmd@adef@currdef␣\endcsname}%
```

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

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

\gmd@adef@indextext

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

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

Default defining commands

Some commands are easy to declare as defining:

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

\pdef

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

\provide

```
\DeclareDefining[star=false]\provide% a gmutils' conditional \def.
```

\pprovide

```
\DeclareDefining[star=false]\pprovide% a gmutils' conditional \pdef.
```

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

\UnDef

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

\UnPdef

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

```
\StoreMacro\UnDef% because the 'hiding' commands relax it.
```

\HideDef \relaxen

```
\def\HideDef{%
  \gm@ifstar\UnDef{\HideDefining\def\relaxen\UnDef}}
```

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

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

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

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

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

```
%\DeclareDefining*\@namedef
```

\newcounter
```
\DeclareDefining*[prefix=\bslash␣c@]\newcounter%
```
this prefix provides indexing also `\c@`⟨*counter*⟩.

\define@key
\define@boolkey
\define@choicekey
```
\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
```

\DeclareOptionX
```
\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`.

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

An example:

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

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

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

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

Now the Heiko Oberdiek's kvoptions package option definitions:

```
\DeclareStringOption
\DeclareBoolOption
areComplementaryOption
\DeclareVoidOption
```

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

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

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

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

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

Suspending ('hiding') and resuming detection

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

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

```
\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 occurred between `\HideAllDefining` and itself.

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

`\HideDefining`

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

`\Hide@DfngOnce`

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

\def\gmd@adef@selfrestore#1{%
  \escapechar\m@ne
  \@ifundefined{gmd@detect@\string#1}{%
    \SMglobal\@xa\StoreMacro
    \csname␣gmd@detect@\string#1\endcsname}{}%
  \global\@nameedef{gmd@detect@\string#1}{%
    \@nx\ifx\@xa\@nx\csname␣gmd@detectname@\string#1%
        \endcsname
    \@nx\macro@pname
    \def\@nx\next{% this \next will be executed in line 3904.
      \SMglobal\RestoreMacro␣% they both are \protected.
      \@xa\@nx\csname␣gmd@detect@\string#1\endcsname
      \@nx\gmd@charbychar}%
    \@nx\fi}% of \@nameedef.
}% of \gmd@adef@selfrestore.
```

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

`\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 CSes

The inner macro indexing macro. `#1` is the `\verb`'s delimiter; `#2` is assumed to be the macro's name with MakeIndex-control chars quoted. `#3` is a macro storing the macro's name, usually `\macro@pname`, built with `\string`ing every char in lines 3735, 3755 and 3767. `#3` is used only to test if the entry should be specially formatted.

```
\index@macro 4651 \newcommand*\index@macro[3][\verbatimchar]{%
4652       \gm@ifundefined{gmd/iexcl/\@xa\detokenize\@xa{#3␣}}%
4653       {% #3 is not excluded from index
4655         \gm@ifundefined{gmd/defentry/\@xa\detokenize\@xa{#3␣}}%
4656         {% #3 is not def entry
4657           \gm@ifundefined{gmd/usgentry/\@xa\detokenize\@xa{#3␣}}%
4658           {% #3 is not usg. entry
4659             \edef\kind@fentry{\CommonEntryCmd}}%
4660           {% #3 is usg. entry
\kind@fentry 4661             \def\kind@fentry{UsgEntry}%
4662             \un@usgentryze{#3}}%
4663         }%
4664         {% #3 is def entry
\kind@fentry 4665           \def\kind@fentry{DefEntry}%
4666           \un@defentryze{#3}%
4667         }% of gmd/defentry/ test's 'else'
4668         \if@pageindex\@pageinclindexfalse\fi% should it be here or there?
                 Definitely here because we'll wish to switch the switch with a declaration.
4671         \if@pageinclindex
4672           \edef\gmu@tempa{gmdindexpagecs{\HLPrefix}{%
                   \kind@fentry}{\EntryPrefix}}%
4673         \else
4674           \edef\gmu@tempa{gmdindexrefcs{\HLPrefix}{%
                   \kind@fentry}{\EntryPrefix}}%
4675         \fi
4676         \edef\gmu@tempa{\IndexPrefix#2\actualchar%
4677           \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).
4681           \encapchar\gmu@tempa}%
4682         \@xa\special@index\@xa{\gmu@tempa}% We give the indexing macro
                 the argument expanded so that hyperref may see the explicit encapchar in order not to add its own encapsulation of |hyperpage when the (default) hyperindex=true option is in force. (After this setting the \edefs in the above may be changed to \defs.)
4694       }{}% closing of gmd/iexcl/ test.
4695       }}

\un@defentryze 4699 \def\un@defentryze#1{%
4700   \ifcsname␣gmd/defentry/\@xa\detokenize\@xa{#1␣}\endcsname
4701   \@xa\g@relaxen\csname␣gmd/defentry/\@xa\detokenize\@xa{#1␣
           }\endcsname
4702   \fi
4703   \ifx\gmd@detectors\empty
4704     \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{%
  \ifcsname␣gmd/usgentry/\@xa\detokenize\@xa{#1␣}\endcsname
  \@xa\g@relaxen\csname␣gmd/usgentry/\@xa\detokenize\@xa{#1␣
      }\endcsname
  \fi}
\@emptify\EntryPrefix%
```
this macro seems to be obsolete now (v0.98d).

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

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

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

Let's initialise `\IndexPrefix`

`\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\gmu@tempa{#1}%
      \ifx\gmu@tempa\@empty␣p.\,\fi}
```
`\HLPrefix`
```
    \providecommand*\HLPrefix{}%
```
it'll be the hypertargets names' prefix in multi-docs. Moreover, it showed that if it was empty, hyperref saw duplicates of the hyper destinations, which was perfectly understandable (`codelinenum.123` made by `\refstepcounter` and `codelinenum.123` made by `\gmhypertarget`). But since v0.98 it is not a problem anymore because during the automatic `\hypertar¦get`ing the lines are labelled `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 7964.

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

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

\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 4659 it's `\edef`ed to allow an 'unmöglich' situation that the user wants to have the common index entries specially formatted. I use this to make *all* the index entries of the driver part to be 'usage', see the source of chapter 1928.

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 `\ifx`ed in lines 4655 and 4657 respectively.

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

\DefIndex
```
\outer\def\DefIndex{\begingroup
  \MakePrivateLetters
  \gm@ifstar
  {\@sanitize\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/\detokenize{#1␣}\endcsname%
      \gmd@justadot% The
      LaTeX \@namedef macro could not be used since it's not 'long'. The space
      to sound with the checker.
  \ifcat\relax\@xa\@nx\@firstofmany#1\@@nil
```
if we meet a CS, then maybe it's a CS to be 'defentryzed' or maybe it's a 'verbatim special' CS. The only way to distinguish those cases is to assume there shouldn't be a verbatim containing only a 'verbatim special' CS.
```
    \@xa\def\@xa\gmu@tempa\@xa{\@allbutfirstof#1\@@nil}%
```

```
      \ifx\gmu@tempa\@empty
        \afterfifi\@firstoftwo% if #1 is a single CS, we \xiistring it.
             Otherwise we \detokenize it.
      \else\afterfifi\@secondoftwo
      \fi
    \else\@xa\@secondoftwo
    \fi
\last@defmark {\xdef\last@defmark{\xiistring#1}}% we \string the argument just
         in case it's a control sequence. But when it can be a CS, we \@defentryze
         in a scope of \escapechar=-1, so there will never be a backslash at the
         beginning of \last@defmark's meaning (unless we \@defentryze \\).
    {\xdef\last@defmark{\detokenize{#1}}}%
    \@xa\gdef\csname␣gmd/isaCS/\last@defmark\endcsname{#2}% #2 is
         either 0 or 1. It is the information whether this entry is a CS or not.
}% of \@defentryze.

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

Initialise `\envirs@toindex`

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

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

```
\CodeUsgIndex \outer\def\CodeUsgIndex{\begingroup
    \MakePrivateLetters
    \gm@ifstar
    {\@sanitize\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
    \gm@ifstar
    {\MakePrivateOthers\@sanitize\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
    \def\im@firstpar{[$]}%
  \else\def\im@firstpar{}%
  \fi
  {\do@properindex% see line 5332.
    \@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{%
  \ifx\relax#1% finish quoting when you meet \relax or:
  \else
    \ifnumo\ifcat\@nx#1\@nx~1\fi\ifcat\@nx#1\relax1\fi>0␣% we
        can meet active char and/or control sequences (made by) verbatim specials, therefore we check whether #1 is an active char and if it is a CS.
      \afterfifi{% we can meet an active char or a CS iff we use verbatim
          specials.
        \ifdefined\verbatim@specials@list
        \afterfi{%
          \begingroup
          \escapechar\@xa\@xa\@xa`\@xa\@firstofmany%
              \verbatim@specials@list\@@nil
          \@xa\endgroup
          \@xa\quote@charbychar\detokenize{#1}% for a CS \detokenize adds a space but if so, it will be ignored by the argument scanner.
        }% of \afterfi.
        \else\PackageError{gmdoc}{Please␣report␣a\space␣bug␣in
          \bslash␣quote@charbychar␣in␣line␣4934}%
        \fi% of \ifdefined\verbatim@specials@list.
      }% of \afterfifi.
    \else
      \quote@char#1%
      \xdef\macro@iname{\macro@iname␣\gmd@maybequote#1}%
      \afterfifi\quote@charbychar
    \fi
  \fi}
```

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

`\TextUsgIndex`
```
\def\TextUsgIndex{\begingroup
  \MakePrivateLetters
  \gm@ifstar{\MakePrivateOthers\Text@UsgIndexStar}{%
      \Text@UsgIndex}}
```

`\Text@UsgIndex`
```
\long\def\Text@UsgIndex#1{%
  \endgroup\@usgentryze#1%
  \text@indexmacro#1}
```

`\Text@UsgIndexStar`
```
\long\def\Text@UsgIndexStar#1{\endgroup\@usgentryze{#1}%
  \text@indexenvir{#1}}
```

`\text@indexenvir`
```
\long\def\text@indexenvir#1{%
  {\verbatim@specials
  \edef\macro@pname{\xiistring#1}%
  \if\bslash\@xa\@firstofmany\macro@pname\@@nil%
```
if `\stringed #1` begins with a backslash, we will gobble it to make MakeIndex not see it.
```
    \edef\gmu@tempa{\@xa\@gobble\macro@pname}%
    \@tempswatrue
  \else
    \let\gmu@tempa\macro@pname
    \@tempswafalse
  \fi
  \@xa\quote@mname\gmu@tempa\relax%
```
we process `\stringed #1` char by char and quote MakeIndex controls. `\relax` is the iterating macro's stopper. The quoted `\stringed #1` shall be the meaning of `\macro@iname`.
```
  \if@tempswa
```
`\quoted@eschar`
```
    \def\quoted@eschar{\quotechar\bslash}%
    \else\@emptify\quoted@eschar\fi%
```
we won't print any backslash before an environment's name, but we will before a CS's name.
```
    \do@properindex%
```
see line 5332.
```
    \index@macro\macro@iname\macro@pname}}
```

`\TextCommonIndex`
```
\def\TextCommonIndex{\begingroup
  \MakePrivateLetters
  \gm@ifstar{\MakePrivateOthers\Text@CommonIndexStar}{%
      \Text@CommonIndex}}
```

`\Text@CommonIndex`
```
\long\def\Text@CommonIndex#1{\endgroup
  \text@indexmacro#1}
```

`\Text@CommonIndexStar`
```
\long\def\Text@CommonIndexStar#1{\endgroup
  \text@indexenvir{#1}}
```

As you see in the lines 4666 and 4662, the markers of special formatting are reset after first use.

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

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

```
    {\egCode@MarginizeMacro}}
```

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

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

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

Here it is:

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

`\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 4381.
    \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 4982 & subs. for commentary.
    {\if@tempswa
        \def\quoted@eschar{\quotechar\bslash}%
        \else\@emptify\quoted@eschar
      \fi
      \index@macro\macro@iname{##1}}}%
```

`\quoted@eschar`

```
}
```

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{% note that since it's \outer, it doesn't have to be \protected.
  \begingroup
  \MakePrivateLetters
```

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

```
  \gm@ifstar{\@sanitize%
    \Code@DefEnvir}{\Code@DefMacro}}
```

`\CodeUsage`

```
\outer\def\CodeUsage{\begingroup
  \MakePrivateLetters
  \gm@ifstar{%
    \@sanitize%
    \MakePrivateOthers
    \Code@UsgEnvir}{\Code@UsgMacro}}
```

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

`\Code@DefMacro`

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

`\Code@UsgMacro`

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

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

`\codeline@wrindex`

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

`\codeline@glossary`

```
\def\codeline@glossary#1{% It doesn't need to establish a group since it is always called in a group.
  \if@pageinclindex
    \edef\gmu@tempa{gmdindexpagecs{\HLPrefix}{relax}{%
        \EntryPrefix}}%
  \else
```

```
      \edef\gmu@tempa{gmdindexrefcs{\HLPrefix}{relax}{%
           \EntryPrefix}}%
  \fi
  \protected@edef\gmu@tempa{%
    \@nx\protected@write\@nx\@glossaryfile{}%
    {\string\glossaryentry{#1\encapchar\gmu@tempa}%
    {\HLPrefix\number\c@codelinenum}}}%
  \gmu@tempa
}
```

`relax` stands for the formatting command. But we don't want to do anything special with the change history entries.

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

```
\AtBeginDocument{%
  \if@pageindex
    \let\special@index=\index
    \let\gmd@glossary\glossary
  \else
    \let\special@index=\codeline@wrindex
    \let\gmd@glossary\codeline@glossary
  \fi}%
```

postponed till `\begin{document}` with respect of doc-like declarations.

And in case we don't want to index:

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

`\Code@DefEnvir`

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

`\Code@UsgEnvir`

```
\DeclareCommand\Code@UsgEnvir\long{om}{%
  \endgroup
  {%
    \IfValueTF{#1}%
    {\addto@estomarginpar{#1}}%
    {\addto@estomarginpar{#2}}%
```

```
      \addto@estoindex{#2}%
      \@usgentryze{#2}}}
\addto@estomarginpar
\long\def\addto@estomarginpar#1{%
\gaddtomacro\envirs@tomarginpar{\do{#1}}}
\addto@estoindex
\long\def\addto@estoindex#1{%
  \gaddtomacro\envirs@toindex{\do{#1}}}
```

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

```
\TextUsage
\def\TextUsage{\begingroup
  \MakePrivateLetters
  \gm@ifstar{\@sanitize\MakePrivateOthers
    \Text@UsgEnvir}{\Text@UsgMacro}}
\Text@UsgMacro
\DeclareCommand\Text@UsgMacro\long{om}{%
  \endgroup
  \IfValueTF{#1}%
  {\Text@Marginize*{#1}{\scanverb*{#1}}}%
  {\Text@Marginize*{#2}{\scanverb*{#2}}}%
  \begingroup\Code@UsgIndex#2% we declare the kind of formatting of the
       entry.
  \text@indexmacro#2}
\Text@UsgEnvir
\DeclareCommand\Text@UsgEnvir\long{om}{%
  \endgroup
  \IfValueTF{#1}%
  {\Text@Marginize*{#1}{\scanverb*{#1}}}%
  {\Text@Marginize*{#2}{\scanverb*{#2}}}%
  \@usgentryze{#2}% we declare the 'usage' kind of formatting of the entry and
       index the sequence #1.
  \text@indexenvir{#2}}
```

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

```
\TextMarginize
\pdef\TextMarginize{\@bsphack\begingroup
  \MakePrivateLetters
  \gm@ifstar{%
    \MakePrivateOthers\egText@MarginizeEnv}{%
         \egText@MarginizeCS}}
\egText@MarginizeEnv
\long\def\egText@MarginizeEnv#1{\endgroup
  \Text@Marginize*{#1}%
  \@esphack}
\egText@MarginizeCS
\long\def\egText@MarginizeCS#1{%
  \endgroup
  \Text@Marginize*{#1}%
}
```

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

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

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

\gmdmarginpar
```
  \long\def\gmdmarginpar#1{%
    \marginpar{\raggedleft\strut
      \hskipoptplus100optminus100opt%
      #1}}%
\else
```
\gmdmarginpar
```
  \long\def\gmdmarginpar#1{}%
\fi
```

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

Note that the above command will just gobble its arguments if the marginpars are disabled.

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

```
\let\marginpartt\narrativett
```

If we print also the narration lines' numbers, then the index entries for CSes and environments marked in the commentary should have codeline numbers not page numbers and that is `\let` in line 5163. On the other hand, if we don't print narration lines' numbers, then a macro or an environment marked in the commentary should have page number not codeline number. This we declare here, among others we add the letter `p` before the page number.

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

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

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

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

```
    \QueerEOL}%
    {\par\addvspace\CodeTopsep\CodeEscapeChar\\}
```

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

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

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

Note that at the end of both the above environments the \'s rôle as the code escape char is restored. This is crafted for the `\SpecialEscapechar` macro's compatibility: this macro influences only the first `macrocode` environment. The situation that the user wants some queer escape char in general and in a particular `macrocode` yet another seems to me "unmöglich, Prinzessin"[10].

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

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

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

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

\@oldmacrocode@launch
```
\def\@oldmacrocode@launch{%
    \emptify\gmd@textEOL% to disable it in \gmd@docstripdirective launched
        within the code.
    \gmd@ctallsetup
    \glet\stored@code@delim\code@delim
    \@makeother\^^B\CodeDelim*\^^B%
    \ttverbatim␣\gmd@DoTeXCodeSpace
    \@makeother\|% because \ttverbatim doesn't do that.
    \MakePrivateLetters% see line 3681.
    \docstrips@percent␣\@makeother\>%
```

sine qua non of the automatic delimiting is replacing possible $\star_{12}$in the environment's name with $\star_{11}$. Not to complicate assume $\star$ 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 $\star$) will be the same in the verbatim text.

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

\foone{\catcode`*11␣}
```
\gm@xistar
```
{\def\gm@xistar{*}}
```
\gmd@currenvxistar
```
\def\gmd@currenvxistar#1*#2\relax{%
```

10 Richard Strauss after Oscar Wilde, *Salome*.

```
\edef\@currenvir{#1\if*#2\gm@xistar\fi}}
```

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

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

`\@oldmacrocode`

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

```
\def\gmd@oldmcfinis{%
  \def\gmu@tempa{\end{\@currenvir}}%
   \@xa\gmu@tempa\@xa\def\@xa\gmd@lastenvir\@xa{%
        \@currenvir}%
  \@xa\CodeDelim\@xa*\stored@code@delim
  \gmd@mchook}%  see line 7600
```

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

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

```
\foone{\catcode`\%\active}
{\def\docstrips@percent{\catcode`\%\active
    \let%\gmd@codecheckifds}}
```

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

`\gmd@codecheckifds`

```
\def\gmd@codecheckifds#1#2{%  note that #1 is just to gobble \gmd@charbychar
      token.
  \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 'private others' to cover the case of other sequence in the argument. (We'll use the `\macro` macro also in the environment for describing and defining environments.)
```
    \gmd@ifonetoken\Hybrid@DefMacro\Hybrid@DefEnvir}%
```

endmacro
```
    {\par\addvspace\MacroTopsep\@codeskipputgtrue}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

environment

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

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

Index exclude list

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

doc takes \index@excludelist to be a \toks register to store the list of expelled CSes. Here we'll deal another way. For each CS to be excluded we'll make (\let, to be precise) a control sequence and then we'll be checking if it's undefined (\ifx-equivalent to \relax).[11]

\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\gmu@tempa{\@nx#1}% My TEX Guru's trick to deal with \fi and such, i.e., to hide from TEX when it is processing a test's branch without expanding.
  \if\gmu@tempa.% a dot finishes expelling
  \else
    \if\gmu@tempa,% The list this macro is put before may contain commas and that's O.K., we just continue the work.
      \afterfifi\dont@index
    \else% what is else shall off the Index be expelled.
      {\escapechar\m@ne
        \xdef\gmu@tempa{\string#1␣}}% its to sound with \detokenizes in tests.
      \@xa\let%
      \csname␣gmd/iexcl/\gmu@tempa\endcsname=\gmd@iedir% In the default case explained e.g. by the macro's name, the last macro's meaning is such that the test in line 4652 will turn false and the subject CS shall not be indexed. We \let not \def to spare TEX's memory.
      \afterfifi\dont@index
    \fi
  \fi}
```

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

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

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

\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}%
```

11 This idea comes from Marcin Woliński.

```
\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 5673.
\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 `\AtBeginDoc¦ument`, `\AtEndDocument`, `\AtEndOfPackage`, `\DeclareOption`, `\DeclareRo¦bustCommand` etc. But the common definitions, such as `\(new|provide)command` and `\(e|g|x)def`s, as the most common, in my opinion excluded should be.

And some my exclusions:

```
\DoNotIndex{\@@input \@auxout \@currentlabel \@dblarg}%
\DoNotIndex{\@ifdefinable \@ifnextchar \@ifpackageloaded}%
\DoNotIndex{\@indexfile \@let@token \@sptoken \^}%
```
the latter comes from CSes like `\^^M`, see sec. 8228.

```
\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 in-
        dexed IMHO.
    \DoNotIndex{\@outerparskip \@partaux \@partlist \@plus}%
    \DoNotIndex{\@sverb \@sxverbatim}%
    \DoNotIndex{\@tempcnta \@tempcntb \@tempskipa \@tempskipb}%
        I think the layout parameters even the kernel, should not be excluded:
        % \@topsep \@topsepadd \abovedisplayskip \clubpenalty etc.
    \DoNotIndex{\@writeckpt}%
    \DoNotIndex{\bfseries \chapter \part \section \subsection}%
    \DoNotIndex{\subsubsection}%
    \DoNotIndex{\char \check@mathfonts \closeout}%
    \DoNotIndex{\fontsize \footnotemark \footnotetext
        \footnotesize}%
    \DoNotIndex{\g@addto@macro \hfilneg \Huge \huge}%
    \DoNotIndex{\hyphenchar \if@partsw \IfFileExists }%
    \DoNotIndex{\include \includeonly \indexspace}%
    \DoNotIndex{\itshape \language \LARGE \Large \large}%
    \DoNotIndex{\lastbox \lastskip \m@th \makeglossary}%
    \DoNotIndex{\maketitle \math@fontsfalse \math@fontstrue
        \mathsf}%
    \DoNotIndex{\MessageBreak \noindent \normalfont
        \normalsize}%
    \DoNotIndex{\on@line \openout \outer}%
    \DoNotIndex{\parbox \part \rmfamily \rule \sbox}%
    \DoNotIndex{\sf@size \sffamily \skip}%
    \DoNotIndex{\textsc \textup \toks@ \ttfamily \vbox}%
 %% \DoNotIndex{\begin*} maybe in the future, if the idea gets popular...
    \DoNotIndex{\hspace* \newcommand* \newenvironment*
        \providecommand*}%
    \DoNotIndex{\renewenvironment* \section* \chapter*}%
  }% of \DefaultIndexExclusions.
```

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

```
\end{MakePrivateLetters}
```

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

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

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

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

```
\UndoDefaultIndexExclusions
5839 \def\UndoDefaultIndexExclusions{%
5840   \StoreMacro\DoNotIndex
5842   \let\DoNotIndex\DoIndex
5844   \DefaultIndexExclusions
5846   \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
\index@prologue
5858 \long\def\IndexPrologue#1{\@bsphack\def\index@prologue{#1}%
       \@esphack}
\indexdiv
5861 \def\indexdiv{\@ifundefined{chapter}{\section*}{\chapter*}}
\index@prologue
5865 \@ifundefined{index@prologue}␣{\def\index@prologue{%
       \indexdiv{Index}%
5866   \markboth{Index}{Index}%
5867   Numbers␣written␣in␣italic␣refer␣to␣the␣\if@pageindex␣
         pages␣\else
5868   code␣lines␣\fi␣where␣the
5869   corresponding␣entry␣is␣described;␣numbers␣underlined␣
         refer␣to␣the
5870   \if@pageindex\else␣code␣line␣of␣the␣\fi␣definition;␣
         numbers␣in
5871   roman␣refer␣to␣the␣\if@pageindex␣pages\else␣code␣lines␣
         \fi␣where
5872   the␣entry␣is␣used.
5873   \if@pageindex\else
5874     \ifx\HLPrefix\@empty
5875       The␣numbers␣preceded␣with␣`p.'␣are␣page␣numbers.
5876     \else␣The␣numbers␣with␣no␣prefix␣are␣page␣numbers.
5877   \fi\fi
5878   \ifx\IndexLinksBlack\relax\else
5879     All␣the␣numbers␣are␣hyperlinks.
5882   \fi
5883   \gmd@dip@hook% this hook is intended to let a user add something without
         redefining the entire prologue, see below.
5885 }}{}
```

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

```
5890 \@emptify\gmd@dip@hook
\AtDIPrologue
5891 \long\def\AtDIPrologue#1{\g@addto@macro\gmd@dip@hook{#1}}
```

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

```
5895 \AtDIPrologue{%
```

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

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

```
\RequirePackage{multicol}
```

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

`\IndexMin`
```
\newdimen\IndexMin␣\IndexMin␣=␣133pt\relax%
```
originally it was set 80 pt, but with my default prologue there's at least 4.7 cm needed to place the prologue and some index entries on the same page.

`\c@IndexColumns`
```
\newcount\c@IndexColumns␣\c@IndexColumns␣=␣3
```
`theindex`
```
\renewenvironment{theindex}
  {\begin{multicols}\c@IndexColumns[\index@prologue][%
      \IndexMin]%
      \IndexLinksBlack
      \IndexParms␣\let\item\@idxitem␣\ignorespaces}%
  {\end{multicols}}
```

`\IndexLinksBlack`
```
\def\IndexLinksBlack{\hypersetup{linkcolor=black}}%
```
To make Adobe Reader work faster.

```
\@ifundefined{IndexParms}
```
`\IndexParms`
```
  {\def\IndexParms{%
      \parindent␣\z@
      \columnsep␣15pt
      \parskip␣0pt␣plus␣1pt
      \rightskip␣15pt
      \mathsurround␣\z@
      \parfillskip=-15pt␣plus␣1␣fil␣%
```
doc defines this parameter rigid but that's because of the stretchable space (more precisely, a `\dot¦fill`) between the item and the entries. But in gmdoc we define no such special delimiters, so we add an infinite stretch.
```
      \small
      \def\@idxitem{\par\hangindent␣30pt}%
```
`\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 `\catcode`s; the directive's text has its `\catcode`s fixed. Then we put the directive's text wrapped with the formatting macro into one macro in order to give just one token the gmdoc's TeX code scanner. Then launch this big TeX code scanning machinery by calling `\gmd@textEOL` which is an alias for the 'narrative' meaning of the line end. This macro opens the verbatim group and launches the char-by-char scanner. That is this scanner because of what we encapsulated the directive's text with the formatting into one macro: to let it pass the scanner.

That's why in the 'old' macrocodes case the active `%` closes the group before launching `\gmd@docstripdirective`.

The 'verbatim' directive macro works very similarly.

```
}
\foone{\@makeother\<\@makeother\>
  \glet\sgtleftxii=<
  \catcode`\^^M=\active}%
{
```

`\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 omit an obsolete remark about then-older MakeIndex's versions.]

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

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

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

\gmd@changes@init
```
\def\gmd@changes@init{%
  \@bsphack\begingroup\@sanitize
  \catcode`\\\z@␣\catcode`\␣10␣\MakePercentIgnore
  \MakePrivateLetters␣\StraightEOL
  \MakeGlossaryControls}
```

\changes
```
\newcommand\changes[4][]{\PackageWarningNoLine{gmdoc}{%
    ^^JThe␣\bslash␣changes␣command␣used␣\on@line
    ^^Jwith␣no␣\string\RecordChanges\space␣declared.
    ^^JI␣shall␣not␣warn␣you␣again␣about␣it}%
```

\changes
```
  \renewcommand\changes[4][]{%
  }}
```

\MakeGlossaryControls
```
\def\MakeGlossaryControls{%
  \edef\actualchar{\string=}\edef\quotechar{\string!}%
```

```
6155    \edef\levelchar{\string>}\edef\encapchar{\xiiclub}}%
```
for the glossary the 'actual', the 'quote' and the 'level' chars are respectively `=`, `!` and `>`, the 'encap' char remains untouched. I decided to preserve the doc's settings for the compatibility.

\changes@
```
6161 \newcommand\changes@[4][\generalname]{%
6164    \if@RecentChange{#3}%
```
if the date is later than the one stored in `\c@Chang¦ % esStartDate`,
```
6166      \@tempswafalse
6167      \ifx\generalname#1%
```
then we check whether a CS-entry is given in the optional first argument or is it unchanged.
```
6169        \ifx\last@defmark\relax\else%
```
if no particular CS is specified in `#1`, we check whether `\last@defmark` contains something and if so, we put it into `\gmu@tempb` scratch macro.
```
6172          \@tempswatrue
6173          \edef\gmu@tempb{%
```
it's a bug fix: while typesetting traditional .dtxes, `\last@defmark` came out with `\` at the beginning (which resulted with `\\`⟨*name*⟩ in the change log) but while typesetting the 'new' way, it occurred without the bslash. So we gobble the bslash if it's present and two lines below we handle the exception of `\last@defmark = {\}` (what would happen if a definition of `\\` was marked in new way gmdocing).
```
6181          \if\bslash\last@defmark\else\last@defmark\fi}%
6182          \ifx\last@defmark\bslash\let\gmu@tempb\last@defmark%
                \fi%
6183          \n@melet{gmd@glossCStest}{gmd/isaCS/\last@defmark}%
6184        \fi
6185      \else%
```
the first argument isx not `\generalname` i.e., a particular CS is specified by it (if some day one wishes to `\changes \generalname`, they should type `\changes[generalname]`…)
```
6189        \@tempswatrue
6190        {\escapechar\m@ne
6191          \xdef\gmu@tempb{\string#1}}%
6192        \if\bslash\@xa\@firstofmany\string#1\relax\@@nil%
```
we check whether `#1` is a CS…

\gmd@glossCStest
```
6194          \def\gmd@glossCStest{1}%
```
… and tell the glossary if so.
```
6195        \fi
6196      \fi
```

\gmd@glossCStest
```
6197      \@ifundefined{gmd@glossCStest}{\def\gmd@glossCStest{0}}{%
            }%
6198      \protected@edef\gmu@tempa{\@nx\gmd@glossary{%
6199          \if\relax\GeneralName\relax\else
6200            \GeneralName%
```
it's for the `\DocInclude` case to precede every `\changes` of the same file with the file name, cf. line 6696.
```
6203          \fi
6204          #2\levelchar%
6205          \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 `\gen¦ eralname`.
```
6210            \gmu@tempb
6211            \actualchar\bslash␣verb*%
6212            \if\verbatimchar\gmu@tempb$\else\verbatimchar\fi
```

```
            \if1\gmd@glossCStest\quotechar\bslash\fi␣%
                \gmu@tempb
            \if\verbatimchar\gmu@tempb$\else\verbatimchar\fi
          \else
            \space\actualchar\generalname
          \fi
          :\levelchar%
          #4%
        }}%
      \gmu@tempa
      \grelaxen\gmd@glossCStest
    \fi% of \if@recentchange
    \endgroup\@esphack}
```

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

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

`\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␣` (`\@tempcnta`) 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␣
      memorised
    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␣%
```
[12] see below.
```
  \edef\gmu@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.}}%
  \gmu@tempa
\fi}
```

(Explanation to line 6276.) My TEX Guru has remarked that the change history tool should be used for documenting the changes that may be significant for the users not only for the author and talking of what may be significant to the user, no changes should be hidden since the first published version. However, the changes' start date may be used to provide hiding the author's 'personal' notes: they should only date the 'public' changes with the four digit year and the 'personal' ones with two digit year and set `\ChangesStart{}{1000/0/0}` or so.

In line 6276 I establish a test value that corresponds to a date earlier than any TEX stuff and is not too small (early) to ensure that hiding the two digit year changes shall not be mentioned in the changes prologue.

"The entries [of a given version number] are sorted for convenience by the name of [the macro explicitly specified as the first argument or] the most recently introduced macro name (i.e., that in the most recent `\begin{macro}` command [or `\Define`]). We therefore provide [`\last@defmark`] to record that argument, and provide a default definition in case `\changes` is used outside a `macro` environment. (This is a wicked hack to get such entries at the beginning of the sorted list! It works providing no macro names start with `!` or `"`.)

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

`\generalname` 6325 `\def\generalname{General}`

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

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

`\RecordChanges` 6337 `\def\RecordChanges{\makeglossary\gmd@DefineChanges`
6338 `  \@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` 6350 `\newdimen\GlossaryMin  \GlossaryMin  = 8opt`

c@GlossaryColumns

`\c@GlossaryColumns` 6352 `\newcount\c@GlossaryColumns  \c@GlossaryColumns = 2`

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

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

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

Here is the MakeIndex style definition:

```
⟨/ 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}
```

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

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

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

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

\gmd@chgs
```
\DeclareCommand\gmd@chgs{o>!m}{%
  \IfValueTF{#1}{%
    \edef\gmu@tempa{\@nx\changes@[\unexpanded{#1}]%
      \@xa\unexpanded\@xa{\gmd@chgs@parse#2\@@nil}}%
  {\edef\gmu@tempa{\@nx\changes@
      \@xa\unexpanded\@xa{\gmd@chgs@parse#2\@@nil}}%
  \gmu@tempa}
```

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

The checksum

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

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

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

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

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

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

And we'll use it in the line 3722 (\stepcounter is \global). See also the \chschange declaration, l. 6594.

However, the check sum mechanism in gmdoc behaves slightly different than in doc which is nicely visible while gmdocing doc: doc states its check sum to be 2171 and our count counts 2126. The mystery lies in the fact that doc's CheckSum mechanism counts the code's backslashes no matter what they mean and the gmdoc's the escape chars so, among others, \\ at the default settings increases doc's CheckSum by 2 while the gmdoc's by 1. (There are 38 occurrences of \\ in doc.dtx `macrocodes`, I counted myself.)[13]

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

In gmdoc we have the `\AtEndInput` hook.

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

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

`\check@checksum`

```
\def\check@checksum{\relax
  \ifnum\check@sum=\z@
    \edef\gmu@tempa{% why \edef—see line 6556
      \@nx\typeout{**********************************^^J%
        *␣The␣input␣file␣\gmd@inputname\space␣has␣no␣
            Checksum
        stated.^^J%
        *␣The␣current␣checksum␣is␣\the\c@CheckSum.^^J%
        \gmd@chschangeline% a check sum changes history entry, see below.
        *␣(package␣gmdoc␣info.)^^J%
        **********************************^^J}}
    \else
    \ifnum\check@sum=\c@CheckSum
      \edef\gmu@tempa{%
        \@nx\typeout{**********+++++++++++++++^^J%
          *␣The␣input␣file␣\gmd@inputname:␣Checksum␣
              passed.^^J%
          \gmd@chschangeline
          *␣(package␣gmdoc␣info.)^^J%
          **********+++++++++++++++^^J}}
    \else
      \edef\gmu@tempa{%
        \@nx\typeout{********\gmd@wykrzykniki^^J%
          *!␣The␣input␣file␣\gmd@inputname:^^J%
          *!␣The␣CheckSum␣stated:␣\the\check@sum\space<>␣my
          count:␣\the\c@CheckSum.^^J%
          \gmd@chschangeline
          *!␣(package␣gmdoc␣info.)^^J%
          ********\gmd@wykrzykniki^^J}}%
    \fi
  \fi
  \gmu@tempa
  \@xa\AtEndDocument\@xa{\gmu@tempa}% we print the checksum notification on the terminal immediately and at end of TEXing not to have to scroll the output far nor search the log.
  \global\check@sum\z@}
```

`\gmd@wykrzykniki`

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

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

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

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

```
\gmd@chschangeline 6570 \def\gmd@chschangeline{%
                   6571   \xiipercent\space\string\chschange
                   6572   {\@ifundefined{fileversion}{v???}{\fileversion}}%
                   6573   {\the\year/\the\month/\the\day}%
                   6574   {\the\c@CheckSum}^^J%
                   6575   \xiipercent\space\string\chschange
                   6576   {\@ifundefined{fileversion}{v???}{\fileversion}}%
                   6577   {\@xa\@gobbletwo\the\year/\the\month/\the\day}%
                   6578   {% with two digit year in case you use \ChangesStart.
                   6579     \the\c@CheckSum}^^J}
```

And here the meaning of such a line is defined:

```
      \chschange 6582 \outer\pdef\chschange{%
                        %  #1  m file version,
                        %  #2  m date,
                        % (#3) c hecksum,
                        % [#4] o the reason of check sum change, possibly short.
                 6589   \@ifQueerEOL
\EOLwasQueer     6590   {\def\EOLwasQueer{11}}{\def\EOLwasQueer{10}}%
\EOLwasQueer     6591   \gmd@changes@init
                 6592   \chschange@}

     \chschange@ 6594 \DeclareCommand\chschange@{mmmo}{%
                 6595   \changes@{#1}{#2}{CheckSum␣#3
                 6596     \IfValueT{#4}{because␣of␣#4}%
                 6597   }% \csname... because \changes is \outer.
                 6599   \CheckSum{#3}%
                 6600   \IfValueF{#4}{%
                 6601     \if\EOLwasQueer
                 6602     \afterfi{%
                 6603       \@ifnextchar\par{%
                 6604         \@xa\gmd@textEOL\gobble}%
                 6605       {}%
                 6606     }% of \afterfi,
                 6607     \fi}% of no value of #4,
                 6608 }% of \chschange@.
```

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

Macros from ltxdoc

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

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

```
\ltxPageLayout 6630 \newcommand*\ltxPageLayout{%
```

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

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

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

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

```
\setlength\marginparwidth{95pt}%
\setlength\oddsidemargin{82pt}%
\setlength\evensidemargin{82pt}}
```

`\DocInclude` and the ltxdoc-like setup

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

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

`\DocInclude` So, define the `\DocInclude` command, that acts

"more or less exactly the same as `\include`, but uses `\DocInput` on a dtx [or .fdd] file, not `\input` on a tex file."

Our version will accept also .sty, .cls, and .tex files.

`\DocInclude`
```
\DeclareCommand\DocInclude{O{}mO{}}{%
    % [#1] o  path (with closing slash), will not be printed
    %  #2  m  file name without extension, will be printed
    % [#3] o  file extension (with dot) if not .sty, .cls, .tex, .dtx nor .fdd
```
originally it took just one argument. Here we make it take two, first of which is intended to be the path (with the closing `/`). This is intended not to print the path in the page footers only the filename.

`\HLPrefix`
```
  \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 `\if`s. 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.

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

```
    \edef\gmd@jobname{\jobname}%
    \edef\gmd@difilename{%
      \@xa\@xa\@xa\@gobble\@xa\string\csname#2\endcsname}%
    \ifx\gmd@jobname\gmd@difilename
```
`\gmd@auxext`
```
      \def\gmd@auxext{auxx}%
    \else
```
`\gmd@auxext`
```
      \def\gmd@auxext{aux}%
    \fi
  \relax
```

```
6714    \clearpage
\currentfile 6716    \gmd@docincludeaux␣\def\currentfile{%
                       gmdoc-IncludeFileNotFound.000}%
6717    \let\fullcurrentfile\currentfile
6718    \@ifnonempty{#3}%
6719    {%
6720      \unless\if.\@firstofmany#3\relax\@@nil
6721        \PackageError{gmdoc}{Optional␣\xiihash3␣of
6722          \string\DocInclude\space
6723          if␣present␣has␣to␣begin␣with␣a␣dot␣(.)}%
6724      \fi
6725      \edef\currentfile{#2#3}%
6726    \IfFileExists{#1\currentfile}{}%
6727    {\PackageError{gmdoc}{\string\DocInclude\space␣file
6728        \currentfile\space␣not␣found}}%
6729    }% of if extension given.
6730    {% if extension not given:
6731      \IfFileExists{#1#2.fdd}{\edef\currentfile{#2.fdd}}{% it's not
                .fdd,
6732        \IfFileExists{#1#2.dtx}{\edef\currentfile{#2.dtx}}{% it's not
                .dtx either,
6734          \IfFileExists{#1#2.sty}{\edef\currentfile{#2.sty}}{% it's
                not .sty,
6736            \IfFileExists{#1#2.cls}{\edef\currentfile{#2.cls}}{%
                  % it's not .cls,
6738              \IfFileExists{#1#2.tex}{\edef\currentfile{#2.tex}}{%
                    % it's not .tex,
6740                \IfFileExists{#1#2.fd}{\edef\currentfile{#2.fd}}{%
                      % so it must be .fd or error.
6742                  \PackageError{gmdoc}{\string\DocInclude%
                        \space␣file
6743                    #1#2.fdd/dtx/sty/cls/tex/fd␣not␣
                          found.}}}}}}}%
6744    }% of if no extension given
6747    \edef\currentfile{\@xa\detokenize\@xa{\currentfile}}%
6748    \edef\fullcurrentfile{#1\currentfile}%
6749    \ifnum\@auxout=\@partaux
6750      \@latexerr{\string\DocInclude\space␣cannot␣be␣nested}%
            \@eha
6751    \else␣\@docinclude{#1}#2#3␣\fi}% Why is #2 delimited with ␣ not braced
         as we are used to, one may ask.

\@docinclude 6757 \def\@docinclude#1#2␣{% To match the macro's parameter string, is an answer.
         But why is \@docinclude defined so? Originally, in ltxdoc it takes one ar-
         gument and it's delimited with a space probably in resemblance to the true
         \input (\@@input in LaTeX).
6762    \clearpage
6764    \if@filesw␣\gmd@writemauxinpaux{#2.\gmd@auxext}\fi% this strange
         macro with a long name is another spurious thing to allow _ in the file-
         names (see line 6829). which are allowed anyway unless active or 14.
6768    \@tempswatrue
6769    \if@partsw␣\@tempswafalse\edef\gmu@tempb{#2}%
6770      \@for␣\gmu@tempa:=\@partlist\do{\ifx\gmu@tempa%
```

```
          \gmu@tempb\@tempswatrue\fi%
6771    \fi
6772    \if@tempswa% the file is on \@partlist
6773      \let\@auxout\@partaux
6774      \if@filesw
6775        \immediate\openout\@partaux␣#2.\gmd@auxext\relax% Yes, only
              #2. It's to create and process the partial .aux(x) files always in the
              main document's (driver's) directory.
6780        \immediate\write\@partaux{\relax}%
6781      \fi
```

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

```
6788      \StoringAndRelaxingDo\gmd@doIndexRelated
6789      \if@ltxDocInclude\part{\currentfile}% In the ltxdoc-like setup we
            make a part title page with only the filename and the file's \maketitle
            will typeset an article-like title.
6792      \else\let\maketitle=\InclMaketitle
6793      \fi% In the default setup we redefine \maketitle to typeset a common
            chapter or part heading.
6795      \if@ltxDocInclude\xdef@filekey\fi
6796      \GetFileInfo{\currentfile}% it's my (GM) addition with the account
            of using file info in the included files' title/heading etc.
6798      \incl@DocInput{\fullcurrentfile}% originally just \currentfile.
6799      \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 in-
            creases the counter.
```

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

```
6805      \RestoringDo\gmd@doIndexRelated
6807      \clearpage
6809      \gmd@writeckpt{#1#2}%
6810      \if@filesw␣\immediate\closeout\@partaux␣\fi
6811    \else% the file isn't on \@partlist
6812      \@nameuse{cp@#1#2}%
6813      \g@emptify\gmd@ABIOnce
6814    \fi
6815    \let\@auxout\@mainaux}% end of \@docinclude.
```

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

\xdef@filekey

```
6819 \def\xdef@filekey{{\@relaxen\narrativett% This assignment is very trick-
       ily crafted: it makes all \narrativetts present in the \filekey's expansion
       unexpandable not only the one added in this step.
6823      \xdef\filekey{\filekey,␣\thefilediv={\narrativett%
            \currentfile}}}}
```

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

\gmd@writemauxinpaux

```
6829 \def\gmd@writemauxinpaux#1{% this name comes from 'write out to main .aux
       to input partial .aux'.
```

We wrap \@input{⟨*partial .aux*⟩} in a $\__{12}$ hacked scope. This hack is especially recommended here since the .aux file may contain a non-\global stuff that should not

be localised by a group that we would have to establish if we didn't use the hack. (Hope you understand it. If not, notify me and for now I'll only give a hint: "Look at it with the TEX's eyes". More uses of this hack are to be seen in gmutils where they are a bit more explained.)

```
\immediate\write\@mainaux{%
  \unexpanded{%
    \bgroup
    \@makeother\_% to allow underscore
    \@makeother\~% to allow paths beginning with ~/
    \firstofone}{\egroup
    \string\@input{#1}}}}
```

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

`\gmd@writeckpt`

```
\def\gmd@writeckpt#1{%
  \immediate\write\@partaux{%
    \unexpanded{%
      \bgroup
      \@makeother\_%
      \@makeother\~%
      \firstofone}\@charlb\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` `\@aalph`

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

A macro that initialises things for `\DocInclude`.

`\gmd@docincludeaux`

```
\def\gmd@docincludeaux{%
```

We set the things for including the files only once.

