**Grzegorz Murzynowski**

# The gmutils Packages Bundle *


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

```
47 ⟨*master⟩
```

(A handful of meta-settings skipped)

```
101 ⟨/ master⟩
102 ⟨*ins⟩
```

\supposedJobname
```
103 \def\supposedJobname{%
104         gmutils%
105 }
```

```
107 \let\xA\expandafter
108 \let\nX\noexpand
109 \long\def\firstofone#1{#1}
```

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

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

`xelatex␣--jobname=`⟨*sth. else*⟩

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

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

\gmBundleName
```
120 \def\gmBundleName{%
121     gmutils%
122 }
```

\currentBundle
```
124 \def\currentBundle{%
125     utilsbundle%
126 }
```

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

```
131 \input␣docstrip.tex
```

```
\NOO 133 \def\NOO{\FromDir\gmBundleFile␣.gmd}
```

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

```
136 \let\skiplines\relax
137 \let\endskiplines\relax
138 \askforoverwritefalse
\MetaPrefixS 140 \def\MetaPrefixS{\MetaPrefix\space}
\perCentS 141 \def\perCentS{\perCent\space}
143 \begingroup
144 \endlinechar=\newlinechar
145 \catcode\newlinechar=12\relax%
147 \catcode`\^=12\relax%
148 \catcode`\ =0\relax␣% Tifinagh Letter Yay
149 \catcode`\\=12 relax␣%
150 catcode` /=12 relax␣%
151 firstofone{ endgroup␣%
153     def preamBeginningLeaf{%
155       RCSInfo
156       MetaPrefixS␣This␣is␣file␣" outFileName"␣generated␣with␣
              the␣DocStrip␣utility.
157       MetaPrefixS
158       ReferenceLines␣%
159       MetaPrefix␣%
160     }% of \preamBeginningLeaf
164     def copyRightLeaf{Copyright␣©␣}%
167     def licenseNoteLeaf{%
168       This␣program␣is␣subject␣to␣the␣LaTeX␣Project␣Public␣
              License.
169       MetaPrefixS␣See␣␣
              http://www.ctan.org/tex-archive/help/Catalogue/licenses.lppl.h
170       MetaPrefixS␣for␣the␣details␣of␣that␣license.
171       MetaPrefix
172       MetaPrefixS␣LPPL␣status:␣"author-maintained".
173       MetaPrefix␣%
174     }% of \licenseNoteLeaf
176     def preamEndingLeaf{%
177       gmBundleFile.{gmd,pdf} gobble{␣or␣\file{%
              Natror-OperaOmnia.{gmd,pdf}}}.
178       MetaPrefixS␣␣%
179     }% of \preamEndingLeaf
181     def providesStatement{%
183       \NeedsTeXFormat{LaTeX2e}
184       \Provides gmFileKind{ gmOutName}
185       space space space space[ gmFileDate space␣
              gmFileVersion space␣ gmFileInfo space␣(GM)]
187     }%
189 }% of \firstofone of changed catcodes.
\beforeDot 191 \def\beforeDot#1.#2\empty{#1}
```

* This file has version number v0.993 dated 2010/10/24.

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