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

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

```
\ifx\filediv\relax
  \ifx\filedivname\relax% (nor \filediv neither \filedivname is 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
```

`\thefilediv`

```
\def\thefilediv{\aalph{\filedivname}}% The files will be numbered with letters, lowercase first.
\@xa\let\csname␣the\filedivname\endcsname=\thefilediv% This line lets \the⟨chapter⟩ etc. equal \thefilediv.
```

`\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:␣{\narrativett\currentfile}␣%
  \else
    \GetFileInfo{\currentfile}%
    File␣\thefilediv:␣{\narrativett\filename}␣%
    Date:␣\filedate\␣%
    Version␣\fileversion
  \fi
  \hfill\thepage}%
\glet\@evenfoot\@oddfoot% see line 6931.
\@xa\def\csname\filedivname␣name\endcsname{File}% we redefine the name of the proper division to 'File'.
\ifx\filediv\section
```

```
      \let\division=\subsection
      \let\subdivision=\subsubsection
      \let\subsubdivision=\paragraph
```

If `\filediv` is higher than `\section` we don't change the three divisions (they are `\section`, `\subsection` and `\subsubsection` by default). `\section` seems to me the lowest reasonable sectioning command for the file. If `\filediv` is lower you should rather rethink the level of a file in your documentation not redefine the two divisions.

```
  \fi}% end of \gmd@docincludeaux.
```

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

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

\SelfInclude
```
\def\SelfInclude{\DocInclude{\jobname}}
```

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

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

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

\@onlypreamble\ltxLookSetup
```

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

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

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

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

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

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

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

Redefinition of `\maketitle`

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

"The macro to generate titles is easily altered in order that it can be used more than once (an article with many titles). In the original, diverse macros were concealed after

use with \relax. We must cancel anything that may have been put into \@thanks, etc., otherwise all titles will carry forward any earlier such setting!"

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

```
\AtBegInput{%
  \providecommand*\maketitle{\par
    \begingroup \def \thefootnote {\fnsymbol {footnote}}%
    \setcounter {footnote}\z@
    \def\@makefnmark{\rlap{\@textsuperscript{\normalfont%
        \@thefnmark}}}%
    \long\def\@makefntext##1{\parindent 1em\noindent
      \hb@xt@1.8em{%
        \hss\@textsuperscript{\normalfont\@thefnmark}}##1}%
    \if@twocolumn \twocolumn [\@maketitle ]%
    \else \newpage \global \@topnum \z@ \@maketitle \fi
```

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

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

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

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

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

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

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

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

```
      \par␣\vskip␣1.5em\relax}%
```

We'd better restore the primary meanings of the macros making a title. (LaTeX 2ε 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 initialise \titlesetup 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 the `\ltxLookSetup` declaration is in force, `\Doc/SelfInclude` will make for an included file a part's title page and an article-like title.

Let's initialise the file division macros.

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

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

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

```
\InclMaketitle  \def\InclMaketitle{%
\and              {\def\and{, ␣}% we make \and just a comma.
                  {\let\thanks=\@gobble% for the toc version of the heading we discard \thanks.
                    \protected@xdef\incl@titletotoc{%
                      \@title\@ifauthor{%
                        \protect\space(\@author)}{}}% we add the author iff the 'files have different authors' and author exists (@variousauthors)
                  }%
\thanks           \def\thanks##1{\footnotemark
                    \protected@xdef\@thanks{\@thanks% to keep the previous \thanks if there were any.
                      \protect\footnotetext[\the\c@footnote]{##1}}}% for some 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
```

```
7163            {\protect\smallerr% this macro is provided by the gmutils package
                    after the relsize package.
7165              \@ifauthor
7166              {\protect\\[0.15em]\@nx\@author
7167                 \ifx\relax\@date\else,␣\fi}% after use, \@date is let to
                         % \relax.
7169              {\ifx\relax\@date\else\protect\\[0.15em]\fi}
```

The default is that all the included files have the same author(s). In this case we won't print the author(s) in the headings. Otherwise we wish to print them. The information which case are we in is brought by the `\if@variousauthors` switch defined in line 7200.

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

```
7181              \protect\@date}}% end of \incl@filedivtitle's brace (2nd
                      or 3rd argument).
7183        }% 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).

```
7187        }% end of the comma-\and's group.
7188        \@xa\filediv\incl@filedivtitle
7189        \@thanks
7190        \g@relaxen\@author␣\g@relaxen\@title␣\g@relaxen\@date
7191        \g@emptify\@thanks
7192    }% end of \InclMaketitle.
```

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

`\if@variousauthors` 7200 `\newif\if@variousauthors`

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

`\PrintFilesAuthors` 7204 `\newcommand*\PrintFilesAuthors{\@variousauthorstrue}`

And the counterpart, if you change your mind:

`\SkipFilesAuthors` 7206 `\newcommand*\SkipFilesAuthors{\@variousauthorsfalse}`

`\@ifauthor` 7208

```
\def\@ifauthor{%
      % #1 what if true
      % #2 what if false
7213    \ifnum\numexpr\if@variousauthors1\else0\fi*
7214        \ifx\@author\relax0\else\ifx\@author\@empty0\else1\fi%
                \fi>0
7215      \@xa\@firstoftwo
7216    \else
7217      \@xa\@secondoftwo
7218    \fi
7219 }
```

The file's date and version information

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

```
\GetFileInfo 7226 \def\GetFileInfo#1{%
  \filename  7227   \def\filename{#1}%
             7228   \def\gmu@tempb##1␣##2␣##3\relax##4\relax{%
  \filedate  7229     \def\filedate{##1}%
\fileversion 7230     \def\fileversion{##2}%
  \fileinfo  7231     \def\fileinfo{##3}}%
             7232   \edef\gmu@tempa{\csname␣ver@#1\endcsname}%
             7233   \@xa\gmu@tempb\gmu@tempa\relax?␣?␣\relax\relax}
```

Since we may documentally input files that we don't load, as `doc` e.g., let's define a declaration to be put (in the comment layer) before the line(s) containing `\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 7244 \newcommand*\FileInfo{%
          7245   \bgroup
          7246   \gmd@ctallsetup
          7247   \bgroup% yes, we open two groups because we want to rescan tokens in 'usual'
                    catcodes. We cannot put \gmd@ctallsetup into the inner macro because
                    when that will be executed, the \inputlineno will be too large (the last
                    not the first line).
          7251   \let\do\@makeother
          7252   \do\␣\do\{\do\}\do\^^M\do\\%
          7253   \gmd@fileinfo}

          7256 \foone{%
          7257   \catcode`!\z@
          7258   \catcode`(\@ne
          7259   \catcode`)\tw@
          7260   \let\do\@makeother
          7261   \do\␣% we make space 'other' to keep it for scanning the code where it may be
                    leading.
          7263   \do\{\do\}\do\^^M\do\\}%
          7264 (%
\gmd@fileinfo 7265 !def!gmd@fileinfo#1Provides#2{#3}#4[#5]#6^^M%
          7266 (!egroup% we close the group of changed catcodes, the catcodes of the arguments
                    are set. And we are still in the group for \gmd@ctallsetup.
          7269 !gmd@writeFI(#2)(#3)(#5)%
          7270 !gmd@FIrescan(#1Provides#2{#3}#4[#5]#6)% this macro will close the group.

          7275 )%
          7276 )

\gmd@writeFI 7278 \def\gmd@writeFI#1#2#3{%
          7280   {\newlinechar=\endlinechar%
          7282     \immediate\write\@auxout{%
          7283       \global\@nx\@namedef{%
          7284         ver@#2.\if␣P\@firstofmany#1\@@nil␣sty\else␣cls%
                           \fi}{#3}}}}

          7286 \foone\obeylines{%
```

```
\gmd@FIrescan 7287   \def\gmd@FIrescan#1{%
              7292  {\newlinechar=\endlinechar\scantokens{#1}}\egroup^^M}}
```

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

```
\ProvideFileInfo 7300 \def\ProvideFileInfo#1{%
              7301   \begingroup
              7302     \catcode`\␣10␣\catcode\endlinechar␣10␣%
              7303     \@makeother\/\@makeother\&%
              7304     \kernel@ifnextchar[{\gmd@providefii{#1}}{\gmd@providefii{%
                         #1}[]}%
              7305   }
\gmd@providefii 7309 \def\gmd@providefii#1[#2]{%
                        (we don't write the file info to .log)
              7311   \@xa\xdef\csname␣ver@#1\endcsname{#2}%
              7312   \endgroup}
```

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

```
\ProvideSelfInfo 7316 \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 7320 \newcommand*\filenote{This␣file␣has␣version␣number␣%
                    \fileversion{}␣dated␣\filedate{}.}
```

And exactly as `\thanks`:

```
\thfileinfo 7322 \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:

```
              7333 \foone{\obeylines}%
              7334 {%
\IndexInput   7335   \pdef\IndexInput#1{%
              7338     \StoreMacro\code@delim%
              7339     \CodeDelim*\^^Z%
\gmd@iihook   7340     \def\gmd@iihook{% this hook is \edefed!
              7341       \@nx^^M%
              7342       \code@delim\relax\@nx\let\@nx\EOFMark\relax}%
              7343     \DocInput{#1}\RestoreMacro\code@delim}%
              7344 }
```

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 7360 \newenvironment{gmlonely}{\StraightEOL}{}
```

```
\skipgmlonely 7362 \newcommand\skipgmlonely[1][]{%
              7363   \def\gmu@tempa{%
\gmd@skipgmltext 7364     \def\gmd@skipgmltext{%
              7365       \g@emptify\gmd@skipgmltext
              7367       #1%
              7368     }}% not to count the lines of the substituting text but only of the text omitted
              7370   \gmu@tempa
              7371   \@xa\AtBegInput\@xa{\gmu@tempa}%
    gmlonely  7372   \renewenvironment{gmlonely}{%
              7373     \StraightEOL
              7374     \@fileswfalse% to forbid writing to .toc, .idx etc.
              7375     \setboxo=\vbox\bgroup}{\egroup\gmd@skipgmltext}}
```

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

```
              7382 \foone{\catcode`\_=8␣}% we ensure the standard \catcode of _.
              7383 {%
   \catother  7384   \newcommand*\catother{${}_{12}$}%
```

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

```
   \catactive 7387   \newcommand*\catactive{${}_{13}$}%
```

and a letter, $_{11}$

```
   \catletter 7389   \newcommand*\catletter{${}_{11}$}% .
              7390 }
```

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 7395 \newenvironment*{copyrnote}{%
              7396   \StraightEOL\everypar{\hangindent3em\relax\hangafter1␣}%
              7397   \par\addvspace\medskipamount\parindent\z@\obeylines}{%
              7398   \@codeskipputgfalse\stanza}
```

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

```
              7402 \StoreEnvironment{quotation}
\gmd@quotationname 7403 \def\gmd@quotationname{quotation}
     quotation 7404 \renewenvironment{quotation}{%
```

The first non-me user complained that `abstract` comes out in quotation marks. That is because `abstract` uses `quotation` internally. So we first check whether the current environment is `quotation` or something else.

```
              7411   \ifx\@currenvir\gmd@quotationname
              7412   \afterfi{\par``\ignorespaces}%
              7413   \else\afterfi{\storedcsname{quotation}}%
              7414   \fi}
              7415 {\ifx\@currenvir\gmd@quotationname
              7416   \afterfi{\ifhmode\unskip\fi''\par}%
              7417   \else\afterfi{\storedcsname{endquotation}}%
              7418   \fi}
```

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

```
\gmdnoindent 7423 \def\gmdnoindent{%
             7424   \ifvmode\leavevmode\hskip-\parindent\ignorespaces
             7425   \fi}%
```
`\ignorespaces` is added to eat a space inserted by `\gmd@textEOL`. Without it it also worked but it was a bug: since `\parindent` is a dimen not skip, TEX looks forward and expands macros to check whether there is a stretch or shrink part and therefore it gobbled the `\gmd@textEOL`'s space.

When a verbatim text occurs in an in-line comment, it's advisable to precede it with `%` if it begins a not first line of such a comment not to mistake it for a part of code. Moreover, if such a short verb breaks in its middle, it should break with the percent at the beginning of the new line. For this purpose provide `\inverb`. It breaks with a `%` at the beginning of new line. Ist starred version puts `%` also at the end of the upper line.

```
\inverb      7439 \pdef\inverb{%
             7441   \gm@ifstar{%
             7442     \def\gmu@tempa{\verbhyphen}% the pre-break.
             7443     \@emptify\gmu@tempb% the no-break.
             7444     \gmd@inverb}%
             7445   {\@emptify\gmu@tempa% the pre-break empty
             7446     \def\gmu@tempb{\gmboxedspace}% the no-break boxed space.
             7447     \gmd@inverb}}
\gmboxedspace 7449 \newcommand*\gmboxedspace{\hbox{\normalfont{␣}}}
\gmd@nlperc  7451 \pdef\gmd@nlperc{%
             7458   \ifhmode\unskip\fi
             7459   \begingroup\hyphenpenalty\inverbpenalty\relax
             7460   \discretionary{\hbox{\gmu@tempa}}% (pre-break). I always put a \hbox
                      here to make this discretionary score the \hyphenpenalty not \exhy¦
                      phenpenalty (The TEXbook p. 96) since the latter may be 10,000 in Polish
                      typesetting.
             7464   {\hbox{\narrationmark}}% (post-break)
             7465   {\gmu@tempb}% (no-break).
             7466   \endgroup
             7467   \penalty10000\hskiposp\relax}
\inverbpenalty 7469 \def\inverbpenalty{-1000}
\gmd@inverb  7471 \pdef\gmd@inverb{%
             7472   \gmd@nlperc
             7473   \ifmmode\hbox\else\leavevmode\null\fi
             7474   \bgroup
             7475   \ttverbatim
             7476   \narrativett
\breakablevisspace 7477   \def\breakablevisspace{%
             7478     \discretionary{\visiblespace}{\narrationmark}{%
                        \visiblespace}}%
\breakbslash 7479   \def\breakbslash{%
             7480     \discretionary{}{\narrationmark\type@bslash}{%
                        \type@bslash}}%
\breaklbrace 7481   \def\breaklbrace{%
             7482     \discretionary
             7483       {\xiilbrace\verbhyphen}%
             7484       {\narrationmark}%
             7485       {\xiilbrace}}%
```

```
    \gm@verb@eol
    \@sverb@chbsl% It's always with visible spaces.
}
```

\nlperc
```
\pdef\nlperc{\newline\narrationmark\ignorespaces}
```

\nlpercent
```
\pdef\nlpercent{%
    \@emptify\gmu@tempa
    \def\gmu@tempb{\gmboxedspace}%
    \gmd@nlperc
}
```

\incs
```
\pdef\incs{% an in-line \cs
    \@emptify\gmu@tempa
    \def\gmu@tempb{\gmboxedspace}%
    \gmd@nlperc\cs
}
```

\inenv
```
\def\inenv{\incs[]}% an in-line \env
```

\incmd
```
\def\incmd{% it has to be \def to let it expand to let \cmd convert its argument
    to a safe string.
        \nlpercent\cmd}
```

\inhash
```
\def\inhash{\nlpercent\hash}
```

As you see, `\inverb` and `\nlpercent` insert a discretionary that breaks to `%` at the beginning of the lower line. Without the break it's a space (alas at its natural width i.e., not flexible) or, with the starred version, nothing. The starred version puts `%` also at the end of the upper line. Then `\inverb` starts sth. like `\verb*` but the breakables of it break to `%` in the lower line.

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

An optional hyphen for CSes in the in-line comment:

\cs
```
\@xa\ampulexdef\csname\string\cs\endcsname
[\ampulexhash1]␣[{{\ampulexhash1}}]␣{\begingroup}␣{%
    \ifdefined}
```

\+
```
{\begingroup␣\def\+{\discre{\gmv@hyphen}{\narrationmark}{}}%
    \ifdefined}
```

\ds
```
\providecommand*\ds{DocStrip}
```

A shorthand for `\CS`:

\CS
```
\pdef\CS{%
    \acro{CS}%
    \@ifnextcat␣a{␣}{}}% we put a space if the next token is ␣. It's the next best
        thing to checking whether the CS consisting of letters is followed by a space.
```

\CSs
```
\pdef\CSs{\CS{}es\@ifnextcat␣a{␣}{}}% for pluralis.
```

\CSes
```
\pdef\CSes{\CS{}es\@ifnextcat␣a{␣}{}}% for pluralis.
```

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

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

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

And for documenting in general:

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

\division
```
\let\division=\section
```

\subdivision
```
\let\subdivision=\subsection
```

\subsubdivision
```
\let\subsubdivision=\subsubsection
```

To kill a tiny little bug in doc.dtx (in line 3299 \gmu@tempb and \gmu@tempc are written plain not verbatim):

gmd@mc
```
\newcounter{gmd@mc}
```

Note it is after the macrocode group

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

\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 \gmu@tempb/c).

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

```
\@emptify\gmd@mcdiag
```

\mcdiagOn
\gmd@mcdiag
```
\def\mcdiagOn{\def\gmd@mcdiag{%
    \typeout{^^J\bslash␣end{\gmd@lastenvir}␣No.\the\c@gmd@mc
    \space\on@line,␣cln.\the\c@codelinenum.}}}
```

\mcdiagOff
```
\def\mcdiagOff{\@emptify\gmd@mcdiag}
```

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

\DeclareEnvironment{enumargs}{o}% the optional argument specifies number of #'s; it's of the o type to inform if it was not given by the user to handle a possible active char touched by argument's catcher; can be 1 (the default), 2 or 4; any else produces one #.

```
{%
  \StraightEOL
  \if@aftercode
    \edef\gmu@tempa{\the\leftskip}%
    \edef\gmu@tempb{\the\hangindent}%
  \fi
  \enumerate
  \if@aftercode
    \leftskip=\glueexpr\gmu@tempa+\gmu@tempb\relax
  \fi
  \edef\gmd@ea@hashes{%
```

```
      \#\ifcase\IfValueTF{#1}{#1}{1}\relax
      \or\or\#\or\or\#\#\#\fi}%

    \@namedef{label\@enumctr}{%
      \env{\if@aftercode\narrationmark\fi
        \relax% to stop \ignorespaces
        \gmd@ea@bwrap
        \gmd@ea@hashes
        \csname␣the\@enumctr\endcsname
        \gmd@ea@ewrap}}% of \label⟨@enumctr⟩.
    \let\mand\item
\gmd@ea@wraps  \provide\gmd@ea@wraps{%
      \emptify\gmd@ea@ewrap
      \emptify\gmd@ea@bwrap}%
    \gmd@ea@wraps
      \opt  \def\opt{%
      \def\gmd@ea@bwrap{[}\def\gmd@ea@ewrap{]}%
    \item
      \gmd@ea@wraps}%

    \settowidth{\@tempdima}{\narrativett␣x\gmd@ea@hashes7x}%
    \edef\gmd@ea@xxxwd{\the\@tempdima}%

       \dc  \DeclareCommand\dc␣!{%
      Q{*>}␣% (1) we check whether there's a sergeant right of the prefix or a star
            to suppress parentheses,
      Q{P!lL\long␣iI}␣% (2) an optional 'bare' prefix for a 'long' argument or
            for ignored
      b␣% (3) prefix(es) in curly braces ( This way we allow the prefix(es) to be
            braced or not at the author's option),
      S{bBcCmoOQsSTAGK\afterassignment}␣% (4) (optional) argument type
            specifier,
      b␣% (5) (optional) default value of the specified argument or (for K and G)
            mandatory.
      b␣% (6) default of K and G.
    }{%
       \IfAmong␣*\among{##1}%
      {% a * suppresses bracket/brace/parentheses decoration.
        \def\gmd@ea@bwrap{\hbox␣to␣\gmd@ea@xxxwd\bgroup\hss}%
        \def\gmd@ea@ewrap{\hss\egroup}%
      }%
      {% if there's no * in #1, be wrap the item label in braces/brackets/parentheses.
        \IfAmong␣##4\among{bB}{% I decide not to print m type arguments in
              braces because the braces are not mandatory for this type.
          \def\gmd@ea@bwrap{\{}%
          \def\gmd@ea@ewrap{\}}%
        }{}%
        \IfAmong␣##4\among{cC}{%
          \def\gmd@ea@bwrap{(}%
          \def\gmd@ea@ewrap{)}%
        }{}%
        \IfAmong␣##4\among{oO}{%
          \def\gmd@ea@bwrap{[}%
          \def\gmd@ea@ewrap{]}%
```

```
      }{}%
      \IfAmong␣##4\among{G}{%
        \def\gmd@ea@bwrap{\detokenize\@xa{\@firstoftwo##5}}%
        \def\gmd@ea@ewrap{\detokenize\@xa{%
             \@secondoftwo##5}}%
      }{}%
      \IfAmong␣##4\among{A}{%
        \def\gmd@ea@bwrap{<}%
        \def\gmd@ea@ewrap{>}%
      }{}%
    }% of if no * in #1.
    \IfAmong␣##4\among{mQsSTK\afterassignment}{%
      \def\gmd@ea@bwrap{\hbox␣to␣\gmd@ea@xxxwd\bgroup\hss}%
      \def\gmd@ea@ewrap{\hss\egroup}%
    }{}%
```

we add a normal space

```
    \addtomacro\gmd@ea@ewrap{{\normalfont\␣}}%
    \IfValueT{##2}{%
      \addtomacro\gmd@ea@ewrap{>\{\string##2\}}}%
    \IfValueT{##3}{%
      \addtomacro\gmd@ea@ewrap{>\{##3\}}}%
    \IfValueT{##4}{%
      \ifx␣s##4%
        \addtomacro\gmd@ea@ewrap{%
          \llap{\metachar[}\scanverb{*}\metachar]}%
      \else\addtomacro\gmd@ea@ewrap{##4}%
      \fi}%
    \IfValueT{##5}{%
      \addtomacro\gmd@ea@ewrap{\{%
%\ttverbatim breakable chars won't work because we are in the item's label's
% \hbox.
        \scanverb*{##5}%
        \}}}%
    \IfValueT{##6}{%
      \addtomacro\gmd@ea@ewrap{\{%
%\ttverbatim breakable chars won't work because we are in the item's label's
% \hbox.
        \scanverb*{##6}%
        \}}}%
    \def\gmd@blubra{%
      \addtomacro\gmd@ea@bwrap{%
        \begingroup
        \relaxen\gmd@ea@hashes
        \@namedef{the\@enumctr}{\<ign.>}%
      }%
      \prependtomacro\gmd@ea@ewrap{%
        \endgroup}%
      \addtomacro\gmd@ea@ewrap{%
        \global␣\advance␣\csname␣c@\@enumctr\endcsname␣\m@ne
      }%
      \emptify\gmd@blubra
    }%
```

```
      \IfIntersect␣{##2}{Ii}{\gmd@blubra}{}%
      \IfIntersect␣{##3}{Ii}{\gmd@blubra}{}%
      \IfAmong␣##4\among{\afterassignment}{\gmd@blubra}{}%
    \item\relax}%
```

`\IfNoValueT{#1}{\@ifnextac\@gobble{}}%` to gobble a possible active line end or active `^^A` or `^^B` that might have occurred because of `\futurelet` of the optional argument checker.

```
  }% of begin definition
{\endenumerate}
```

The starred version is intended for lists of arguments some of which are optional: to align them in line.

enumargs*
\gmd@ea@wraps

```
\newenvironment*{enumargs*}{%
  \def\gmd@ea@wraps{%
    \def\gmd@ea@bwrap{␣}\def\gmd@ea@ewrap{␣}}%
  \enumargs}{\endenumargs}
```

doc-compatibility

My TEX Guru recommended me to write hyperlinking for doc. The suggestion came out when writing of gmdoc was at such a stage that I thought it to be much easier to write a couple of `\let`s to make gmdoc able to typeset sources written for doc than to write a new package that adds hyperlinking to doc. So…

The doc package makes `%` an ignored char. Here the `%` delimits the code and therefore has to be 'other'. But only the first one after the code. The others we may re`\catcode` to be ignored and we do it indeed in line 2527.

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

\CharacterTable

```
\def\CharacterTable{\begingroup
  \@makeother\{\@makeother\}%
  \Character@Table}

\foone{%
  \catcode`\[=1␣\catcode`\]=2␣%
  \@makeother\{\@makeother\}%
[
```

\Character@Table

```
  \def\Character@Table#1{#2}[\endgroup
    \message[^^J^^J␣gmdoc.sty␣package:^^J
    ====␣The␣input␣file␣contains␣the␣\bslash␣
        CharacterTable.^^J
    ====␣If␣you␣really␣need␣to␣check␣the␣correctness␣of␣
        the␣chars,^^J
    ====␣please␣notify␣the␣author␣of␣gmdoc.sty␣at␣the␣
        email␣address^^J
    ====␣given␣in␣the␣legal␣notice␣in␣gmdoc.sty.^^J^^J]%
  ]]
```

Similarly as doc, gmdoc provides `macrocode`, `macro` and `environment` environments. Unlike in doc, `\end{macrocode}` *does not* require to be preceded with any particular number of spaces. Unlike in doc, it *is not* a kind of `verbatim`, however, which means the code and narration layers remains in force inside it which means that any text

after the first % in a line will be processed as narration (and its control sequences will be executed). For a discussion of a possible workaround see line 8184.

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

\DescribeMacro
\DescribeEnv

The \DescribeMacro and \DescribeEnv commands seem to correspond with my \TextUsage macro in its plain and starred version respectively except they don't typeset their arguments in the text i.e., they do two things of the three. So let's \def them to do these two things in this package, too:

\DescribeMacro

```
\outer\def\DescribeMacro{%
  \@bsphack
  \begingroup\MakePrivateLetters
  \gmd@ifonetoken\Describe@Macro\Describe@Env}
```

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

\DescribeEnv

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

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

\Describe

```
\outer\def\Describe{% It doesn't typeset its argument in the point of occurrence.
  \leavevmode
  \@bsphack
  \begingroup\MakePrivateLetters
  \gm@ifstar{\MakePrivateOthers\Describe@Env}{%
      \Describe@Macro}}
```

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

\Describe@Macro

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

\Describe@Env

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

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

My understanding of 'minimality' includes avoiding too much freedom as causing chaos not beauty. That's the philosophical and æ sthetic reason why I don't provide \MacroFont. In my opinion there's a noble tradition of typesetting the TEX code in

\MacroFont

\tt font and this tradition sustained should be. If one wants to change the tradition, let them redefine \tt, in TEX it's no problem. I suppose \MacroFont is not used explicitly, and that it's (re)defined at most, but just in case let's \let:

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

\CodeIndent
\MacroIndent

We have provided \CodeIndent in line 2344. And it corresponds with doc's \MacroIndent so

\MacroIndent

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

And similarly the other skips:

\MacrocodeTopsep

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

\MacroTopsep

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

\SpecialEscapechar

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

\theCodelineNo
\LineNumFont

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

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

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

I hope this lack will not cause big trouble.

For further notifications let's define a shorthand:

\noeffect@info

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

\PrintDescribeMacro
\PrintMacroName
\PrintDescribeEnv
\PrintEnvName

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

Let us just notify the user.

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

```
7943    \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

```
7948 \def\CodelineNumbered{\AtBeginDocument{\gag@index}}
7949 \@onlypreamble\CodelineNumbered
```

Note that if the linesnotnum option is in force, this declaration shall not revert its effect.

I assume that if one wishes to use doc's interface then they'll not use gmdoc's options but just the default.

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

```
7961 \let\CodelineIndex\@pageindexfalse
7962 \@onlypreamble\CodelineIndex

7964 \let\PageIndex\@pageindextrue
7966 \@onlypreamble\PageIndex
```

The next two declarations I find useful and smart:

\DisableCrossrefs

```
7970 \def\DisableCrossrefs{\@bsphack\gag@index\@esphack}
```

\EnableCrossrefs
\DisableCrossrefs

```
7972 \def\EnableCrossrefs{\@bsphack\ungag@index
7973   \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

```
7981 \newcommand*\AlsoImplementation{\@bsphack
7982   \long\def\StopEventually##1{\gdef\Finale{##1}}% we define \Fi¦
         % nale just to expand to the argument of \StopEventually not to to
         add anything to the end input hook because \Finale should only be exe-
         cuted if entire document is typeset.
   %\init@checksum is obsolete in gmdoc at this point: the CheckSum counter is reset
       just at the beginning of (each of probably numerous) input(s).
7993       \@esphack}

7995 \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

```
8002 \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"

```
8006       ##1\endinput}\@esphack}
```

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

```
8011 \@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 argu-*

ment⟩>.

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:

```
8023 \let\SpecialMainIndex=\DefIndex
```

\SpecialMainEnvIndex
```
8026 \def\SpecialMainEnvIndex{\csname␣CodeDefIndex\endcsname*}%
```
we don't type \DefIndex explicitly here because it's \outer, remember?

\SpecialIndex
```
8031 \let\SpecialIndex=\CodeCommonIndex
```

\SpecialUsageIndex
```
8033 \let\SpecialUsageIndex=\TextUsgIndex
```

\SpecialEnvIndex
```
8035 \def\SpecialEnvIndex{\csname␣TextUsgIndex\endcsname*}
```

\SortIndex
```
8037 \def\SortIndex#1#2{\index{#1\actualchar#2}}
```

"All these macros are usually used by other macros; you will need them only in an emergency."

Therefore I made the assumption(s) that 'Main' indexing macros are used in my 'Code' context and the 'Usage' ones in my 'Text' context.

\verbatimchar

Frank Mittelbach in doc provides the \verbatimchar macro to (re)define the \verb[*]'s delimiter for the index entries. The gmdoc package uses the same macro and its default definition is {&}. When you use doc you may have to redefine \verbatimchar if you use (and index) the \+ control sequence. gmdoc does a check for the analogous situation (i.e., for processing \&) and if it occurs it takes $ as the \verb*'s delimiter. So strange delimiters are chosen deliberately to allow any 'other' chars in the environments' names. If this would cause problems, please notify me and we'll think of adjustments.

\verbatimchar
```
8057 \def\verbatimchar{&}
```

\IndexPrologue

\IndexPrologue is defined in line 5858. And other doc index commands too.

```
8073 \@ifundefined{main}{}{\let\DefEntry=\main}
8075 \@ifundefined{usage}{}{\let\UsgEntry=\usage}
```

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

StandardModuleDepth
```
8083 \newcounter{StandardModuleDepth}
```

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

\
```
8088 \noeffect@info{DocstyleParms}
```

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

\DontCheckModules
```
8093 \@emptify\DontCheckModules␣\noeffect@info{DontCheckModules}
```
\CheckModules
```
8094 \@emptify\CheckModules␣\noeffect@info{CheckModules}
```

\Module

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

\AltMacroFont
```
8098 \@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
```
8104 \def\MakePercentIgnore{\catcode`\%9\relax}
```
\MakePercentComment
```
8105 \def\MakePercentComment{\catcode`\%14\relax}
```

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

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

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

TEXing the original doc with my humble[14] package was a challenge and a milestone experience in my TEX life.

One of minor errors was caused by my understanding of a 'shortverb' char: due to gmverb, in the math mode an active 'shortverb' char expands to itself's 'other' version thanks to `\string` (It's done with `|` in mind). doc's concept is different, there a 'shortverb' char should in the math mode work as shortverb. So let it be as they wish: gmverb provides `\OldMakeShortVerb` and the old-style input commands change the inner macros so that also `\MakeShortVerb` works as in doc (cf. line 8146).

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`

```
8143 \def\OldDocInput{%
8145   \AtBegInputOnce{\StraightEOL
8146     \let\@MakeShortVerb=\old@MakeShortVerb
8148     \OldMacrocodes}%
8149   \bgroup\@makeother\_%  it's to allow _ in the filenames. The next macro will
                                close the group.
8151   \Doc@Input}
```

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

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

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

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

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

```
8184 %␣\AVerySpecialMacro␣%␣delete␣the␣first␣%␣when. ..
```

was got with

14 What a *false* modesty! ;-)

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

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

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

\OCRInclude

I realised that I want to print all my TEX source files verbatim just in case my computers and electronic memories break so that I can reconstruct them via OCR . For this purpose I provide `\OCRInclude`. It takes the same arguments as `\DocInclude` only typesets a file with no index nor line numbers.

\OCRInclude
\incl@DocInput

```
\DeclareCommand\OCRInclude{O{}mO{}}{%
  \StoreMacro\incl@DocInput
  \def\incl@DocInput##1{%
    \begingroup
    \CodeSpacesBlank
    \@beginputhook
    \title{\currentfile}\maketitle
    \noverbatimspecials
    \relaxen\@xverbatim
    \relaxen\check@percent
    \RestoreMacro\@verbatim
    \verbatimleftskip\z@skip
    \verbatim
    \@makeother\|% because \ttverbatim doesn't do that.
    \texcode@hook% we add some special stuff, e.g. in gmdocc.cls we
    \@input{##1}%
    \endgroup}%
  \csname\string\DocInclude\endcsname{#1}{#2}{#3}%
  \RestoreMacro\incl@DocInput
}
```

Polishing, development and bugs

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

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

• Marcin Woliński suggests to add the marginpar clauses for the AMS classes as we did for the standard ones in the lines 2180–2185. Most probably I can do it on request when I only know the classes' names and their 'marginpar status'.

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

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

• Is `\RecordChanges` really necessary these days? Shouldn't be the `\makeglossary` command rather executed by default?[15]

• Do you use `\listoftables` and/or `\listoffigures` in your documentations? If so, I should 'EOL-straighten' them like `\tableofcontents`, I suppose (cf. line 2630).

• Some lines of non-printing stuff such as `\Define`… and `\changes` connecting the narration with the code resulted with unexpected large vertical space. Adding a fully blank line between the printed narration text and not printed stuff helped.

• Specifying `codespacesgrey,␣codespacesblank` results in typesetting all the spaces grey including the leading ones.

• About the DocStrip verbatim mode directive see above.

[No] ⟨*eof*⟩

Until version 0.99i a file that is `\DocInput` had to be ended with a comment line with an `\EOF` or `\NoEOF` CS that suppressed the end-of-file character to make input end properly. Since version 0.99i however the proper ending of input is achieved with `\everyeof` and therefore `\EOF` and `\NoEOF` become a bit obsolete.

If the user doesn't wish the documentation to be ended by '⟨*eof*⟩', they should redefine the `\EOFMark` CS or end the file with a comment ending with `\NoEOF` macro defined below[16]:

```
8306 \foone{\catcode`\^^M\active␣}{%
\@NoEOF 8307   \def\@NoEOF#1^^M{%
8308     \@relaxen\EOFMark\endinput}%
 \@EOF  8309   \def\@EOF#1^^M{\endinput}}

\NoEOF  8311 \def\NoEOF{\QueerEOL\@NoEOF}
 \EOF   8312 \def\EOF{\QueerEOL\@EOF}
```

As you probably see, `\[No]EOF` have the 'immediate' `\endinput` effect: the file ends even in the middle of a line, the stuff after `\(No)EOF` will be gobbled unlike with a bare `\endinput`.

```
8325 \endinput
8327 ⟨/ package⟩
```

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

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

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

Written by Natror (Grzegorz Murzynowski),
natror at 02 dot pl


```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{gmdocc}
              [2010/03/02 v0.84 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.

noindex

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.

nochanges

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

1 This file has version number v0.84 dated 2010/03/02.

Both of the above options turn the *writing out to the files* off. They don't turn off `\PrintIndex` nor `\PrintChanges`. (Those two commands are no-ops by themselves if there's no .ind (n)or .gls file respectively.)

outeroff

One more option is `outeroff`. It's intended for compiling the documentation of macros defined with the `\outer` prefix. It `\relax`es this prefix so the '`\outer`' macros' names can appear in the arguments of other macros, which is necessary to pretty mark and index them.

I decided not to make discarding `\outer` the default because it seems that LaTeX writers don't use it in general and gmdoc.sty *does* make some use of it.

debug

This class provides also the `debug` option. It turns the `\if@debug` Boolean switch True and loads the trace package that was a great help to me while debugging gmdoc.sty.

The default base document class loaded by gmdocc.cls is Marcin Woliński mwart. If you have not installed it on your computer, the standard article will be used.

Moreover, if you like MW's classes (as I do) and need `\chapter` (for multiple files' input e.g.), you may declare another mwcls with the option homonymic with the class's name: `mwrep` for mwrep and `mwbk` for mwbk. For the symmetry there's also `mwart` option (equivalent to the default setting).

mwrep
mwbk
mwart

The existence test is done for any MW class option as it is in the default case.

Since version 0.99g (November 2007) the bundle goes XeTeX and that means you can use the system fonts if you wish, just specify the `sysfonts` option and the three basic XeTeX-related packages (fontspec, xunicode and xltxtra) will be loaded and then you can specify fonts with the fontspec declarations. For use of them check the driver of this documentation where the TeX Gyre Pagella font is specified as the default Roman.

sysfonts

There are also some options for mono and sans fonts, see the changes history for details.

minion

The `minion` option sets Adobe Minion Pro as the main font, the `pagella` sets TeX Gyre Pagella as the main font.

pagella

cronos

The `cronos` option sets Adobe Cronos Pro as the sans serif font, the `trebuchet` option sets MS Trebuchet as sans serif.

trebuchet

cursor

The `cursor` (working only with XeTeX & fontspec) option sets TeX Gyre Cursor as the typewriter font. It emboldens it to the optical weight of Computer/Latin Modern Mono in the code (`embolden=2.5`) and leaves light (`embolden=1`) for verbatims in the narrative. Moreover, this option also prepares a condensed version (`extend=0.87`) for verbatims in the marginpars.

Note that with no option for the monospaced font the default (with XeTeX) will be Latin Modern Mono and then Latin Modern Mono Light Condensed is set for verbatims in marginpars (if available).

\verbatimspecials

This class sets `\verbatimspecials/«»[¿]` if the engine is XeTeX, see the gmverb documentation to learn about this declaration. Remember that `\verbatimspecials` whatever would they be, have no effect on the code layer.

\EOFMark

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

□

The Code

178 `\RequirePackage{xkeyval}`

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

```
\gm@DOX 183 \newcommand*\gm@DOX{\DeclareOptionX[gmcc]<>}
\gm@EOX 184 \newcommand*\gm@EOX{\ExecuteOptionsX[gmcc]<>}
```

We define the `class` option. I prefer the mwcls, but you can choose anything else, then the standard article is loaded. Therefore we'd better provide a Boolean switch to keep the score of what was chosen. It's to avoid unused options if article is chosen.

```
\ifgmcc@mwcls 193 \newif\ifgmcc@mwcls
```

Note that the following option defines `\gmcc@class#1`.

```
class 196 \gm@DOX{class}{% the default will be Marcin Woliński class (mwcls) analogous to
          article, see line 357.
\gmcc@CLASS 198   \def\gmcc@CLASS{#1}%
        199   \@for\gmcc@resa:=mwart,mwrep,mwbk\do␣{%
        200     \ifx\gmcc@CLASS\gmcc@resa\gmcc@mwclstrue\fi}%
        201 }
mwart 203 \gm@DOX{mwart}{\gmcc@class{mwart}}% The mwart class may also be de-
          clared explicitly.
mwrep 206 \gm@DOX{mwrep}{\gmcc@class{mwrep}}% If you need chapters, this option
          chooses an MW class that corresponds to report,
mwbk 210 \gm@DOX{mwbk}{\gmcc@class{mwbk}}% and this MW class corresponds to book.
article 213 \gm@DOX{article}{\gmcc@class{article}}% you can also choose article. A meta-
          remark: When I tried to do the most natural thing, to \ExecuteOptionsX
          inside such declared option, an error occurred: 'undefined control sequence
          % \XKV@resa␣->␣\@nil'.
outeroff 221 \gm@DOX{outeroff}{\let\outer\relax}% This option allows \outer-prefixed
          macros to be gmdoc-processed with all the bells and whistles.
\if@debug 225 \newif\if@debug
debug 227 \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 232 \gm@DOX{noindex}{%
        233   \PassOptionsToPackage{noindex}{gmdoc}}% This option turns the writ-
              ing out to .idx file off.
\if@gmccnochanges 237 \newif\if@gmccnochanges
nochanges 239 \gm@DOX{nochanges}{\@gmccnochangestrue}% This option turns the writing
          out to .glo file off.
```

Since version 0.99g the gmdoc bundle goes XeTeX. That means that if XeTeX is detected, we may load the fontspec package and the other two of basic three XeTeX-related, and then we `\fontspec` the fonts. But the default remains the old way and the new way is given as the option below.

```
\ifgmcc@oldfonts 263 \newif\ifgmcc@oldfonts
sysfonts 265 \gm@DOX{sysfonts}{\gmcc@oldfontsfalse}
mptt 274 \gm@DOX{mptt}[17]{\relax}% now a no-op, left only for backwards compati-
          bility. It was an option for setting the marginpar typewriter font.
\gmcc@tout 284 \def\gmcc@tout#1{\typeout{^^J@@@@␣gmdocc␣class:␣#1^^J}}
```

```
\gmcc@setfont 286 \def\gmcc@setfont#1{%
              287   \gmcc@oldfontsfalse% note that if we are not in XeTeX, this switch will be
                        turned true in line 415
              289   \AtEndOfClass{%
              290     \ifdefined\zf@init\afterfi{%
              291       \gmcc@tout{Main␣font␣set␣to␣#1}%
   \gmcc@dff  292       \def\gmcc@dff{Numbers={OldStyle,␣Proportional}}
              293       \@xa\setmainfont\@xa[\gmcc@dff,␣Mapping=tex-text]{%
                            #1}%
              303       \@xa\defaultfontfeatures\@xa{\gmcc@dff,␣Scale=MatchLowercase}%
                            when put before \setmainfont,
              305       \gmath
\LineNumFont  306       \def\LineNumFont{%
              307         \normalfont\scriptsize\addfontfeature{%
                              Numbers=Monospaced}}%
              308     }%
              309     \else\afterfi{\gmcc@tout{I~can␣set␣main␣font␣to␣#1␣
                          only␣in
              310         XeTeX/fontspec}}%
              311     \fi
              312   }}
    minion    314 \gm@DOX{minion}{\gmcc@setfont{Minion␣Pro}}
   pagella    315 \gm@DOX{pagella}{\gmcc@setfont{TeX␣Gyre␣Pagella}
              317 }
   cronos     318 \gm@DOX{cronos}{%
              319   \AtEndOfClass{\setsansfont[Mapping=tex-text]{Cronos␣Pro}}}
 trebuchet    320 \gm@DOX{trebuchet}{%
              322   \AtEndOfClass{\setsansfont[Mapping=tex-text]{Trebuchet␣
                        MS}}}
   myriad     323 \gm@DOX{myriad}{%
              325   \AtEndOfClass{\setsansfont[Mapping=text-text]{Myriad␣Web␣
                        Pro}}}
      lsu     326 \gm@DOX{lsu}{%
              328   \AtEndOfClass{\setsansfont[Mapping=tex-text]{Lucida␣Sans␣
                        Unicode}}}
   cursor     330 \gm@DOX{cursor}{%
              336   \AtEndOfClass{%
              337     \setmonofont[FakeBold=2.5,␣BoldFeatures={FakeBold=0},
              338     FakeStretch=0.87,␣Ligatures=NoCommon
              339     ]{TeX␣Gyre␣Cursor}%
\marginpartt  340     \def\marginpartt{\tt\addfontfeature{FakeBold=2,
              341         FakeStretch=0.609}%
              342       \color{black}}% to provide proper color when marginpar occurs be-
                            tween lines that break a coloured text.
\narrativett  344     \def\narrativett{\ttfamily\addfontfeature{FakeBold=1}}%
              345     \let\UrlFont\narrativett
              346   }% of \AtEndOfClass.
              347 }% of the cursor option.
  fontspec    353 \gm@DOX{fontspec}{\PassOptionsToPackage{#1}{fontspec}}
              357 \gm@EOX{class=mwart}% We set the default basic class to be mwart.
              363 \PassOptionsToPackage{countalllines}{gmdoc}%
```

```
\DeclareOptionX*{\PassOptionsToPackage{\CurrentOption}{gmdoc}}
\ProcessOptionsX[gmcc]<>
\ifgmcc@mwcls
  \IfFileExists{\gmcc@CLASS.cls}{}{\gmcc@mwclsfalse}% As announced, we do the ontological test to any mwcls.
\fi
\ifgmcc@mwcls
  \LoadClass[fleqn, oneside, noindentfirst, 11pt, withmarginpar,
  sfheadings]{\gmcc@CLASS}%
\else
  \LoadClass[fleqn, 11pt]{article}% Otherwise the standard article is loaded.
\fi
\RequirePackage{gmutils}[2008/10/08]% we load it early to provide \@ifXeTeX, but after loading the base class since this package redefines some environments.
\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. Some why the accents didn't want to work at my XeTeX settings so below I define them for XeTeX as respective chars.

```
\agrave  \def\agrave{\`a}%
\cacute  \def\cacute{\'c}%
\eacute  \def\eacute{\'e}%
\idiaeres  \def\idiaeres{\"\i}%
\nacute  \def\nacute{\'n}%
\ocircum  \def\ocircum{\^o}%
\oumlaut  \def\oumlaut{\"o}%
\uumlaut  \def\uumlaut{\"u}%
\else% this case happens only with XeTeX.
  \let\do\relaxen
  \do\Finv\do\Game\do\beth\do\gimel\do\daleth% these five caused the 'already defined' error.
```

```
    \let\@zf@euenctrue\zf@euencfalse
     \XeTeXthree%
\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{geometry}
\gmdoccMargins@params  \def\gmdoccMargins@params{{top=77pt,␣height=687pt,␣% =53 lines but
       the lines option seems not to work 2007/11/15 with TeX Live 2007 and
       XeTeX0.996-patch1
       left=4cm,␣right=2.2cm}}
\gmdoccMargins  \def\gmdoccMargins{%
    \@xa␣␣\newgeometry\gmdoccMargins@params}

\@xa\geometry\gmdoccMargins@params

\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
\marginpartt    \def\marginpartt{\fontspec{LMTypewriter10␣
          LightCondensed}}%
\fi

\raggedbottom

\setcounter{secnumdepth}{0}% We wish only the parts and chapters to be
       numbered.
```

\thesection
```
\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.
```

\titlesetup
```
\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 out to the .glo file.
```

```
\RecordChanges% We turn the writing the \changes out to 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 underscore.
```

```
\exhyphenpenalty\hyphenpenalty% 'cause mwcls set it =10000 due to Polish customs.
```

```
\RequirePackage{amssymb}
```

\EOFMark
```
\def\EOFMark{\rightline{\ensuremath{\square}}}
```

```
\DoNotIndex{\@nx␣\@xa␣%
}
```

\ac
```
\provide\ac{\acro}
```

\+
```
\def\+{\-\hskip\z@\penalty\@M}␣% a discretionary hyphen that allows further hyphenation
```

```
\Xedekfracc
```

```
\let\mch\metachar
```

```
\ATfootnotes
\AtBegInput{\ATfootnotes}
```

```
\UrlFix
```

```
\GMverbatimspecials
```

```
\let\texcode@hook\makestarlow
```

```
\endinput
```

c. 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}
      [2010/03/04 v0.991 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 omitted.

Contents of the gmutils.zip archive

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

gmutils.sty
README
gmutils.pdf
gmutils.tds.zip

```
\ifx\XeTeXversion\relax
  \let\XeTeXversion\@undefined% If someone earlier used
      % \@ifundefined{XeTeXversion} to test whether the engine is XeTeX,
      then \XeTeXversion is defined in the sense of ε-TeX tests. In that case
      we \let it to something really undefined. Well, we might keep sticking
```

1 This file has version number v0.991 dated 2010/03/04.

to \@ifundefined, but it's a macro and it eats its arguments, freezing their catcodes, which is not what we want in line 6155.

```
\fi
\ifdefined\XeTeXversion
\XeTeXinputencoding␣utf-8␣%  we use Unicode dashes later in this file.
\fi%  and if we are not in XeTeX, we skip them thanks to XeTeX-test.
```

Options

\ifgmu@quiet
quiet

```
\newif\ifgmu@quiet
\DeclareOption{quiet}{\gmu@quiettrue}
\ProcessOptions
```

A couple of abbreviations

\@xa
\@nx

```
\let\@xa\expandafter
\let\@nx\noexpand
```

\@xau

```
\def\@xau#1{\unexpanded\@xa{#1}}
```

Note that there's only one \expandafter, after \unexpanded. It's because \unexpanded expands expandable tokens and gobbles \relaxes while looking for an opening brace or \bgroup. Note also that it *requires* a text in braces.

Note also that (since v0.991) this is a 1-parameter macro so doesn't expand subsequent tokens until meets a ⟨*balanced text*⟩ but just takes first single token or ⟨*text*⟩.

\pdef

```
\def\pdef{\protected\def}
```

And this one is defined, I know, but it's not \long with the standard definition and I want to be able to \gobble a \par sometimes.

\gobble
\@gobble
\gobbletwo

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

\provide

```
\long\pdef\provide#1{%
  \ifdefined#1%
    \ifx\relax#1\afterfifi{\def#1}%
    \else\afterfifi{\gmu@gobdef}%
    \fi
  \else\afterfi{\def#1}%
  \fi}
```

\gmu@gobdef

```
\long\def\gmu@gobdef#1#{%
  \def\gmu@tempa{}%  it's a junk macro assignment to absorb possible prefixes.
  \@gobble}
```

\pprovide

```
\def\pprovide{\protected\provide}
```

Note that both \provide and \pprovide may be prefixed with \global, \outer, \long and \protected because the prefixes stick to \def because all before it is expandable. If the condition(s) is false (#1 is defined) then the prefixes are absorbed by a junk assignment.

Note moreover that unlike LaTeX's \providecommand, our \(p)provide allow any parameters string just like \def (because they just *expand* to \def).

```
\@nameedef  282 \long\def\@nameedef#1#2{%
            283   \@xa\edef\csname#1\endcsname{#2}}
```

\ifs as a four-argument LaTeX command robust to unbalanced \ifs and \fis

```
\gmu@if     287 \long\def\gmu@if#1#2{%
            288   \csname␣if#1\endcsname#2\@xa\@firstoftwo\else\@xa%
                      \@secondoftwo\fi
            289 }
```

\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.

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

```
\scantwo    304 \long\pdef\scantwo#1#2{
            305   \begingroup\endlinechar\m@ne
            306   \@xa\endgroup\scantokens{#1#2}%
            307 }
```

```
\@firstofthree   309 \long\def\@firstofthree#1#2#3{#1}
\@secondofthree  310 \long\def\@secondofthree#1#2#3{#2}
\@thirdofthree   311 \long\def\@thirdofthree#1#2#3{#3}
\@secondoffive   312 \long\def\@secondoffive#1#2#3#4#5{#2}
```

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

```
\@allbutfirstof 322 \long\def\@allbutfirstof#1#2\@@nil{#2}
```

\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 \afterfi and pals are sensitive to \fis that may occur in macros' arguments so probably the safest and expandable way is the \expandafter\@(first|second)oftwo trick.

```
\longafterfi  353 \def\longafterfi{%
```

```
\afterfi   354   \long\def\afterfi##1##2\fi{\fi##1}}
           355 \longafterfi
```

And two more of that family:

```
\afterfifi     357 \long\def\afterfifi#1#2\fi#3\fi{\fi\fi#1}
\afteriffifi   359 \long\def\afteriffifi#1#2\fi#3\fi{\fi#1}
```

Notice the refined elegance of those macros, that cover both 'then' and 'else' cases thanks to `#2` that is discarded.

```
\afterififfifiifi   363 \long\def\afterififfifiifi#1#2\fi#3\fi#4\fi{\fi#1}
\afteriffifiifi     364 \long\def\afteriffifiifi#1#2\fi#3\fi#4\fi{\fi\fi#1}
\afterfifiifi       365 \long\def\afterfifiifi#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}
```

\foone

The next command, `\foone`, is intended as two-argument for shortening of the `\begingroup...% \firstofone{\endgroup...}` hack.

```
\foone       372 \long\def\foone#1{\begingroup#1\relax\egfirstofone}
\egfirstofone 374 \long\def\egfirstofone#1{\endgroup#1}
\fooatletter 376 \def\fooatletter{\foone\makeatletter}
```

\@ifempty, \IfAmong, \IfIntersect \@ifinmeaning

After a short deliberation I make the `\IfAmong`...`\among` and `\IfIntersect` macros' names apeless since they are intended for the macro and class writers, at the same level of abstraction as `\DeclareCommand`.

```
\@ifempty    386 \long\pdef\@ifempty#1{%
                   % #1 the token(s) we test, it may be just {},
                   % #2 the stuff executed if #1 is empty (is {}),
                   % #3 the stuff executed if #1 consists of at least one token.
\gmu@reserveda 392   \def\gmu@reserveda{#1}%
               393   \ifx\gmu@reserveda\@empty\@xa\@firstoftwo
               394   \else\@xa\@secondoftwo
               395   \fi}
\@ifnonempty 397 \long\def\@ifnonempty#1#2#3{\@ifempty{#1}{#3}{#2}}
\IfAmong     399 \long\pdef\IfAmong#1\among#2{%
                   % #1 the token(s) whose presence we check,
                   % #2 the list of tokens in which we search #1,
                   % #3 the 'if found' stuff,
                   % #4 the 'if not found' stuff.
\gmu@among@  408   \long\def\gmu@among@##1#1##2\gmu@among@{%
             409     \@ifempty{##2}\@secondoftwo\@firstoftwo}%
             410   \gmu@among@#2#1\gmu@among@}
\IfIntersect 412 \long\def\IfIntersect#1#2{% this (not expandable) macro checks whether
                   the list of tokens #1 and #2 have nonempty intersection and if so executes
                   % \@firstoftwo or else \@secondoftwo, so it's a 4-argument com-
                   mand:
                   % #1 first list to match,
```

```
                % #2 second list to match,
                % #3 if match (nonempty intersection),
                % #4 if not match (empty intersection).
422   \let\IfIntersect@next\@secondoftwo
423   \gmu@foreach#1\gmu@foreach{%
424     \@xa\IfAmong\gmu@forarg\among{#2}
425     {\let\IfIntersect@next\@firstoftwo}{}}%
426   \IfIntersect@next}
```

We need a macro that iterates over every token on a list. LaTeX's `\@for` iterates over a list separated with commas so it's not the case. Our macro is much simpler.

```
\gmu@foreach  431 \long\def\gmu@foreach#1\gmu@foreach#2{%
 \gmu@forer   432   \long\def\gmu@forer##1{%
              433     \ifx\gmu@foreach##1\else
 \gmu@forarg  434     \long\def\gmu@forarg{##1}%
              435     #2%
              436     \@xa\gmu@forer
              437     \fi}%
              438   \gmu@forer#1\gmu@foreach}
```

```
\@ifinmeaning  442 \long\pdef\@ifinmeaning#1\of#2{%
                        % #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.
\gmu@reserveda  458   \def\gmu@reserveda{\IfAmong#1\among}%
                459   \@xa\gmu@reserveda\@xa{#2}}
```

Global Boolean switches

The `\newgif` declaration's effect is used even in the LaTeX 2_ε source by redefining some particular user defined ifs (UD-ifs henceforth) step by step. The goal is to make the UD-if's assignment global. I needed it at least twice during gmdoc writing so I make it 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  472 \pdef\newgif#1{%
         473   {\escapechar\m@ne
         474     \global\let#1\iffalse
         475     \@gif#1\iftrue
         476     \@gif#1\iffalse
         477   }}
```

'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  483 \def\@gif#1#2{%
       484   \protected\@xa\gdef\csname\@xa\@gobbletwo\string#1%
       485   g% the letter g for '\global'.
       486   \@xa\@gobbletwo\string#2\endcsname
       487   {\global\let#1#2}}
```

```
\pdef\newif#1{% We not only make \newif \protected but also make it to
        define \protected assignments so that premature expansion doesn't
        affect \if...\fi nesting.
  \count@\escapechar␣\escapechar\m@ne
  \let#1\iffalse
  \@if#1\iftrue
  \@if#1\iffalse
  \escapechar\count@}
```

\@if
```
\def\@if#1#2{%
  \protected␣\@xa\def\csname\@xa\@gobbletwo\string#1%
  \@xa\@gobbletwo\string#2\endcsname
  {\let#1#2}}
```

\hidden@iffalse
\hidden@iftrue
```
\pdef\hidden@iffalse{\iffalse}
\pdef\hidden@iftrue{\iftrue}
```

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
```
\pdef\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.

2008/08/10 I spent all the night debugging `\penalty 10000` that was added after a hypertarget in vertical mode. I didn't dare to touch hyperref's guts, so I worked it around with ensuring every `\grefstepcounter` to be in hmode:

\hgrefstepcounter
```
\pdef\hgrefstepcounter#1{%
  \ifhmode\leavevmode\fi\grefstepcounter{#1}}
```

By the way I read some lines from *The TEXbook* and was reminded that `\unskip` strips any last skip, whether horizontal or vertical. And I use `\unskip` mostly to replace a blank space with some fixed skip. Therefore define

\hunskip
```
\pdef\hunskip{\ifhmode\unskip\fi}
```

Note the two macros defined above are `\protected`. I think it's a good idea to make `\protected` all the macros that contain assignments. There is one more thing with `\ifhmode`: it can be different at the point of `\edef` and at the point of execution.

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{%
  \edef#1{\@xa\unexpanded\@xa{#1}\unexpanded{#2}}%  \edef and \unexpanded to double the hashes.
}
```

And for use in the very document,

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

2008/08/09 I need to prepend something not add at the end—so

\prependtomacro
```
\long\def\prependtomacro#1#2{%
  \edef#1{\unexpanded{#2}\@xa\unexpanded\@xa{#1}}}
```

Note that `\prependtomacro` can be prefixed.

\addtotoks
```
\long\def\addtotoks#1#2{%
  #1=\@xa{\the#1#2}}
```

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

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

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

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

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

\g@relaxen
\grelaxen
```
\newcommand*\g@relaxen{\global\@relaxen}
\@ifdefinable\grelaxen{\let\grelaxen\g@relaxen}
```

`\gm@ifundefined`—a test that doesn't create any hash entry unlike `\@ifundefined`

I define it under another name not redefine `\@ifundefined` because I can imagine an odd case when something works thanks to `\@ifundefined`'s 'relaxation' effect.

\gm@ifundefined
```
\long\def\gm@ifundefined#1{%  not \protected because expandable.
  \ifcsname#1\endcsname%  defined…
    \@xa\ifx\csname␣#1\endcsname\relax%  but only as \relax—then 2nd argument.
      \afterfifi\@firstoftwo%
    \else%  defined and not \relax—then 3rd argument.
      \afterfifi\@secondoftwo%
    \fi
  \else%  not defined—then 3rd.
    \afterfi\@firstoftwo%
```

```
  \fi}
```

\gm@testdefined

```
\long\def\gm@testdefined#1\iftrue{%
```

This is a macro that expands to \iftrue or \iffalse depending on whether its argument is defined in the LaTeX sense. Its syntax requires an \iftrue to force balancing \ifs with \fis.

```
  \csname
  \ifdefined#1%
    \ifx#1\relax
       iffalse%
    \else␣iftrue%
    \fi
  \else␣iffalse%
  \fi\endcsname
}
```

\gm@testundefined

```
\long\def\gm@testundefined#1\iftrue{%
```

we expand the last macro two levels. We repeat the parameter string to force the proper syntax.

```
  \@xa\@xa\@xa\unless\gm@testdefined#1\iftrue}
```

Some 'other' and active stuff

Here I define a couple of macros expanding to special chars made 'other'. It's important the CS are expandable and therefore they can occur e.g. inside \csname…\endcsname unlike e.g. CS'es \chardefed.

```
\foone{\catcode`\_=8␣}
```

\subs

```
{\let\subs=_}
```

```
\foone{\catcode`\^=7␣}
```

\sups

```
{\let\sups=^}
```

```
\foone{\@makeother\_}
```

\xiiunder

```
{\def\xiiunder{_}}
```

```
\let\all@unders\xiiunder
\foone{\catcode`\_=8␣}
{\addtomacro\all@unders{_}}
\foone{\catcode`\_=11␣}
{\addtomacro\all@unders{_}}
\foone{\catcode`\_=13␣}
{\addtomacro\all@unders{_}}
```

Now \all@unders bears underscores of categories 8, 11, 12 and 13.

\all@stars

```
\def\all@stars{*}
\foone{\catcode`\*=11␣}
{\addtomacro\all@stars{*}}
\foone{\catcode`\*=13␣}
{\addtomacro\all@stars{*}}
```

And \all@stars bears stars of categories 11, 12 and 13.

```
\ifdefined\XeTeXversion
  \chardef\_="005F
\fi
```

```
\foone{\@makeother\`}%
```

\backquote

```
{\def\backquote{`}}
```

```
\foone{\catcode`\[=1 \@makeother\{
  \catcode`\]=2 \@makeother\}}%
[%
  \def\xiilbrace[{]%
  \def\xiirbrace[}]%
]% of \firstofone
```

Note that LaTeX's `\@charlb` and `\@charrb` are of catcode 11 ('letter'), cf. The LaTeX 2ε Source file k, lines 129–130.

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}
{%
  \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\\}
{!newcommand*!xiibackslash{\}}
```

```
\let\bslash=\xiibackslash
```

```
\foone{\@makeother\%}
{\def\xiipercent{%}}
```

```
\foone{\@makeother\&}%
{\def\xiiand{&}}
```

```
\foone{\@makeother\␣}%
{\def\xiispace{␣}}
```

```
\foone{\@makeother\#}%
{\def\xiihash{#}}
```

We introduce `\visiblespace` from Will Robertson's xltxtra if available. It's not sufficient `\@ifpackageloaded{xltxtra}` since `\xxt@visiblespace` is defined only unless `no-verb` option is set. 2008/08/06 I recognised the difference between `\xiispace` which has to be plain 'other' char (used in `\xiistring`) and something visible to be printed in any font.

```
\AtBeginDocument{%
  \ifdefined\xxt@visiblespace
    \let\visiblespace\xxt@visiblespace
\@visiblespace@fallback    \def\xxt@visiblespace@fallback{{%
        \fontspec{Latin␣Modern␣Mono}\textvisiblespace}}%
  \else
    \let\visiblespace\xiispace
  \fi}
```

```
\foone\obeyspaces{\def\gmu@activespace{␣}}
```

```
\foone\obeylines{\def\activeM{^^M}}
```

\@ifnextcat, \@ifnextac, catcode-independent \gm@ifstar, \@ifnextsingle, \@ifnextgroup

As you guess, we \def \@ifnextcat à la \@ifnextchar, see LaTeX 2ε source dated 2003/12/01, file d, lines 253–271. The difference is in the kind of test used: while \@ifnextchar does \ifx, \@ifnextcat does \ifcat which means it looks not at the meaning of a token(s) but at their \catcode(s). As you (should) remember from *The TeXbook*, the former test doesn't expand macros while the latter does. But in \@ifnextcat the peeked token is protected against expanding by \noexpand. Note that the first parameter is not protected and therefore it shall be expanded if it's expandable. Because an assignment is involved, you can't test whether the next token is an active char. But you can test if the next token is {$_1$ or }$_2$: \@ifnextcat\bgroup..., \@ifnextcat\egroup....

\@ifnextcat

```
\long\pdef\@ifnextcat#1#2#3{%
  \def\reserved@d{#1}%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
  \futurelet\@let@token\@ifncat}
```

\@ifncat

```
\def\@ifncat{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifncat
  \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=␣}␣\global\:␣%
```

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.

Having \@ifnextcat defined, let's apply it immediately. Similar thing does probably the xspace package.

\spifletter

```
\def\spifletter{\@ifnextcat␣a{\space}{}}
```

\@ifnextif

```
\long\pdef\@ifnextif#1#2#3{%
```

(the future token in \noexpanded, unlike #1)

```
  \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␣% #1 of \@ifnextif is not \if-equivalent the future token. But this
          may be because the future token is active so it would be \if-equivalent
          if not passed through \futurelet. Let's manage this case.
      \begingroup
      \edef\gmu@tempa{%
        \lccode`\@nx~=`\reserved@d
      }\gmu@tempa
      \lowercase{\endgroup
        \ifx~}\@let@token
        \let\reserved@c\reserved@a
      \else
        \let\reserved@c\reserved@b
      \fi
    \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\pdef\@ifnextac#1#2{%
  \@ifnextsingle
  {\gm@ifnac{#1}{#2}}%
  {#2}}
```

\gm@ifnac
```
\long\def\gm@ifnac#1#2#3{%
  \ifcat\@nx~\@nx#3%
    \@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi{#1#3}{#2#3}}
```

Yes, it won't work for an active char `\let` to {$_{11}$, 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` not just recatcode it to $_1$?)

Having defined such tools, let's redefine `\gm@ifstar` to make it work with whatever catcode * may be. make a version of `\gm@ifstar` that would work with $\star_{11}$.

\gmu@lowstar
```
\pdef\gmu@lowstar{%
  \iffontchar\font"22C6␣\char"22C6␣\else\gmu@lowstarfake\fi}% we
        could define it to be expandable (with ^^^^22c6) but the only goal of it
        would be allowing this low star in \csname…\endcsname. But it would
        be misleading (not everyone can easily distinguish * from ⋆) so IMHO it's
        better to raise an error should such an active occur in a \csname. This
```

macro is intended to be \let to an active star only in verbatims. For usual text there's

Commands around making star low (via \activeation) are defined in line 7048 because they use \DeclareCommand.

```
\def\gmu@tempa{*}
\foone{\catcode`\*=\active}
{\def\gmu@tempb{*}%
```

it's defined in line ?? to make * defined (when it was undefined, \newcommand's \gm@ifstar test turned true, the next undefined token was gobbled and raised an error).

```
  \let*=\gmu@lowstar}

\edef\gmu@tempa{%
  \long\pdef\@nx\gm@ifstar##1##2{% ]
    \@nx\@ifnextif\gmu@tempa%
    {\@nx\@firstoftwo{##1}}%
    {%
      \@nx\@ifnextchar\@xa\@nx\gmu@tempb
      {\@nx\@firstoftwo{##1}}{##2}}%
  }% of \gm@ifstar.
}\gmu@tempa
```

A test whether we can pick a single token. We have to check whether we are not next to { and whether we are not next to }.

\@ifnextsingle

```
\long\pdef\@ifnextsingle#1#2{%
```

This macro checks whether the next token is able to be picked or is it a braced list of tokens or is it a group closer so there's no token to be picked.

```
  \@ifnextcat\bgroup{#2}{%
    \@ifnextcat\egroup{#2}%
    {#1}}}
```

\@ifnextgroup

```
\long\pdef\@ifnextgroup#1#2{%
```

Note this macro turns true both before a group opener and before a group closer.

```
  \@ifnextsingle{#2}{#1}}
```

now let's apply this to sth. useful (used in gmverse and in some typesetting, e.g. for prof. JSB.

\ignoreactiveM

```
\pdef\ignoreactiveM{%
  \@ifnextgroup{}{\gmu@checkM}}

\foone\obeylines{%
```

we know it's a single token since we use this macro only in \@ifnextgroup's 'else'.

\gmu@checkM

```
  \long\pdef\gmu@checkM#1{%
    \ifx
    #1\@xa\ignoreactiveM%
    \else\@xa#1\fi}}
```

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\gmu@reserveda\*%
  \def\*{%
    \let\*\gmu@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.

Now a test if the next token is an active line end. I use it in gmdoc and later in this package for active long dashes.

\@ifnextMac
```
\foone\obeylines{%
  \long\pdef\@ifnextMac#1#2{%
    \@ifnextchar^^M{#1}{#2}}}
```

Storing and restoring the meanings of CSes

First a Boolean switch of globalness of assignments and its verifier.

\ifgmu@SMglobal
```
\newif\ifgmu@SMglobal
```

\SMglobal
```
\pdef\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 meaning back (when grouping is not recommended):

\StoreMacro
```
\pdef\StoreMacro{%
  \begingroup\makeatletter\gm@ifstar\egStore@MacroSt%
      \egStore@Macro}
```

The unstarred version takes a CS and the starred version a text, which is intended for special control sequences. For storing environments there is a special command in line 1177.

\egStore@Macro
```
\long\def\egStore@Macro#1{\endgroup\Store@Macro{#1}}
```
\egStore@MacroSt
```
\long\def\egStore@MacroSt#1{\endgroup\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.
1055      \@xa\@nx\csname#1\endcsname}
1056    \gmu@smtempa
1057    \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 CSes and stores each of them. The CS'es may be separated with commas but they don't have to.

```
\StoreMacros  1073 \long\pdef\StoreMacros{\begingroup\makeatletter\Store@Macros}
\Store@Macros 1074 \long\def\Store@Macros#1{\endgroup
1075    \gmu@setsetSMglobal
1076    \let\gml@StoreCS\Store@Macro
1077    \gml@storemacros#1.}

\gmu@setsetSMglobal 1080 \def\gmu@setsetSMglobal{%
1081    \ifgmu@SMglobal
1082      \let\gmu@setSMglobal\gmu@SMglobaltrue
1083    \else
1084      \let\gmu@setSMglobal\gmu@SMglobalfalse
1085    \fi}
```

And the inner iterating macro:

```
\gml@storemacros 1088 \long\def\gml@storemacros#1{%
\gmu@reserveda   1089    \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.
1092    \if\gmu@reserveda.%  a dot finishes storing.
1093      \global\gmu@SMglobalfalse
1094    \else
1095      \if\gmu@reserveda,%  The list this macro is put before may contain com-
                        mas and that's O.K., we just continue the work.
1097        \afterfifi\gml@storemacros
1098      \else%  what is else this shall be stored.
1099        \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.
1102        \afterfifi{\gmu@setSMglobal\gml@storemacros}%
1103      \fi
1104    \fi}
```

And for the restoring

```
\RestoreMacro 1110 \pdef\RestoreMacro{%
1111    \begingroup\makeatletter\gm@ifstar\egRestore@MacroSt%
                \egRestore@Macro}

\egRestore@Macro   1113 \long\def\egRestore@Macro#1{\endgroup\Restore@Macro{#1}}
\egRestore@MacroSt 1114 \long\def\egRestore@MacroSt#1{\endgroup\Restore@MacroSt{#1}}

\Restore@Macro 1116 \long\def\Restore@Macro#1{%
```

```
    \escapechar92
    \gmu@ifstored#1{%
      \ifgmu@SMglobal\afterfi\global\fi
      \@xa\let\@xa#1\csname␣/gmu/store\string#1\endcsname
      \global\gmu@SMglobalfalse}%
    {\unless\ifgmu@quiet
        \PackageWarning{gmutils}{\@nx#1␣is␣not␣stored,␣I␣do␣
            nothing␣with
          it}%
      \fi
    }%
}
```

\gmu@ifstored

```
\long\def\gmu@ifstored#1#2#3{%
  \gm@ifundefined{/gmu/store%
    \if\relax\@nx#1\else\bslash\fi
    \string#1}{#3}{#2}%
}
```

\gmu@storeifnotyet

```
\long\pdef\gmu@storeifnotyet#1{%
  \if\relax\@nx#1% we check if it's a CS
  \gmu@ifstored{#1}{}{\StoreMacro#1}%
  \fi}
```

\Restore@MacroSt

```
\long\def\Restore@MacroSt#1{%
  \gm@ifundefined{/gmu/store\bslash#1}%
  {\unless\ifgmu@quiet
    \PackageWarning{gmutils}{\bslash#1␣is␣not␣stored.␣I~do␣
        nothing}%
    \fi}%
  {\edef\gmu@smtempa{%
      \ifgmu@SMglobal\global\fi
      \@nx\let\@xa\@nx\csname#1\endcsname
      \@xa\@nx\csname/gmu/store\bslash#1\endcsname}% cf. the commentary in line 1049.
    \gmu@smtempa}%
  \global\gmu@SMglobalfalse}
```

\RestoreMacros

```
\long\pdef\RestoreMacros{\begingroup\makeatletter%
    \Restore@Macros}
```

\Restore@Macros

```
\long\def\Restore@Macros#1{\endgroup
  \gmu@setsetSMglobal
  \let\gml@StoreCS\Restore@Macro% we direct the core CS towards restoring and call the same iterating macro as in line 1077.
  \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{\begingroup\makeatletter\Stored@Macro}
\long\def\Stored@Macro#1{\endgroup\Restore@Macro#1#1}
```

To be able to call a stored CS without restoring it.

\storedcsname

```
\def\storedcsname#1{%
```

```
    \csname␣/gmu/store\bslash#1\endcsname}
```

2008/08/03 we need to store also an environment.

\StoreEnvironment
```
\pdef\StoreEnvironment#1{%
  \StoreMacro*{#1}\StoreMacro*{end#1}}
```

\RestoreEnvironment
```
\pdef\RestoreEnvironment#1{%
  \RestoreMacro*{#1}\RestoreMacro*{end#1}}
```

It happened (see the definition of \@docinclude in gmdoc.sty) that I needed to \relax a bunch of macros and restore them after some time. Because the macros were rather numerous and I wanted the code more readable, I wanted to \do them. After a proper defining of \do of course. So here is this proper definition of \do, provided as a macro (a declaration).

\StoringAndRelaxingDo
```
\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 because the latter is defined in Till Tantau's beamer class another way) (both arguments should be text):

\n@melet
```
\long\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
```
\long\def\gn@melet#1#2{%
  \edef\gmu@nl@reserveda{%
    \global\let\@xa\@nx\csname#1\endcsname
    \@xa\@nx\csname#2\endcsname}%
  \gmu@nl@reserveda}
```

\envirlet
```
\long\pdef\envirlet#1#2{%
  \n@melet{#1}{#2}%
```

```
    \n@melet{end#1}{end#2}%
}
```

\DeclareCommand and \DeclareEnvironment

The code of this section is based on the xparse package version 0.17 dated 1999/09/10, the version available in TEX Live 2007-13, in Ubuntu packages at least. Originally considered a stub 'im Erwartung' (Schönberg) for the LATEX3 bundle, it evolved to quite a nice tool I think.

I rewrote the ancient xparse's code to use the usual catcodes (only with `@` a letter) and not to use the ldcsetup package (which caused an error of undefined CS `\KV@def`). Then I added the `\protected` prefix to the first macro. And I added my concept of `S/T` arg. spec. And of `b` and `B` spec. And the `Q` argument type. And made `\DeclareEnvironment` makes the arguments passed also to the end macro.

After a short deliberation I rename the command to `\DeclareCommand` which is much shorter than original `\DeclareDocumentCommand` and more adequate at least in my case: I don't only use this powerful declaration for 'document' commands.

In the args specification you can use:

`m` for mandatory argument (braced or not),

`O{`⟨*default*⟩`}` for optional argument (in square brackets) with default value ⟨*default*⟩ (which is passed as `#`⟨*n*⟩ if the argument is missing),

`o` equivalent of `O{\NoValue}`: optional argument (in square brackets) lack of which makes the `\NoValue` macro passed as `#`⟨*n*⟩. You can check it with `\If[No]Value(T|F|TF){\#`⟨*n*⟩`}`,

`s` for optional star (if star is present, it becomes respective `#`⟨*n*⟩, or else `\NoValue` is at the `#`⟨*n*⟩ (unlike in xparse!).

`S{`⟨*list*⟩`}{`⟨*default*⟩`}` for a Single token, checks whether on the respective position there's an unbraced token one of ⟨*list*⟩ and returns it on `#`⟨*n*⟩ if present or `{`⟨*default*⟩`}` if absent. The `{`⟨*default*⟩`}` is a braced optional. If absent, `\NoValue` is the default.

`T` as 'Token', just an alias for `S`.

Like in xparse, `s` is a shorthand for `S{*}`, but *unlike* in xparse, in the parsed arguments you get `*` or `\NoValue` as `#`⟨*n*⟩. You can now declare

```
\DeclareCommand\mimbla{S{+-}m}{%
 \leavevmode
 \ifx#1+\raise\fi
 \ifx#1-\lower\fi
 \ifx#1\NoValue\@tempdima=\fi
 7pt  \hbox{#2}%
 }
```

and after `\mimbla+{raised}␣\mimbla-{lowered}␣\mimbla␣{untouched}` get:

raised lowered untouched

This example is rather silly but shows that you spend only one `#` for two symbols while in xparse you would spend two. What if you wanted 10 options for the optional symbol? Here it's no problem. And with any number k of tokens on the list the next `#` is always $n+1$ not $n+k+1$.

`Q{`⟨*list*⟩`}` as 'se**Q**uence', a sequence of symbols from ⟨*list*⟩; for the fundamental usage see line 2244. More clearly, an argument specified with `Q{`⟨*tokens*⟩`}` is a word over the alphabet ⟨*tokens*⟩∪␣ where ␣ is ignored and shall not be passed in the respective `#`⟨*n*⟩.

`c` for an optional argument in parentheses. Historically it comes from a 'coordinate' argument and may serve as such: if you declare `\DCcoordinate`, then its catcher will be redefined to check for presence of a comma and pass the argument as `{%` {⟨*before comma*⟩} {⟨*after comma*⟩} } if comma present. `\NoValue` is passed if absent. By default `\DCnocoordinate` is executed that defines the `c` type argument as just an optional in parentheses.

`C{`⟨*default*⟩`}` as `c`, with default value.

`b` for for an *optional* argument in curly braces. That's strange for anyone acquainted with LaTeX and contrary to its basic convention, but practised by Til Tantau in the beamer class. If missing, `\NoValue` is passed as `#`⟨*n*⟩ and therefore all the tests `If[No]Value(T|F|TF)` apply.

`B{`⟨*default*⟩`}` for a braced *optional* argument with the default value ⟨*default*⟩.

`K{`⟨*#-string*⟩`}[{`⟨*replacement*⟩`}]` for a *mandatory* **K**nuthian delimited or undelimited macro parameters. This concept comes from Stephen Hicks' suggestion at BachoTeX 2009 of implementing arguments delimited with `#{`. So, in the mandatory first argument to `K` you give an arbitrary parameters string as described in Chapter 20 of *The TeXbook*. If you don't provide the replacement, only `#1` of those parameters will be passed to the core macro of your command as `#`⟨*n*⟩. In both arguments you use single `#` char.

`G{`⟨*pair of tokens*⟩`}`[default value] a **G**eneral catcher of an optional. For instance, to declare an optional in angles (used e.g. in Till Tantau's beamer), declare `G{<>}`. Again, if second argument is absent, `\NoValue` is assumed.

`A` (for 'angles') a shorthand for `G{<>}` (accepts the second braced as the default value).

`\afterassignment{`⟨*register*⟩`}` a pseudo-argument that in fact *interrupts* parsing arguments to let an assignment to ⟨*register*⟩. The default ⟨*register*⟩ is `\@tempcnta`. For example,

```
\DeclareCommand\llf{\afterassignment}
 {{\lastlinefit\@tempcnta\par}}
```

defines `\llf` to be a command that expects a value for a numerical assignment, assigns it to (`\@tempcnta` and then to) the `\lastlinefit` special register and executes `\par` with that setting inside a group.

Actually, ⟨*register*⟩ may be empty `{}` and then each time our command is used the left side of the assignment has to be given explicitly and may even be different each time.

Doesn't add anything to the arguments' list.

This pseudo-argument type should be considered experimental.

Note that you could limit acceptable values of a mandatory or an optional-in-brackets argument to a list inside definition of the command, using `\IfAmong`…`\among`… defined in line 399.

The `S/T`, `s` and `Q` arguments are always 'long', they allow `\par` as their value that is. They have to be unbraced to be parsed so there's no danger of "runaway argument".

`\long!lL` By default, all the `c`, `C`, `o`, `O`, `m`, `b` and `B` are 'short', they don't allow `\par` in them that is. Note however that `\par` is allowed in the default values. If you wish to allow `\par` for all the arguments, you can say `\DeclareCommand\mycommand!` … — the optional `!` makes all the arguments 'long'. Instead of `!` you can use `L` or `l` for 'long' or just `\long` itself.

In the arguments specification string you can write `>[{]`⟨*prefix*⟩`[}]` to make subsequent argument 'long' or ignored:

`Pp!lL \long \par iI` Any of `Pp!lL\long\par` to make a particular argument 'long', allowing `\par` in it that is, and/or any of `iI` to make the argument ignored (just gobbled).

(Note that also the `c` and `C` arguments may be made 'long'. That's because I use them not as coordinates but as just another kind of optional argument.

The concept of ignored arguments came to my head when I was declaring a command with three braced optionals and put optional stars only to distinguish the braced optionals.

For example, after

```
\DeclareCommand\GLBTQKi{%
    G{&&}{\#1 default}
    >LB{\#2 default}
    >iT{*\ht}
    Q{0123456789}{0}
    K{#1\par#2\par}{{\text{#1}\over\text{#2}}}
```

and you get `\GLBTQKi` 4-argument with:
`&#1& G{&&}` optional short in a pair of `&` with `\NoValue` as the default,
`{#2} >{L}B` optional long in curly braces,
⟨*ign.*⟩`>{i}T{*\ht}` star or `\ht` control sequence,
`#3 Q{0123456789}{0}` optional sequence of decimal digits with default 0,
`#4 K{#1\par#2\par}{{\text{#1}\over\text{#2}}}` mandatory sequence of two arguments both delimited with `\par`, that will be passed the inner macro as `{\text{#1}\over\text{#2}}`.

`\IfLong` The arguments may be tested inside the command with `\IfLong{#`⟨*n*⟩`}{`⟨*what if long*⟩`}{`⟨*what if short*⟩`}`. The test looks for `\par` at any level of nesting (`{\par}`—`\par` in braces will not hide) since it `\detokenize`s its argument.

`\global` `\outer` If you wish to define your command `\global`ly, you can specify `\DeclareCommand%` `\mycommand\global`. If you wish to forbid usage of your command in arguments of macros, add the `\outer` prefix. As with original TEX's `\def` and like, the prefixes are allowed in any order and in any number only here they come between the command's name and the arguments specification. You can also add `\long`, as we mentioned above, and, for the symmetry, also `\protected`, although the latter is *always* added since the command is not expandable.

Handling of white spaces with optionals seems to me too complicated compared to the estimated weight of the problem and I haven't faced it so far so I don't provide anything. But!—but there are some commands that should be invisible in the typeset text, such as indexing commands and font declarations. For those there is a working LATEX mechanism of `\@bsphack`—`\@esphack` and to use it I provide yet another 'prefix':
`iIwW` `\sphack` if you type `W` or `w` (for 'white') or `I` or `i` (for 'invisible') or, if you prefer a prefix-like, `\sphack`[2] between the CS to be defined and arguments spec, `\@bsphack` and `\@es¦phack` will be added in proper places.

The original inner macros of the ancient `xparse` had names like `\@dc@o` etc. According to my TEX Guru's advice I changed them to `\ArgumentCatcher@`⟨*letter(s)*⟩ to make the error messages less confusing. Well, I don't know if they are but `\Argu¦mentCatcher@PO` looks better than `\@dc@PO` doesn't it?

`\IfDCMessages` `\DCMessagesfalse` `\DCMessagestrue` Talking of messages, there's a Boolean switch `\IfDCMessages`. Its default setting is `\DCMessagesfalse` but if you set `\DCMessagestrue`, every `\command` created with my `\DeclareCommand` will issue a message "Parsing arguments for `\command`" at the beginning of its execution.

`.qQ` If you are positive that no such message will ever be useful, you can suppress the very placing of it in the command's definition with an optional argument to `\Decla¦recommand` with all other 'prefixes', `.` or `q` or `Q` for 'quiet'.

2 I don't define the `\sphack` control sequence and don't assume it's defined. I use it only as a marker and my use of it doesn't create an entry in the hash table.

To sum up, `\DeclareCommand` takes the following arguments:

`#1` `>{P}m` the command to be defined (can be even `\par` if you really wish),

`#2` `Q{\long\global\outer\protected!lL.qQiIwW\sphack}` optional ε-TEX's prefix(es) and/or symbols for making all the arguments long (`!`, `l` or `L`) and/or to suppress placing of the diagnostic message in the definition (`.`, `q`, `Q`) and/or for placing `\@sphack`—`\@esphack` (`i`, `I`, `w`, `W`, `\sphack`),

`#3` `>{P}m` the arguments specification (can contain `\par` as you see),

`#4` `>{P}m` the definition body. You refer to the arguments with `#`⟨*n*⟩ and can test their presence and absence with `\If[No]Value(T|F|TF){#`⟨*n*⟩`}` and if the argument was specified with the `>P` prefix (allows `\par` in itself), you can test it with `\IfLong{#`⟨*n*⟩`}{`⟨*if long*⟩`}{`⟨*if short*⟩`}`. You are also provided the `\IfAmong...%` `\among` and `\IfIntersect` tests defined earlier in this package to process the arguments, especially of the `S`/`T` and `Q` type.

`\DeclareEnvironment` There is also the `\DeclareEnvironment` command to define environments with sophisticated optionals. It takes the arguments analogous to those of `\DeclareCommand`.

The `iIwW\sphack` specifier however acts different: it doesn't add `\@bsphack` nor `\@esphack` but only `\@ignoretrue` to the end macro so the spaces following `\end{%` `myenvir}` will be ignored. (I tried the space hack but it's problematic ("bad space factor" error) if an environment begins in the vertical mode and ends in horizontal.

So the arguments to `\DeclareEnvironment` are:

`#1` `>{P}m` the environment's name; you may wonder why it allows `\par`; it's to allow environments like `\string\par`—I met such an environment once.

`#2` `Q{\long\outer\global\protected!lL.qQiIwW\sphack}` to prefix the command (note that `\outer` prefix will actually not work since the command is called with `\csname`), make all the arguments 'long' (`\long`, `!`, `l` or `L`), not to place the message issuer in the command (`.`, `q`, `Q`) or to make the environment ignores spaces following its end (`iIwW\sphack`).

`#3` `>{P}m` the arguments specification (both for the begin and for the end macros)

`#4` `>{P}m` the begin definition,

`#5` `>{P}m` the end definition; it can use the same parameters (`#`⟨*n*⟩'s) as the begin definition. Note however that there is only one specification of the arguments and both begin and end have to have 9 parameters in total at most.

```
1582 \unless\ifdefined\@temptokenb
\@temptokenb 1583 \newtoks\@temptokenb
1584 \fi

\if@dc@alllong@ 1586 \newif\if@dc@alllong@␣%  for an option of all arguments long.
\if@dc@quiet@ 1587 \newif\if@dc@quiet@␣%  for suppressing the message of using of declared command.

\IfDCMessages 1590 \newif\IfDCMessages␣%  a global switch to suppress the message about parsing.

\@dc@long@yes 1594 \def\@dc@long@yes{P}

\@dc@arguments 1596 \newtoks\@dc@arguments

\@dc@catchernum 1598 \newcount\@dc@catchernum
\@dc@argnum 1599 \newcount\@dc@argnum

\if@dc@ignore 1601 \newif\if@dc@ignore

1605 \long\pdef\DeclareCommand#1#2#3{%
        % #1 command to be defined,
        % #2 arguments specification,
```

```
                % #3 definition body.
\@dc@catchernum 1616  \@dc@catchernum\z@
                1617  \@dc@argnum\z@
                1618  \toks@{}% in this register we will store the created sequence of argument catch-
                          ers.
                1622  \@temptokena\toks@␣% in this register we will store the sequence of #1#2….
                1623  \@temptokenb\toks@␣% in this register we will store the sequences of m's.
                1627  \emptify\@dc@long@
                1628  \@dc@ignorefalse
                1629  \@dc#2X% X is the sentinel of parsing.
                1630  \protected\edef#1{%
                1631    \@nx\@dc@␣␣␣␣{\the\toks@}%
                1632    \@xa\@nx\csname\string#1\endcsname
                1633    \@nx␣#1%
                1634  }%
                1635  \edef\gmu@reserveda{%
                1636    \long\def\@xa\@nx\csname\string#1\endcsname
                1637    \the\@temptokena{%
                1639      \unexpanded{#3}}}%
                1640  \gmu@reserveda}% of the 'draft' \DeclareCommand.

      \@dc@     1643 \long\def\@dc@
                1644  #1% the argument catchers in a pair of braces (their arguments may contain \par
                          so the macro is long).
                1646  #2% \\thecommand
                1647  #3% \thecommand
                1648  {%
                1649  \ifx\protect\@typeset@protect
                1650    \@xa\@firstofone
                1651  \else
                1652    \protect#3\@xa\@gobble
                1653  \fi
                1654  {\@dc@arguments{#2}#1\the\@dc@arguments}}

        \@dc    1657 \def\@dc#1{% The main parsing macro.
                1658  \ifx#1X% if we meet the sentinel we stop. Note there may be some m's in
                          % \@temptokenb and we might not to care about them since they denote
                          mandatory arguments and as such they will be grabbed by
                          % \\⟨declared command⟩. However, if any of them contains \par (while it
                          shouldn't), then it wouldn't be noticed since \\↙<declared␣command>
                          is always long. Therefore:
                1666    \@dc@add@ms
                1667  \else
                1668    \ifx#1>%
                1669      \let\next@dc\grab@prefix
                1670    \else␣% not X neither >
                1671      \if@dc@alllong@\let\@dc@long@\@dc@long@yes\fi
                1672      \ifx#1\afterassignment\@dc@ignoretrue\fi␣% if we parse the
                              assigned pseudo-argument, we don't want to add anything to the ar-
                              guments' toks register so we apply general mechanism of ignoring an
                              argument.
                1676      \ifx#1\gobblespace\@dc@ignoretrue\fi␣% again, as above for ig-
                              noring spaces.
```

```
      \ifnum\ifx#1mo\else1\fi\ifx\@dc@long@\@empty 0\else
          2\fi
          \if@dc@ignore 1\else0\fi=0 % we check whether #1 is
              % m and not prefixed.
        \addto@hook\@temptokenb m% as you see, \@temptokenb serves
            as a storage of m's.
      \else % (not m) or prefixed
        \@dc@add@ms
        \if@dc@ignore\addtotoks\toks@{\@dc@ignorearg}\fi
        \@xa\addtotoks\@xa\toks@\@xa{%
          \csname ArgumentCatcher@\@dc@long@\detokenize{#1}\endcsname}%
              argument catcher's name is \ArgumentCatcher@>[P>]/<arg.
              % type>.
        \@temptokenb{}%
      \fi% of \if short m or not
      \advance\@dc@catchernum\@ne
      \unless\if@dc@ignore
        \advance\@dc@argnum\@ne
        \@temptokena\@xa{%
          \the\@xa\@temptokena\@xa##\the\@dc@argnum}%
      \else
        \@dc@ignorefalse
      \fi
      \IfAmong#1\among{ABCO}%
      {\let\next@dc\grab@default}%
      {\IfAmong#1\among{GQST}%
        {\let\next@dc\grab@defaults}% G, Q, S and T have second argu-
            ment, braced optional, which is the default value (\NoValue by
            default).
        {\gmu@if  x {K#1}%
          {\let\next@dc\@dc@define@K}%
          {\gmu@if x {\afterassignment#1}%
            {\let\next@dc\dc@grab@afterass}%
            {\let\next@dc\@dc}%
          }%
        }%
      }%
      \unless\ifx\next@dc\@dc@define@K
        \emptify\@dc@long@
      \fi
  \fi % of \if#1>.
    \@xa\next@dc
  \fi
}%
```

\@dc@add@ms

```
\def\@dc@add@ms{%
  \@xa\@ifempty\@xa{\the\@temptokenb}%
  {}%
  {\toks@\@xa{%
      \the\@xa\toks@
      \csname ArgumentCatcher@\the\@temptokenb\endcsname}}}
```

\grab@default

```
\long\def\grab@default#1{% the default value or the list of tokens for S may
        contain \par (why not?), that's why we make this macro \long.
```

```
    \toks@\@xa{\the\toks@{#1}}%
    \let\next@dc\@dc
    \@dc
}
\grab@defaults \long\def\grab@defaults#1{% default (when default is absent)
    \toks@\@xa{\the\toks@{#1}}%
    \@ifnextchar\bgroup
    {\grab@default}%
    {\grab@default{\NoValue}}%
}
\dc@grab@afterass \def\dc@grab@afterass{%
    \@ifnextchar\bgroup
    {\grab@default}%
    {\grab@default{\@tempcnta}}%
}
\@dc@addargum \long\def\@dc@addargum{%
    \addtotoks\@dc@arguments}
\StoreMacro\@dc@addargum
\@dc@ignorearg \pdef\@dc@ignorearg{%
\@dc@addargum   \long\def\@dc@addargum##1{%
      \RestoreMacro\@dc@addargum}}
```

Parsing of Knuthian parameters string is easier to implement with full-feature `\DeclareCommand` so we postpone it till line 2363

```
\grab@prefix \def\grab@prefix#1{%
    \IfIntersect{#1}{\long!lL\par␣Pp}%
\@dc@long@  {\def\@dc@long@{P}}{}%
    \IfIntersect{#1}{Ii}%
    {\@dc@ignoretrue}{}%
    \@dc
}
\ArgumentCatcher@o \long\def\ArgumentCatcher@o#1\@dc@arguments{% parsing of an optional
        with no default
    \@ifnextchar[%
\@dc@parse@next {\def\@dc@parse@next{#1}\ArgumentCatcher@o@}% the last macro has
        to be short so we can't give it all the arguments' catchers since some de-
        fault may contain \par. Therefore we hide all the argument catchers in
        % \@dc@parse@next.
    {\@dc@addargum\NoValue#1\@dc@arguments}}
\ArgumentCatcher@o@ \def\ArgumentCatcher@o@[#1]{%
    \@dc@addargum{{#1}}\@dc@parse@next\@dc@arguments}
\ArgumentCatcher@Po \long\def\ArgumentCatcher@Po#1\@dc@arguments{% parsing of an optional
        with no default.
    \@ifnextchar[%
    {\ArgumentCatcher@Po@{#1}}%
    {\@dc@addargum\NoValue#1\@dc@arguments}}
\ArgumentCatcher@Po@ \long\def\ArgumentCatcher@Po@#1[#2]{%
    \@dc@addargum{{#2}}#1\@dc@arguments}
```

```
\ArgumentCatcher@O
1791 \long\def\ArgumentCatcher@O#1#2\@dc@arguments{% parsing of optional with default value.
                % #1 the default value,
                % #2 the tail of args parser.
1797    \@ifnextchar[%
\@dc@parse@next
1798    {\def\@dc@parse@next{#2}\ArgumentCatcher@o@}% note that even when an optional is short, its default may contain \par.
1800    {\@dc@addargum{{#1}}#2\@dc@arguments}}
\ArgumentCatcher@PO
1803 \long\def\ArgumentCatcher@PO#1#2\@dc@arguments{% parsing of optional with default value.
                % #1 the default value,
                % #2 the tail of args parser.
1809    \@ifnextchar[%
1810    {\ArgumentCatcher@Po@#2}% note that even when an optional is short, its default may contain \par.
1812    {\@dc@addargum{{#1}}#2\@dc@arguments}}
\ArgumentCatcher@b
1815 \long\def\ArgumentCatcher@b#1\@dc@arguments{% parsing of an optional braced short argument.
1816    \@ifnextcat\bgroup
\@dc@parse@next
1817    {\def\@dc@parse@next{#1}\ArgumentCatcher@m@}% the last macro has to be short so we can't give it all the arguments' parsers since some default may contain \par. Therefore we hide all the argument catchers in % \@dc@parse@next.
1821    {\@dc@addargum\NoValue#1\@dc@arguments}}
\ArgumentCatcher@Pb
1824 \long\def\ArgumentCatcher@Pb#1\@dc@arguments{% parsing of a braced and long optional with \NoValue default.
1826    \@ifnextcat\bgroup
1827    {\ArgumentCatcher@Pm#1\@dc@arguments}%
1828    {\@dc@addargum\NoValue#1\@dc@arguments}%
1829 }
\ArgumentCatcher@B
1831 \long\def\ArgumentCatcher@B#1#2\@dc@arguments{% parsing of a braced optional with default value.
                % #1 the default value,
                % #2 the tail of args catchers.
1837    \@ifnextcat\bgroup
\@dc@parse@next
1838    {\def\@dc@parse@next{#2}\ArgumentCatcher@m@}% note that even when an optional is short, its default may contain \par.
1840    {\@dc@addargum{{#1}}#2\@dc@arguments}}
\ArgumentCatcher@PB
1842 \long\def\ArgumentCatcher@PB#1#2\@dc@arguments{% parsing of a braced and long optional with default value.

                #1 the default value,
                #2 the tail of args catchers.
1849    \@ifnextcat\bgroup
1850    {\ArgumentCatcher@Pm#2\@dc@arguments}%
1851    {\@dc@addargum{{#1}}#2\@dc@arguments}}
\ArgumentCatcher@S@
1854 \long\def\ArgumentCatcher@S@% parsing of a Single continued:
1855 #1% the list to search #4 on,
1856 #2% the default value,
1857 #3% the tail of args parsers,
```

#4% the token we search in #1 (it's an unbraced single token as we checked in line 1873).

```
{%
  \IfAmong#4\among{#1}%
  {\@dc@addargum{{#4}}#3\@dc@arguments}%
  {\@dc@addargum{{#2}}#3\@dc@arguments#4}%
}
```

\ArgumentCatcher@S

\long\def\ArgumentCatcher@S␣% parsing of a **S**ingle token. It's always long since we look for a single unbraced token so there is no danger of "runaway argument".

#1% the list we search optional token on,
#2% the default value,
#3% the tail of args parser.

```
\@dc@arguments{%
  \@ifnextsingle
  {\ArgumentCatcher@S@{#1}{#2}{#3}}%
  {\@dc@addargum{{#2}}#3\@dc@arguments}% if we parse an opening or closing brace, then we are sure it's not any expected Single.
}
```

\let\ArgumentCatcher@PS\ArgumentCatcher@S␣% as above: we always act 'long' with single tokens.

\let\ArgumentCatcher@T\ArgumentCatcher@S␣% we allow T (for Token) as an alias of S.

```
\edef\ArgumentCatcher@s{%
  \@nx\ArgumentCatcher@S{\@xa\@nx\all@stars}{\@nx\NoValue}}% the s arg spec is a shorthand for S{*}. Except the star may be of any category from {11, 12, 13}.
\let\ArgumentCatcher@Ps\ArgumentCatcher@s
```

\ArgumentCatcher@Q

\long\def\ArgumentCatcher@Q% parsing of a se**Q**uence of tokens from the list. It's always long since we look for unbraced tokens so there is no danger of "runaway argument".

#1% the list we search optional tokens on,
#2% the default value,
#3% the tail of args parser.
\@dc@arguments␣% delimiter

```
{%
```

\@dc@parse@next

```
  \def\@dc@parse@next{#3}%
  \emptify\@dc@seQuence
  \ArgumentCatcher@Q@{#1}{#2}}
```

\ArgumentCatcher@PQ

\let\ArgumentCatcher@PQ\ArgumentCatcher@Q

\ArgumentCatcher@Q@

```
\long\def\ArgumentCatcher@Q@#1#2{% list, default
  \@ifnextsingle
  {\ArgumentCatcher@Q@@{#1}{#2}}%
  {\@dc@seQuence@finish{#2}}%
}
```

\ArgumentCatcher@Q@@

\long\def\ArgumentCatcher@Q@@#1#2#3{% list, default, token to be checked we only launch this macro when not before opening curly brace. #1 is the list of acceptable values, #2 the default value, #3 is the token to be sought on the list.

```
    \IfAmong␣#3\among{#1}%
    {\addtomacro\@dc@seQuence#3%
      \ArgumentCatcher@Q@{#1}{#2}}%
    {\@dc@seQuence@finish{#2}#3}%
}
\@dc@seQuence@finish
\long\def\@dc@seQuence@finish#1{%
    \ifx\@dc@seQuence\@empty\def\@dc@seQuence{#1}\fi
    \@xa\@dc@addargum\@xa{\@xa{\@dc@seQuence}}%
    \@dc@parse@next\@dc@arguments␣}
\ArgumentCatcher@c
\@dc@parse@next
\long\def\ArgumentCatcher@c#1\@dc@arguments{%
    \@ifnextchar(%
    {\def\@dc@parse@next{#1}\ArgumentCatcher@c@}%
    {%
      \@dc@addargum{{\NoValue}}#1\@dc@arguments}%
}
\DCcoordinate
\ArgumentCatcher@c@
\def\DCcoordinate{%
    \def\ArgumentCatcher@c@(##1){%
      \IfAmong,\among{##1}%
      {\@xa\@dc@addargum\@dc@commasep##1\@dc@commasep}%
      {\@dc@addargum{{##1}}}\@dc@parse@next\@dc@arguments}%
\@dc@commasep
    \def\@dc@commasep##1,##2\@dc@commasep{{{{##1}{##2}}}}%
\ArgumentCatcher@Pc@
    \long\def\ArgumentCatcher@Pc@(##1){%
    \IfAmong,\among{##1}%
    {\@xa\@dc@addargum\@dc@commasep##1\@dc@commasep}%
    {\@dc@addargum{{##1}}}\@dc@parse@next\@dc@arguments}%
}
\DCnocoordinate
\ArgumentCatcher@c@
\def\DCnocoordinate{%
    \def\ArgumentCatcher@c@(##1){%
      \@dc@addargum{{##1}}\@dc@parse@next\@dc@arguments}%
\ArgumentCatcher@Pc@
    \long\def\ArgumentCatcher@Pc@(##1){%
      \@dc@addargum{{##1}}\@dc@parse@next\@dc@arguments}%
}

\DCnocoordinate
\ArgumentCatcher@C
\@dc@parse@next
\long\def\ArgumentCatcher@C#1#2\@dc@arguments{%
    \@ifnextchar(%
    {\def\@dc@parse@next{#2}\ArgumentCatcher@c@}%
    {\@dc@addargum{{#1}}#2\@dc@arguments}}
\ArgumentCatcher@Pc
\@dc@parse@next
\long\def\ArgumentCatcher@Pc#1\@dc@arguments{%
    \@ifnextchar(%
    {\def\@dc@parse@next{#1}\ArgumentCatcher@Pc@}%
    {\@dc@addargum{{\NoValue}}#1\@dc@arguments}%
}
\ArgumentCatcher@PC
\@dc@parse@next
\long\def\ArgumentCatcher@PC#1#2\@dc@arguments{%
    \@ifnextchar(%
    {\def\@dc@parse@next{#2}\ArgumentCatcher@Pc@}%
    {\@dc@addargum{{#1}}#2\@dc@arguments}}
\ArgumentCatcher@Pm
\long\def\ArgumentCatcher@Pm#1\@dc@arguments#2{%
```

```
  \@dc@addargum{{#2}}#1\@dc@arguments}
\gmu@hashes
\def\gmu@hashes#1#2{% this is a fully expandable loop analogous to that of The
                    ε-TEX Manual p. 9.
  \ifnum#1<#2%
  ####\number#1
  \expandafter\gmu@hashes
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}% of \gmu@hashes.
\gmu@hashesbraced
\def\gmu@hashesbraced#1#2{%
  \ifnum#1<#2%
  {####\number#1}%
  \expandafter\gmu@hashesbraced
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}% of \gmu@hashesbraced.

\@tempcnta=1
\@whilenum\@tempcnta<9\do{%
  \edef\gmu@ms{\romannumeral\numexpr\@tempcnta*1000␣}%
  \edef\gmu@tempa{%
    \long\def\@xa\@nx\csname
        ArgumentCatcher@\gmu@ms
    \endcsname
    ##1\@nx\@dc@arguments{\def\@nx\@dc@parse@next{##1}%
      \@xa\@nx\csname
        ArgumentCatcher@\gmu@ms␣@%
      \endcsname}%
    \def\@xa\@nx\csname
    ArgumentCatcher@\gmu@ms␣@%
    \endcsname
    \gmu@hashes1{\numexpr\@tempcnta+1}%
    {\@nx\@dc@addargum{%
        \gmu@hashesbraced1{\numexpr\@tempcnta+1}}%
      \@nx\@dc@parse@next\@nx\@dc@arguments
    }% of \ArgumentCatcher@⟨m's⟩@.
  }% of \edef.
  \gmu@tempa
  \advance\@tempcnta1\relax
}% of the loop.
```

The loop above defines 8 pairs of macros as we see thanks to the code below:

```
\if␣1␣1
\@tempcnta\@ne
\@whilenum\@tempcnta<9\do{%
  \edef\gmu@ms{\romannumeral\numexpr\@tempcnta*1000}%
  \typeout{\bslash␣␣ArgumentCatcher@\gmu@ms:%
    ^^J\@xa\meaning\csname␣ArgumentCatcher@\gmu@ms%
        \endcsname}%
  \typeout{\bslash␣␣ArgumentCatcher@\gmu@ms␣@:%
    ^^J\@xa\meaning\csname␣ArgumentCatcher@\gmu@ms␣@%
        \endcsname}%
```

```
2043   \advance\@tempcnta\@ne
2044 }
2045 \fi
```

The code above produces on the terminal:

```
\ArgumentCatcher@m:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@m@
\ArgumentCatcher@m@:
macro:#1->\addto@hook \@dc@arguments {{#1}}\@dc@parse@next
    \@dc@arguments
\ArgumentCatcher@mm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mm@
\ArgumentCatcher@mm@:
macro:#1#2->\addto@hook \@dc@arguments
    {{#1}{#2}}\@dc@parse@next \@dc@arguments
\ArgumentCatcher@mmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmm@
\ArgumentCatcher@mmm@:
macro:#1#2#3->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}}\@dc@parse@next \@dc@arguments
\ArgumentCatcher@mmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmm@
\ArgumentCatcher@mmmm@:
macro:#1#2#3#4->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}}\@dc@parse@next \@dc@arguments
\ArgumentCatcher@mmmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmm@
\ArgumentCatcher@mmmmm@:
macro:#1#2#3#4#5->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}}\@dc@parse@next \to
ks@
\ArgumentCatcher@mmmmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmmm@
\ArgumentCatcher@mmmmmm@:
macro:#1#2#3#4#5#6->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}{#6}}\@dc@parse@ne
xt \@dc@arguments
\ArgumentCatcher@mmmmmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmmmm@
\ArgumentCatcher@mmmmmmm@:
macro:#1#2#3#4#5#6#7->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}{#6}{#7}}\@dc@pa
rse@next \@dc@arguments
\ArgumentCatcher@mmmmmmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmmmmm@
```

```
\ArgumentCatcher@mmmmmmmmm@:
macro:#1#2#3#4#5#6#7#8->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}}\@
dc@parse@next \@dc@arguments
```

umentCatcher@mmmmmmmmm
```
\def\ArgumentCatcher@mmmmmmmmm\the\@dc@arguments
#1#2#3#4#5#6#7#8#9{% yes, it has to be delimited (well, actually: started) by
    % \the\@dc@arguments not only \@dc@arguments. And it doesn't need
    to be long and launch inner short macro because there's nothing more to catch.
  \@dc@addargum{%
    {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}\the\@dc@arguments}
```

\ArgumentCatcher@Pm
```
\long\def\ArgumentCatcher@Pm#1\@dc@arguments#2{% catcher of a long
        mandatory.
  \@dc@addargum{{#2}}#1\@dc@arguments}
```

\ArgumentCatcher@K
```
\long\def\ArgumentCatcher@K␣% a Knuthian delimited or undelimited arguments parsed and passed to the command's body in a digest (Knuthian replacement)
#1% the particular Knuthian macro
#2% the tail of args parser.
\@dc@arguments␣% tail delimiter
```

\@dc@parse@next
```
{\def\@dc@parse@next{#2}#1}
```

```
\let\ArgumentCatcher@PK\ArgumentCatcher@K
```

\gmu@twostring
```
\long\def\gmu@twostring#1#2{\string#1\string#2}
```

\ArgumentCatcher@G
```
\long\def\ArgumentCatcher@G␣% general catcher
#1% left and right delimiter
#2% default value
#3% the tail of args parser.
\@dc@arguments␣% tail delimiter
{%
  \edef\@dc@Gname{ArgumentCatcher@G\gmu@twostring#1}%
  \unless\ifcsname␣\@dc@Gname␣\endcsname
    \@xa\def\csname␣\@dc@Gname\@xa\endcsname
    \@dc@Gparams#1{\@dc@addargum{{##2}}##1\@dc@arguments}%
  \fi
  \@xa\@ifnextchar\@firstoftwo#1%
```

\dc@parse@next
```
  {\def\dc@parse@next{#3}% we hide possible \pars.
    \csname\@dc@Gname\endcsname\dc@parse@next}%
  {\@dc@addargum{{#2}}#3\@dc@arguments}%
}
```

\ArgumentCatcher@PG
```
\long\def\ArgumentCatcher@PG␣% general catcher
#1% left and right delimiter
#2% default value
#3% the tail of args parser.
\@dc@arguments␣% tail delimiter
{%
  \edef\@dc@Gname{ArgumentCatcher@PG\gmu@twostring#1}%
  \unless\ifcsname␣\@dc@Gname␣\endcsname
    \long\@xa\def\csname␣\@dc@Gname\@xa\endcsname
    \@dc@Gparams#1{\@dc@addargum{{##2}}##1\@dc@arguments}%
  \fi
```

```
    \@xa\@ifnextchar\@firstoftwo#1%
    {\csname\@dc@Gname\endcsname{#3}}%
    {\@dc@addargum{{#2}}#3\@dc@arguments}%
}
```

`\@dc@Gparams` `\long\def\@dc@Gparams#1#2{##1#1##2#2}%` expands to parameters string with `%` `#1` delimited with first argument and `#2` delimited with second. In fact it's intended to pass `#1` further (it will always be braced) and to catch an argument put in a pair of tokens given `\@`dc@Gparams as the arguments.

`\ArgumentCatcher@A` `\long\def\ArgumentCatcher@A{\ArgumentCatcher@G{<>}}`
`\ArgumentCatcher@PA` `\long\def\ArgumentCatcher@PA{\ArgumentCatcher@PG{<>}}`

`\ArgumentCatcher@a` `\long\def\ArgumentCatcher@a{\ArgumentCatcher@G{<>}{\NoValue}}`
`\ArgumentCatcher@Pa` `\long\def\ArgumentCatcher@Pa{\ArgumentCatcher@PG{<>}{%`
`\NoValue}}`

`\NoValue` `\def\NoValue{-NoValue-}`

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

```
    \@xa\ifx\@firstofmany#1\@empty\@@nil\NoValue␣\afterfi%
        \@firstoftwo
    \else\afterfi\@secondoftwo
    \fi}
```

`\IfNoValueT` `\long\def\IfNoValueT#1#2{\IfNoValueTF{#1}{#2}\@empty}`

`\IfNoValueF` `\long\def\IfNoValueF#1#2{\IfNoValueTF{#1}\@empty{#2}}`

`\IfValueTF` `\long\def\IfValueTF#1#2#3{\IfNoValueTF{#1}{#3}{#2}}`

`\PutIfValue` `\long\def\PutIfValue#1{\IfValueT{#1}{#1}}`

`\IfValueT` `\let\IfValueT\IfNoValueF`

`\IfValueF` `\let\IfValueF\IfNoValueT`

`\IfLong` `\long\pdef\IfLong#1{%`
`% #1` the argument to check for `\par`,
`% #2` what if `\par` found,
`% #3` what if `\par` not found.

```
    \edef\gmu@reserveda{\detokenize{#1}}%
    \edef\gmu@reserveda{%
      \IfAmong\string\par\@nx\among{\gmu@reserveda}}%
    \gmu@reserveda}
```

`\afterassignment` pseudo-argument

`\long\@namedef{ArgumentCatcher@\detokenize{%`
`\afterassignment}}%`
`#1%` register used in the assignment
`#2%` the tail of args parser.
`\@dc@arguments{%`

`\@dc@parse@next` `\def\@dc@parse@next{%`
`\@dc@addargum\NoValue␣␣%` to make ignoring mechanism work
`#2\@dc@arguments}%`
`\afterassignment\@dc@parse@next`
`#1}%` optional space and = is left at user's discretion. In fact, `#1` may be empty and then each time our command is used the left side of the assignment has to be given explicitly and may even be different each time.

```
\n@melet{ArgumentCatcher@P\detokenize{\afterassignment}}
{ArgumentCatcher@\detokenize{\afterassignment}}
```

`\gobblespace` pseudo-argument: useful between optional and Knuthian-type arguments and after all parsing. Equivalent `>iT{\somedummymacro}`. At first I wanted mark it `\ignorespaces` but the gobbling doesn't expand macros: it doesnt't use `\ignorespaces` but `\@ifnextchar`'s space trick.

```
\long\@namedef{ArgumentCatcher@\detokenize{\gobblespace}}%
{\ArgumentCatcher@S{\dc@gobblespace@dummy}{\NoValue}}

\n@melet{ArgumentCatcher@P\detokenize{\gobblespace}}
{ArgumentCatcher@\detokenize{\gobblespace}}
```

`\dc@gobblespace@dummy`

```
\def\dc@gobblespace@dummy{\dc@gobblespace@dummy}

\let\gobblespace\dc@gobblespace@dummy
```

end of catchers' definitions

`\DeclareCommand`

```
\DeclareCommand\DeclareCommand␣␣␣%
```

This is the final version by now. Note we introduce the 'prefixes' at the `#2` position only at this point so we have yet to prefix every argument specifier separately. Note also we only here 'teach' `\DeclareCommand` to pass the name of its main argument in the `\dc@bsname` macro.

```
{>Pm␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%
```

(1) command to be defined (may be `\par` if you really wish),

```
  Q{\global\protected\outer\long!lL.qQiIwW\sphack\envhack}%
```

(2) an optional se**Q**uence of ε-TEX's prefixes and/or `!` or `l` or `L` for 'all long' and/or `.` or `q` or `Q` for 'quiet' (message about calling the command is suppressed) and/or `i` or `I` as 'invisible' or `W` or `w` as 'white' or `\sphack` to make the command 'invisible' in the leading with the `\@bsphack…% \@esphack`[3],

```
  >Pm␣%
```

(3) arguments specification (may contain `\par`),

```
  >Pm␣%
```

(4) the definition body; may contain nearly anything including `\par` and tests for presence/absence of arguments `% \If[No]Value(T|F|TF)` and for their longness `\IfLong`; you refer to the arguments with `#`⟨*n*⟩.

```
}{%
  \edef\dc@bsname{\string#1}%
```

we'll use it in K-type arg. catcher and to allow `#1 == \par` (`\PackageWarning` is short and `\typeout` too!).

```
  \gm@testdefined#1\iftrue
    \IfDCMessages
      \PackageWarning{gmutils␣(gmparse)}{Redefining␣command␣
          \dc@bsname\space
        with␣\string\DeclareCommand}%
    \fi
  \else
    \IfDCMessages
```

`\dc@bsname`

```
      \typeout{^^J\string\DeclareCommand\dc@bsname\space␣
          (new)^^J}%
    \fi
  \fi
```

3 To deal with an 'invisible' environment, there is one more acceptable value of this argument: `\envhack`, which suppresses placing `\@bsphack`—`\@esphack` and places `\@ignoretrue` instead in the end macro.

First we parse the declaration options to know whether all the arguments are to be long and whether we should suppress the message "Parsing arguments for…".

```
\@dc@alllong@false
\@dc@quiet@false
\IfIntersect{#2}{\long!lL}{%
  \@dc@alllong@true}{}%
\IfIntersect{#2}{.qQ}{\@dc@quiet@true}{}%
\IfIntersect{#2}{iIwW\sphack}%
{% if 'invisibility' is specified:
  \def\@dc@bsphack@{\@nx\@bsphack}%
  \def\@dc@esphack@{\@nx\@esphack}%
}%
{\emptify\@dc@bsphack@\emptify\@dc@esphack@}% if 'invisibility' is not specified.
\IfAmong\envhack\among{#2}%
{% but if we define an environment:
  \emptify\@dc@bsphack@\emptify\@dc@esphack@}%
{% and if we don't define an environment:
  \emptify\@dc@env@argstoend}%
\relaxen\@dc@global@␣\relaxen\@dc@outer@
\IfAmong\outer\among{#2}{\let\@dc@outer@\outer}{}%
\IfAmong\global\among{#2}{\let\@dc@global@\global}{}%
```

We initialise the scratch count and toks registers before parsing of the arguments specifiers.

```
\@dc@catchernum\z@
\@dc@argnum\z@
\toks@{}%
\@temptokena\toks@
\@temptokenb\toks@
```

We parse the arguments specifiers:

```
\@dc#3X% X is the sentinel of parsing.
```

Now we define the basic macro:

```
\@dc@outer@\@dc@global@␣% possibly the prefixes,
\protected\edef#1{% always \protected ;-)
  \@dc@bsphack@␣% perhaps \@bsphack
  \unless\if@dc@quiet@
    \unexpanded{%
      \IfDCMessages
      \typeout}{Parsing␣arguments␣for␣\dc@bsname.}%
      \@nx\fi
  \fi
  \@nx\@dc@{\the\toks@}% in the braces appear the argument catchers.
  \@xa\@nx\csname\dc@bsname\endcsname
  \ifx\@dc@outer@\outer
    \@xa\@nx\csname␣\@xa\gobble\string#1␣\endcsname␣% note the blank space after #1! If the command is to be \outer, we can't put it itself, so we put another, whose name is the same plus a space. Anyway, since #1 becomes \protected, this case will never be executed.
    \else␣\@nx␣#1%
    \fi
```

```
    }%
    \edef\gmu@reserveda{%
      \@dc@global@␣% perhaps \global
      \long\def\@xa\@nx\csname\string#1\endcsname% for
           a command \command, define \\command
      \the\@temptokena% it's #1#2#3...
      {% the definition body:
        \@dc@env@argstoend␣% if we define an environment, this macro will
             provide passing the arguments to the end macro (see line 2409).
        \unexpanded{#4}%
        \@dc@esphack@}%
    }% of \edef.
    \gmu@reserveda
}% of \DeclareCommand.
```

Now we have a convenient tool to implement parsing of a Knuthian argument(s).

Knuthian argument's default replacement

\@dc@K@defaultrepl

```
\def\@dc@K@defaultrepl{##1}
```

2009/11/22 inner pair of braces removed (as leading to additional group causing errors).

\@dc@define@K

```
\DeclareCommand\@dc@define@K\long{mb}{% first we add the particular CS
      to the toks register:
    \edef\gmu@tempa{%
      \unexpanded{\toks@\@xa}%
      {\@nx\the\toks@{%
           \@xa\@nx\csname\dc@bsname␣@K\the\@dc@catchernum%
                \endcsname}}%
    }\gmu@tempa
```

and define this macro:

```
    \edef\gmu@tempa{%
      \def\@xa\@nx\csname\dc@bsname␣@K\the\@dc@catchernum%
           \endcsname
      \unexpanded{#1}{%
      \@nx\@dc@addargum{{%
           \IfValueTF{#2}{\unexpanded{#2}}{\@xau{%
                \@dc@K@defaultrepl}}%
        }}% we add the Knuthian replacement to the toks register as #n.
      \@nx\@dc@parse@next\@dc@arguments}% and continue parsing.
    }% of \edef. Now we execute this temporary macro, which means we \def\⟨command⟩@K⟨n
    \@dc@global@␣% set properly before \@dc#3X.
    \if␣P\@dc@long@\relax\long\fi
    \gmu@tempa
```

Now we clean up and continue construction of arguments parser.

```
    \emptify\@dc@long@
    \@dc
}
```

\DeclareEnvironment

```
\DeclareCommand\DeclareEnvironment\long
{m% Pm the environment's name; you may wonder why it allows \par. Once I met
        an environment with a name containing
        % \string\par.
```

```
2394      Q{\outer\global\protected\long!lL.qQiIwW\sphack\envhack}%
```
a sequence of ε-TEX's prefixes and/or specifiers to make all the arguments 'long' (`\long`, `!`, `l` or `L`) and/or not to place the message issuer in the command (`.`, `q`, `Q`) and/or to make the environment ignore spaces following its end (`i`, `I`, `w`, `W` or `\sphack`),

```
2402    m␣␣␣␣%
```
the arguments specification (both for the begin and the end definitions),

```
2404  m␣%
```
the begin definition,

```
2405  m␣%
```
the end definition; it *can* contain any `#`⟨*n*⟩—the arguments of the environment are passed to it, too.

```
2408 }{%
```

`\@dc@env@argstoend`

```
2409    \def\@dc@env@argstoend{%
2410      \edef
2411      \@xa\@nx\csname␣\bslash#1(arguments)\endcsname
2412      {\unexpanded{%
2413           \@xa\@xa\@xa\unexpanded\@xa\@xa\@xa{\@xa\@gobble%
                \the\@dc@arguments}%
2414        }% of outer \unexpanded,
2415      }% of \edef's body.
```

Note that `\@dc@env@argstoend` will be put in an `\edef` and the arguments bearer it sets will be defined as custom for this particular environment (its name contains the name of the environment) and inside the environment's group and will bear a list of tokens {⟨*arg.1*⟩} {⟨*arg.2*⟩} … not the CS `\@dc@arguments` so it's robust to nested environments and even to `\DeclareEnvironment`'s occurrences inside the environment.

```
2424    }% of \@dc@env@argstoend.

2426    \def\gmu@reserveda{\envhack}%
```
we add the information we define an environment.

```
2428    \IfValueT{#2}%
2429    {\addtomacro\gmu@reserveda{#2}}%
```
we pass all the 'prefixes' to `\Declare%` `Command`

```
2432    \@xa\DeclareCommand\csname#1\@xa\endcsname
2433    \gmu@reserveda{#3}{#4}%
```

Now the begin definition is done. Let's get down to the end definition.

```
2438    \@dc@global@␣%
```
note this CS was for sure redefined by the last `\Declare%` `Command` (and by nothing else) and that's exactly what we want.

```
2441    \@nameedef{end#1}%
```
It's `\edef` inside.

```
2442    {\@nx\@xa\@xa\@nx\csname\bslash␣end#1\endcsname
2443      \@xa\@nx\csname␣\bslash#1(arguments)\endcsname}%
```

We `\edef` the `\end`⟨*env. name*⟩ macro to be

`\expandafter\\end`⟨*env. name*⟩`\\`⟨*env. name*⟩`(arguments)`

(it's `\expandafter` and two control sequences, the latter with `(arguments)` in its name).

```
2449    \IfIntersect{#2}{iIwW\sphack}%
2450    {\@dc@global@\@xa\addtomacro
2451      \csname␣end#1\endcsname{\@ignoretrue}}{}%
2452    \edef\gmu@reserveda{%
2453      \@dc@global@\long\def␣%
```
the inner end macro is always `\long` and perhaps defined `\global`ly.

```
2455      \@xa\@nx\csname\bslash␣end#1\endcsname
2456      \the\@temptokena%
```
it's `#1#2#3`…—the inner end macro takes the same number of parameters as the begin macro.

```
      {% the definition body
        \unexpanded{#5}}%
    }% of \edef.
    \gmu@reserveda
}
```

Ampulex Compressa-like modifications of macros

Ampulex Compressa is a wasp that performs brain surgery on its victim cockroach to lead it to its lair and keep alive for its larva. Well, all we do here with the internal LaTeX macros resembles Ampulex's actions but here is a tool for a replacement of part of macro's definition.

The `\ampulexdef` command takes its `#2` which has to be a macro and replaces part of its definition delimited with `#5` and `#6` with the replacement `#7`. The redefinition may be prefixed with `#1`. `#2` may have parameters and for such a macro you have to set the parameters string and arguments string (the stuff to be taken by the one-step expansion of the macro) as the optional `[#3]` and `[#4]`. . If `\ampulexdef` doesn't find the start and end tokens in the meaning of the macro, it does nothing to it. You have to write `####` instead of `#` or you can use `\ampulexhash` as well. For an example use see line 3366.

`\ampulexdef`

```
\DeclareCommand\ampulexdef\long
{Q{\outer\long\global\protected}␣% (1) (optional) prefix(es); allowed is
      any sequence of them in any order, just like for the original TeX's \def.
  S{\def\edef\gdef\xdef\pdef}␣% (2) (optional) kind of definition; if not
      specified, \def will be used.
  m␣% (3) macro to be redefined,
  O{}␣% (4) \def's parameters string; empty by default,
  O{}␣% (5) definition body's parameters to be taken in a one-step expansion of
      the redefined macro; empty by default,
  m␣% (6) start token(s),
  m␣% (7) end token(s)
  m␣% (8) the replacement
}{% For the example of usage see 3366.
  \def\gmu@tempa{#6}%
  \def\gmu@tempb{#7}%
  \def\gmu@tempc{#8}% we wrap the start, end and replacement tokens in
      macros to avoid unbalanced \ifs.
  \edef\gmu@tempd{%
    \long\def\@nx\gmu@tempd
    ####1\detoken@xa\gmu@tempa
    ####2\detoken@xa\gmu@tempb
    ####3\@nx\gmu@tempd{%
      \unexpanded{%
        \@ifempty{##3}{\hidden@iffalse}{\hidden@iftrue}}}}%
  \gmu@tempd% it defines \gmu@tempc to produce an open \if depending on
      whether the start and end token(s) are found in the meaning of #3.

  \edef\gmu@tempe{%
    \@nx\gmu@tempd\all@other#3% note that #3 may have parameters so we
        have to look at its meaning not at its one-level expansion, which could
        produce an 'extra \}' error.
    \detoken@xa\gmu@tempa
```

```
      \detoken@xa\gmu@tempb\@nx\gmu@tempd
    }%
    \gmu@tempe% we apply the checker and it produces an open \if.
    \edef\gmu@tempd{%
      \long\def\@nx\gmu@tempd
      ####1\@xa\unexpanded\@xa{\gmu@tempa}%
      ####2\@xa\unexpanded\@xa{\gmu@tempb}%
      ####3\@nx\ampulexdef{% we define a temporary macro with the parameters delimited with the 'start' and 'end' parameters of \ampulexdef.
        \@nx\unexpanded{####1}%
        \@nx\@xa\@nx\unexpanded
        \@nx\@xa{\@nx\gmu@tempc}% we replace the part of the redefined macro's meaning with the replacement text.
        \@nx\unexpanded{####3}}}%
    \gmu@tempd
    \edef\gmu@tempf{#5}%
    \edef\gmu@tempe{%
      \IfValueT{#1}{#1}\IfValueTF{#2}{#2}{\def}%
      \@nx#3#4{%
        \@xa\@xa\@xa\gmu@tempd\@xa#3\gmu@tempf\ampulexdef}}%
    \gmu@tempe
    \fi}
\ampulexhash \def\ampulexhash{####}% for your convenience (not to count the hashes).
```

For the heavy debugs I was doing while preparing gmdoc, as a last resort I used `\showlists`. But this command alone was usually too little: usually it needed setting `\showboxdepth` and `\showboxbreadth` to some positive values. So,

```
\gmshowlists \def\gmshowlists{\showboxdepth=1000␣\showboxbreadth=1000␣%
      \showlists}
```

```
  \nameshow \newcommand\nameshow[1]{\@xa\show\csname#1\endcsname}
\nameshowthe \newcommand\nameshowthe[1]{\@xa\showthe\csname#1\endcsname}
```

Note that to get proper `\showthe\my@dimen14` in the 'other' `@`'s scope you write `\nameshowthe{my@dimen}14` .

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 \long\def\xiistring#1{%
    \if\@nx#1\xiispace
      \xiispace
    \else
      \afterfi{\string#1}% to make the same error as bare \string would cause in case of empty #1.
    \fi}
```

The next macro is applied to a `\detokenized` nonempty string to convert the spaces into 'other'.

```
 \@xiispaces \def\@xiispaces#1␣#2\@@nil{%
    #1%
    \ifx\@xiispaces#2\@xiispaces
    \else
```

```
\xiispace
\afterfi{\@xiispaces#2\@@nil}%
\fi}
```

Environments redefined

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`

```
\long\def\@begnamedgroup#1{%
  \edef\@prevgrouplevel{\the\currentgrouplevel}%
```
added 2009/03/24 to handle special pseudo-environments that don't increase `\currentgrouplevel` (such as `document`). Note it's `\edef`ed *outside* the environment's group.

```
  \@ignorefalse% not to ignore blanks after group
  \begingroup\@endpefalse
  \edef\@prevenvir{\@currenvir}%
```
Note we `\edef` it inside the group (for obvious reason), unlike the 'previous' grouplevel.

```
  \edef\@currenvir{#1}%
```
We could do recatcoding through `\string` or `% \detokenize` but all the name 'other' and 10 could affect a thousand packages so we don't do that and we'll recatcode in a testing macro, see line 2748.

```
  \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`.)

Let us make it the starred version of `\begin`.

`\begin*` `\begin`

```
\def\begin{\gm@ifstar{\@begnamedgroup}{%
        \@begnamedgroup@ifcs}}
```

`\@begnamedgroup@ifcs`

```
\def\@begnamedgroup@ifcs#1{%
  \ifcsname#1\endcsname\afterfi{\@begnamedgroup{#1}}%
  \else\afterfi{\@latex@error{Environment␣#1␣undefined}%
        \@eha}%
  \fi}%
```

`\@ifenvir` and improvement of `\end`

It's very clever and useful that `\end` checks whether its argument is `\ifx`-equivalent `\@currenvir`. However, in standard LaTeX 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 catcodes of chars are irrelevant (until they are `\active`, ₁, ₂ etc.). Thus should be also in the `\end`'s test and therefore we ensure the compared texts are both expanded and made all 'other'.

First a (not expandable) macro that checks whether current environment is as given in `#1`. Why is this macro `\long`?—you may ask. It's `\long` to allow environments such as `\string\par`.

`\@ifenvir`

```
\long\def\@ifenvir#1{%
    % #1 enquired environment name which will be confronted with \@currenvir
    % #2 what if true (if the names are equivalent[4])
    % #3 what if false
\@ifedetokens{\@currenvir}{#1}}
```

`\@ifedetokens`

```
\long\pdef\@ifedetokens#1#2{%
% #1 first list of tokens to be expanded and detokenized
% #2 second list
% #3 if agree
% #4 else
  \edef\gmu@tempa{#1}% to get #1 fully expanded.
  \edef\gmu@reserveda{\@xa\detokenize\@xa{\gmu@tempa}}% with our brave new \begin, \@currenvir is fully expanded, remember?
  \edef\gmu@tempa{#2}% to get #2 fully expanded.
  \edef\gmu@reservedb{\@xa\detokenize\@xa{\gmu@tempa}}%
  \ifx\gmu@reserveda\gmu@reservedb\@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi}
```

`\@ifjobname`

```
\def\@ifjobname#1{\@ifedetokens{\jobname}{#1}}
```

`\@ifprevenvir`

```
\long\def\@ifprevenvir#1{%
    % #1 enquired environment name which will be confronted with \@prevenvir
    % #2 what if true (if the names are equivalent[5])
    % #3 what if false
\@ifedetokens{\@prevenvir}{#1}}
```

Note that `\@ifjobname` and `\@ifenvir` are expandable and in an `\edef` they expand to

`\@ifedetokens{`⟨*current jobname*⟩`}{`⟨*arg.1*⟩`}`

and

`\@ifedetokens{`⟨*current envir*⟩`}{`⟨*arg.1*⟩`}`

resp. which may be useful for some TEXvert.

`\@checkend`

```
\def\@checkend#1{\@ifenvir{#1}{}{\@badend{#1}}}
```

4 The names are checked whether they produce the same `\csname`. They don't have to have the same catcodes.

5 The names are checked whether they produce the same `\csname`. They don't have to have the same catcodes.

Thanks to it you may write `\begin{macrocode*}` with $\star_{12}$ and end it with `\end{%` `macrocode*}` with $\star_{11}$ (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*}.
```

You might also write also `\end{macrocode\star}` where `\star` is defined as 'other' star or letter star.

Storing and restoring the catcodes of specials

```
\gmu@storespecials
2787 \DeclareCommand\gmu@storespecials{o}{% we provide a possibility of adding
       stuff. For usage see line ??.
2789   \def\do##1{\catcode`\@nx##1=\the\catcode`##1\relax}%
\gmu@restorespecials
2790   \edef\gmu@restorespecials{%
2791     \dospecials\do\^^M}\IfValueT{#1}{#1}}
\gmu@septify
2793 \pdef\gmu@septify{% restoring the standard catcodes of specials. The name is
                        the opposite of 'sanitize' :-) . It restores also the original catcode of ^^M.
2796   \def\do{\relax\catcode`}%
2797   \do\␣10\do\\0\do\{1\do\}2\do\$3\do\&4%
2798   \do\#6\do\^7\do\_8\do\%14\do\~13\do\^^M5\relax
 %%␣\let\do\@makeother
 %%␣\doo\do1\do2\do3\do4\do5\do6\do7\do8\do9\relax
2801 }
```

From relsize

As file relsize.sty, v3.1 dated July 4, 2003 states, LaTeX 2_ε version of these macros was written by Donald Arseneau asnd@triumf.ca and Matt Swift swift@bu.edu after the LaTeX 2.09 smaller.sty style file written by Bernie Cosell cosell@WILMA.BBN.COM.

I take only the basic, non-math mode commands with the assumption that there are the predefined font sizes.

`\relsize` `\smaller` `\larger` `\smallerr` `\largerr`

You declare the font size with `\relsize{`$\langle n \rangle$`}` 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.

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

```
\relsize
2831 \pdef\relsize#1{%
2832   \ifmmode␣\@nomath\relsize\else
2833     \begingroup
2834      \@tempcnta␣% assign number representing current font size
2835        \ifx\@currsize\normalsize␣4\else␣␣␣% funny order is to have
             most ...
2836          \ifx\@currsize\small␣3\else␣␣␣␣␣␣␣␣% ...likely sizes checked
               first
2837            \ifx\@currsize\footnotesize␣2\else
2838              \ifx\@currsize\large␣5\else
2839                \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: \normal|
                          size 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
\smaller
\textlarger
\textsmaller
\largerr
\smallerr

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

Meta-symbols

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

\meta

```
\pdef\meta#1{%
```

"Since the old implementation of \meta could be used in math we better ensure that this is possible with the new one as well. So we use \ensuremath around \langle and \rangle. However this is not enough: if \meta@font@select below expands to \itshape it will fail if used in math mode. For this reason we hide the whole thing inside an \nfss@text box in that case."

```
    {\meta@fontsetting\ensuremath\langle}%
    \ifmmode\@xa\nfss@text\fi
    {% this has to be a begin-group because \nfss@text becomes \hbox in math
        mode.
     \gmu@activespaceblank
      \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.

```
      #1\/%
    }%
    {\meta@fontsetting\ensuremath\rangle}%
}% of \meta.
```

\gmu@activespaceblank
\gmu@activespace

```
\pdef\gmu@activespaceblank{%
  \@xa\def\gmu@activespace{\space\ignorespaces}% note the subtle perversity of this definition: if we meet more than one subsequent active spaces, then the first of them will typeset \space and its \ignorespaces will gobble \space of the second and will stop at \ignorespaces and this \ignorespaces will gobble the next \space and so on.
}
```

\meta@fontsetting

```
\def\meta@fontsetting{\color{red!50!black}}
```

\meta@font@select

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

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.

The below \meta's drag[6] is a version of *The TEXbook*'s one.

\<...>

```
\def\<#1>{\meta{#1}}
```

\metachar
\metacharfont

```
\pdef\metachar#1{\begingroup\metacharfont␣#1\endgroup}
\def\metacharfont{\meta@fontsetting\rm}
```

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 line break just before nor just after themselves. The \discretionary TEX primitive has three arguments: #1 'before break', #2 'after break', #3 'without break', remember?

\discre

```
\pdef\discre#1#2#3{\leavevmode\kernosp%
  \discretionary{#1}{#2}{#3}\penalty10000\hskiposp\relax}
```

\discret

```
\pdef\discret#1{\discre{#1}{#1}{#1}}
```

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

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

```
\def\:{%
  \ifmmode\afterfi{\mskip\medmuskip}%
  \else\afterfi{\discre{\null}{}{}}% \null to get \hyphenpenalty not \exhyphenpenalty.
  \fi}

\let\gmu@discretionaryhyphen\-␣% for the cases when we don't redefine \ but use \
```

\bihyphen
\gmu@discretionaryhyphen
\gmu@bihyphen@char
\gmu@bihyphen@corr

```
\DeclareCommand\bihyphen{O{*}}{% a redefinition of \- that makes it optional-argument to allow further hyphenation
  \DeclareCommand\gmu@discretionaryhyphen{T{#1}ca}{%
    \IfValueT{##2}{%
      \@ifempty{##2}{}{%
        \def\gmu@bihyphen@char{##2}}%
    }%
    \IfValueT{##3}{%
      \@ifempty{##3}{}{%
        \def\gmu@bihyphen@corr{##3}}%
    }%
    \IfValueTF{##1}\discre\discretionary
    {% before break
      \IfValueTF{##2}{%
        \@ifempty{##2}{\gmu@bihyphen@char}{##2}%
      }{%
        \ifnum\hyphenchar\font>\z@
          \char\hyphenchar\font
        \fi}}% end of before break
    {% after break
      \IfValueT{##3}{%
        \@ifempty{##3}{\gmu@bihyphen@corr}{##3}%
      }%
    }%
    {% without break
    }% almost as in The TeXbook: unlike The TeXbook, we allow hyphenchars ≥ 255 as we are XeTeX.
  }% of \DeclareCommand\-
  \gmu@storeifnotyet\-% original \- is a TeX's primitive, therefore we should store it.
  \let\-\gmu@discretionaryhyphen
  \let\@dischyph\gmu@discretionaryhyphen␣% to override framed.sty
}% of \bihyphen

\relaxen\gmu@bihyphen@corr
```

\vs

```
\pdef\vs{\discre{\visiblespace}{}{\visiblespace}}
```

Then we define a macro that makes the spaces visible even if used in an argument (i.e., in a situation where re\catcodeing has no effect).

\printspaces
\gm@pswords

```
\def\printspaces#1{{\let~=\vs␣\let\␣=\vs␣\gm@pswords#1␣%
    \@@nil}}
\def\gm@pswords#1␣#2\@@nil{%
```

```
    \ifx\relax#1\relax\else#1\fi
    \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 sentinel
        space: it has been already by \printspaces.
\sfname \pdef\sfname#1{\textsf{\printspaces{#1}}}
\gmu@discretionaryslash \def\gmu@discretionaryslash{\discre{/}{\hbox{}}{/}}% the
        second pseudo-argument nonempty to get \hyphenpenalty
        not \exhyphenpenalty.
\file \pdef\file#1{\gmu@printslashes#1/\gmu@printslashes}
\gmu@printslashes \def\gmu@printslashes#1/#2\gmu@printslashes{%
    \sfname{#1}%
    \ifx\gmu@printslashes#2\gmu@printslashes
    \else
    \textsf{\gmu@discretionaryslash}%
    \afterfi{\gmu@printslashes#2\gmu@printslashes}\fi}
```

it allows the spaces in the filenames (and prints them as ␣).

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

```
\pk \pdef\pk#1{\textsf{#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 \DeclareCommand\cs{O{\type@bslash\penalty\@M\hskip\z@skip}}{%
        % [#1] O{\bslash} the control sequence's prefix, by default it's \ allowing
            hyphenation of subsequent word,
        %  #2  m the control sequence or anything to be typeset in typewriter font.
    \begingroup
    \ifdefined\verbatim@specials\verbatim@specials\fi
    \edef\-{\discretionary{%
        \ifdefined\gmv@hyphen\gmv@hyphen
        \else\unexpanded{{\normalfont-}}%
        \fi}{}{}}%
    \def\{{\type@lbrace\yeshy}\def\}{\char`\}}%
    \narrativett
    \edef\narrativett@storedhyphenchar{\the\hyphenchar\font}%
    \hyphenchar\font=\ifdefined\gmv@hyphenchar\gmv@hyphenchar
    \else␣"A6␣\fi
    \cs@inner{#1}}
\cs@inner \pdef\cs@inner#1#2{%
    #1#2%
    \hyphenchar\font=\narrativett@storedhyphenchar\relax
    \endgroup}
\narrativett \def\narrativett{\ttfamily}% such name because I introduce it to distin-
        guish the narrative verbatims from the code in gmdoc.
\env \long\pdef\env{\cs[]}
```

And for the special sequences like ^^A:

```
\foone{\@makeother\^}
  {\pdef\hathat{\cs[^^]}}
\AtBeginDocument{%
  \@ifpackageloaded{gmdoc}{\def\hash{\cs[\#]}}{}}
```

\hathat

\hash

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

```
\def\possfil{\hfil\penalty1000\hfilneg}
```

\possfil

```
\def\possvfil{\vfil
  \penalty\numexpr
  \gmu@minnum{\clubpenalty+\widowpenalty}{9999}%
  \relax␣% eaten by \gmu@maxnum
  \relax␣% eaten by \numexpr
  \vfilneg}
```

\possvfil

```
\long\def\gmu@extremnum#1#2#3{%
        % #1 inequality sign
        % #2 left side of comparison
        % #3 right side of comparison
  \@xa\ifx\@firstofmany#3\@@nil\relax␣% this complicated test is to allow arguments beginning with some \if⟨...⟩ as in \possvfil e.g.
    \@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi
  {#2}%
  {%
    \ifnum\numexpr#2\relax#1\numexpr#3\relax
      \@xa\@firstoftwo
    \else\@xa\@secondoftwo
    \fi
    {\gmu@extremnum#1{#2}}%
    {\gmu@extremnum#1{#3}}%
  }%
}
```

\gmu@extremnum

```
\def\gmu@maxnum{\gmu@extremnum>}
\def\gmu@minnum{\gmu@extremnum<}
```

\gmu@maxnum
\gmu@minnum

Typesetting arguments and commands

\arg

We define a conditional and iterating command `\arg` that in math mode does what it used to do was in math and outside math it typesets mandatory, optional and picture (parenthesed) and angled arguments and optional stars. You can write

```
\arg[gefilte]*<fisch>(mit){baigele}
```

to get

[⟨*gefilte*⟩][*]{⟨*fisz*⟩}(⟨*mit*⟩){⟨*bajgele*⟩}

or even

```
\verb+\MoltoAdagio/arg*{Dankgesang}<an>[die␣Gottheit]+
```

(where / is the escape char in verbatims) to get

\MoltoAdagio[*]{⟨*Dankgesang*⟩}<⟨*an*⟩>[⟨*die Gottheit*⟩]

(in der lydischen Tonart).

For more complicated arguments configurations consider using gmdoc's environment `enumargs`.

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
```
\long\def\cmd#1{\@xa\cs\@xa{\@xa\cmd@to@cs\string#1}}%
```
it has to be un `\protected`! It has so many `\expandafter`s to allow `\cmd\par` and still keep the `\cs` command 'short'.

\cmd@to@cs
```
\def\cmd@to@cs#1#2{\char\number`#2\relax}
```
It can be short since it never gets actual control sequence as an argument only a string of 'other' tokens (and maybe spaces).

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

\marg
```
\pdef\marg#1{{\narrativett\type@lbrace}\meta{#1}{%
    \narrativett\char`\}}}
```

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

\oarg
```
\pdef\oarg{\@ifnextchar[\@oargsq\@oarg}
\pdef\@oarg#1{{\narrativett[}\meta{#1}{\narrativett]}}
\pdef\@oargsq[#1]{\@oarg{#1}}
```

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

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

\pdef\aarg{\@ifnextchar<\@aarga\@aarg}
\def\@aarg#1{{\narrativett<}\meta{#1}{\narrativett>}}
\def\@aarga<#1>{\@aarg{#1}}

\def\@verbaarga#1#2>{\@aarg{#2}\arg@dc}

\foone{\catcode`>\active}{%
  \def\@verbaargact#1#2>{\@aarg{#2}\arg@dc}%
}

\foone{\@makeother\{\@makeother\}%
  \catcode`[=\@ne\catcode`]=\tw@}
[%
  \def\@verbmargm#1#2}[% for an argument in curly braces in a verbatim, where the braces are not groupers and not necessary 'other'. We'll know by \@ifnextif that the future token is an opening brace. Note this macro has 2nd parameter delimited with 'other' closing brace (so may not act correctly when braces are nested (then hide them with special verbatim groupers)).
    \marg[#2]%
    \arg@dc
  ]%
]% of \foone
```

\arg@dc
```
\DeclareCommand\arg@dc!{%
```

```
3238    s␣% (1)
3239    o␣% (2)
3240    c␣% (3)
3241    b␣% (4)
3242    a␣% (5)
3243    S{\arg}␣% (6) just gobbled (for backwards compatibility)
3244 }{% This command iterates while it has arguments and typesets them in brackets,
        parentheses or curly braces. Note it gobbles subsequent \args and just iter-
        ates.
3247    \def\next{o}%
3248    \IfValueT{#1}%
3249    {\metachar[\scanverb{*}\metachar]\def\next{1}}%
3250    \IfValueT{#2}{\@oarg{#2}\def\next{1}}%
3251    \IfValueT{#3}{\@parg{#3}\def\next{1}}%
3252    \IfValueT{#4}{\marg{#4}\def\next{1}}%
3253    \IfValueT{#5}{\aarg{#5}\def\next{1}}%
3254    \@ifnextchar\egroup{\endgroup}{%
3255      \if1\next\@xa\arg@dc
3256      \else␣% it's crucial that we look for verbatim braces after we checked there
              were no #4, otherwise there would be an error.
3259        \def\next{%
3260          \@ifnextif\xiilbrace{\@verbmargm}%
3261          {% not active or other lbrace
3262            \@ifnextif<{% then we look for angles
3263              \ifnum\catcode`>=\active
3264                \@xa\@verbaargact
3265              \else\@xa\@verbaarga
3266              \fi}%
3267            {% and if not angles neither verbatim braces, then
3268              \endgroup␣␣% if we have no more arguments to typeset, we
                      close the group opened in line 3286
3270            }%
3271          }%
3272        }%
3273        \@xa␣\next
3274      \fi
3275    }% of not egroup
3276 }% of \arg@dc
```

Now define the front-end macro of the `\arg` command:

```
3280 \foone{\obeylines}{%
3281   \AtBeginDocument{%
3282     \let\math@arg\arg␣%
\arg 3283     \pdef\arg{\ifmmode\math@arg␣%
3285       \else\afterfi{%
3286         \begingroup␣%
3288         \ifdefined\@ifQueerEOL\@ifQueerEOL{%
3289           \def^^M{\unskip\space}% in the 'queer' EOLs scope we keep
                  line end active in case we have \arg {⟨arg.⟩} ending a line: the
                  next char peeper touches line end or, if the line end was 5, gob-
                  bles the space it turns into so the comment layer would 'leak'
                  to the code layer.
3295         }{}\fi␣%
```

```
3296          \arg@dc}%
3297        \fi}% of \arg,
3298    }% of \AtBeginDocument,
3299 }% of \foone.
```

Now you can write

`\arg{mand. \arg}␣[opt. \arg]␣(pict. \arg)`
to get {⟨*mand. arg*⟩}␣[⟨*opt. arg*⟩]␣(⟨*pict. arg*⟩). (Yes, with only one `\arg`!)
And `$\arg(1+i)␣=␣\pi/4$` for $\arg(1+i) = \pi/4$.

\cat
```
3311 \DeclareCommand\cat{Q{'"0123456789ABCDEF}}{%
3312   ${}_{\the\numexpr#1}\m@th$\@ifnextcat␣a\space{}}
```

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 IMHO.

\not@onlypreamble
```
3328 \newcommand\not@onlypreamble[1]{{%
3329   \def\do##1{\ifx#1##1\else\@nx\do\@nx##1\fi}%
3330   \xdef\@preamblecmds{\@preamblecmds}}}

3332 \not@onlypreamble\@preamblecmds
3333 \not@onlypreamble\@ifpackageloaded
3334 \not@onlypreamble\@ifclassloaded
3335 \not@onlypreamble\@ifl@aded
3336 \not@onlypreamble\@pkgextension
```

And let's make the message of only preamble command's forbidden use informative a bit:

\gm@notprerr
```
3341 \def\gm@notprerr{␣can␣be␣used␣only␣in␣preamble␣(\on@line)}

3343 \AtBeginDocument{%
3344   \def\do#1{\@nx\do\@nx#1}%
3345   \edef\@preamblecmds{%
3346     \def\@nx\do##1{%
3347       \def##1{\@nx\PackageError{gmutils/LaTeX}%
3348         {\@nx\string##1␣\@nx\gm@notprerr}\@nx\@eha}}%
3349     \@preamblecmds}}
```

A subtle error raises: the LaTeX standard `\@onlypreamble` and what `\document` does with `\@preamblecmds` makes any two of 'only preamble' CS's `\ifx`-identical inside `document`. And my change makes any two CS's `\ifx`-different. The first it causes a problem with is standard LaTeX's `\nocite` that checks `\ifx\@onlypreamble\document`. So hoping this is a rare problem, we circumvent it. 2008/08/29 a bug is reported by Edd Barrett that with natbib an 'extra }' error occurs so we wrap the fix in a conditional.

\gmu@nocite@ampulex
```
3366 \def\gmu@nocite@ampulex{%  we wrap the stuff in a macro to hide an open \if.
                                And not to make the begin-input hook too large. the first optional argument
                                is the parameters string and the second the argument for one-level expansion
                                of \nocite. Both hash strings are doubled to pass the first \def.
3372   \ampulexdef\nocite[####1][{{####1}}]% note the double brace around
           % #3.
3374   \ifx
```

```
    {\@onlypreamble\document}%
    \iftrue}

\AtBeginDocument{\gmu@nocite@ampulex}%
```

Third person pronouns

Is a reader of my documentations 'she' or 'he' and does it make a difference?

Previous versions of this documentation were consequently alternating 'he' and 'she' and provided specific macros for that purpose. Now I'm not that queer and gender so I take what is normally used, 'they' that is.

(The issue of human sexes and genders (certainly much more numerous than 2) is complex and delicate and a TEX macro package is probably not the best place to discuss it.)

```
\heshe   \def\heshe{they}
\hisher  \def\hisher{their}
\himher  \def\himher{them}
\hishers \def\hishers{theirs}

\HeShe   \def\HeShe{They}
\HisHer  \def\HisHer{Their}
\HimHer  \def\HimHer{Them}
\HisHers \def\HisHers{Theirs}
```

Improvements to mwcls sectioning commands

That is, 'Expe-ri-mente'[7] 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 `\label`s,
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   \pdef\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:

```
ectionsoutofmainmatter  \def\gm@dontnumbersectionsoutofmainmatter{%
                          \if@mainmatter\else␣\HeadingNumberedfalse␣\fi}
earpagesduetoopenright  \def\gm@clearpagesduetoopenright{%
```

7 A. Berg, *Wozzeck*.

```
\if@openright\cleardoublepage\else␣\clearpage\fi}
```

To avoid `\def`ing of `\mw@sectionxx` if it's undefined, we redefine `\def` to gobble the definition and restore the original meaning of itself.

Why shouldn't we change the ontological status of `\mw@sectionxx` (not define if undefined)? Because some macros (in gmdocc e.g.) check it to learn whether they are in an mwcls or not.

But let's make a shorthand for this test since we'll use it three times in this package and maybe also somewhere else.

`\@ifnotmw`

```
\long\def\@ifnotmw#1#2{\gm@ifundefined{mw@sectionxx}{#1}{#2}}
```

The kernel of MW's sectioning commands:

```
\@ifnotmw{}{%
```

`\mw@sectionxx`

```
\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 `\gm@ifundefined` to make it work in classes other than mwbk

```
      \gm@ifundefined{if@mainmatter}{}{%
          \gm@dontnumbersectionsoutofmainmatter}
  \fi
%   \ifHeadingNumbered
%     \refstepcounter{#1}%
%     \protected@edef\HeadingNumber{\csname
    the#1\endcsname\relax}%
%   \else
%     \let\HeadingNumber\@empty
%   \fi
```

`\HeadingRHeadText` `\HeadingTOCText` `\HeadingText` `\mw@HeadingType`

```
  \def\HeadingRHeadText{#2}%
  \def\HeadingTOCText{#3}%
  \def\HeadingText{#4}%
  \def\mw@HeadingType{#1}%
  \if\mw@HeadingBreakBefore
    \if@specialpage\else\thispagestyle{closing}\fi
    \gm@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
  \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.

- [#1] the flags, e.g. breakbefore, breakafter;
- #2 the sectioning name, e.g. chapter, part;
- #3 preskip;
- #4 heading type;
- #5 postskip

\SetSectionFormatting

```
\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}%
    \@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
  \gm@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 distincter,
             % #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{%
    \gm@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. the backslash) 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}%
      \edef\gmu@reserveda{\unexpanded{#2}\@xa\unexpanded{%
           \gmu@reserveda}}%
      \n@melet{#1@head}{gmu@reserveda}%
}
```

```
}% of \@ifnotmw's else.
```

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{%
  \gm@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 3721.
    {\@xa\mw@getflags\@xa{\gmu@prevsec}%
      \glet\gmu@reserveda\mw@HeadingBreakAfter}%
    \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
    \gm@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 \gm@ifundefined{gmu@prevsec} 'else'.
}% of \def\gmu@checkaftersec.
```

\ParanoidPostsec

```
\def\ParanoidPostsec{% this version of \ifgmu@postsec is intended for the
    special 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's else.
```

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
```
\DeclareCommand\WPheadings{T{\chapter}}{%
  \SetSectionFormatting[breakbefore,wholewidth]
      {part}{\z@\@plus1fill}{}{\z@\@plus3fill}%
  \IfValueF{#1}{%
    \gm@ifundefined{chapter}{}{%
      \SetSectionFormatting[breakbefore,wholewidth]
      {chapter}
      {66\p@}% {67\p@} for Adventor/Schola 0,95.
      {\FormatHangHeading{\LARGE}}
      {27\p@\@plus0,2\p@\@minus1\p@}%
    }%
  }% of unless #1

  \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@\@pluso,4\p@\@minus6\p@}
        {\FormatHangHeading␣{\large}}% 12/14 pt
        {6\p@\@pluso,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@\@pluso,2\p@\@minus1\p@}
        {\FormatHangHeading␣{\normalsize}}
        {3\p@\@pluso,1\p@}% those little skips should be smaller than you cal-
            culate out of a geometric progression, because the interline skip en-
            larges them.

  \SetSectionFormatting[runin]{paragraph}
        {7\p@\@pluso,15\p@\@minus1\p@}
        {\FormatRunInHeading{\normalsize}}
        {2\p@}%
  \SetSectionFormatting[runin]{subparagraph}
        {4\p@\@plus1\p@\@minuso,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}%
```

```
    \StoreMacros{\partmark\chaptermark\sectionmark%
        \subsectionmark\subsubsectionmark\paragraphmark}%
\gm@secxx  \def\gm@secxx#1#2[#3]#4{%
      \ifx\gm@secstar\@empty
```

a little trick to allow a special version of the heading just to the running head.

```
        \@namedef{#1mark}##1{%
```

we redefine \⟨*sec*⟩mark to gobble its argument and to launch the stored true marking command on the appropriate argument.

```
          \storedcsname{#1mark}{#2}%
          \RestoreMacro*{#1mark}%
```

after we've done what we wanted we restore original \#1mark.

```
        }%
\gm@secstar  \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.

```
      \fi
      \@xa\@xa\csname\gm@secini#1\endcsname
      \gm@secstar{#4}}%
} {%
```

we are in mwcls and want to reverse MW's optionals order i.e., if there's just one optional, it should go both to toc and to running head.

```
\gm@secini  \def\gm@secini{gm@mw}%
    \let\gm@secmarkh\@gobble%
```

in mwcls there's no need to make tricks for special version to running headings.

```
\gm@secxx  \def\gm@secxx#1#2[#3]#4{%
      \@xa\@xa\csname\gm@secini#1\endcsname
      \gm@secstar[#2][#3]{#4}}%
}

\gm@sec  \def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
\gm@secx  \def\gm@secx#1[#2]{%
    \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}%
```

if there's only one optional, we double *it* not the mandatory argument.

```
\gm@straightensec  \def\gm@straightensec#1{%
```

the parameter is for the command's name.

```
    \gm@ifundefined{#1}{}{%
```

we don't change the ontological status of the command because someone may test it.

```
      \n@melet{\gm@secini#1}{#1}%
      \@namedef{#1}{%
\gm@secstar    \gm@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
\gm@secstar      \def\gm@secstar{}\gm@sec{#1}}}%
}%

\let\do\gm@straightensec
\do{part}\do{chapter}\do{section}\do{subsection}\do{%
    subsubsection}
\@ifnotmw{}{\do{paragraph}}%
```

this 'straightening' of \paragraph with the 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*
```
\@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

\pdef\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
\StoreMacro\TeX
\AtBeginDocument{\RestoreMacro\TeX}
```

\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\DeclareTextCommand\@nx#2{PD1}{\gmu@reserveda}}
                 \gmu@reserveda
                 \DeclareTextCommandDefault#2{#3}%
        \pdef    \pdef#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."

```
\gm@ifundefined{AmSTeX}
\AmSTeX  {\def\AmSTeX{\leavevmode\hbox{$\mathcal␣A\kern-.2em%
           \lower.376ex%
             \hbox{$\mathcal␣M$}\kern-.2em\mathcal␣S$-\TeX}}{}
\BibTeX  \DeclareLogo\BibTeX{{\rmfamily␣B\kern-.05em%
           \textsc{i{\kern-.025em}b}\kern-.08em% the kern is wrapped in braces
                for my \fakescaps' sake.
           \TeX}}
\SliTeX  \DeclareLogo\SliTeX{{\rmfamily␣S\kern-.06emL\kern-.18em%
           \raise.32ex\hbox
             {\scshape␣i}\kern␣-.03em\TeX}}
\PlainTeX \DeclareLogo\PlainTeX{\textsc{Plain}\kern2pt\TeX}
    \Web  \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:

```
\TeXbook \DeclareLogo[The␣TeX␣book]\TeXbook{\textsl{The␣\TeX␣book}}
         \let\TB\TeXbook% TUG Boat uses this.
    \eTeX \DeclareLogo[e-TeX]\eTeX{%
           \iffontchar\font"03B5{\itshape␣ε}\else
           \ensuremath{\varepsilon}\fi-\kern-.125em\TeX}% definition sent by
                Karl Berry from TUG Boat itself.

         \StoreMacro\eTeX
\pdfeTeX \DeclareLogo[pdfe-TeX]\pdfeTeX{pdf\eTeX}
 \pdfTeX \DeclareLogo\pdfTeX{pdf\TeX}
\pdfLaTeX \DeclareLogo\pdfLaTeX{pdf\LaTeX}

         \gm@ifundefined{XeTeX}{%
\XeTeX     \DeclareLogo\XeTeX{X\kern-.125em\relax
             \gm@ifundefined{reflectbox}{%
               \lower.5ex\hbox{E}\kern-.1667em\relax}{%
               \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
             \TeX}}{}

         \gm@ifundefined{XeLaTeX}{%
```

```
\XeLaTeX 4097  \DeclareLogo\XeLaTeX{X\kern-.125em\relax
4098    \gm@ifundefined{reflectbox}{%
4099      \lower.5ex\hbox{E}\kern-.1667em\relax}{%
4100      \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
4101    \LaTeX}}
```

As you see, if TEX doesn't recognise `\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.

```
\LuaTeX 4109 \DeclareLogo[LuaTeX]\LuaTeX{\textsc{Lua}\TeX}
```

Expandable turning stuff all into 'other'

A shorthand. Note that it takes an undelimited argument not requires ⟨*balanced text*⟩.

```
\detoken@xa 4128 \long\def\detoken@xa#1{\detokenize\@xa{#1}}
```

The next macro originates from the ancient era when I didn't know about ε-TEX's `\detokenize`. A try to redefine it to `\detokenize\@xa{#1}` resulted in error so (v0.991) I leave it and use as is.

Note however it acts different than `\detoken@xa` for a macro with parameters: while `\detoken@xa` produces and 'extra }' error, `\all@other` expands to the detokenised meaning.

```
\all@other 4138  \long\def\all@other#1{\@xa\gm@gobmacro\meaning#1}
```

The `\gm@gobmacro` macro above is applied to gobble the `\meaning`'s beginning, `long␣macro:->` all 'other' that is.

```
4143  \edef\gmu@tempa{%
\gm@gobmacro 4144    \def\@nx\gm@gobmacro##1\@xa\@gobble\string\macro:##2->{}}
4145  \gmu@tempa
```

Brave New World of XETEX

```
\@ifXeTeX 4150 \def\@ifXeTeX{% two-argument command
4151    \ifdefined\XeTeXversion
4152    \unless\ifx\XeTeXversion\relax\afterfifi\@firstoftwo\else%
            \afterfifi\@secondoftwo\fi
4153    \else\afterfi\@secondoftwo\fi}
\XeTeXifprefix 4155 \def\XeTeXifprefix{% to be used as prefix to an \if... test.
4156    \@ifXeTeX{}{\unless}}
\XeTeXthree 4158 \DeclareCommand\XeTeXthree{o}{%
4162    \@ifXeTeX{%
4163      \IfValueT{#1}{\PassOptionsToPackage{#1}{fontspec}}%
4164      \@ifpackageloaded{gmverb}{\StoreMacro\verb}{}%
4165      \RequirePackage{xltxtra}% since v 0.4 (2008/07/29) this package re-
            defines \verb and verbatim*, and quite elegantly provides an option
            to suppress the redefinitions, but unfortunately that option excludes also
            a nice definition of \xxt@visiblespace which I fancy.
4172      \@ifpackageloaded{gmverb}{\RestoreMacro\verb}{}%
4173      \AtBeginDocument{%
```

```
4174        \RestoreMacro\LaTeX\RestoreMacro*{LaTeX␣}% my version of
                the LaTeX logo has been stored just after defining, in line 4023.
4177        \RestoreMacro\eTeX}%
```

\adddefaultfontfeatures

```
4179      \pdef\adddefaultfontfeatures##1{%
4180        \addtomacro\zf@default@options{#1,}}
4181    }% of \@ifXeTeX's first argument,
4182    {}% \@ifXeTeX's second argument,
4183 }% of \XeTeXthree's body.
```

The `\udigits` declaration causes the digits to be typeset uppercase. I provide it since by default I prefer the lowercase (nautical) digits.

```
4189 \AtBeginDocument{%
4190   \@ifpackageloaded{fontspec}{%
```

\udigits

```
4191     \pdef\udigits{%
4192       \addfontfeature{Numbers=Uppercase}}%
4193   }{%
4194     \emptify\udigits}}
```

Fractions

\Xedekfracc

```
4199 \def\Xedekfracc{\gm@ifstar\gmu@xedekfraccstar%
         \gmu@xedekfraccplain}
```

(plain) The starless version turns the font feature `frac` on.

(⋆) But nor my modification of Minion Pro neither TEX Gyre Pagella 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 `\@namedef`s below) and the `numr` and `dnom` features with the fractional slash otherwise (via `\gmu@dekfracc`).

(⋆⋆) But Latin Modern Sans Serif Quotation doesn't support the numerator and denominator positions so we provide the double star version for it, which takes the char from font if it exist and typesets with lowers and kerns otherwise.

\gmu@xedekfraccstar

```
4214 \def\gmu@xedekfraccstar{%
```

\gmu@xefraccdef

```
4215   \def\gmu@xefraccdef##1##2{%
4216       \iffontchar\font␣##2
4217         \@namedef{gmu@xefracc##1}{\char##2␣}%
4218       \else
4219         \n@melet{gmu@xefracc##1}{relax}%
4220       \fi}%
```

\gmu@dekfracc

```
4222     \def\gmu@dekfracc##1/##2{%
4223       {\addfontfeature{VerticalPosition=Numerator}##1}%
               \gmu@numeratorkern
4224       \char"2044␣\gmu@denominatorkern
4225       {\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.

```
4229     \gmu@xefraccdef{1/4}{"BC}%
4230     \gmu@xefraccdef{1/2}{"BD}%
4231     \gmu@xefraccdef{3/4}{"BE}%
4232     \gmu@xefraccdef{1/3}{"2153}%
4233     \gmu@xefraccdef{2/3}{"2154}%
4234     \gmu@xefraccdef{1/5}{"2155}%
```

```
      \gmu@xefraccdef{2/5}{"2156}%
      \gmu@xefraccdef{3/5}{"2157}%
      \gmu@xefraccdef{4/5}{"2158}%
      \gmu@xefraccdef{1/6}{"2159}%
      \gmu@xefraccdef{5/6}{"215A}%
      \gmu@xefraccdef{1/8}{"215B}%
      \gmu@xefraccdef{3/8}{"215C}%
      \gmu@xefraccdef{5/8}{"215D}%
      \gmu@xefraccdef{7/8}{"215E}%
\dekfracc@args
      \pdef\dekfracc@args##1/##2{%
        \gm@ifundefined{gmu@xefracc\detokenize{##1/##2}}{%
          \gmu@dekfracc{##1}/{##2}}{%
          \csname␣gmu@xefracc\detokenize{##1/##2}\endcsname}%
        \if@gmu@mmhbox\egroup\fi
      }% of \dekfracc@args.
      \gm@ifstar{\let\gmu@dekfracc\gmu@dekfraccsimple}{}%
    }
\gmu@xedekfraccplain
\dekfracc@args
\def\gmu@xedekfraccplain{% 'else' of the main \gm@ifstar
      \pdef\dekfracc@args##1/##2{%
        \ifmmode\hbox\fi{%
          \addfontfeature{Fractions=On}%
          ##1/##2}%
        \if@gmu@mmhbox\egroup\fi
      }% of \dekfracc@args
    }
\if@gmu@mmhbox
\newif\if@gmu@mmhbox% we'll use this switch for \dekfracc and also for \thous
                    (hacky thousand separator).
\dekfracc
\pdef\dekfracc{%
  \ifmmode\hbox\bgroup\@gmu@mmhboxtrue\fi
  \dekfracc@args}
\gmu@numeratorkern
\gmu@denominatorkern
\def\gmu@numeratorkern{\kern-.055em\relax}
\def\gmu@denominatorkern{\kern-.05em\relax}
```

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 ½ 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 `\gm@ifstar` we wrote 8 (eight) `#`s to get the correct definition and in the second argument 'only' 4. (The LaTeX 2ε Source claims that that is changed in the 'new implementation' of `\gm@ifstar` so maybe it's subject to change.)

A simpler version of `\dekfracc` is provided in line 5722.

`\resizegraphics`

```
\resizegraphics
\pdef\resizegraphics#1#2#3{%  2009/11/17 works bad with a file whose
                              name contains spaces so I return \XeTeXpicfile
  \resizebox{#1}{#2}{%
    \edef\gmu@tempa{\@nx\csname␣XeTeX\@nx\@ifendswithpdf{%
```

```
          \@xa\string\csname#3\endcsname}{pdf}{pic}file\@nx%
              \endcsname}%
      \gmu@tempa␣"#3"\relax}}

\edef\gmu@tempa{%
  \def\@nx\@ifendswithpdf##1{%
    \unexpanded{%
      \ifnum
      \if\relax\gmu@pdfdetector}##1%
    \detokenize{pdf}\unexpanded{\relax0\else1\fi}% we expand to
        1 if #1 ends with lowercase 'pdf' of cat. 12
    \unexpanded{\if\relax\gmu@PDFdetector}##1%
    \detokenize{PDF}\unexpanded{\relax0\else1\fi}% we expand to
        1 if #1 ends with uppercase 'PDF' of cat. 12
    >0
    \unexpanded{\@xa\@firstoftwo\else\@xa\@secondoftwo\fi}%
  }% of \@ifendswithpdf

  \def\@nx\gmu@pdfdetector##1\detokenize{pdf}{}%
  \def\@nx\gmu@PDFdetector##1\detokenize{PDF}{}%
}\gmu@tempa

\textsuperscript@@
\pdef\textsuperscript@@#1{%
  \@textsuperscript{\selectfont#1}}%
\@textsuperscript
\def\@textsuperscript#1{%
  {\m@th\ensuremath{^{\mbox{\fontsize\sf@size\z@#1}}}}}

\let\textsuperscript@@\textsuperscript

\GMtextsuperscript
\def\GMtextsuperscript{%
  \@ifXeTeX{%
\textsuperscript
    \DeclareCommand\textsuperscript{sm}{%
      \IfValueTF{##1}{\textsuperscript@@{##2}}%
      {%
        \begingroup
        \addfontfeature{VerticalPosition=Numerator}##2%
        \endgroup}}%
  }{\truetextsuperscript}}

\truetextsuperscript
\def\truetextsuperscript{%
  \let\textsuperscript\textsuperscript@@
}
```

Settings for mathematics in main font

I used these terrible macros while typesetting E. Szarzyński's *Letters* in 2008. The `\gmath` declaration introduces math-active digits and binary operators and redefines Greek letters and parentheses, the `\garamath` declaration redefines the quantifiers and is more Garamond Premier Pro-specific.

\gmath
\garamath

\gmath

So, when you set default fonts (in the preamble), put `\gmath` to set what possible from them to math. This sets the `normal` math version. If you want to use another set of fonts elsewhere in your document and have according math for them, set them 'for a while' in the preamble and in their scope declare `\gmath[`⟨*version*⟩`]`. Then put `\mathversion{normal}` right after `\begin{document}` and `\gmath[`⟨*version*⟩`]` where you want those other fonts.

`\gmath` takes second optional argument, in parentheses, which should be (sth. that expands to) a `\fontspec` font selecting command. A font selected by this command will be declared and used when some char in basic font is missing and only else the default math font will be left for such a char.

So,

`\gmath[`⟨*version*⟩`]`(⟨*rescue font(spec-ification)*⟩)

Note that `\gmath` without first optional argument has always to come first because otherwise it overwrite settings of your math version.

`\gmu@getfontstring`
```
\def\gmu@getfontstring{%
  \xdef\gmu@fontstring{%
    \gmu@fontstring@}}
```

`\gmu@fontstring@`
```
\def\gmu@fontstring@{%
  \@xa\@xa\@xa\gmu@fontstring@@\@xa\meaning\the\font\@@nil}
```

`\gmu@fontstring@@`
```
\def\gmu@fontstring@@#1"#2"#3\@@nil{"#2"}
```

`\gmu@getfontscale`
```
\def\gmu@getfontscale#1Scale#2=#3,{%
  \ifx\gmu@getfontscale#3\else
    \def\gmu@tempa{MatchLowercase}%
    \def\gmu@tempb{#3}%
    \ifx\gmu@tempa\gmu@tempb
      \gmu@calc@scale{5}%
      \@xa\@firstoftwo
    \else
      \def\gmu@tempa{MatchUppercase}%
      \ifx\gmu@tempa\gmu@tempb
        \gmu@calc@scale{8}%
        \afterfifi\@firstoftwo
        \else\afterfifi\@secondoftwo
      \fi
    \fi
    {\xdef\gmu@fontscalebr{[\gmu@fontscale]␣}}%
    {\xdef\gmu@fontscalebr{[#3]␣}}%
    \afterfi\gmu@getfontscale
  \fi
}
```

`\gmu@getfontdata`
```
\def\gmu@getfontdata#1{%
  \global\emptify\gmu@fontscalebr
  \begingroup
  #1%
  \@xa\@xa\@xa\gmu@getfontscale
  \csname␣zf@family@options\f@family\endcsname
  ,Scale=\gmu@getfontscale,%
  \gmu@getfontstring
  \xdef\gmu@theskewchar{\the\skewchar\font}%
  \endgroup}
```

`\gmu@stripchar`
```
\def\gmu@stripchar#1"{"}
```

`\gmath@getfamnum`
```
\DeclareCommand\gmath@getfamnum{C{\gmath@fam}}{%
  \edef\gmath@famnum{\@xa\gmu@stripchar\meaning#1}%
}
```

\XeTeXmathcode⟨*char slot*⟩ [⟨=⟩] ⟨*type*⟩ ⟨*family*⟩ ⟨*char slot*⟩

\gmathFams
```
\DeclareCommand\gmathFams{o␣␣%  the name of math version
  C{\NoValue}␣%  'rescue' font. Will be accessible via \symgmathRoman/<version>
       (math family) and \gmath@fontt/<version> (font).
}{%
  \IfValueT{#1}{%
    \DeclareMathVersion{#1}% this sets the defaults so no need to define
         them explicitly
    }% of if #1 given

  \gmu@getfontdata{\rmfamily\itshape}%
  \edef\gmu@tempa{%
    \IfValueTF{#1}{\@nx\SetSymbolFont{letters}{#1}}%
    {\@nx\DeclareSymbolFont{letters}}%
    {\encodingdefault}{gmathit\PutIfValue{#1}}{m}{it}%
    \IfValueT{#1}{%
      \@nx\DeclareSymbolFont{letters#1}%
      {\encodingdefault}{gmathit\PutIfValue{#1}}{m}{it}%
      }%
    \@nx\DeclareFontFamily{\encodingdefault}{gmathit%
         \PutIfValue{#1}}{%
      \skewchar\font\gmu@theskewchar\space}%
    \@nx\DeclareFontShape{\encodingdefault}{gmathit%
         \PutIfValue{#1}}{m}{it}{%
      <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
    \IfValueT{#1}{%
      \@nx\SetMathAlphabet\@nx\mathit{#1}{%
           \encodingdefault}{gmathit#1}{m}{it}}%
  }\gmu@tempa
  \IfValueT{#2}{%
    \gmu@getfontdata{#2\gmu@ifstored{\upshape}{%
         \storedcsname{upshape}}{\upshape}}%
    \edef\gmu@tempa{%
      \@nx\DeclareSymbolFont{gmathRoman\PutIfValue{#1}}%
      {\encodingdefault}{gmathRm\PutIfValue{#1}}{m}{n}%
      \@nx\DeclareFontFamily{\encodingdefault}{gmathRm%
           \PutIfValue{#1}}{%
        \skewchar\font\gmu@theskewchar\space}%
      \@nx\DeclareFontShape{\encodingdefault}{gmathRm%
           \PutIfValue{#1}}{m}{n}{%
        <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
    }\gmu@tempa
    \@xa\font\csname␣gmath@fontt\PutIfValue{#1}\endcsname
    =\gmu@fontstring\relax

    \gmu@getfontdata{#2\gmu@ifstored{\itshape}{%
         \storedcsname{itshape}}{\itshape}}%
    \edef\gmu@tempa{%
      \@nx\DeclareSymbolFont{gmathItalic\PutIfValue{#1}}%
      {\encodingdefault}{gmathIt\PutIfValue{#1}}{m}{n}%
      \@nx\DeclareFontFamily{\encodingdefault}{gmathIt%
           \PutIfValue{#1}}{%
        \skewchar\font\gmu@theskewchar\space}%
```

```
        \@nx\DeclareFontShape{\encodingdefault}{gmathIt%
             \PutIfValue{#1}}{m}{n}{%
          <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
      }\gmu@tempa
    }% of if #2 given

    \gmu@getfontdata{\rmfamily\upshape}%
    \edef\gmu@tempa{%
      \IfValueTF{#1}{\@nx\SetSymbolFont{gmathroman}{#1}}%
      {\@nx\DeclareSymbolFont{gmathroman}}%
      {\encodingdefault}{gmathrm\PutIfValue{#1}}{m}{n}%
      \@nx\DeclareFontFamily{\encodingdefault}{gmathrm%
           \PutIfValue{#1}}{%
        \skewchar\font\gmu@theskewchar\space}%
      \@nx\DeclareFontShape{\encodingdefault}{gmathrm%
           \PutIfValue{#1}}{m}{n}{%
        <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
      \IfValueT{#1}{%
        \@nx\SetMathAlphabet\@nx\mathrm{#1}{%
             \encodingdefault}{gmathrm#1}{m}{n}}%
    }\gmu@tempa
      \@xa\font\csname␣gmath@font\PutIfValue{#1}\endcsname
      =\gmu@fontstring\relax
      \gmathfamshook
}

\emptify\gmathfamshook

\gmathbase  \DeclareCommand\gmathbase{oC{\NoValue}}{%
      \gmathFams[#1](#2)%
\gmath@do   \DeclareCommand\gmath@do{%
      m␣%  (1) the character or CS to be declared,
      o␣%  (2) the Unicode to be assigned,
      m␣%  (3) math type (CS like \mathord etc.)
      C{\gmath@fam}␣%  font family
    }{%
      \gmath@getfamnum(##4)%
      \IfValueTF{##2}{%
        \edef\gmu@tempa{%
          =␣\mathchar@type##3\space
          \gmath@famnum\space
          "##2\relax}%
        \if\relax\@nx##1%
          \gmu@ifstored{##1}{}{\StoreMacro##1}%
          \edef\gmu@tempa{%
            \XeTeXmathchardef␣\@nx##1\gmu@tempa}%
          \else
          \edef\gmu@tempa{%
            \XeTeXmathcode␣`##1␣\gmu@tempa}
          \fi%
        }%
        {% no value of ##2
          \edef\gmu@tempa{%
            \XeTeXmathcode␣`##1␣=
```

```
4542            \mathchar@type##3\space
4543            \gmath@famnum\space
4545            `##1\relax}%
4546        }%
4547        \gmu@tempa
4550    }% of \gmath@do
\gmath@doif 4552    \DeclareCommand\gmath@doif{%
4553        m␣␣␣␣␣␣␣␣% (1) the Unicode hex of char enquired,
4554        m␣␣␣␣␣␣␣% (2) the char or CS to be declared,
4555        m␣␣␣␣␣␣␣% (3) math type CS(\mathord etc.),
4556        o␣␣␣␣␣␣␣% (4) second-choice Unicode (taken if first-choice is absent),
4557        o␣␣␣␣␣␣␣% (5) third-choice Unicode (as above if second-choice is absent
                from font).
4558        B{\gmath@fam}␣% (6) never used in this package. Why?
4559        C{\NoValue}
4560    }{%
4561        \gmu@storeifnotyet{##2}%
4562        \@xa\let\@xa\gmath@ft\csname
4563          gmath@font%
4564          \ifx\gmath@fam##6\else␣t\fi
4565          \gmath@version
4566        \endcsname
4567        \iffontchar\gmath@ft"##1␣\gmath@do##2[##1]##3(##6)%
4568        \else
4569          \IfValueTF{##4}{%
4570            \iffontchar\gmath@ft"##4␣%
                    \gmath@do##2[##4]##3(##6)%
4571            \else
4572              \IfValueTF{##5}{%
4573                \iffontchar\gmath@ft"##5␣%
                        \gmath@do##2[##5]##3(##6)%
4574                \else
4575                  \gmath@restore{##1}{##2}{##3}{##4}{##5}{%
                            ##7}%
4576                \fi}%
4577              {\gmath@restore{##1}{##2}{##3}{##4}{##5}{##7}}%
4578            \fi}%
4579          {\gmath@restore{##1}{##2}{##3}{##4}{##5}{##7}}%
4580        \fi
4581    }% of \gmath@doif In the command above we try to define math char or
          a CS in the family given as ##6. If there're no respective chars, we try
          the same with the family given (as a word) in ##7.
\gmath@restore 4585    \def\gmath@restore##1##2##3##4##5##6{%
4586        \IfValueT{##6}%
4587        {\ifcsname␣##6\endcsname
4588            \edef\gmu@tempa{%
4589              \unexpanded{\gmath@doif{##1}{##2}{%
                        ##3}[##4][##5]}%
4590              {\@xa\@nx\csname##6\endcsname}\relax
4591            }\gmu@tempa
4592            \@xa\@gobbletwo␣% if family ##6 is defined, we gobble the other
                  branch
```

```
          \fi
        }%
        \firstofone
        {\if\relax\@nx##2%
             \RestoreMacro##2%
          \fi
        }%
      }%
\iffalse % doesn't work in a non-math font.
\gmath@delc
      \DeclareCommand\gmath@delc{mo}{%
           %  #1   the char or CS to be declared,
           % [#2]  the Unicode (if not the same as the char).
      \gmath@getfamnum
      \IfValueTF{##2}{%
        \edef\gmu@tempa{%
          =\gmath@famnum\space"##2\relax}%
        \edef\gmu@tempa{%
          \XeTeXdelcode`##1\gmu@tempa}
      }%
      {%
        \edef\gmu@tempa{%
          \XeTeXdelcode`##1=
          \gmath@famnum\space
          `##1\relax}%
      }%
      \gmu@tempa
    }% of \gmath@delc
\gmath@delcif
    \def\gmath@delcif##1##2{%
           %  #1   the Unicode enquired,
           %  #2   the char to be delcode-declared
      \iffontchar\gmath@font"##1\gmath@delc##2[##1]\fi}
\fi% of iffalse
\gmath@delimif
    \def\gmath@delimif##1##2##3{%
           %  #1   the Unicode enquired,
           %  #2   the CS defined as \XeTeXdelimiter,
           %  #3   the math type CS (probably \mathopen or \mathclose).
      \iffontchar\gmath@font"##1
        \gmath@getfamnum
        \protected\edef##2{\@nx\ensuremath{%
             \XeTeXdelimiter\mathchar@type##3\space
             \gmath@famnum\space"##1\relax}}%
      \fi}% of \gmath@delimif.
\rmopname
    \pdef\rmopname##1##2##3{%
      \mathop{##1\kern\z@\mathrm{##3}}\csname n##2limits@%
          \endcsname
    }%
\gmu@dogmathbase
    \DeclareCommand\gmu@dogmathbase{oC{\NoValue}}{%
      \RestoreMacro\mathchar@type%
      \IfValueT{##1}{\mathversion{##1}}%
      \edef\gmath@version{\PutIfValue{##1}}%
      \@xa\let\@xa\gmath@fam\csname symgmathroman%
```

```
\endcsname
\edef\gmath@famm{symgmathRoman\gmath@version}%
```
as you see, this is not a font family (number) but a macro containing the name: of the secondary ('rescue') family.

```
\typeout{@@@ gmutils.sty: taking some math chars from
    the font^^J \gmu@fontstring@}%
\gmath@do+\mathbin
\gmath@doif{2212}-\mathbin[2013](\gmath@famm)%
```
minus sign if present or else en dash

```
\gmath@do=\mathrel
\gmath@do0\mathord
\gmath@do1\mathord
\gmath@do2\mathord
\gmath@do3\mathord
\gmath@do4\mathord
\gmath@do5\mathord
\gmath@do6\mathord
\gmath@do7\mathord
\gmath@do8\mathord
\gmath@do9\mathord
\gmath@doif{2264}\le\mathrel(\gmath@famm)%
\let\leq\le
\let\leeng\le
\gmath@doif{2265}\ge\mathrel(\gmath@famm)%
\let\geq\ge
\let\geeng\ge
\gmath@doif{2A7D}\xleq\mathrel(\gmath@famm)%
\gmath@doif{2A7E}\xgeq\mathrel(\gmath@famm)%
\@ifpackageloaded{polski}{%
  \ifdefined\xleq
  \gmu@storeifnotyet\leq
  \let\leq=\xleq
  \let\le=\leq
  \fi
  \ifdefined\xgeq
  \gmu@storeifnotyet\geq
  \let\geq=\xgeq
  \let\ge=\geq
  \fi}{}%
\gmath@do.\mathpunct
\gmath@do,\mathpunct
\gmath@do;\mathpunct
\gmath@do…\mathpunct
\gmath@do(\mathopen
\gmath@do)\mathclose
\gmath@do[\mathopen
\gmath@do]\mathclose
\gmath@doif{00D7}×\mathbin (\gmath@famm)%
\gmath@do:\mathrel
\gmath@doif{00B7}·\mathbin(\gmath@famm)%
\gmath@doif{22C6}⋆\mathbin(\gmath@famm)%
```
low star

```
\gmath@doif{2300}\varnothing\mathord(\gmath@famm)%
```

```
4725    \gmath@doif{221E}\infty\mathord(\gmath@famm)%
4726    \gmath@doif{2248}\approx\mathrel(\gmath@famm)%
4727    \gmath@doif{2260}\neq\mathrel(\gmath@famm)%
4728    \let\ne\neq
4729    \gmath@doif{00AC}\neg\mathbin(\gmath@famm)%
4730    \gmath@doif{00AC}\nego\mathord(\gmath@famm)%
4731    \gmath@do/\mathop
4732    \gmath@do<\mathrel
4734    \gmath@do>\mathrel
4736    \gmath@doif{2329}\langle\mathopen(\gmath@famm)%
4737    \gmath@doif{232A}\rangle\mathclose(\gmath@famm)%
4738    \gmath@doif{2202}\partial\mathord(\gmath@famm)%
4739    \gmath@doif{00B1}\pm\mathbin(\gmath@famm)%
4740    \gmath@doif{007E}\sim\mathrel(\gmath@famm)%
4741    \gmath@doif{2190}\leftarrow\mathrel(\gmath@famm)%
4742    \gmath@doif{2192}\rightarrow\mathrel(\gmath@famm)%
4743    \gmath@doif{2194}\leftrightarrow\mathrel(\gmath@famm)% if not
          present, \gmathfurther will take care of it if left and right arrows are
          present.
4746    \gmath@doif{2191}\uparrow\mathrel(\gmath@famm)% it should be
          a delimiter (declared with \gmath@delimif) but in a non-math font
          the delimiters don't work (2008/11/19) and I don't think I'll ever need
          up- and down- arrows as delimiters.
4750    \gmath@doif{2193}\downarrow\mathrel(\gmath@famm)%
4752    \gmath@doif{2208}\in\mathrel[03F5][0454](\gmath@famm)%
```

As a fan of modal logics I allow redefinition of `\lozenge` and `\square` iff both are in the font. I don't accept the 'ballot box' U+2610.

```
4756    \if\iffontchar\gmath@font"25CA␣0\else␣1\fi
4757       \iffontchar\gmath@font"25FB␣0\else\iffontchar%
             \gmath@font"25A1␣0\else␣2\fi\fi
4758      \gmath@do\lozenge[25CA]\mathord
4759      \gmath@doif{25FB}\square\mathord[25A1](\gmath@famm)% 'medium
            white square (modal operator)' of just 'white square'.
4761    \fi
4762    \gmath@doif{EB08}\bigcircle\mathbin(\gmath@famm)%
4763    \gmath@doif{2227}\wedge\mathbin(\gmath@famm)%
4764    \gmath@doif{2228}\vee\mathbin(\gmath@famm)%
4766    \gmath@doif{0393}\Gamma\mathalpha(\gmath@famm)%
4767    \gmath@doif{0394}\Delta\mathalpha(\gmath@famm)%
4768    \gmath@doif{0398}\Theta\mathalpha(\gmath@famm)%
4769    \gmath@doif{039B}\Lambda\mathalpha(\gmath@famm)%
4770    \gmath@doif{039E}\Xi\mathalpha(\gmath@famm)%
4772    \gmath@doif{03A3}\Sigma\mathalpha(\gmath@famm)%
4773    \gmath@doif{03A5}\Upsilon\mathalpha(\gmath@famm)%
4774    \gmath@doif{03A6}\Phi\mathalpha(\gmath@famm)%
4775    \gmath@doif{03A8}\Psi\mathalpha(\gmath@famm)%
4776    \gmath@doif{03A9}\Omega\mathalpha(\gmath@famm)%
4778    \@xa\let\@xa\gmath@fam\csname␣symletters\gmath@version%
4779    \endcsname
4780    \edef\gmath@famm{symgmathItalic\gmath@version}%
4782    \gmath@doif{03B1}\alpha\mathalpha(\gmath@famm)%
4783    \gmath@doif{03B2}\beta\mathalpha(\gmath@famm)%
```

```
      \gmath@doif{03B3}\gamma\mathalpha(\gmath@famm)%
      \gmath@doif{03B4}\delta\mathalpha(\gmath@famm)%
      \gmath@doif{03F5}\epsilon\mathalpha(\gmath@famm)%
      \gmath@doif{03B5}\varepsilon\mathalpha(\gmath@famm)%
      \gmath@doif{03B6}\zeta\mathalpha(\gmath@famm)%
      \gmath@doif{03B7}\eta\mathalpha(\gmath@famm)%
      \gmath@doif{03B8}\theta\mathalpha(\gmath@famm)%
      \gmath@doif{03D1}\vartheta\mathalpha(\gmath@famm)%
      \gmath@doif{03B9}\iota\mathalpha(\gmath@famm)%
      \gmath@doif{03BA}\kappa\mathalpha(\gmath@famm)%
      \gmath@doif{03BB}\lambda\mathalpha(\gmath@famm)%
      \gmath@doif{03BC}\mu\mathalpha(\gmath@famm)%
      \gmath@doif{03BD}\nu\mathalpha(\gmath@famm)%
      \gmath@doif{03BE}\xi\mathalpha(\gmath@famm)%
      \gmath@doif{03C0}\pi\mathalpha(\gmath@famm)%
      \gmath@doif{03A0}\Pi\mathalpha(\gmath@famm)%
      \gmath@doif{03C1}\rho\mathalpha(\gmath@famm)%
      \gmath@doif{03C3}\sigma\mathalpha(\gmath@famm)%
      \gmath@doif{03DA}\varsigma\mathalpha(\gmath@famm)% 03C2?
      \gmath@doif{03C4}\tau\mathalpha(\gmath@famm)%
      \gmath@doif{03C5}\upsilon\mathalpha(\gmath@famm)%
      \gmath@doif{03D5}\phi\mathalpha(\gmath@famm)%
      \gmath@doif{03C7}\chi\mathalpha(\gmath@famm)%
      \gmath@doif{03C8}\psi\mathalpha(\gmath@famm)%
      \gmath@doif{03C9}\omega\mathalpha(\gmath@famm)%
      \if␣1␣1%
      \iffontchar\gmath@font"221A
        \fontdimen61\gmath@font=1pt
        \edef\sqrtsign{%
          \XeTeXradical␣\@xa\gmu@stripchar\meaning%
              \symgmathroman\space␣"221A\relax}%
      \fi
      \fi% of if 1 1.
\max  \def\max{\rmopname␣\relax␣m{max}}%
\min  \def\min{\rmopname␣\relax␣m{min}}%
\lim  \def\lim{\rmopname␣\relax␣m{lim}}%
\sin  \def\sin{\rmopname␣\relax␣o{sin}}%
\cos  \def\cos{\rmopname␣\relax␣o{cos}}%
 \tg  \def\tg{\rmopname␣\relax␣o{tg}}%
\ctg  \def\ctg{\rmopname␣\relax␣o{ctg}}%
\tan  \def\tan{\rmopname␣\relax␣o{tan}}%
\ctan \def\ctan{\rmopname␣\relax␣o{ctan}}%
    }% of \gmu@dogmathbase
    \AtBeginDocument{\gmu@dogmathbase[#1](#2)%
      \let\gmathbase\gmu@dogmathbase
    }% of atbd
      \not@onlypreamble\gmathbase
}% of \gmathbase
```

The \gmatbase declaration defines a couple of gmath defining commands and then launches them for the default font at begin document and becomes only that launching.

```
\@onlypreamble\gmathbase
```

It's a bit tricky: if \gmathbase occurs first time in a document inside document then an error error is raised. But if \gmathbase occurs first time in the preamble, then it removes itself from the only-preamble list and redefines itself to be only the inner macro of the former itself.

\gmathfurther

```
\pdef\gmathfurther{%
  \def\do##1##2##3{\gmu@storeifnotyet##1%
    \def##1{%
      \mathop{\mathchoice{\hbox{%
            \rm
            \edef\gma@tempa{\the\fontdimen8\font}%
            \larger[3]%
            \lower\dimexpr(\fontdimen8\font-\gma@tempa)/2␣%
              \hbox{##2}}}{\hbox{%
              \rm
              \edef\gma@tempa{\the\fontdimen8\font}%
              \larger[2]%
              \lower\dimexpr(\fontdimen8\font-%
                   \gma@tempa)/2␣%
              \hbox{##2}}}%
          {\mathrm{##2}}{\mathrm{##2}}}##3}}%
    \iffontchar\gmath@font"2211␣␣␣␣\do\sum{\char"2211}{}%
        \fi%
    \do\forall{\gma@quantifierhook␣\rotatebox[origin=c]{%
        180}{A}%
      \gmu@forallkerning
    }{\nolimits}%
    \def\gmu@forallkerning{\setbox0=\hbox{A}\setbox2=\hbox{%
        \scriptsize␣x}%
      \kern\dimexpr\ht2/3*2␣-\wd0/2\relax}%
    \do\exists{\rotatebox[origin=c]{180}{\gma@quantifierhook␣
        E}}\nolimits%
    \def\do##1##2##3{\gmu@storeifnotyet##1%
      \def##1{##3{%
          \mathchoice{\hbox{\rm##2}}{\hbox{\rm##2}}%
          {\hbox{\rm\scriptsize##2}}{\hbox{\rm%
              \tiny##2}}}}%
    \unless\iffontchar\gmath@font"2227
      \do\vee{\rotatebox[origin=c]{90}{<}}\mathbin%
    \fi
    \unless\iffontchar\gmath@font"2228
      \do\wedge{\rotatebox[origin=c]{-90}{<}}\mathbin
    \fi
    \unless\iffontchar\gmath@font"2194
      \if\iffontchar\gmath@font"2190␣0\else1\fi
        \iffontchar\gmath@font"2192␣0\else2\fi
        \do\leftrightarrow{\char"2190\kern-0,1em␣%
            \char"2192}\mathrel
    \fi\fi
    \def\do##1##2##3{\gmu@storeifnotyet##1%
      \def##1####1{##2{\hbox{%
            \rm
```

```
                \setbox0=\hbox{####1}%
                \edef\gma@tempa{\the\ht0}%
                \edef\gma@tempb{\the\dp0}%
                ##3%
                \setbox0=\hbox{####1}%
               \lower\dimexpr(\ht0␣+␣\dp0)/2-\dp0␣-((%
                    \gma@tempa+\gma@tempb)/2-\gma@tempb)␣%
               \box0}}}%
    \do\bigl\mathopen\larger
    \do\bigr\mathclose\larger
    \do\Bigl\mathopen\largerr
    \do\Bigr\mathclose\largerr
    \do\biggl\mathopen{\larger[3]}%
    \do\biggr\mathclose{\larger[3]}%
    \do\Biggl\mathopen{\larger[4]}%
    \do\Biggr\mathclose{\larger[4]}%
    \addtotoks\everymath{%
      \def\do##1##2{\gmu@storeifnotyet##1%
        \def##1{\ifmmode##2{\mathchoice
             {\hbox{\rm\char`##1}}{\hbox{\rm\char`##1}}%
             {\hbox{\rm\scriptsize\char`##1}}{\hbox{\rm\tiny%
                  \char`##1}}}%
           \else\char`##1\fi}}%
      \do\{\mathopen
      \do\}\mathclose
      \def\={\mathbin{=}}%
      \def\neqb{\mathbin{\neq}}%
      \let\neb\neqb
      \def\do##1{\gmu@storeifnotyet##1%
        \edef\gma@tempa{%
          \def\@xa\@nx\csname␣\@xa\gobble\string##1r%
               \endcsname{%
            \@nx\mathrel{\@nx##1}}}%
        \gma@tempa}%
      \do\vee␣\do\wedge␣\do\neg
      \def\fakern{\mkern-3mu}%
      \thickmuskip=8mu␣plus␣4mu\relax
      \gma@gmathhook
    }% of \everymath.
    \everydisplay\everymath
    \ifdefined\Url
      \ampulexdef\Url{\let\do}\@makeother
      {\everymath{}\let\do\@makeother}% I don't know why but the url
           package's \url typesets the argument inside a math which caused
           digits not to be typewriter but Roman and lowercase.
   \fi% of \ifdefined\Url.
}% of \def\gmathfurther.

\StoreMacro\mathchar@type

\gmath \DeclareCommand\gmath{oC{\NoValue}}{%
    \gmathbase[#1](#2)%
    \gmathfurther
```

```
    \IfValueT{#1}{\csname␣gmathhook#1\endcsname}% this allows adding
        version-specific stuff (I first used this for Fell fonts rescued with Garamond
        Premier)
}
\gmathscripts \pdef\gmathscripts{%
    \addtotoks\everymath{\catcode`\^=7\relax␣\catcode`\_=8%
        \relax␣}%
    \everydisplay\everymath}
\gmathcats \pdef\gmathcats{%
    \addtotoks\everymath{\gmu@septify}%
    \everydisplay\everymath}

\emptify\gma@quantifierhook
\quantifierhook \def\quantifierhook#1{%
\gma@quantifierhook   \def\gma@quantifierhook{#1}}

\emptify\gma@gmathhook
\gmathhook \def\gmathhook#1{\addtomacro\gma@gmathhook{#1}}

\gma@dollar \def\gma@dollar$#1${{\gmath$#1$}}%
\gma@bare \def\gma@bare#1{\gma@dollar$#1$}%
\gma@checkbracket \def\gma@checkbracket{\@ifnextchar\[%
    \gma@bracket\gma@bare}
\gma@bracket \def\gma@bracket\[#1\]{{\gmath\[#1\]}\@ifnextchar\par{}{%
        \noindent}}
\gma \def\gma{\@ifnextchar$%
    \gma@dollar\gma@checkbracket}

\garamath \DeclareCommand\garamath{%
  O{\rm}% the font command
}{%
```

Before 2009/10/19 all the stuff was added to `\everymath` which didn't work.

```
    \quantifierhook{\addfontfeature{OpticalSize=800}}%
\gma@arrowdash   \def\gma@arrowdash{{%
        \setbox0=\hbox{\char"2192}\copy0\kern-0,6\wd0
        \bgcolor\rule[-\dp0]{0,6\wd0}{\dimexpr1,07\ht0+\dp0}%
            \kern-0,6\wd0}}%
\gma@gmathhook   \def\gma@gmathhook{%
      \def\do####1####2####3{\gmu@storeifnotyet####1%
        \def####1{####3{%
\mathchoice             \mathchoice{\hbox{#1####2}}{\hbox{#1####2}}%
            {\hbox{#1\scriptsize####2}}{\hbox{#1%
                \tiny####2}}}}}%
      \do\mapsto{\rule[0,4ex]{0,1ex}{0,4ex}\kern-0,05em%
        \gma@arrowdash\kern-0,05em\char"2192}\mathrel
      \do\cup{\scshape␣u}\mathbin
      \do\varnothing{\setbox0=\hbox{\gma@quantifierhook%
          \addfontfeature{Scale=1.272727}0}%
        \setbox2=\hbox{\char"2044}%
        \copy0␣\kern-0,5\wd0␣\kern-0,5\wd2␣\lower0,125\wd0␣%
            \copy2
        \kern0,5\wd0\kern-0,5\wd2}{}% of \varnothing
```

```
      \do\leftarrow{\char"2190\kern-0,05em\gma@arrowdash}%
           \mathrel
      \do\shortleftarrow{\char"2190}\mathrel
      \do\rightarrow{\gma@arrowdash\kern-0,05em\char"2192}%
           \mathrel
      \do\shortrightarrow{\char"2192\relax}\mathrel
      \do\in{\gma@quantifierhook\char"0454}\mathbin
      \do\prec{\gma@quantifierhook
        \rotatebox[origin=c]{-90}{%
          \glyphname{u03A5.a}}}\mathrel␣% added 2009/9/11
  }% of \gma@gmathhook
}% of \garamath.
```

Minion and Garamond Premier kerning and ligature fixes

»Ws« shall not make long »s« because long »s« looks ugly next to »W«.

```
\def\gmu@tempa{\kern-0,08em\penalty10000\hskiposp\relax
  s\penalty10000\hskiposp\relax}
\protected\edef\Vs{V\gmu@tempa}
\protected\edef\Ws{W\gmu@tempa}
```

`\Wz`
```
\pdef\Wz{W\kern-0,05em\penalty10000\hskiposp\relax␣z}
```

A left-slanted font

Or rather a left Italic *and* left slanted font. In both cases we sample the skewness of the itshape font of the current family, we reverse it and apply to `\itshape` in `\litshape` and `\textlit` and to `\sl` in `\lsl`. Note a slight asymmetry: `\litshape` and `\textlit` take the current family while `\lsl` and `\textlsl` the basic Roman family and basic (serif) Italic font. Therefore we introduce the `\lit` declaration for symmetry, that declaration left-slants `\it`.

I introduced them first while typesetting E. Szarzyński's *Letters* to follow his (elaborate) hand-writing and now I copy them here when need left Italic for his *Albert Camus'* The Plague to avoid using bold font.

Of course it's rather esoteric so I wrap all that in a declaration.

`\leftslanting@`
`\litdimen`
`\litcorrection`
`\litkern`
`\dilitkern`
`\litshape`
```
\pdef\leftslanting@{%
  \def\litdimen{\strip@pt\fontdimen1\font␣ex}%
  \def\litcorrection{%
    \ifhmode\null\nobreak\hskip\litdimen\relax\fi}%
  \def\litkern{% note it's to be used inside the left slanted font, unlike \litcorrection, intended to be used before switching to left slant/italic.
    \leavevmode\null
    \kern-\litdimen\relax}%
  \def\dilitkern{\kern\litdimen\litkern}%
  \pdef\litshape{%
    \litcorrection
    \itshape
    \@tempdima=-2\fontdimen1\font
    \advance\leftskip␣by\strip@pt\fontdimen1\font␣ex␣% to assure at least the lowercase letters not to overshoot to the (left) margin. Note this has any effect only if there is a \par in the scope.
```

```
      \litcorrection
      \edef\gmu@tempa{%
        \@nx\addfontfeature{FakeSlant=\strip@pt\@tempdima}}%
```
when not \edefed, it caused an error, which is perfectly understandable.
```
      \gmu@tempa}%
\textlit  \pdef\textlit##1{%
      {\litshape##1}}%
\lit  \pdef\lit{\rm\litshape}%
\lsl  \pdef\lsl{{%
        \litcorrection
        \it
        \@tempdima=-\fontdimen1\font
        \litcorrection
        \xdef\gmu@tempa{%
          \@nx\addfontfeature{RawFeature={slant=\strip@pt%
              \@tempdima}}}}%
      \rm␣␣%
```
Note in this declaration we left-slant the basic Roman font not the itshape of the current family.
```
      \gmu@tempa}%
```

Now we can redefine \em and \emph to use left Italic for nested emphasis. In Polish typesetting there is bold in nested emphasis as I have heard but we don't like bold since it perturbs homogeneous greyness of a page. So we introduce a three-cycle instead of two-: Italic, left Italic, upright.

```
\em   \pdef\em{%
      \ifdim\fontdimen1\font=\z@␣␣\itshape
      \else
        \ifdim\fontdimen1\font>\z@␣\litshape
        \else␣\upshape
        \fi
      \fi}%
  \pdef\emph##1{%
      {\em##1}}%
}% of \leftslanting@.
\leftslanting  \pdef\leftslanting{\AtBeginDocument\leftslanting@}
\AtBeginDocument{\let\leftslanting\leftslanting@}
```

Fake Old-style Numbers

While preparing documentation of this package I faced an æsthetic problem of lack of old-style numbers in a font I fancy. The font is for the sans serif and the digits occur only in the date in title so it would be a pity not too use a nice font when only one or two numbers are needed.

```
\romorzero  \def\romorzero#1{%
      \ifnum#1=0␣zero\else\romannumeral#1␣\fi}
\fakeonum  \DeclareCommand\fakeonum{%
      0␣%
```
fake bold for the digit »2« (for which emboldening improves look),
```
      >Pm␣␣␣%
```
the text to fake old-style numbers in.
```
}{%
```
I tried to use this command as a declaration but active digits are very uncomfortable, e.g. you can't define macros with arguments.

```
    \gmu@if@onum{#2}{%
      \begingroup
      \edef\gmu@tempa{#2}%
      \makeatletter%
      \IfValueT{#1}{%
        \prependtomacro\fake@onum@ii{%
          \begingroup\addfontfeature{FakeBold=#1}}%
        \addtomacro\fake@onum@ii\endgroup
      }%
      \endlinechar\m@ne␣% to suppress the line end added by \scantokens,
           especially in active ^^M's scopes.
      \gmu@dofakeonum
      \@xa\scantokens\@xa{\gmu@tempa}%
      \endgroup
    }% of \gmu@ifonum 'else'.
}% of \fakeonum.
```

\gmu@dofakeonum

```
\def\gmu@dofakeonum{%
    \def\do##1{%
      \catcode`##1\active
      \scantokens{%
        \@xa\let\@xa##1%
        \csname␣fake@onum@\@xa\romorzero\string##1\endcsname%
             \empty}}%
    \do0\do1\do2\do3\do4\do5\do6\do7\do8\do9%
}

\def\do#1#2{%
    \@namedef{fake@onum@\romorzero#1}{#2}}

\def\gmu@tempa#1{%
    \do#1{\leavevmode
      \gmu@calculateslant{#1}% uses \gmu@tempa and \gmu@tempb, there-
           fore goes first. And defines \gmu@tempd.
      \gmu@measurewd{#1}% the width of char #1 is in \gmu@tempa without
           kerning and in \gmu@tempb with kerning.
      \edef\gmu@tempc{\the\fontcharht\font`#1}%
      \hbox␣to␣\gmu@tempb␣{%
        \hss\resizebox{\gmu@tempa}%
        {\dimexpr\fontdimen5\font+\gmu@tempc-\fontdimen8%
             \font}%
        {\gmu@tempd#1}\hss}}}
```

\gmu@measurewd

```
\def\gmu@measurewd#1{%
    \edef\gmu@tempa{\the\fontcharwd\font`#1}%
    \settowidth{\@tempdimb}{% to preserve kerning
      \char`#1\char`#1\char`#1\char`#1\char`#1\char`#1%
      \char`#1\char`#1\char`#1\char`#1\char`#1\char`#1%
      \char`#1\char`#1\char`#1\char`#1\char`#1\char`#1%
      \char`#1\char`#1\char`#1}%
    \edef\gmu@tempb{\the\dimexpr(\@tempdimb-\gmu@tempa)/20}%
}

\gmu@tempa0␣% \fake@onum@zero
\gmu@tempa1␣% \fake@onum@i
```

```
\gmu@tempa2 % \fake@onum@ii

\def\gmu@tempa#1{%
  \do#1{\leavevmode
    \gmu@measurewd{#1}%
    \lower
    \dimexpr\fontdimen8\font-\fontdimen5\font\relax
    \hbox to \gmu@tempb {\hss#1\hss}}%
}

\gmu@tempa3 % \fake@onum@iii
\gmu@tempa4 % \fake@onum@iv
\gmu@tempa5 % \fake@onum@v
\gmu@tempa7 % \fake@onum@vii
\gmu@tempa9 % \fake@onum@ix

\def\gmu@tempa#1{% to preserve pseudo-kerning in digits sequences.
  \do#1{\leavevmode
  \gmu@measurewd#1%
  \hbox to \gmu@tempb {\hss#1\hss}}}

\gmu@tempa6 % \fake@onum@vi
\gmu@tempa8 % \fake@onum@viii
```

\gmu@if@onum

```
\protected\def\gmu@if@onum{%
  \edef\gmu@tempa{\@xa\meaning\the\font}%
  \@xa\@ifinmeaning\detokenize{+onum}\of\gmu@tempa
}
```

Thus `\gmu@if@onum` becomes a two-argument command that executes its `#1` if there is `+onum` in current font specification or its `#2` if `+onum` is absent.

One could easily generalise `\gmu@if@onum` to `\@if@fontfeature`, i.e. to a test for an arbitrary font feature, probably with employing that very nice feature specification of fontspec, so that you could write `\IfFontFeature{Numbers=OldStyle}{}{fake old-style digits}`.

\gmu@getslant

```
\pdef\gmu@getslant{% we define \gmu@tempa to the (fake) slant of current font.
  \edef\gmu@tempa{\@xa\meaning\the\font\detokenize{slant=0,}}%
  \edef\gmu@tempb{%
    \def\@nx\gmu@tempb####1%
    \detokenize{slant=}%
    ####2,####3}%
  \gmu@tempb\@@nil{##2}%
  \edef\gmu@tempa{\@xa\gmu@tempb\gmu@tempa\@@nil pt}%
}
```

\gmu@calculateslant

```
\def\gmu@calculateslant#1{%
  \gmu@getslant
  \edef\gmu@tempa{\the\numexpr\dimexpr\fontdimen1\font +%
      \gmu@tempa}% \gmu@tempa bears the number of scaled points of total slant ( \fontdimen1\font + slant=... if present) per 1pt of #1.
  \edef\gmu@tempa{\the\numexpr \gmu@tempa *
    \numexpr\fontdimen5\font\relax/\numexpr\fontcharht%
        \font`#1
    \relax}% we scale the total slant of #1 by the ratio of original and scaled height of #1.
```

```
    \edef\gmu@tempd{%
      \the\dimexpr␣\gmu@tempa␣sp␣␣-␣\fontdimen1\font}% and we subtract slant-fontdimen from the scaled total slant.
    \ifdim\gmu@tempd=\z@␣\emptify\gmu@tempd
    \else\edef\gmu@tempd{%
      \@nx\addfontfeature{FakeSlant=\strip@pt\dimexpr%
          \gmu@tempd}}%
    \fi}
\gmu@cepstnof \DeclareCommand\gmu@cepstnof{O{\gmu@tempa}% a cs to be \xdefed the font specification,
    s% not used really,
    m% \fontspec token or name of feature font ( Italic, Bold, SmallCaps, % BoldItalic ),
    O{,␣Scale=MatchLowercase}% fontspec font features (key=val)
}
{% \gmu@cepstnof's body
\gmu@reservedc \def\gmu@reservedc##1:##2:\@@nil{%
      \ifx:##2:\else␣RawFeature={\gmu@maybestripcomma##2,,%
          \@@nil}␣\fi}%
\gmu@reservedd \def\gmu@reservedd##1/##2/\@@nil{% to check whether font name contains / (which may not be true!)
      \ifx&##2&\@xa\@firstoftwo\else\@xa\@secondoftwo\fi}%
\gmu@reservede \def\gmu@reservede##1:##2:\@@nil{% to check whether the font name contains : when it doesn't contain /.
      \ifx&##2&\@xa\@firstoftwo\else\@xa\@secondoftwo\fi}%

    \edef\gmu@reserveda{%
```

now, reserved B parses the font name and features. It uses an auxiliary reserved C because after / may be or may not be features specification.

```
      \@xa\@xa\@xa\gmu@reservedd\@xa\meaning\the\font//\@@nil
      {% font name doesn't contain a slash
        \@xa\@xa\@xa\gmu@reservede\@xa\meaning\the\font::%
            \@@nil
        {% nor does it contain a colon
          \def\@nx\gmu@reservedb\detokenize{select␣font
            "}####1\detokenize{"}####2\@nx\@@nil{%
            \ifx\fontspec#3%
              \@nx\@nx\@nx\fontspec[\@gobble#4\@empty]{####1}% gobble a comma
            \else
              #3Font={####1},␣#3Features={\@gobble␣#4%
                  \@empty}%
            \fi
          }%
        }%
        {% no slash but there is a colon
          \def\@nx\gmu@reservedb\detokenize{select␣font
            "}####1:####2\detokenize{"}####3\@nx\@@nil{%
            \ifx\fontspec#3%
              \@nx\@nx\@nx\fontspec[\@nx\gmu@reservedc####2::%
                  \@nx\@@nil#4]{####1}%
```

```
            \else
              #3Font={####1}, #3Features={\@nx%
                   \gmu@reservedc####2::\@nx\@@nil#4}%
            \fi
          }%
        }%
      }% of 'no slash' case
      {% font name contains a slash
        \def\@nx\gmu@reservedb\detokenize{select font
          "}####1/####2\detokenize{"}####3\@nx\@@nil{%
          \ifx\fontspec#3%
            \@nx\@nx\@nx\fontspec[\@nx\gmu@reservedc####2::%
                 \@nx\@@nil#4]{####1}%
          \else
            #3Font={####1}, #3Features={\@nx%
                 \gmu@reservedc####2::\@nx\@@nil#4}%
          \fi
        }% of \gmu@reservedb
      }% of 'slash present' case
    }\gmu@reserveda
    \xdef#1{\@xa\@xa\@xa\gmu@reservedb\@xa\meaning\the\font%
        \@@nil}%
}%
```

`\gmu@maybestripcomma`
```
\def\gmu@maybestripcomma#1,,#2\@@nil{#1}
```

`\gmu@setbasefont`
```
\pdef\gmu@setbasefont{\@xa\let\@xa\gmu@basefont\the\font}

\let\setbasefont\gmu@setbasefont
```

`\gmu@calc@scale`
```
\DeclareCommand\gmu@calc@scale{%
  O{1}% a factor,
  m% number of the fontdimen
  }
{\begingroup
```

We 'descale' the current font:

```
  \gmu@cepstnof\fontspec[, Scale=1]\gmu@tempa
  \@xa\let\@xa\gmu@currfont@descaled\the\font
  \gmu@basefont
  \gmu@cepstnof\fontspec[, Scale=1]\gmu@tempa
```

now also the base font is descaled.

```
  \xdef\gmu@fontscale{%
    \strip@pt
    \dimexpr  1pt *
     \numexpr\dimexpr#1\fontdimen#2\font\relax\relax /
    \numexpr\fontdimen#2\gmu@currfont@descaled\relax
    \relax}%
  \endgroup}
```

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 `\␣`.

`\AtBeginDocument{%` to bypass redefinition of `\~` as a text command with various encodings

`\texttilde`

```
\pdef\texttilde{%
  \@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 `\active`ate the space but also will (re)define it as the `\␣` primitive. So define `\gmobeyspaces` that obeys this requirement.

(This definition is repeated in gmverb.)

`\gmobeyspaces`

```
\foone{\catcode`\␣\active}%
{\def\gmobeyspaces{\let␣\␣\catcode`\␣\active}}
```

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.

`\defobeylines`

```
\foone{\catcode`\^^M\active}% the comment signs here are crucial.
{\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}
```

`\dekmedbigskip`

```
\def\dekmedbigskip{\vskip\glueexpr␣\medskipamount+%
    \smallskipamount}
```

`\dekmedskip`

```
\def\dekmedskip{\vskip\medskipamount}
```

`\dekbigskip`

```
\def\dekbigskip{\vskip\bigskipamount}
```

`\hfillneg`

```
\def\hfillneg{\hskip␣opt␣plus␣-1fill\relax}
```

A mark for the **TO-DO!**s:

`\TODO`

```
\newcommand*{\TODO}[1][]{{%
    \sffamily\bfseries\huge␣TO-DO!\if\relax#1\relax\else%
        \space\fi#1}}
```

I like two-column tables of contents. First I tried to provide them by writing `\begin{multicols}{2}` and `\end{multicols}` out to 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
```
\DeclareCommand\enoughpage{%
  Q{+-0123456789}% (optional short version (number of \baselineskips))
  B{2\baselineskip}% (2) optional (formerly mandatory) long version of required room on a page
  >is
  B{}␣% (3) what if the room is enough
  >isB{\newpage}␣% (4) what if there's to little room on a page
}{%
  \if
  \ifdim\dimexpr\pagegoal-\pagetotal<\dimexpr
  \IfValueTF{#1}{#1\baselineskip}{#2}\relax
   1\else0\fi
   \unless\ifdim\dimexpr\pagegoal-\pagetotal<\z@
    1\else2\fi
```

We check both whether space left on page is larger than we check *and* whether it's nonnegative (the latter happens when we are already on the next page). Therefore this test will not work properly for large values of `#1` or values close to `\textwidth`.

```
    \@xa\@firstoftwo
  \else
    \@xa\@secondoftwo
  \fi
  {#4}{#3}%
}
```

An equality sign properly spaced:

\equals
```
\pdef\equals{\hunskip${}={}$\ignorespaces}
```

And for the LaTeX's pseudo-code statements:

\eequals
```
\pdef\eequals{\hunskip${}=={}$\ignorespaces}
```

\·
```
\pdef\·{\hunskip${}·{}$\ignorespaces}
```

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 `\edef`s.

\freeze@actives
```
\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`.

```
\ShowFont
\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.

```
\liturgiques
\newcommand*\liturgiques[1][red]{%  Requires the color package.
  \gmu@RPfor{xcolor}\color%
\czerwo
  \newcommand*\czerwo{\small\color{#1}}% environment
\czer
  \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.
  \StoreMacro\*%
\*
  \def\*{\czer{\storedcsname{*}}}%
\+
  \def\+{\czer{†}}%
\nieczer
  \newcommand*\nieczer[1]{\textcolor{black}{##1}}%
}
```

After the next definition you can write `\gmu@RP[`⟨*options*⟩`]{`⟨*package*⟩`}{`⟨*CS*⟩`}` to get the package `#2` loaded with options `#1` if the CS`#3` is undefined.

```
\gmu@RPfor
\newcommand*\gmu@RPfor[3][]{%
  \ifx\relax#1\relax\emptify\gmu@resa
\gmu@resa
  \else␣\def\gmu@resa{[#1]}%
  \fi
  \@xa\RequirePackage\gmu@resa{#2}}
```

Since inside `document` we cannot load a package, we'll redefine `\gmu@RPfor` to issue a request before the error issued by undefined CS.

```
\AtBeginDocument{%
\gmu@RPfor
  \renewcommand*\gmu@RPfor[3][]{%
    \unless\ifdefined#3%
      \@ifpackageloaded{#2}{}{%
        \typeout{^^J!␣Package␣`#2'␣not␣loaded!!!␣(%
            \on@line)^^J}}%
    \fi}}
```

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
\pprovide\continuum{%
  \gmu@RPfor{eufrak}\mathfrak\ensuremath{\mathfrak{c}}}
```

And this macro I saw in the ltugproc document class and I liked it.

```
\iteracro
\def\iteracro{%
\acro
  \pdef\acro##1{%
    \begingroup
```

```
      \acropresetting
      \gmu@acrospaces##1␣\gmu@acrospaces
      \endgroup
  }%
}
\emptify\acropresetting
\iteracro
```

`\gmu@acrospaces`

```
\def\gmu@acrospaces#1␣#2\gmu@acrospaces{%
  \gmu@acroinner#1\gmu@acroinner
  \ifx\relax#2\relax\else
    \space
    \afterfi{\gmu@acrospaces#2\gmu@acrospaces}% when #2 is nonempty,
```

it is ended with a space. Adding one more space in this line resulted in an infinite loop, of course.

```
  \fi}
```

`\gmu@acroinner`

```
\def\gmu@acroinner#1{%
  \ifx\gmu@acroinner#1\relax\else
    \ifcat␣a\@nx#1\relax%
      \ifnum`#1=\uccode`#1%
        {\acrocore{#1}}%
      \else{#1}% tu było \smallerr
      \fi
    \else#1%
    \fi
    \afterfi\gmu@acroinner
  \fi}
```

We extract the very thing done to the letters to a macro because we need to redefine it in fonts that don't have small caps.

`\acrocore`

```
\pdef\acrocore{\smaller␣% was: \scshape\lowercase
}
```

Since the fonts I am currently using do not support required font feature, I skip the following definition.

`\IMHO` `\AKA`

```
\def\IMHO{\acro{IMHO}}
\def\AKA{\acro{AKA}}
```

`\usc`

```
\pdef\usc#1{{\addfontfeature{Letters=UppercaseSmallCaps}#1}}
```

`\uscacro`

```
\def\uscacro{\let\acro\usc}
```

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]{gm¦
```

docc} it's loading gmdocc.cls with all the bells and whistles except the error message.

Cf. *The TEXbook* ex. 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.

\gmu@dekfraccsimple
```
\def\gmu@dekfraccsimple#1/#2{\leavevmode\kern.1em
  \raise.7ex\hbox{%
    \gmu@fracfontsetup#1}\gmu@numeratorkern
  \dekfraccslash\gmu@denominatorkern
  {%
    \gmu@fracfontsetup#2}%
  \if@gmu@mmhbox\egroup\fi}
```

\gmu@fracfontsetup
```
\def\gmu@fracfontsetup{%
  \smaller[3]\addfontfeature{FakeBold=1}}
```

\dekfraccsimple
```
\def\dekfraccsimple{%
  \let\dekfracc@args\gmu@dekfraccsimple
}
```

\dekfraccslash
```
\@ifXeTeX{\def\dekfraccslash{\char"2044␣}}
```

\dekfraccslash
```
{\def\dekfraccslash{/}}␣% You can define it as the fraction
      slash, \char"2044␣
\dekfraccsimple
```

A macro that acts like `\,` (thin and unbreakable space) except it allows hyphenation afterwards:

\ikern
```
\newcommand*\ikern{\,\penalty\@M\hskip\z@skip\relax}
```

And a macro to forbid hyphenation of the next word:

\nohy
```
\pdef\nohy{\leavevmode\kernosp\relax}
```

\yeshy
```
\pdef\yeshy{\leavevmode\penalty\@M\hskip\z@skip}
```

In both of the above definitions 'osp' not `\z@` to allow their writing to and reading from files where @ is 'other'.

`\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
```
\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
```
\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}}
```

Switching on and off parts of one file

The `\include` facility is very nice only it forces you to split your source in many files. Therefore I provide a tool analogous to `\include` and using the same `\includeonly` mechanism/list to switch on and off parts of the same source file.

`\filepart`

```
\def\filepart#1{\relax
  \ifnum\@auxout=\@partaux
  \@latex@error{\string\filepart\space␣cannot␣be␣nested}%
      \@eha
  \else\afterfi{\@filepart#1␣}\fi}
```

`\@filepart`

```
\def\@filepart#1␣{%
  \clearpage
  \edef\gmu@filepartname{#1}% we'll use it later
  \if@filesw
```

```
      \immediate\write\@mainaux{\string\@input{#1.aux}}%
    \fi
    \@tempswatrue
    \if@partsw
      \@tempswafalse
      \@for\reserved@a:=\@partlist\do{%
        \ifx\reserved@a\gmu@filepartname\@tempswatrue\fi}%
    \fi
    \if@tempswa
      \let\@auxout\@partaux
      \if@filesw
        \immediate\openout\@partaux␣#1.aux
        \immediate\write\@partaux{\relax}%
      \fi
      \@xa\@firstoftwo
    \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@filepartname}%
      \let\@auxout\@mainaux
      \@xa\@secondoftwo
    \fi
    {\iftrue}%
    {\let\endfilepart\fi
      \csname␣gm@skipped@#1\endcsname
      \def\next{\RestoreMacro\endfilepart
        \@ifnextchar\bgroup{\show\NextBgroup\@gobble}{}}%
      \@xa\next\iffalse}%
}
```

`\endfilepart`

```
\DeclareCommand\endfilepart{b}{%
```

Note the argument is not used really. Maybe later we'll use it for checking of proper matching. Or maybe not.

```
    \inclasthook
    \clearpage
    \@writeckpt{\gmu@filepartname}%
    \if@filesw
    \immediate\closeout\@partaux
    \fi
    \fi% this \fi closes \Iftrue put by line 5841.
    \let\@auxout\@mainaux
}

\StoreMacro\endfilepart
```

`\nofileparts` `\endfilepart`

```
\def\nofileparts{%
  \let\filepart\@gobble
  \DeclareCommand\endfilepart{b}{}%
}
```

Fix of including when fontspec is used

The fontspec package creates counters for font families. If a fontspec command is used in a part of a document and then such a part is skipped, an error occurs 'No counter

zf@fam@… defined'. Now we fix that by ensuring all the counters are defined before they are set.

Note it's a draft version which doesn't support resetting of one counter within another.

```
\includecountfix 5890 \def\includecountfix{%
\@wckptelt       5891   \def\@wckptelt##1{%
                 5892     \immediate\write\@partaux{%
                 5893       \providecounter{##1}% to provide the font counters defined in parts
                                                    of the document.
                 5896       \string\setcounter{##1}{\the\@nameuse{c@##1}}}}%
                 5897 }
\providecounter  5899 \pdef\providecounter#1{%
#1               5900   \unless\ifcsname␣c@#1\endcsname\newcounter{#1}\fi}
```

Faked small caps

```
\gmu@scapLetters 5905 \def\gmu@scapLetters#1{%
                 5906   \ifx#1\relax\relax\else% two \relaxes to cover the case of empty #1.
                 5907     \ifcat␣a\@nx#1\relax
                 5908       \ifnum\the\lccode`#1=`#1\relax
                 5909         {\fakescapscore\MakeUppercase{#1}}% not Plain \uppercase
                                                                    because that works bad with inputenc.
                 5911       \else#1%
                 5912       \fi
                 5913     \else#1%
                 5914     \fi%
                 5915     \@xa\gmu@scapLetters
                 5916   \fi}%
\gmu@scapSpaces  5918 \def\gmu@scapSpaces#1␣#2\@@nil{%
                 5919   \ifx#1\relax\relax
                 5920   \else\gmu@scapLetters#1\relax
                 5921   \fi
                 5922   \ifx#2\relax\relax
                 5923   \else\afterfi{\␣\gmu@scapSpaces#2\@@nil}%
                 5924   \fi}
\gmu@scapss      5926 \def\gmu@scapss#1\@@nil{{\def~{{\nobreakspace}}%
\nobreakspace    5927     \gmu@scapSpaces#1␣\@@nil}}% %␣\def\\{{\newline}}\relax adding
                                                      redefinition of \\ caused stack overflow. Note it disallows hyphenation
                                                      except at \-.
\fakescaps       5931 \pdef\fakescaps#1{{\gmu@scapss#1\@@nil}}
                 5933 \let\fakescapscore\gmu@scalematchX
```

Experimente z akcentami patrz no3.tex.

```
\tinycae         5936 \def\tinycae{{\tiny\AE}}% to use in \fakescaps[\tiny]{…}
                 5938 \RequirePackage{calc}
```

wg \zf@calc@scale pakietu fontspec.

```
                 5942 \@ifpackageloaded{fontspec}{%
\gmu@scalar      5943   \def\gmu@scalar{1.0}%
  \zf@scale      5944   \def\zf@scale{}%
\gmu@scalematchX 5945   \def\gmu@scalematchX{%
```

```
      \begingroup
\gmu@scalar     \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 synthesised
            uppercase height.
        \divide\@tempdimb␣by1000\relax
        \divide\@tempdima␣by\@tempdimb
        \setlength{\@tempdima}{\@tempdima*\real{\gmu@scalar}}%
        \gm@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}

\fakescextrascale 5971 \def\fakescextrascale#1{\def\fakesc@extrascale{#1}}
\fakesc@extrascale
```

See above/see below

To generate a phrase as in the header depending of whether the respective label is before of after.

```
\wyzejnizej 5977 \newcommand*\wyzejnizej[1]{%
5978   \edef\gmu@tempa{\gm@ifundefined{r@#1}{\arabic{page}}{%
5979       \@xa\@xa\@xa\@secondoftwo\csname␣r@#1\endcsname}}%
5980   \ifnum\gmu@tempa<\arabic{page}\relax␣wy\.zej\fi
5981   \ifnum\gmu@tempa>\arabic{page}\relax␣ni\.zej\fi
5982   \ifnum\gmu@tempa=\arabic{page}\relax␣\@xa\ignorespaces\fi
5983 }
```

`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 5993 \def\napapierkistretch{0,3pt}%
```
It's quite much for 11/13pt leading.

```
\napapierkicore 5995 \def\napapierkicore{\advance\baselineskip%
5996   by␣optplus\napapierkistretch\relax}

6003 \DeclareEnvironment{napapierki}{s}{%
6005   \par\IfValueT{#1}{\global
6006   }%
6007   \napapierkicore}
6008 {%
6009   \par
```

```
    \IfValueT{#1}{\global\baselineskip=1\baselineskip\relax
    }%
}% so that you can use \napapierki* ›...\endnapapierki* in interlacing environments.
```

\gmu@luzniej
```
\newcount\gmu@luzniej
```

\luzniejcore
```
\newcommand*\luzniejcore[1][1]{%
    \advance\gmu@luzniej\@ne% We use this count to check whether we open the environment or just set \looseness inside it again.
    \ifnum\gmu@luzniej=\@ne␣␣\multiply\tolerance␣by␣2␣\fi
    \looseness=#1\relax}
```

After `\begin{luzniej}` we may put the optional argument of `\luzniejcore`

luzniej
```
\newenvironment*{luzniej}{\par\luzniejcore}{\par}
```

The starred version sets `\looseness` in `\everypar`, which has its advantages and disadvantages.

luzniej*
```
\newenvironment*{luzniej*}[1][1]{%
    \multiply\tolerance␣by␣2\relax
    \everypar{\looseness=#1\relax}}{\par}
```

\nawj
```
\newcommand*\nawj{\kern0,1em\relax}% a kern to be put between parentheses and letters with descendants such as j or y in certain fonts.
```

The original `\pauza` of polski has the skips rigid (one is even a kern). We make the skips flexible. Moreover, our `\pauza` begins with `\ifhmode` to be usable also at the beginning of a line where it marks a part of a dialogue.

\pauza@skipcore
```
\def\pauza@skipcore{\hskip0.2em␣plus0.1em\relax
    \pauzacore
    \@ifnextchar,{% 2009/11/22 added a special case of a comma following pauza
    }{\hskip.2em␣plus0.1em\relax\ignorespaces}}%
```

\ppauza@skipcore
```
\def\ppauza@skipcore{\unskip\penalty10000\hskip0.2em␣
    plus0.1em\relax
        \ppauza@dash\hskip.2em␣plus0.1em\ignorespaces}
```

```
\AtBeginDocument{%
```

\pauza
```
    \pdef\pauza{%
      \ifhmode
        \unskip\penalty10000
        \hskip0.2em␣plus0.1em\relax
        \pauzacore\hskip.2em␣plus0.1em\relax\ignorespaces%
      \else
        \pauzadial
      \fi}%
```

According to *Instrukcja technologiczna. Skład ręczny i maszynowy* the dialogue dash (in Polish) should be followed by a rigid hskip of ½ em.

\pauzadial
```
    \pdef\pauzadial{%
      \leavevmode\pauzacore\penalty10000\hskip0,5em%
          \ignorespaces}
```

And a version with no space at the left, to begin a `\noindent`ed paragraph explaining e.g. a quotation:

\lpauza
```
    \pdef\lpauza{%
```

```
      \leavevmode
      \pauzacore\hskip.2em␣plus0.1em\ignorespaces}%
```

We define \ppauza as an en dash surrounded with thin stretchable spaces and sticking to the upper line or bare but discretionary depending on the next token being space$_{10}$. Of course you'll never get such a space after a literal CS so an explicit \ppauza will always result with a bare discretionary en dash, but if we \let−\ppauza…

\ppauza
```
    \pdef\ppauza{%
      \ifvmode␣␣␣␣\PackageError{gmutils}{%
        command␣\bslash␣ppauza␣(en␣dash)␣not␣intended␣for␣
            vmode.}{%
        Use␣\bslash␣ppauza␣(en␣dash)␣only␣in␣number␣and␣
            numeral␣ranges.}%
      \else
        \unskip\discretionary
        {\ppauza@dash}{\ppauza@dash}{\ppauza@dash}%
      \fi}%
}% of at begin document
```

```
\ifdefined\XeTeXversion
\AtBeginDocument{% to be independent of moment of loading of polski.
```

\−
```
  \pdef\—{%
    \ifhmode
      \unskip\penalty10000
      \afterfi{%
        \@ifnextspace{\pauza@skipcore}%
        {\@ifnextchar,{\pauza@skipcore}% a special case of comma added 2009/11/22
          {\@ifnextMac{\pauza@skipcore}%
            {\pauzacore\penalty\hyphenpenalty\hskip%
                \z@skip}}}%
      }% of \afterfi's argument
      \else
```

According to *Instrukcja technologiczna. Skład ręczny i maszynowy* the dialogue dash should be followed by a rigid hskip of ½ em.

```
      \leavevmode\pauzacore\penalty10000\hskip0,5em%
          \ignorespaces
    \fi}%
```

The next command's name consists of letters and therefore it eats any spaces following it, so \@ifnextspace would always be false, therefore we don't use it.

\−
```
  \pdef\—{%
    \ifvmode␣␣␣␣\PackageError{gmutils}{%
      command␣\bslash␣ppauza␣(en␣dash)␣not␣intended␣for␣
          vmode.}{%
      Use␣\bslash␣ppauza␣(en␣dash)␣only␣in␣number␣and␣
          numeral␣ranges.}%
    \else
      \afterfi{%
        \@ifnextspace{\ppauza@skipcore}{%
          \@ifnextMac\ppauza@skipcore
          {\unskip\discretionary
            {\ppauza@dash}{\ppauza@dash}{\ppauza@dash}}}%
      }%
```

```
6125      \fi
6126    }%
\emdash 6128    \def\emdash{\char`\— }
6129 }% of at begin document

\longpauza 6131 \def\longpauza{\def\pauzacore{— }}
\pauzacore 6132 \longpauza
\shortpauza 6133 \def\shortpauza{%
\pauzacore 6134    \def\pauzacore{\hbox{—\kern.23em\relax\llap{—}}}}%
\ppauza@dash 6135 \def\ppauza@dash{—}%

6138 \else␣% not XeTeX
\longpauza 6139 \def\longpauza{\def\pauzacore{---}}
\pauzacore 6140 \longpauza
\shortpauza 6141 \def\shortpauza{%
\pauzacore 6142    \def\pauzacore{--\kern.23em\relax\llap{--}}}%
\ppauza@dash 6143 \def\ppauza@dash{--}%

6146 \fi% of if XeTeX.
```

If you have all the three dashes on your keyboard (as I do), you may want to use them for short instead of `\pauza`, `\ppauza` and `\dywiz`. The shortest dash is defined to be smart in math mode and result with –.

```
6154 \ifdefined\XeTeXversion
6155 \foone{\catcode`— \active␣\catcode`—\active␣\catcode`-\active}{%
         %
\adashes 6156    \def\adashes{\AtBeginDocument\adashes}% because \pauza is defined
             at begin document.
\adashes 6158    \AtBeginDocument{\def\adashes{%
    \— 6159          \catcode`— \active␣\def— {\— }%
     \- 6160         \catcode`—\active␣\def—{\—}%
        6162         \addtomacro\dospecials{\do\—\do\— }%
        6163         \addtomacro\@sanitize{\@makeother\—\@makeother\— }%
        6164         \addtomacro\gmu@septify{\do\—13\do\—13\relax}%
6165 }}}
6166 \else
6167 \relaxen\adashes
6168 \fi
```

The hyphen shouldn't be active IMHO because it's used in TeX control such as `\hskip-2pt`. Therefore we provide the `\ahyphen` declaration reluctantly, because sometimes we need it and always use it with caution. Note that my active hyphen in vertical and math modes expands to $-_{12}$.

```
\gmu@dywiz 6177 \def\gmu@dywiz{\ifmmode-\else
6178    \ifvmode-\else\afterfifi\dywiz\fi\fi}%

6180 \foone{\catcode`-\active}{% aktivnyj diefis aktywny dywiz active hyphen
\ahyphen 6181      \def\ahyphen{\let-\gmu@dywiz\catcode`\-\active}}
```

To get current time. Works in ε-TeXs, including XeTeX. `\czas` typesets 8.18 and `\czas[:]` typesets 8:18.

```
\czas 6186 \newcommand*\czas[1][.]{%
6187    \the\numexpr(\time-30)/60\relax#1%
6188    \@tempcnta=\numexpr\time-(\time-30)/60*60\relax
6189    \ifnum\@tempcnta<10␣0\fi\the\@tempcnta}
```

```
6192 \@ifXeTeX{%
\textbullet 6193   \pdef\textbullet{%
6196     \iffontchar\font"2022␣\char"2022␣\else\ensuremath{%
             \bullet}\fi}%
\glyphname 6198   \pprovide\glyphname#1{%
6200     \XeTeXglyph␣\numexpr\XeTeXglyphindex␣"#1"\relax\relax}% since
             XeTeX ... \numexpr is redundant.
6202 }
\textbullet 6203 {\def\textbullet{\ensuremath{\bullet}}}
  tytulowa 6205 \newenvironment*{tytulowa}{\newpage}{\par\thispagestyle{%
         empty}\newpage}
```

To typeset peoples' names on page 4 (the editorial page):

```
\nazwired 6208 \def\nazwired{\quad\textsc}
```

Typesetting dates in my memoirs

A date in the `YYYY-MM-DD` format we'll transform into 'DD mmmm YYYY' format or we'll just typeset next two tokens/{...} if the arguments' string begins with `--`. The latter option is provided to preserve compatibility with already used macros and to avoid a starred version of `\thedate` and the same time to be able to turn `\datef` off in some cases (for SevSev04.tex).

```
\polskadata 6222 \pdef\polskadata{%
\gmu@datefsl 6223   \DeclareCommand\gmu@datefsl{%
6224     Q{0123456789\bgroup}>iT{/-}␣% (1) year
6225     Q{0123456789\bgroup}>iT{/-}␣% (2) month
6226     Q{0123456789\bgroup}␣% (3) day
6227     T{,}␣K{##1\gmu@datefsl}␣% (4, 5) additional stuff after comma
6228   }{%
6229       \IfValueF{##2}{\PutIfValue{##3}}%
6230       \IfValueT{##2}{%
6231         \@tempcnta=0##3\relax\the\@tempcnta
6232         \ifcase##2\relax\or␣stycznia\or␣lutego%
6233         \or␣marca\or␣kwietnia\or␣maja\or␣czerwca\or␣
               lipca\or␣sierpnia%
6234         \or␣września\or␣października\or␣listopada\or␣
               grudnia\else
6235         {}%
6236         \fi}%
6237         \IfValueT{##1}{\space␣##1}%
6238         \PutIfValue{##4}\IfValueT{##5}{␣##5}%
6239       }% of \gmu@datefsl.
6240   }% of \polskadata
6242 \polskadata
```

For documentation in English:

```
\englishdate 6245 \pdef\englishdate{%
\gmu@datefsl 6246   \DeclareCommand\gmu@datefsl{%
6247     Q{0123456789\bgroup}>iT{/-}␣% (1) year
6248     Q{0123456789\bgroup}>iT{/-}␣% (2) month
6249     Q{0123456789\bgroup}␣% (3) day
6250     T{,}␣␣K{##1\gmu@datefsl}␣% (4, 5) additional stuff after comma
```

```
  }{%
    \IfValueF{##2}{\PutIfValue{##3}}%
    \IfValueT{##2}{%
      \ifcase##2\relax\or␣January\or␣February%
        \or␣March\or␣April\or␣May\or␣June\or␣July\or␣August%
        \or␣September\or␣October\or␣November\or␣December%
        \else
        {}%
      \fi}%
    \space
    \@tempcnta=##3\relax\the\@tempcnta,
    \IfValueT{##1}{␣##1}%
    \PutIfValue{##4}\IfValueT{##5}{␣##5}%
  }% of \gmu@datefsl.
}%
```

Dates for memoirs to be able to typeset them also as diaries.

\ifdate
\bidate

```
\newif\ifdate
\pdef\bidate#1{%
  \gmu@datefsl#1\gmu@datefsl
}
```

\linedate
\linedate@@
\linedate@

```
\pdef\linedate{\gm@ifstar\linedate@@\linedate@}
\pdef\linedate@@#1{\linedate@{--}{}{}#1}}
\pdef\linedate@#1{\par
  \linedate@hook{#1}%
  \ifdate\addvspace{\dateskipamount}%
  \possvfil% if we put it before \addvspace, the v-space is always added.
  \date@line{\footnotesize\itshape␣\bidate{#1}}%
  \nopagebreak
  \else% %\ifnum\arabic{dateinsection}>0\dekbigskip\fi
    \addvspace{\bigskipamount}\possvfil
  \fi}% end of \linedate.
\let\linedate@hook\@gobble
\let\dateskipamount\medskipamount
```

\rdate

```
\pdef\rdate{\let\date@line\rightline␣\linedate}
```

\date@left

```
\def\date@left#1{\par{%
    \raggedright#1%
    \leftskip\z@skip
    \@@par}}%
```

\ldate

```
\pdef\ldate{%
  \let\date@line\date@left
  \linedate}
```

\runindate

```
\newcommand*\runindate[1]{%
  \paragraph{\footnotesize\itshape␣\gmu@datef#1\gmu@datef}%
  \stepcounter{dateinsection}}
```

I'm not quite positive which side I want the date to be put to so let's `let` for now and we'll be able to change it in the very documents.

```
\let\thedate\ldate
```

```
\zwrobcy 6316 \pdef\zwrobcy#1{\emph{#1}}␣%
```
ostinato, allegro con moto, garden party etc., także kompliment

```
\tytul 6319 \pdef\tytul#1{\emph{#1}}
```

Maszynopis w świecie justowanym zrobi delikatną chorągiewkę. (The `maszynopis` environment will make a delicate ragged right if called in a justified world.)

```
maszynopis 6325 \newenvironment{maszynopis}[1][]{#1\ttfamily
6326   \hyphenchar\font=45\relax% this assignment is global for the font.
6327   \@tempskipa=\glueexpr\rightskip+\leftskip\relax
6328   \ifdim\gluestretch\@tempskipa=\z@
6329   \tolerance900
```
it worked well with tolerance = 900.
```
6331   \advance\rightskip␣by\z@␣pluso,5em\relax\fi
6332   \fontdimen3\font=\z@% we forbid stretching spaces…
 %␣␣\fontdimen4\font=\z@ but allow shrinking them.
6334   \hyphenpenaltyo␣% not to make TEX nervous: in a typewriting this marvellous algorithm of hyphenation should be turned off and every line broken at the last allowable point.
6337   \StoreMacro\pauzacore
\pauzacore 6338   \def\pauzacore{-\rlap{\kern-o,3em-}-}%
6339 }{\par}
```

```
\justified 6343 \pdef\justified{%
6344   \leftskip=1\leftskip% to preserve the natural length and discard stretch and shrink.
6346   \rightskip=1\rightskip
6347   \parfillskip=1\parfillskip
6348   \advance\parfillskip␣by␣osp␣plus␣1fil\relax
6350   \let\\\@normalcr}
```

To conform Polish recommendation for typesetting saying that a paragraph's last line leaving less than `\parindent` should be stretched to fill the text width:

```
\fullpar 6355 \DeclareCommand\fullpar{%
6356   T{+-}%
6357   Q{+-0123456789}␣% optional looseness (most probably negative)
6358 }{%
6359   \begingroup
6360   \IfValueT{#1}{\looseness=#1\IfValueTF{#2}{#2}{1}\relax
6361     \multiply\tolerance␣by␣\tw@
6362   }%
6363   \fullparcore
6364   \par
6365   \endgroup}
```

```
\fullparcore 6367 \pdef\fullparcore{%
6368   \hunskip
6369   \parfillskip\z@skip}
```

To conform Polish recommendation for typesetting that says that the last line of a paragraph has to be 2`\parindent` long at least. The idea is to set `\parfillskip` naturally rigid and long as `\textwidth-2\parindent`, but that causes non-negligible shrinking of the inter-word spaces so we provide a declaration to catch the proper glue where the parindent is set (e.g. in footnotes parindent is 0 pt)

```
\twoparinit 6379 \newcommand*\twoparinit{% the name stands for 'last paragraph line's length
```

*min*imum *two* \parindent.

\twopar@defts
```
\def\twopar@defts{%
  \hsize-\leftskip-\rightskip-\fontcharwd\font`…}%
```
\twopar@atleast
```
\def\twopar@atleast{2\@parindent}%
```
\twopar
```
\DeclareCommand\twopar{%
  T{+-}% (1) you can specify loosening the paragraph by one only by typing
        single + and tightening by one by typing single -.
  Q{+-0123456789}% (2)
  A{\twopar@atleast}% (3)
  >iT{\cipolagwa}}{%
  \begingroup
  \IfValueT{##1}{%
    \looseness=##1\IfValueTF{##2}{##2}{1}\relax
    \multiply\tolerance␣by2
  }%
  \twoparcore<##3>%
  \par
  \endgroup
}% of \twopar.
```

```
\ifdefined\XeTeXversion
```
\twoparcore
```
  \DeclareCommand\twoparcore{%
    A{\twopar@atleast}>iT{\cipolągwa}}{%
    \hunskip␣% it's O.K. it's in a group, it'll work anyway.
    \edef\gmu@tempa{\the\dimexpr\twopar@defts-##1\relax}%
    \parfillskip=\glueexpr\gmu@tempa␣minus␣\gmu@tempa
    \relax% to delimit \glueexpr.
    \relax% to delimit the assignment.
  }%
\else␣% not XeTeX—doesn't use \fontcharwd.
```
\twoparcore
```
  \DeclareCommand\twoparcore{%
    A{\twopar@default}>iT{\cipolagwa}}{%
    \hunskip␣% it's O.K. it's in a group, it'll work anyway.
    {\setbox0=\hbox{\dots}%
      \xdef\gmu@tempa{\the\wd0}}%
    \edef\gmu@tempa{%
      \the\dimexpr\hsize-\leftskip-\rightskip
      -\gmu@tempa-2\@parindent\relax}%
    \parfillskip=\glueexpr\gmu@tempa␣minus␣\gmu@tempa
    \relax% to delimit \glueexpr.
    \relax% to delimit the assignment.
  }%
\fi
```

```
\AtBeginDocument{%
  \unless\ifdefined\@parindent
```
\@parindent
```
    \newskip\@parindent
    \@parindent=\parindent
  \fi
```
\restoreparindent
```
  \def\restoreparindent{\parindent\@parindent}%
}% of \AtBeginDocument.
}% of \twoparinit.
```

For dati under poems

Or explanations under results of time.

\wherncore
```
\DeclareCommand\wherncore{om}{%
    % [#1] o ptional value of \hskip of (left) indent of the parbox. If absent, parbox is aligned right;
    % [#2] o  text for the datum parbox.
  \IfValueTF{#1}{\leftline{%
      \whernfont
      \hskip#1\relax\parbox
      {\dimexpr\textwidth-\leftskip-\rightskip-#1}%
      {#2}% of \parbox,
    }% of \leftline,
  }% of ValueT{#1}.
  {% ValueF{#1}:
    \rightline
    {\whernfont
      \whern@parbox{#2}%
    }% of \rightline,
    \setprevdepth
  }% of ValueF{#1},
}% of \whercore.
```

\whern@parbox
```
\DeclareCommand\whern@parbox{%
  S{\leftskip\rightskip}% horizontal alignment of resulting box (the side to be ragged)
  O{t}␣% vertical alignment of parbox
  >is␣% separator
  O{0,7666\textwidth}␣% (3) width of parbox
  m␣% (4) parbox contents
}{%
 %  #1  S  the skip of the ragged side,
 %  #2  S  the \parbox's contents.
  \parbox[#2]{#3}{%
    \IfValueTF{#1}{#1}{\leftskip}=0sp␣plus␣\textwidth
    \parfillskip0sp\relax
    \let\\\linebreak
    \disobeylines
    \whernfont␣#4\unskip\strut\endgraf
    \getprevdepth
  }% of \parbox,
}% of \whern@parbox.
```

\whern
```
\def\whern{%
  \endgraf\nopagebreak
  \gm@ifstar{\whercore}%
  {\vskip\whernskip\whercore}}
\let\whernfont\footnotesize
```

\whernskip
```
\newskip\whernskip
\whernskip2\baselineskip␣minus␣2\baselineskip\relax
```

\whernup
```
\DeclareCommand\whernup{%
```

```
o␣%  a vskip before
>is␣%  separating star (ignored)
o␣␣%  (2) custom width of parbox
>Pm}{\par
\IfValueT{#1}{\vskip#1\relax}%
\leftline{%
  \IfValueTF{#2}{\whern@parbox\rightskip[b][#2]}%
  {\whern@parbox\rightskip[b]}%
  {#3}%
}%
\setprevdepth
\nopagebreak\relax
\@ifenvir{quote}{\noindent\ignorespaces}{}}
```

Thousand separator

\thousep

```
\pdef\thousep#1{%  a macro that'll put the thousand separator between every two three-digit groups.
```

First we check whether we have at least five digits.

```
  \gmu@thou@fiver#1\relax\relax\relax\relax\relax%  we put five \relaxes after the parameter to ensure the string will meet \gmu@thou@fiver's definition.
  \gmu@thou@fiver{#1}{%  if more than five digits:
    \emptify\gmu@thou@put
    \relaxen\gmu@thou@o\relaxen\gmu@thou@i\relaxen%
        \gmu@thou@ii
    \@tempcnta\z@
    \gmu@thou@putter#1\gmu@thou@putter
    \gmu@thou@put
  }}
```

\gmu@thou@fiver

```
\def\gmu@thou@fiver#1#2#3#4#5\gmu@thou@fiver#6#7{%  this macro only checks if the text delimited with itself consists of at least five tokens/braces
  \ifx\relax#5\relax\@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi{#6}{#7}}
```

\gmu@thou@putter

```
\def\gmu@thou@putter#1#2{%  we are sure to have at least five tokens before the sentinel \gmu@thou@putter.
  \advance\@tempcnta\@ne
  \@tempcntb\@tempcnta
  \divide\@tempcntb3\relax
  \@tempcnta=\numexpr\@tempcnta-\@tempcntb*3
  \edef\gmu@thou@put{\@xau{\gmu@thou@put}\unexpanded{#1}%
    \ifx\gmu@thou@putter#2\else
      \ifcase\@tempcnta
        \gmu@thou@o\or\gmu@thou@i\or\gmu@thou@ii%  all three CSes are yet \relax so we may put them in an \edef safely.
      \fi
    \fi}%  of \edef
  \ifx\gmu@thou@putter#2%  if we are at end of the digits…
    \edef\gmu@tempa{%
      \ifcase\@tempcnta
        \gmu@thou@o\or\gmu@thou@i\or\gmu@thou@ii
```

```
      \fi}%
    \@xa\let\gmu@tempa\gmu@thousep% ... we set the proper CS...
  \else% or ...
    \afterfi{% iterate.
      \gmu@thou@putter#2}% of \afterfi
  \fi% of if end of digits.
}% of \gmu@thou@putter.
```

\gmu@thousep
```
\def\gmu@thousep{\,}% in Polish the recommended thousand separator is a thin space.
```

So you can type `\thousep{7123123123123}` to get 7 123 123 123 123. But what if you want to apply `\thousep` to a count register or a `\numexpr`? You should write one or two `\expandafter`s and `\the`. Let's do it only once for all:

\xathousep
```
\pdef\xathousep#1{\@xa\thousep\@xa{\the#1}}
```

Now write `\xathousep{\numexpr␣10*9*8*7*6*120}` to get 3 628 800.

\shortthousep \thous
```
\def\shortthousep{%
  \DeclareCommand\thous{Q{+-0123456789}}{%
```

we declare it as a command with Q-type argument to allow spaces between digits.

```
    \ifmmode\hbox\bgroup\@gmu@mmhboxtrue\fi
    \IfValueTF{##1}{% we are given a sequence of digits
      \@tempcnta=##1\relax
      \ifnum\@tempcnta<0␣$-$%
        \@tempcnta=-\@tempcnta
      \fi
      \xathousep\@tempcnta
      \if@gmu@mmhbox\egroup
      \else\@xa\spifletter
      \fi
    }%
    {% no bare digits given, then we assume the argument is braced.
      \thousep
    }%
  }% of \thous.
}% of \shortthousep.
```

And now write `\thous␣3628800` to get 3 628 800 even with a blank space (beware of the range of TEX's counts).

hyperref's \nolinkurl into \url*

\urladdstar \url
```
\def\urladdstar{%
  \AtBeginDocument{%
    \@ifpackageloaded{hyperref}{%
      \StoreMacro\url
      \pdef\url{\gm@ifstar{\nolinkurl}{\storedcsname{url}}}%
    }{}}}
\@onlypreamble\urladdstar
```

Footnotes suggested by Andrzej Tomaszewski

\ATfootnotes
```
\DeclareCommand\ATfootnotes{s}{%
```

We make the footnote mark in the footnote `\scriptsize` not `\scriptscriptsize`.

```
    \IfValueT{#1}% the following setting is suitable for old style numbers in footnote marks, therefore I place it in the starred version of the command.
    {\prependtomacro\gmu@ATfootnotes{%
        \pdef\@makefnmark{%
          \mbox␣{\normalfont␣\textsuperscript␣{\smaller[3]%
              \@thefnmark␣}}}%
      }% of prepend,
    }% of \IfValueT.

    \gmu@ATfootnotes
    \gmu@AT@ampulex\maketitle% without hyperref
    \ifdefined\HyOrg@maketitle
      \afterfi{\gmu@AT@ampulex\HyOrg@maketitle}% with hyperref
    \fi
}
\gmu@AT@ampulex
\pdef\gmu@AT@ampulex#1{%
    \ampulexdef#1{\def\@makefnmark}%
    \if@twocolumn
    {\gmu@ATfootnotes\if@twocolumn}% Ampulex redefinition of \maketitle for the standard classes.
\@makefntext
    \ampulexdef#1{\long\def\@makefntext}%
    \if@twocolumn{\gmu@ATfootnotes\if@twocolumn}% Ampulex redefinition of \maketitle for mwcls.
}
\gmu@ATfootnotes
\pdef\gmu@ATfootnotes{%
```

And we make the footnote number not be in superscript but on the base line, according to Andrzej Tomaszewski's suggestion on BachoTEX 2008, and the same size as in the footnote mark.

```
\@makefntext
    \long\pdef\@makefntext##1{%
      \ifdefined\@parindent␣\parindent\@parindent
      \else␣\parindent␣1em\relax
      \fi
      \indent{\ATf@font\scriptsize%
        {\@thefnmark}}%
      \gmu@fnhook
      \enspace\ignorespaces##1}%
}
\let\ATf@font\normalfont

\emptify\gmu@fnhook
\rrthis
\pdef\rrthis{% 'rag right this': make only the current paragraph ragged right (e.g. if the paragraph consists of a long URL).
    \begingroup\rightskip=0sp␣plus␣\hsize␣\endgraf\endgroup}
\centerthis
\pdef\centerthis{% 2009/12/15
    \begingroup
    \rightskip=1\rightskip␣plus␣\hsize
    \leftskip=1\leftskip␣plus\hsize
    \parfillskip=\z@skip
    \endgraf\endgroup}
\balsmiley
\def\balsmiley#1␣{}% to balance parentheses and brackets in smileys. ;-)
```

```
% \balsmiley(␣;-) .
```

\scantnoline
```
\long\pdef\scantnoline#1{% 'rescan tokens without adding line end'
  {\endlinechar\m@ne\scantokens{#1}}}
```

A fix to the url package

It happened that a URLs typeset with the `\url` command of the url package came out sort of spaced because kerning was off because of the math mode. So I provide a redefinition of the internal macros of the url package which in my version uses not math mode but `\scantokens` and not `\relpenalty` and `\binoppenalty` but `\hyphenpenalty` (as it is in the paragraph) and `\discretionary`. I tried putting explicit penalties after the symbols but that spoiled kerning.

The rules of line breaking are somewhat different, too: in the original url package line breaks are forbidden between any two symbols listed in `\UrlBigBreaks`. In my version line breaks are forbidden between any two *identical* 'URL Breaks' and 'URL Big Breaks'.

There are some more differences in formatting some chars, i.a. `~`, `%` and angle brackets which I don't treat specially and just take from font assuming the font provides ASCII chars and checking whether it provides the angle brackets.

```
\@ifXeTeX{%
```

\UrlFix
```
  \pdef\UrlFix{\AtBeginDocument{%
      \@ifpackageloaded{url}{\gm@UrlFix}{}}%
    \relaxen\UrlFix}%
  \AtBeginDocument{%
```

\UrlFix
```
    \pdef\UrlFix{%
      \@ifpackageloaded{url}{\gm@UrlFix}{}%
      \relaxen\UrlFix}}%
}
{%
```

\UrlFix
```
  \pdef\UrlFix{\PackageWarning{gmutils}{!!!␣The␣\string%
      \UrlFix\space
      declaration␣works␣only␣with␣XeTeX}}%
}
```

```
\@ifXeTeX{}{%
  \edef\gmu@restoreUpUpUp{\catcode`\@nx\^^=\the\catcode`%
      \^^}%
  \AtEndOfPackage\gmu@restoreUpUpUp
  \catcode`\^^=9␣}
```

\gm@UrlFix
```
\def\gm@UrlFix{%
```

default style assignments

\UrlBreaks
```
  \def\UrlBreaks{\do\.\do\@\do\\\do\/\do\!\do\_\do\|\do\;%
      \do\]%
    \do\)\do\,\do\?\do\'\do\"\do\+\do\=\do\#\do\%\do\~\do%
        \_\do\|%
    \do\{\do\}\do\$}%
```

\UrlBigBreaks
```
  \def\UrlBigBreaks{\do\:}%
```

\UrlNoBreaks
```
  \def\UrlNoBreaks{\do\(\do\[\do\{}%
```

\UrlSpecials
```
  \def\UrlSpecials{%
    \do\␣{\hbox{\visiblespace}}\do\^^M{\hbox{%
        \visiblespace}}}%
```

```
\Url@Format 6741 \def\Url@Format##1{%
6742   \UrlFont
6743   \ifdefined\verbatim@specials
6744     \catcode`\>\active
6745     \verbatim@specials
6746     \verbatim@mathhack
6747   \fi␣% setting of the escape char, begin and end group and optionally math
          shift, defined in gmverb.
6750   \gm@UrlSetup
6751   \UrlLeft
6752   \edef\gmu@theendlinechar{\the\endlinechar}%
6753   \endlinechar\m@ne
6754   \kern\z@% to forbid hyphenating the first word if the URL begins with
          a word
6756   \hyphenchar\font=\UrlHyphenchar\relax
6757   \let\-\gmu@discretionaryhyphen
6758   \scantokens{##1}%
6759   \endlinechar\gmu@theendlinechar\relax
6760   \UrlRight
6761 }% of \Url@Format.

6763 \edef\UrlHyphenchar{%
6764   \ifdefined\gmv@hyphenchar\gmv@hyphenchar
6765   \else"A6␣\fi}% broken bar, ¦ or the same as provided in gmverb for verba-
          tims. You can redefine it as you please. This char is used as the hyphen-
          ation char in URLs and therefore should be different from - (hyphen),
          which is often a part of an URL. The broken bar seems to be quite unlikely
          in URLs and/or file names.

\verbatim@mathhack 6773 \def\verbatim@mathhack{%
6774   \ifdefined\verbatim@specials@list
6775     \@xa\verbatim@mathhack@\verbatim@specials@list
6776   \fi
6777 }%

\verbatim@mathhack@ 6779 \def\verbatim@mathhack@##1##2##3##4##5##6{%
6780   \IfValueT{##4}{%
6781     \edef\gmu@thinmuskip{\the\thinmuskip}%
6782     \edef\gmu@medmuskip{\the\medmuskip}%
6783     \edef\gmu@thickmuskip{\the\thickmuskip}%
6784     \begingroup
6785     \lccode`\~=`##4\lowercase{%
6786       \endgroup\def~####1~}%
\thinmuskip 6787     {$\thinmuskip\gmu@thinmuskip\relax
6788       \medmuskip\gmu@medmuskip\relax
6789       \thickmuskip\gmu@thickmuskip\relax
6790       ####1%
6791       $}%
6792     \catcode`##4\active
6793   }%
6794 }%

\gm@UrlSetup 6796 \def\gm@UrlSetup{%
6797   \medmuskip\Urlmuskip␣\thickmuskip\medmuskip␣%
          \thinmuskipomu%
```

```
      \relpenalty\UrlBigBreakPenalty␣\binoppenalty%
          \UrlBreakPenalty
      \def\do{\gmu@doUrlMath\UrlBreakPenalty}\UrlBreaks␣% bin ( \hy|
          phenpenalty anyway)
      \def\do{\gmu@doUrlMath\UrlBigBreakPenalty}\UrlBigBreaks␣% rel
          ( \hyphenpenalty anyway)
      \def\do{\gmu@doUrlMath\@M}\UrlNoBreaks␣% open (no break)
      \def\do{\gmu@doUrlMathAc\UrlBreakPenalty}% ( \hyphenpenalty)
      \UrlSpecials
      \if␣\iffontchar\font"2329␣1\else0\fi\iffontchar%
          \font"232A␣1\else2\fi
```

we check whether the font provides both left and right angle brackets.

```
        \gmu@measurewd{^^^^2329}%
        \edef\gmu@tempa{%
          \@nx\gmu@doUrlMathAc\@M\@nx\<{%
            \hbox␣to\gmu@tempb{\unexpanded{\hss\char"2329␣%
                \hss}}}%
          }\gmu@tempa
        \gmu@measurewd{^^^^232a}%
        \edef\gmu@tempa{%
          \@nx\do\@nx\>{%
            \hbox␣to\gmu@tempb{\unexpanded{\hss\char"232A␣%
                \hss}}}%
          }\gmu@tempa
      \else
        \gmu@doUrlMathAc\@M\<{\langle}\do\>{\rangle}%
      \fi
      \iffontchar\font"22C6␣% low star
        \do\*{\hbox{\char"22C6␣}}%
      \else␣\do\**%
      \fi
      \ifx\do@url@hyp\@empty
        \gmu@measurewd{-}% this macro is defined in line 5197.
        \edef\gmu@tempa{%
          \unexpanded{\gmu@doUrlMathAc\@M\-}%
          {\hbox␣to␣\gmu@tempb{\unexpanded{\hss-\hss}}%
            \@nx\-}% hyphen is a good point for hyphenation, but the hyphen-
                ation char should be sth. else, and it is indeed: ¦ (broken bar,
                \char"A6). See also line 6765
        }\gmu@tempa
      \fi
      \addfontfeature{Ligatures=NoCommon,␣Mapping=none}% instead of
          'doing' \verbatim@nolig@list.
    }% of \gm@UrlSetup.
```

\gmu@doUrlMath

```
    \def\gmu@doUrlMath##1##2{%
      % #1 value of the penalty (used as a Boolean: if < 10 000,
        % \hyphenpenalty will be used anyway, if ≥ 10 000, there will be no
        % \discretionary),
      % #2 the char, given as \⟨char⟩.
      \begingroup
      \lccode`\~=`##2\lowercase{%
        \endgroup\def~{\@ifnextchar~}%
```

```
      \@xa\addtomacro\@xa~}% of \lowercase.
    \ifnum##1<\@M
    {%
      {\char`##2\csname␣gmu@dbl\string##2kern\endcsname}% if next
           is the same char
      {\ifmmode\char`##2% else
        \else\gmu@urlbreakable{##1}{##2}%
        \fi}%
    }% of \addtomacro's argument \ifnum true.
    \else
    {%
      {\char`##2\csname␣gmu@dbl\string##2kern\endcsname}{%
           \char`##2}%
    }% of \addtomacro's argument \ifnum false.
    \fi
      \catcode`##2=\active
    }% of \gmu@doUrlMath.
\gmu@doUrlMathAc  \def\gmu@doUrlMathAc##1##2##3{%
      % #1 (value of) a penalty (see the remark to ##1 of the previous macro),
      % #2 the char (as \⟨char⟩),
      % #3 the definition.
    \begingroup
    \lccode`\~=`##2\lowercase{%
      \endgroup\def~{\@ifnextchar~}%
      \@xa\addtomacro\@xa~}% of \lowercase.
    \ifnum␣##1<\@M
    {%
      {\ifmmode\char`##2\else$##3\m@th$\fi}%
      {\ifmmode\char`##2%
        \else\discretionary{\hbox{$##3\m@th$}}{}{\hbox{$##3%
             \m@th$}}%
        \fi}%
    }% of \addtomacro's argument if num true.
    \else
    {%
      {\ifmmode\char`##2\else$##3\m@th$\fi}{\ifmmode%
           \char`##2\else$##3\m@th$\fi}%
    }% of \addtomacro's argument if num false.
    \fi
    \catcode`##2=\active
  }% of \gmu@doUrlMathAc.
\gmu@url@rigidbreak  \pdef\gmu@url@rigidbreak##1##2{\discretionary{\char`##2}{%
       }{\char`##2}}%
\gmu@url@flexbreak  \pdef\gmu@url@flexbreak##1##2{\penalty\@M␣\hskip\z@␣
       pluso,o3em
    \char`##2\penalty##1\hskip\z@␣pluso,o3em\relax}%
  \let\gmu@urlbreakable\gmu@url@flexbreak
\Url@z  \def\Url@z##1{%
```

Do any hyper referencing due to hyperref (or perform a url-def)

```
    \Url@HyperHook
```

Now do the formatting in a group (can also have `\Url@HyperHook` take this as an argument).

```
      {\Url@Format{##1}}%
      \endgroup}%
  \DeclareUrlCommand\file{\urlstyle{sf}}%
  \emptify\Url@moving% with our settings \url is pretty allowed in moving
        arguments, I hope.
}% of \gmu@UrlFix.
```

\UrlSlashKern

```
\DeclareCommand\UrlSlashKern{O{tt}m}%
{\AtBeginDocument{%
    \@nameedef{url@#1style}{\def\@nx\UrlFont{%
        \@xa\@nx\csname#1family\endcsname
        \def\@xa\@nx\csname␣gmu@dbl\string\/kern\endcsname
        {\kern#2\relax}%
      }% of \UrlFont
    }% of \url#1style
    \urlstyle{#1}%
  }% of \AtBeginDocument
}% of \UrlSlashKern
```

Conditional tilde

Polish typesetting standards say that for 12 dd and 10 dd ⟨*czcionki*⟩ if leading is narrower that 3½ ⟨*kwadratu*⟩, 3½ × 48dd, then hanging letters are allowed, which also applies to 8 and 6 dd ⟨*czcionki*⟩ in less than 3 ⟨*kwadraty*⟩ leading. I treat this recommendation not strictly but as an inspiration, that is I translate »dd« to »pt«.

\TrzaskaTilde

```
\def\TrzaskaTilde{%
  \@xa\DeclareCommand\@xa\gm@smarttilde
  \@xa{\@xa␣S\@xa{\all@stars~}}{%
    \IfValueTF{##1}{\nobreakspace{}}%
    {\ifdim\dimexpr\hsize-\leftskip-\rightskip
      -\ifdim\hangindent<\z@-\fi\hangindent␣% the last parameter is
          used with respect to the floatflt package.
      >%
        \ifdim\f@size␣pt>\dimexpr10opt-1sp\relax
          168dd
        \else
          144dd
        \fi
        \nobreakspace␣{}%
      \else
        \␣%
      \fi
    }% of \gm@ifstar's else,
  }% of \gm@smarttilde,
  \let~\gm@smarttilde
}% of \TrzaskaTilde.
```

\getprevdepth

```
\pdef\getprevdepth{%
  \endgraf
  \xdef\setprevdepth{\prevdepth=\the\prevdepth\relax}%
```

```
}
```

Storing the catcode of line end

\StoreCatM
```
\def\StoreCatM{%
  \protected\edef\RestoreCatM{%
    \catcode`\@nx\^^M=\the\catcode`\^^M\relax}%
}
```

\RestoreCatM
```
\pdef\RestoreCatM{\PackageE{gmutils}{first␣store␣the␣catcode␣
    of
    ^\empty^\empty␣M␣with␣\string\StoreCatM.}%
}
```

A really empty page

Copied from Marcin Woliński's macros.

\clearemptydoublepage
```
\newcommand{\clearemptydoublepage}{%
          \newpage{\pagestyle{empty}\cleardoublepage}}
```

\setspaceskip
```
\DeclareCommand\setspaceskip{%
  A{}% optional factor for all three components
  O{\fontdimen2\font}%
 >is
 O{\fontdimen3\font}%
 >is
 O{\fontdimen4\font}}
{\spaceskip=#1#2plus#1#3minus#1#4\relax}
```

```
\foone\obeylines{%
```

\disobeylines
```
  \def\disobeylines{% for arguments in which line end is active to simulate
      normal behaviour
    \ifnum\catcode`\^^M=\active%
```

\@ifnextgroup
```
      \pdef^^M{\@ifnextgroup{\ifhmode\unskip\space\fi}{%
```

\gmu@disMinner
```
          \gmu@disMinner}}%
      \def\gmu@disMinner##1{%
        \ifx^^M##1\endgraf%
        \else\afterfi{\ifhmode\unskip\space\fi}\fi##1}%
    \fi}%
}
```

\makestarlow
```
\def\makestarlow{%
  \begingroup\lccode`\~=`\*\lowercase{%
```

* \gmu@lowstar
```
    \endgroup\def~{\gmu@lowstar}}% 2009/10/19 \let changed to \def
        to allow redefinitions of \gmu@lowstar.
  \catcode`\*=\active
  \defLowStarFake
}
```

\defLowStarFake
```
\DeclareCommand\defLowStarFake{%
  Q{+-0123456789,.}{0,5}% fraction of fontchar depth of the star glyph
}%
{%
```

```
\gmu@lowstarfake 7034   \def\gmu@lowstarfake{%
7035     \leavevmode\vbox{\hbox{*}\kern#1\fontchardp\font`*}%
7036   }%
7037 }
```

```
\gmu@lowstarfake 7040 \def\gmu@lowstarfake{*}␣%
```
useful for next command where normal star is low.

The `\*` CS and active `*` should be defined different to make them distinguishable by tests, especially with `\gm@ifstar` in mind.

```
\*  7048 \DeclareCommand\*{Q{0123456789}{1}}
7050 {\gmu@star@loop0{#1}}
```

```
\gmu@star@loop 7052 \def\gmu@star@loop#1#2{%
```
this is an expandable loop as in The ε-TEX Manual p. 9.

```
7054     \ifnum#1<\numexpr#2\relax%
7055       \gmu@lowstar
7056       \@xa\gmu@star@loop
7057       \@xa{\number\numexpr#1+1\@xa}%
7058       \@xa{\number#2\@xa}%
7059     \fi}
```

```
7063 \endinput
```

d. The gmiflink Package[1]

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

LPPL status: "author-maintained".

```
45 \NeedsTeXFormat{LaTeX2e}
46 \ProvidesPackage{gmiflink}
47       [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 omitted.

Contents of the gmiflink.zip archive

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

1 This file has version number v0.97 dated 2006/08/16.

gmiflink.sty
README
gmiflink.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 `\noexpand`ed in `\edef`s.

```
199 \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%
}}
```

e. The gmverb Package[1]

March 4, 2010

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

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

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmverb}
    [2010/03/04 v0.93 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 their 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`.

`\nobreakbslash` (If you don't like line breaking at backslash, there's the `\nobreakbslash` declaration (observing the common scoping rules, hence OCSR) and an analogous declaration for the left brace: `\nobreaklbrace`.)

`\nobreaklbrace`

The default 'hyphen' is `%` since it's the default comment char. If you wish another char to appear at the line break, use the `\VerbHyphen` declaration that takes \⟨*char*⟩ as the only argument. This declaration is always global.

`\VerbHyphen`

`\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 v0.93 dated 2010/03/04.

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, gmverb provides a macro `\dekclubs` as a shorthand for `\MakeShortVerb(*)\|`.

\dekclubs

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

\DeleteShortVerb

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., `$2|o$` to get 2|o 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 `\Old¦MakeShortVerb` instead, what I do when I like to display short verbatims in display-math.

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

\edverbs There's one more declaration, `\edverbs` that makes `\[` checks if the next token is an active char and opens an `\hbox` if so. That is done so that you can write (in `\edverbs`' and `\dekclubs`' scope)

```
\[|<verbatim stuff>|\]
```

instead of

```
\[\hbox{|<verbatim stuff>|}\]
```

to get a displayed shortverb.

Both versions of `\dekclubs` OCSR.

The `verbatim` environment inserts `\topsep` before and after itself, just as in standard version (as if it was a `list`).

In August 2008 Will Robertson suggested grey visible spaces for gmdoc. I added a respective option to gmdoc but I find them so nice that I want to make them available for all verbatim environments so I bring here the declaration `\VisSpacesGrey`. It redefines only the visible spaces so affects `\verb*` and `verbatim*` and not the unstarred versions. The colour of the visible spaces is named `visspacesgrey` and you can redefine it xcolor way.

\VisSpacesGrey visspacesgrey

\verbatimspecials We also provide the `\verbatimspecials` declaration that takes six arguments:

`#1` `m` a char for verbatim escape char (for catcode 0), has to be unbraced[2],
`#2` `m` a char for group starter (for catcode 1), has to be unbraced,
`#3` `m` a char for group ender (for catcode 2), has to be unbraced,
`[#4]` (optional) a char for verbatim math shift (for catcode 3); it has to be in square brackets if present. If absent, nothing is set for the verbatim math shift,
`[#5]` (optional) a char for the shorthand for `\metachar`; it has to be in square brackets if present.

2 To be precise, the arguments cannot be wrapped in curly braces because those are recatcoded to 'other'. But if you make some other pair of chars category 1 and 2 that are not on the `\dospecials` list, then you can wrap the arguments in those chars, but what for?

`{#6} b` optional in curly braces, additional stuff (commands) to be executed in a verbatim.

For example, after telling TEX

```
\verbatimspecials␣⁄␣«␣»␣[␣¿␣]␣[␣›␣]␣{\def\|{$\vert$}}
```

(the slash is Unicode Fractional Slash, spaces are ignored) you can write

```
|\macro⁄arg«arg. ¿n+1¿»\›[No›]Value›(T⁄|F›)|
```

to get

\macro{⟨*arg.* $n+1$⟩}\[No]Value(T|F)

To get a 'verbatim special' verbatim, precede it with the 'verbatim escape': `⁄⁄␣⁄«␣⁄»␣⁄¿␣⁄›` in the example above.

Note also that `⁄|` is a control sequence so it doesn't delimit the short verbatim `|`'s argument.

The `\verbatimspecials` declaration OCSR. Subsequent uses of it override the previous settings. If you specified the optionals at first and then specify `\verbatim¦specials` without optionals, the previous optional settings are forgotten.

`\noverbatimspecials` To turn the 'verbatim specials' off write `\noverbatimspecials`, which OCSR too.

Note that although we don't provide a 'verbatim superscript' nor 'verbatim subscript', you have the `\sups` and `\subs` CS'es defined by gmutils.

`\sups` `\subs` The 4th argument for the math shift is optional because you can use LATEX's `\(` and `\)`.

`\(` `\)` `alltt` The `\verbatimspecials` declaration goes a step further than LATEX's `alltt` and `semiverbatim` Til Tantau's beamer's `semiverbatim` environments. To get their effect, declare

```
\verbatimspecials\{}
```

`\scanverb` There is something for verbatims in arguments of commands: `\scanverb[*]{`⟨*text*⟩`}`. However there are some limitations: if `%` is the comment char (which is usual situation), then you cannot use `%` in ⟨*text*⟩, or rather, `%` will act as comment char anyway. Moreover, spaces are ignored. This last limitation may be worked around if you declare `\verba¦timspecials`, say `⁄` (fraction slash) as the escape char. Then you can use `⁄␣` to put a space which will be typeset blank in the unstarred version and visible with star.

Not so long ago I started to use the 'broken bar' (U+00A6, ¦) character as a hyphen in hyperlinks, because it seems not to occur in hyperlinks at all unlike hyphen. I suggest the `erbDiscretionaryHyphen` same char for verbatims, but if you don't like it, there's the `\verbDiscretionary¦Hyphen` declaration that takes two arguments. Broken bar is declared as

```
\verbDiscretionaryHyphen{"A6}{¦}
```

The package options

As many good packages, this also does not support any options.

The remarks about installation and compiling of the documentation are analogous to those in the chapter gmdoc.sty and therefore omitted.

Contents of the gmverb.zip archive

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

gmverb.sty
README

gmverb.pdf
gmverb.tds.zip

This package requires another package of mine, gmutils, also available on CTAN.

The code

Preliminaries

```
\RequirePackage{gmutils}[2008/10/08]
```

For `\firstofone`, `\afterfi`, `\gmobeyspaces`, `\@ifnextcat`, `\foone` and `\noexpand`'s and `\expandafter`'s shorthands `\@nx` and `\@xa` resp.

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 $\backslash\langle char\rangle$ form (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'.

\breaklbrace
```
\def\breaklbrace{%
  \discretionary{\type@lbrace\verbhyphen}{}{\type@lbrace}%
  \yeshy}
```

```
\foone{\catcode`\[=1␣\catcode`\{=\active␣\catcode`\]=2␣}%
[%
```

\dobreaklbrace
```
   \def\dobreaklbrace[\catcode`\{=\active
   \def{%
```

\breaklbrace
```
     [\breaklbrace\gm@lbracehook]]%
]
```

Now we only initialise 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{\}%
  }% of \foone.
```

\breakbslash
```
\def\breakbslash{%
  \discretionary{\verbhyphen}%
```

```
    {\type@bslash}{\type@bslash}\yeshy% it seems that we allow hyphen-
        ation after backslash but hyphenation will be allowed iff \hyphenchar
        % \font is nonnegative.
}% of \breakbslash.
```

Sometimes line breaking at a backslash may be unwelcome. The basic case, when the first CS in a verbatim breaks at the line end leaving there `%`, is covered by line 801. For the others let's give the user a counter-crank:

\nobreakbslash
\breakbslash

```
\pdef\nobreakbslash{\def\breakbslash{\type@bslash\yeshy}}% to use
    due to the common scoping rules. But for the special case of a backslash open-
    ing a verbatim scope, we deal specially in the line 801.
```

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

\nobreaklbrace
\breaklbrace

```
\pdef\nobreaklbrace{\def\breaklbrace{\type@lbrace\yeshy}}
```

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

\dobreakbslash
\breakbslash

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

\breakablevisspace

```
\def\breakablevisspace{\discretionary{\visiblespace}{}{%
    \visiblespace}}
```

The `\visiblespace` macro is `\let` in gmutils to `\xiispace` or to `\xxt@visi¦blespace` of xltxtra if available.

```
\foone\obeyspaces% it's just re\catcode'ing.
{%
\newcommand*\dobreakvisiblespace{\def␣{\breakablevisspace}%
    \obeyspaces}% \defing it caused a stack overflow disaster with
    gmdoc.
\newcommand*\dobreakblankspace{\let␣=\space\obeyspaces}%
}
```

\dobreakvisiblespace
\breakablevisspace
\dobreakblankspace

```
\foone{\@makeother\|}{%
  \def\xiiclub{|}}
```

\xiiclub

Almost-Knuthian `\ttverbatim`

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

\ttverbatim

```
\pdef\ttverbatim{%
  \let\do=\do@noligs␣\verbatim@nolig@list
  \let\do=\@makeother␣\dospecials
  \dobreaklbrace\dobreakbslash
  \dobreakspace
  \makeatletter
  \ifhmode
    \setspaceskip
  \fi
  \tt
  \edef\gmv@storedhyphenchar{\the\hyphenchar\font}%
```

```
  \hyphenchar\font=\gmv@hyphenchar
  \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 \VerbT \ttverbatim@hook

```
\def\VerbT1{\def\ttverbatim@hook{\fontencoding{T1}%
      \selectfont}}
```

We wish the visible spaces to be the default.

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

The core: from shortvrb

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

\MakeShortVerb \@shortvrbdef \@shortvrbdef

```
\def\MakeShortVerb{%
  \gm@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}
```

\DeleteShortVerb

```
\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 CSes `\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 begin definition of your environment and then necessarily `\endverbatim` in its end definition. Of course (or maybe *surprisingly*), the commands written in the begin definition after `\verbatim` will also be executed at `\begin{`⟨*environment*⟩`}`.

```
verbatim        \def\verbatim{%
\verbatim         \edef\gmv@hyphenpe{\the\hyphenpenalty}%
                  \edef\gmv@exhyphenpe{\the\exhyphenpenalty}%
                  \@beginparpenalty␣\predisplaypenalty␣\@verbatim
                  \frenchspacing␣\gmobeyspaces␣\@xverbatim
```

```
    \hyphenpenalty=\gmv@hyphenpe\relax
    \exhyphenpenalty=\gmv@exhyphenpe
    \hyphenchar\font=\m@ne
}% in the LaTeX version there's \@vobeyspaces instead of \gmobeyspaces.
\@namedef{verbatim*}{\@beginparpenalty␣\predisplaypenalty␣%
    \@verbatim
    \@sxverbatim␣% it's the same as \@xverbatim and defines the verbatim end
        (a macro delimited with \end{/<curr.envir.>}).
}
\def\endverbatim{\@@par
    \hyphenchar\font=\gmv@storedhyphenchar␣% hyphenchar assignments
        are always global.
    \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%
!gdef!@xverbatim[%
    [!endlinechar!m@ne␣!everyeof[!@nx]%
      !edef!verbatim@currenvir[%
        !@xa!scantokens!@xa[!@currenvir]%
      ]% of \verbatim@currenvir. This macro is defined as the meaning of
          % \@currenvir rescanned. It's done specially for the active star in my
          verbatims. \@currenvir is fully expanded but my active star is \pro¦
          tected.
      !@xa]% and here a little trick with groups:
      !@xa!def!@xa!verbatim@currenvir
    !@xa[!verbatim@currenvir]%
    !edef!verbatim@edef[%
    !def!@nx!verbatim@end%
    ####1!noexpand\end!@nx{%
      !@xa!unexpanded!@xa[!verbatim@currenvir]%
    }[%
      ####1!@nx!end[!@currenvir]]]%
    !verbatim@edef
    !verbatim@end]%
!endgroup

\let\@sxverbatim=\@xverbatim
```

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

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

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

```
    \parsep\parskip
```

From `\@trivlist`:

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

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

```
    \@@par\addvspace\@topsep
    \if@minipage\else\vskip\parskip\fi
    \advance\@totalleftmargin\verbatimleftskip
    \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
    \verbatim@specials
}
```

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

\verbatimleftskip
```
\newskip\verbatimleftskip
\verbatimleftskip\leftmargini
```

\verbatimhangindent
```
\newskip\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* \verb
```
\def\verb{%
  \relax\ifmmode\hbox\else\leavevmode\null\fi
  \bgroup
  \ttverbatim
  \verbatim@specials
  \gm@verb@eol
  \gm@ifstar
  {\verb@lasthook\@sverb@chbsl}%
  {\gmobeyspaces\frenchspacing\verb@lasthook\@sverb@chbsl}}%
```
in the LaTeX version there's \@vobeyspaces instead of \gmobeyspaces.
```
\emptify\verb@lasthook
```

\@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.

\DefineTypeChar
```
\DeclareCommand\DefineTypeChar{mmo}{%
%  #1  m  the char as a CS,
%  #2  m  short name of the char.
% [#3] o  the cs of the char in 'other' catcode.
  \@namedef{gmu@#2wd@name}{#2␣wd␣of
    \@xa\meaning\the\font\space␣at␣\detoken@xa\f@size␣pt}
  \@namedef{gmu@measure#2}{%
    \unless\ifcsname\csname␣gmu@#2wd@name\endcsname\endcsname
    \gmu@measurewd{#1}%
```
\edefs \gmu@tempa as the width of the char and
```
    % \gmu@tempb as the width of the char among 20 copies of itself.
    \@xa\gn@melet\csname␣gmu@#2wd@name\endcsname{gmu@tempb}%
```
here we let the CS with the name contained in \gmu@⟨*char-name*⟩wd@name to the expanded value of width of the char measured among copies of it.
```
    \typeout{@@@@␣package␣gmverb:
      \@xa\string
      \csname\csname␣gmu@#2wd@name\endcsname\endcsname
      \space␣is
      \csname\csname␣gmu@#2wd@name\endcsname\endcsname}%
    \fi
  }%
```
of \gmu@measure⟨*char-name*⟩.
```
  \@nameedef{type@#2}{%
```

```
      \@nx\leavevmode
      \@xa\@nx\csname␣gmu@measure#2\endcsname
      \hbox␣to␣\@nx\csname
      \@xa\@nx\csname␣gmu@#2wd@name\endcsname\@nx\endcsname
      {\IfValueTF{#3}{\@nx#3}{\@xa\@nx\csname␣#2\endcsname}%
        \@nx\hss}%
   }% of \type@⟨char-name⟩,
}% of \DefineTypeChar.
```

```
\DefineTypeChar\\{bslash}% this defines \type@bslash and its aides \gmu@measurebsl
      and \gmu@bslashwd@name.
```

```
\DefineTypeChar\{{lbrace}[\xiilbrace]% this defines \type@lbrace and
      its auxilia analogous to the above.
```

\check@bslash

```
\def\check@bslash{%
  \@ifnextchar\@def@breakbslash
  {\type@bslash\yeshy\@gobble}% note we allow hyphenation but actually
       this will have effect if \hyphenchar\font allows hyphenation (when it's
       not > 0 ).
  {}}
```

```
\let\verb@balance@group\@empty
```

\verb@egroup

```
\def\verb@egroup{\global\let\verb@balance@group\@empty
  \hyphenchar\font=\gmv@storedhyphenchar
  \egroup}
```

\gm@verb@eol

```
\let\gm@verb@eol\verb@eol@error
```

The latter is a LaTeX 2_ε kernel macro that `\active`ates 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}% here we make the delimiter to be the
        macro closing the verbatim group.
}
\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{\gm@ifstar{\MakeShortVerb*\|}{\MakeShortVerb%
                   \|}}
    \olddekclubs \def\olddekclubs{\OldMakeShortVerb\|}
```

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\[{%
      \@ifnextac\gmv@disverb\gmv@dismath}%
    \relaxen\edverbs}%
   \gmv@disverb \def\gmv@disverb{%
    \gmv@dismath
    \hbox\bgroup\def\]{\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
  \gm@ifstar{\eg@MakeShortVerbStar}{\eg@MakeShortVerb}}
\eg@MakeShortVerbStar \def\eg@MakeShortVerbStar#1{\MakeShortVerb*#1\endgroup}
\eg@MakeShortVerb \def\eg@MakeShortVerb#1{\MakeShortVerb#1\endgroup}
```

Grey visible spaces

In August 2008 Will Robertson suggested grey spaces for gmdoc. I added a respective option to that package but I like the grey spaces so much that I want provide them for any verbatim environments, so I bring the definition here. The declaration, if put in the preamble, postpones redefinition of `\visiblespace` till `\begin{document}` to recognise possible redefinition of it when xltxtra is loaded.

```
\let\gmd@preambleABD\AtBeginDocument
\AtBeginDocument{\let\gmd@preambleABD\firstofone}

\RequirePackage{xcolor}% for \providecolor
\VisSpacesGrey \def\VisSpacesGrey{%
  \providecolor{visspacesgrey}{gray}{0.5}%
  \gmd@preambleABD{%
    \edef\visiblespace{%
      \hbox{\@nx\textcolor{visspacesgrey}%
        {\@xa\unexpanded\@xa{\visiblespace}}}}%
  }}
```

Verbatim specials—CSes in verbatims

```
\verbatimspecials \pdef\verbatimspecials{% This declaration only defines a bearer of the 'verbatim specials'.
    %  #1  m  char for verbatim escape char (for catcode 0), has to be unbraced,
    %  #2  m  char for verbatim group begin (for catcode 1), has to be unbraced,
    %  #3  m  char for verbatim group end (for catcode 2), has to be unbraced,
    % [#4] o  char for verbatim math shift (for catcode 3),
    % [#5] o  char for a shorthand for \metachar.
    % {#6} b  (optional braced) additional stuff (commands) to be executed at the
    %          beginning of the verbatims.
  \@bsphack
  \begingroup
  \let\do\@makeother
  \dospecials
  \catcode`\␣=10
  \verbatim@specials@iii}
\verbatim@specials@iii \pdef\verbatim@specials@iii#1#2#3{% as you see, we take only first three
      arguments in a despecialized group. It's to avoid \futurelet of the optionals' parser to touch (and thus spoil) subsequent token. Yes, we could handle the case of a space or single line end but handling the case of a backslash would be somewhat difficult.
  \endgroup
verbatim@specials@list \def\verbatim@specials@list{#1#2#3}%
  \@ifnextchar[%
```

```
    {\begingroup\let\do\@makeother\dospecials
      \catcode`\␣=9
      \verbatim@specials@iv}%
    {\addtomacro\verbatim@specials@list{\NoValue\NoValue}%
      \verbatim@specials@vi}%
}% of \verbatim@specials@iii.
```

\verbatim@specials@iv

```
\pdef\verbatim@specials@iv[#1]{%
    \endgroup
    \addtomacro\verbatim@specials@list{#1}%
    \@ifnextchar[%
    {\begingroup\let\do\@makeother\dospecials
      \catcode`\␣=9
      \verbatim@specials@v}%
    {\addtomacro\verbatim@specials@list{\NoValue}%
    \verbatim@specials@vi}%
}% of \verbatim@specials@iv.
```

\verbatim@specials@v

```
\pdef\verbatim@specials@v[#1]{%
    \endgroup
    \addtomacro\verbatim@specials@list{#1}%
    \verbatim@specials@vi
}% of \verbatim@specials@v.
```

\verbatim@specials@vi

```
\DeclareCommand\verbatim@specials@vi\long{b}
{\addtomacro\verbatim@specials@list{{#1}}%
\@esphack}
```

\verbatim@specials

```
\def\verbatim@specials{% this is the macro that actually sets the chars given
      in \verbatim@specials@list as the escape char, group begin and group
      end.
    \ifdefined\verbatim@specials@list
    \@xa\verbatim@specials@\verbatim@specials@list
    \fi
}% of \verbatim@specials.
```

\verbatim@specials@

```
\long\def\verbatim@specials@#1#2#3#4#5#6{%
    \catcode`#1=0
    \protected\@namedef{#1}{#1}%
    \catcode`#2=1
    \protected\@namedef{#2}{#2}%
    \catcode`#3=2
    \protected\@namedef{#3}{#3}%
    \edef\gmu@tempa{\the\endlinechar}%
    \endlinechar\m@ne␣% we have to suppress adding of a line end by \scant¦
        okens since it would turn into an active char ^^M and raise an error (which
        actually did happen).
    \scantokens{%
      #1let#1bgroup=#2%
      #1let#1egroup=#3%
      #1catcode#1backquote#1h=6#1relax%
      #1pdef#1<h1>#2#1meta#2h1#3#3%
      #1catcode#1backquote#1h=11#1relax%
    }%
    \endlinechar\gmu@tempa\relax
```

```
    \IfValueT{#4}{%
      \catcode`#4=3
      \protected\@namedef{#4}{#4}}%
    \IfValueT{#5}{%
      \begingroup
      \lccode`\~=`#5\lowercase{\endgroup\let~\metachar}%
      \protected\@namedef{#5}{#5}%
      \catcode`#5=\active
    }% of if value #5.
    \PutIfValue{#6}%
}
```

`\noverbatimspecials`

```
\pdef\noverbatimspecials{\let\verbatim@specials@list%
      \@undefined}
```

`\GMverbatimspecials`

```
\def\GMverbatimspecials{%
  \gm@testdefined\XeTeXversion\iftrue
  \verbatimspecials␣⁄«»[¿][›]%
  {\def\|{\metachar{$\vert$}}%
    \makestarlow
    \relaxen\'% to provide a CS separator (space is not 10 in verbatims).
  }% of #6.
  \fi
}% of \GMverbatimspecials.
```

Partial `\verb` in arguments

Now command for partial verbatims in arguments of commands:

```
\let\gmu@tempa\all@stars
\@xa\addtomacro\@xa\gmu@tempa\@xa{\all@unders}
```

`\gmv@hashhalfing` `\xiihash`

```
\foone{\catcode`#=\active}
{\def\gmv@hashhalfing{%
    \def#{\xiihash\@ifnextchar#\gobble{}}%
    \catcode`#\active}%
}
```

```
\foone{\@makeother\^^R}{%
  \@xa\DeclareCommand\@xa\scanverb\@xa{%
    \@xa␣Q\@xa{\gmu@tempa}>Pm}{%
        %  #1  Q{*_}
        %  #2  m  the stuff to be rescanned and typeset verbatim. Note that % will be executed during first scan so at best will disappear.
```

Spaces are ignored (because of detokenizers that add a space after a CS) but if you declare some `\verbatimspecials`, then you can use ⁄⁄␣ where ⁄ denotes the escape char in verbatim.

```
    \begingroup
    \gmu@septify
    \endlinechar=\m@ne
    \@xa\IfIntersect\@xa{\all@stars}{#1}%
    {\def\␣{\breakablevisspace}}%
    {\let\␣=\space}%
    \@xa\IfIntersect\@xa{\all@unders}{#1}%
    {}{% We make spaces ignored only if there was no underscore in #1 and if #2 doesn't contain \␣.
```

```
      \IfAmong\␣\among{#2}%
      {}{\addtomacro\verb@lasthook{\catcode`\␣=9␣}}}%
    \addtomacro\verb@lasthook{\gmv@hashhalfing␣}%
    \@makeother\^^R%
    \edef\gmu@tempa{%
      \@nx\scantokens{%
        \bslash␣verb%
        ^^R\detokenize{#2}^^R% we delimit the \verb's argument with
                              'other' ^^R assuming this char to be used very seldom if at all.
      }% of \scantokens,
    }\gmu@tempa
    \endgroup
  }% of \scanverb,
}% of \foone.

\def\verbDiscretionaryHyphen#1#2{%
  \def\gmv@hyphenchar{\numexpr#1\relax}%
  \def\gmv@hyphen{#2}%
}

\verbDiscretionaryHyphen{"A6}{¦}

\def\gmu@tempa{%
  \DeclareCommand\verbLongDashes{
    >iS{–}% to memorise which dash we set
    B{1.41}␣% expansion of en-dash
    >iS{— }␣% as above
    B{2}␣% expansion of em-dash
  }%
}

\def\gmu@tempb{\catcode`–\active␣\catcode`—\active}

\foone{\catcode`–\active\catcode`—\active}
{%
  \edef\gmu@tempa{\@xau\gmu@tempa
    {%
      \@nx\addtomacro\@nx\ttverbatim@hook{%
        \@xau\gmu@tempb
        \def\@nx–{\@nx\scalebox{##1}[1]{\string–}}%
        \def\@nx—{\@nx\scalebox{##2}[1]{\string— }}%
      }%
    }%
  }%
}%
\gmu@tempa

\endinput% for the Tradition.
```

\verbDiscretionaryHyphen
\gmv@hyphenchar
\gmv@hyphen
\verbLongDashes

f. The gmoldcomm Package[1]

March 4, 2010

This is a package for handling the old comments in LaTeX 2ε 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 occur 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 \gmoc@ocname

```
\def\gmoc@maccname{macrocode}
\def\gmoc@ocname{oldcomments}

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

\gmoc@checkenv

```
[\def\gmoc@checkenv[%
  \@ifnextchar{%
    [\gmoc@checkenvinn][]]%
```

\gmoc@checkenvinn \gmoc@resa

```
\def\gmoc@checkenvinn{#1}[%
  \def\gmoc@resa[#1]%
  \ifx\gmoc@resa\gmoc@maccname
    \def\next[%
```

1 This file has version number v0.99 dated 2007/11/10.

```
      \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 , a-0

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. 90. a-6287

gmdoc v0.991
- General:
  - CheckSum 6134 because of compatibilising the `enumargs` environment with `\DeclareCommand`of gmutils v.0.991; abandoning gmeometric, a-0
  - put to CTAN on 2010/03/04, a-0

gmdoc v0.99a
- General:
  - CheckSum 4479 , a-0

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, a-8327

gmdoc v0.99c
- General:
  - A bug fixed in `\DocInput` and all `\expandafter`s changed to `\@xa` and `\noexpand`s to `\@nx`, a-8327
  - The TEX-related logos now are declared with `\DeclareLogo` provided in gmutils, a-8327
- `\DocInput`:
  - added ensuring the code delimiter to be the same at the end as at the beginning, a-2518
- `\gmd@bslashEOL`:
  - a bug fix: redefinition of it left solely to `\QueerEOL`, a-3565

gmdoc v0.99d
- General:
  - `\@namelet` renamed to `\n@melet` to solve a conflict with the beamer class (in gmutils at first), a-8327
  - `\afterfi` & pals made two-argument, a-8327
- `\FileInfo`:
  - added, a-7253

gmdoc v0.99e
- General:
  - a bug fixed in `\DocInput` and `\IndexInput`, a-8327
  - CheckSum 4574 , a-0

gmdoc v0.99g
- General:
  - CheckSum 5229 , a-0
  - 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, a-8327
- `hyperref`:
  - added bypass of encoding for loading url, a-2226
- `\inverb`:
  - added, a-7439
- `\OldDocInput`:
  - obsolete redefinition of the `macro` environment removed, a-8143

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, a-8327

gmdoc v0.99i
- General:
  - A "feature not bug" fix: thanks to `\everyeof` the `\[No]EOF` is now not necessary at the end of `\DocInput` file., a-8327
  - CheckSum 5247 , a-0

gmdoc v0.99j
- General:
  - CheckSum 5266 , a-0

quotation:
Improved behaviour of redefined quotation to be the original if used by another environment., a-7404

gmdoc v0.99k

General:
CheckSum 5261 , a-0

hyperref:
removed some lines testing if XeTeX colliding with tikz and most probably obsolete, a-2244

gmdoc v0.99l

General:
CheckSum 5225 , a-0

\CodeSpacesGrey:
added due to Will Robertson's suggestion, a-2752

codespacesgrey:
added due to Will Robertson's suggestion, a-2205

\gmd@FIrescan:
\scantokens used instead of \write and \@@input which simplified the macro, a-7287

macrocode:
removed \CodeSpacesBlank, a-5369

\SelfInclude:
Made a shorthand for \Docinclude\jobname instead of repeating 99% of \DocInclude's code, a-6976

gmdoc v0.99m

\@oldmacrocode:
renamed from \VerbMacrocodes, a-5463

^^M:
there was \let^^M but \QueerEOL is better: it also redefines \^^M, a-2501

General:
CheckSum 5354 , a-0
CheckSum 5356 , a-0
Counting of all lines developed (the countalllines package option), now it uses \inputlineno, a-8327

\changes:
changed to write the line number instead of page number by default and with codelineindex option which seems to be more reasonable especially with the countalllines option, a-5148

\DocInclude:
resetting of codeline number with every \filedivname commented out because with the countalllines option it caused that reset at \maketitle after some lines of file, a-6912

\FileInfo:
\egroup of the inner macro moved to the end to allow \gmd@ctallsetup. From the material passed to \gmd@FIrescan ending ^^M stripped not to cause double labels., a-7270

\gmd@bslashEOL:
also \StraightEOL with countalllines package option lets \^^M to it, a-3565

\thefilediv:
let to \relax by default, a-7127

theglossary:
added \IndexLinksBlack, a-6358

gmdoc v0.99n

General:
CheckSum 5409 , a-0
CheckSum 5547 , a-0
In-line comments' alignment developed, a-8327

\CS:
added, a-7558

\CSes:
added, a-7566

\CSs:
added, a-7564

\finish@macroscan:
the case of \␣ taken care of, a-3895

\gmd@eatlspace:
\afterfifi added—a bug fix, a-3019

\gmd@nlperc:
added \hboxes in \discretionary to score \hyphenpenalty not \exhyphenpenalty, a-7451

\gmd@percenthack:
\space replaced with a tilde to forbid a line break before an in-line comment, a-3122

\HideDef:
added the starred version that calls \UnDef, a-4420

\HideDefining:
Added the starred version that hides the defining command only once, a-4585

\ilrr:
added, a-3236

\mcdiagOff:
developed for the case of in-line comment, a-7626

\nostanza:
added adding negative skip if in vmode and \par, a-2439

\UnDef:

a bug fixed: \gmd@charbychar appended to \next—without it a subsequent in-line comment was typeset verbatim, a-4408

\verbcodecorr:
added, a-3257

gmdoc v0.990

\@codetonarrskip:
a bug fix: added \@nostanzagtrue, a-3382

\mcdiagOff:
added the optional argument which is the number of hashes (1 by default or 2 or 4), a-7626

gmdoc v0.99p

General:
CheckSum 5607 , a-0

\DeclareCommand:
added, a-4441

\mcdiagOff:
added optional arguments' handling, a-7626

gmdoc v0.99q

General:
CheckSum 5603 , a-0

gmdoc v0.99r

General:
CheckSum 5607 , a-0
put to CTAN on 2008/11/22, a-0

\toCTAN:
added, a-6431

gmdoc v0.99s

General:
\@bsphack—\@esphack added to \TextMarginize, \Describe, \DescribeMacro and \DescribeEnv, a-8327
\gmd@ifinmeaning moved to gmutils and renamed to \@ifinmeaning, a-8327
CheckSum 5974 because of enumargs handling the argument types of \DeclareCommand; handling \verbatimspecials, including writing them to index; introduction of \narrativett including \ampulexdef of gmverb internals, a-0

\check@checksum:
!*!*!… sequence changed to ! ! !… for better distinction, a-6548

\chgs:
added, a-6453

\DeclareOption:
declared as defining if without star because \DeclareOption* doesn't define a named option and so it doesn't have a text argument, a-4454

\egText@MarginizeEnv:
a bug fixed: braces added around #1, a-5279

\gmd@ABIOnce:
deferred till the end of package to allow adding titles \AtBegInputOnce, a-2696

\gmd@nlperc:
\newcommand* replaced with \pdef and optional argument's declaration removed since nothing is done to #1 in the body of now-macro. Wrapped in a group for setting \hyphenpenalty, a-7451

\gmd@writeFI:
added assignment of \newlinechar, a-7280

\gmd@wykrzykniki:
added, a-6561

\InclMaketitle:
a bug fixed: \if\relax\@date changed to \ifx, a-7137

\mcdiagOff:
added \StraightEOLto let the in-line comment continue after this environment, a-7626

\narrationmark:
added and introduced—\code@delim forked to what delimits the code (\code@delim) and what is typeset at the boundary of code: \narrationmark, a-2316

\narrativett:
introduced in gmutils and employed in the narrative verbatims, including \ampulexdef of the gmverb macros, a-5244

\TextMarginize:
added \@sanitize in the starred version, a-5270

\TextUsage:
added \@sanitize in the starred version, a-5233

\toCTAN:
made a shorthand for \chgs not \changes, a-6431

gmdoc v0.99t

General:
Since geometry v.5.2 gmeometric is obsolete so was removed, a-8327

gmdoc v1.0

\gmd@chgs:
made \long (consider it a bug fix), a-6456

\gmd@chgs@parse:
made \long (consider it a bug fix), a-6467

gmdocc v0.74

\edverbs:
used to simplify displaying shortverbs, b-529

gmdocc v0.75
General:
CheckSum 130 , b-0

gmdocc v0.76
\+:
The gmeometric option made obsolete and the gmeometric package is loaded always, for XeTeX-compatibility. And the class options go xkeyval., b-569
General:
CheckSum 257 , b-0

gmdocc v0.77
\+:
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, b-569
General:
CheckSum 262 , b-0

gmdocc v0.78
\+:
Added the pagella option not to use Adobe Minion Pro that is not freely licensed, b-569
General:
CheckSum 267 , b-0

gmdocc v0.79
General:
CheckSum 271 , b-0

gmdocc v0.80
General:
CheckSum 275 , b-0
CheckSum 276 , b-0
gmcc@fontspec:
added, b-353

gmdocc v0.81
General:
put to CTAN on 2008/11/22, b-0

gmdocc v0.82
General:
CheckSum 303 , b-0
CheckSum 316 because of \verbatimspecials, hyphenation in verbatims etc., b-0
CheckSum 320 , b-0
\ac:
added, b-548
countalllines:
gmdoc option here executed by default, b-363
gmcc@cronos:
added, for Iwona sans font, b-318
gmcc@cursor:
added, for TeX Gyre Cursor mono font, which I embolden a little and shrink horizontally a little, b-330
subtly distinguished weights of the TeX Gyre Cyursor typewriter font in the code and in verbatims in the commentary, b-330
\gmcc@dff:
I commented out setting of Latin Modern fonts for sans serif and monospaced: XeTeX/fontspec does that by default., b-293
gmcc@lsu:
added, for Lucida Sans Unicode sans font, b-326
gmcc@myriad:
added, for Myriad Web Pro sans font, b-323
gmcc@trebuchet:
added, for Trebuchet MS sans font, b-320
\LineNumFont:
added, b-306

gmdocc v0.83
General:
CheckSum 332 because of abandoning gmeometric since geometry v.5.2 provides \newgeometry, b-0

gmutils v0.74
\@begnamedgroup@ifcs:
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, c-2683
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, c-7063

gmutils v0.75
\@ifnextcat:
\let for #1 changed to \def to allow things like \noexpand ~ , c-794
\@ifnextif:
\let for #1 changed to \def to allow things like \noexpand ~ , c-836
\@ifnif:
added, c-876

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, c-7063

\freeze@actives:
added, c-5536

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, c-7063

gmutils v0.78
General:
\@namelet renamed to \n@melet to solve a conflict with the beamer class. The package contents regrouped, c-7063

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, c-3312

gmutils v0.80
General:
CheckSum 1689 , c-0

\hfillneg:
added, c-5433

gmutils v0.81
\dekfraccslash:
moved here from pmlectionis.cls, c-5728

\ifSecondClass:
moved here from pmlectionis.cls, c-5697

gmutils v0.82
\ikern:
added, c-5736

gmutils v0.83
\texttilde:
postponed to \begin{document} to avoid overwriting by a text command and made sensible to a subsequent /, c-5389

gmutils v0.84
General:
CheckSum 2684 , c-0

gmutils v0.85
General:
CheckSum 2795 , c-0
fixed behaviour of too clever headings with gmdoc by adding an \ifdim test, c-7063

gmutils v0.86
\texttilde:
renamed from \~ since the latter is one of LaTeX's accents, c-5389

gmutils v0.87
General:
CheckSum 4027 , c-0
the package goes ε-TeX even more, making use of \ifdefined and the code using UTF-8 chars is wrapped in a XeTeX-condition, c-7063

gmutils v0.88
General:
CheckSum 4040 , c-0

\RestoreEnvironment:
added, c-1181

\storedcsname:
added, c-1172

\StoreEnvironment:
added, c-1177

gmutils v0.89
General:
removed obsolete adjustment of pgf for XeTeX, c-7063

gmutils v0.90
General:
CheckSum 4035 , c-0

\XeTeXthree:
adjusted to the redefinition of \verb in xlxtra 2008/07/29, c-4158

gmutils v0.91
General:
CheckSum 4055 , c-0
removed \jobnamewoe since \jobname is always without extension. \xiispace forked to \visiblespace \let to \xxt@visiblespace of xltxtra if available. The documentation driver integrated with the .sty file, c-7063

gmutils v0.92
\@checkend:
shortened thanks to \@ifenvir, c-2771

\@gif:
added redefinition so that now switches defined with it are \protected so they won't expand to a further expanding or unbalanced \iftrue/false in an \edef, c-489

\@ifenvir:
added, c-2707

\@ifprevenvir:
added, c-2748

General:
CheckSum 4133 , c-0

gmutils v0.93
\@nameedef:
added, c-282

General:

A couple of `\DeclareRobustCommand*` changed to `\pdef`, c-7063
CheckSum 4140 , c-0
CheckSum 4501 , c-0
The numerical macros commented out as obsolete and never really used, c-7063

`\ampulexdef`:
added, c-2523

`\em`:
added, c-5110, c-5119

`\gmu@RPfor`:
renamed from `\gmu@RPif` and #3 changed from a csname to CS, c-5600

`\litshape`:
copied here from E. Szarzyński's *The Letters*, c-5068

`\lsl`:
copied here from E. Szarzyński's *The Letters*, c-5093

`\nocite`:
a bug fixed: with natbib an 'extra }' error. Now it fixes only the standard version of `\nocite`, c-3349

`\pdef`:
added, c-240

`\pprovide`:
added, c-269

`\provide`:
added, c-254

`\textlit`:
added, c-5085

`\thousep`:
added, c-6513

gmutils v0.94

`\@xau`:
added, c-225

General:
`\bgroup` and `\egroup` in the macro storing commands and in `\foone` changed to `\begingroup` and `\endgroup` since the former produce an empty `\mathord` in math mode while the latter don't, c-7063
CheckSum 4880 , c-0
removed `\unex@namedef` and `\unex@nameuse`, probably never really used since they were incomplete: `\edef@other` undefined, c-7063
The code from ancient xparse (1999) of TeXLive 2007 rewritten here, c-7063

`\afterfifi`:
`\if` removed from parameters' string, c-357

`\ampulexdef`:
made xparse-ish and `\ampulexset` removed, c-2523

`\dekfracc`:
made to work also in math mode, even with math-active digits, c-4265

`\gm@ifundefined`:
added. All `\@ifundefined`s used by me changed to this, c-623
made robust to unbalanced `\if`s and `\fi`s the same way as LaTeX's `\@ifundefined` (after a heavy debug :-), c-623

`\gmathfurther`:
removed definition of `\⟨letter⟩`s and `\⟨digit⟩`s, c-4844

`\ldate`:
`\leftline` replaced with `\par`…`\par` to work well with floatflt, c-6303

`\prependtomacro`:
order of arguments reversed, c-589

`\resizegraphics`:
`\includegraphics` works well in XeTeX so I remove the complicated version with `\XeTeXpicfile`, c-4294

`\textbullet`:
the XeTeX version enriched with `\iffontchar` due to lack of bullets with the default settings reported by Morten Høgholm and Edd Barrett, c-6193

gmutils v0.95

General:
CheckSum 4908 , c-0

`\gm@testdefined`:
added, c-641

`\gm@testundefined`:
added, c-656

gmutils v0.96

General:
CheckSum 5363 , c-0
put to CTAN on 2008/11/21, c-0, c-7063

`\glyphname`:
moved here from my private document class, c-6198

`\gmathfurther`:
Greek letters completed. Wrapped with `\addtotoks` to allow using in any order with `\garamath` and others, c-4844
the `\everymath`'s left brace moved here: earlier all the stuff was put into `\everymath`, c-4913

`\gmathscripts`:
added, c-4959

`\LuaTeX`:
added, c-4109

`\pk`:

\textup removed to allow slanting the name in titles (that are usually typeset in Italic font), c-3053

gmutils v0.97

*:
removed (it was a text tilde, available as \texttilde), c-5381

\@allbutfirstof:
added, c-322

General:
CheckSum 5375 , c-0
put to CTAN on 2008/11/22, c-0

\pdfLaTeX:
added, c-4086

gmutils v0.98

\@allbutfirstof:
renamed from \@secondofmany, c-322

\@ifedetokens:
rewritten not to make entries in the hash table, thanks to \detokenize and made robust to open \ifs in the arguments thanks to substitution of explicit parameters with \@firstoftwo and \@secondoftwo, c-2722

\@ifinmeaning:
moved here from gmdoc and rewritten, including split to \IfAmong because the latter is needed in \DeclareCommand, c-442

\@ifnextsingle:
added test for \egroup, c-956
extracted from \@ifnextac, c-951

General:
CheckSum 5429 , c-0
CheckSum 5577 because of \DeclareCommand, c-0
CheckSum 5627 because of \fakeonum, c-0
CheckSum 6035 because of \DeclareCommand, \arg, c-0
CheckSum 6129 because of \DeclareEnvironment, c-0
CheckSum 6147 because of \arg in verbatims, c-0
CheckSum 6535 because of \UrlFix, c-0
CheckSum 6656 because of type settings for gmdoc (\narrativett) and for \verbatimspecials, c-0

\addto@macro:
\toks@ replaced with some \expandafters because use of \toks@ here caused a disaster in \DeclareCommand, c-576

\ampulexdef:
a bug fixed: added \unexpanded, c-2539
first argument (the prefix) made of the Q type, c-2523

\arg:
made iterating thanks to \DeclareCommand\arg@dc, c-3299

\ATfootnotes:
moved here from my own private macro package to allow the beauty of these footnotes for the general audience, c-6611

\balsmiley:
moved here from my personal macro package since I needed it in the documentation of gmutils, c-6674

\cs:
added \verbatim@specials, c-3067
made \- switch to \normalfont because IMHO this underlines the fact that a CS belongs to the narrative. \hyphenchar set to 45 as in usual texts, c-3067

\DeclareCommand:
added the \@bsphack—\@esphack option, c-2262
added the Q{⟨*tokens*⟩} argument type (a word over the alphabet ⟨*tokens*⟩), c-2262
added the S/T spec parsing, the s spec parsing rewritten to be a shorthand for S{*}, unused code removed, c-1605

\enoughpage:
\par removed since to let it be used for \pagebreak inside a paragraph, c-5495
made ε-TEX-ish, c-5495
made ifthenelse-like with 'then' and 'else' optional default ⟨*nothing*⟩ and \newpageresp., c-5495

\fakeonum:
added, c-5148

\gmu@lowstar:
renamed from \@ifstar since something redefines \@ifstar, c-935

\IfAmong:
split from \IfAmong, c-399

\IfNoValueTF:
the xparse tests for the presence of value redefined and much simplified (43 CS'es less). Moreover, they are now fully expandable: \IfNoValueTF, \IfNoValueT, \IfNoValueF, \IfValueTF, \IfValueT, \IfValueF, c-2162

\napapierkicore:
added optional star controlling globalness, c-6003

\rrthis:
added, c-6661
\scantnoline:
added, c-6682
gmutils v0.99
\@ifedetokens:
moved to a separate macro from \@ifenvir and made symmetric to both arguments, c-2722
gmutils v0.991
\@ifnextif:
inner macro fixed to handle active chars more properly (more as \if would do it), c-836
\@xau:
made 1-parameter, c-225
General:
CheckSum 8257 because of some shorthands and major development of \DeclareCommand, including 'Knuthian' and general optional arguments and '\afterassignment' pseudo-argument, c-0
put to CTAN on 2010/03/04, c-0
gmverb v0.79
\edverbs:
added, e-860
gmverb v0.80
\edverbs:
debugged, i.e. \hbox added back and redefinition of \[, e-860
\xiiclub:
\ttverbatim@hook added, e-436
gmverb v0.81
General:
\afterfi made two-argument (first undelimited, the stuff to be put after \fi, and the other, delimited with \fi, to be discarded, e-1144
gmverb v0.82
General:
CheckSum 663 , e-0
gmverb v0.83
General:
added a hook in the active left brace definition intended for gmdoc automatic detection of definitions (in line 371), e-1144
CheckSum 666 , e-0
gmverb v0.84
General:
CheckSum 658 , e-0
gmverb v0.85
General:
added restoring of \hyphenpenalty and \exhyphenpenalty and setting \hyphenchar=-1, e-1144
CheckSum 673 , e-0
gmverb v0.87
General:
CheckSum 661 , e-0
visible space tidied and taken from xltxtra if available. gmutils required. The \xii... CSes moved to gmutils. The documentation driver moved into the .sty file, e-1144
gmverb v0.88
General:
CheckSum 682 , e-0
\VisSpacesGrey:
added, or rather moved here from gmdoc, e-934
gmverb v0.89
General:
\dekclubs, \dekclubs* and \olddekclubs made more consistent, shorthands for \MakeShortVerb\|, \MakeShortVerb*\| and \OldMakeShortVerb\| respectively., e-1144
CheckSum 686 , e-0
gmverb v0.90
General:
CheckSum 684 , e-0
some \(b|e)group changed to \(begin|end)group, e-1144
gmverb v0.91
General:
CheckSum 686 , e-0
put to CTAN on 2008/11/21, e-0
\verbatimleftskip:
added, e-703
gmverb v0.92
General:
CheckSum 979 because of \verbatimspecials, hyphenation in verbatims, low star in verbatims, kerning of backslash in shrunk fonts, e-0
\breakbslash:
renamed from \fixbslash, e-398
\breaklbrace:
renamed from \fixlbrace, e-405
\ttverbatim:
added \makeatletter to sound with the 'verbatim specials', namely to allow control sequences containing @, e-450
gmverb v0.93
General:

CheckSum 1035 because of a bug fix in `\scanverb` (halfing the hashes), e-0

put to CTAN on 2010/03/04, e-0

`\gmv@hashhalfing`:

cut out as separate macro, e-1064

`\verbDiscretionaryHyphen`:

added to synchronise hyphen chars in `gmdoc`'s documentation, e-1109

`\xiihash`:

mandatory argument made long (a bug fix), e-1072

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=gmutils.sty, d=gmiflink.sty, e=gmverb.sty, f=gmoldcomm.sty

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

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

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

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

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

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

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

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

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

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

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

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

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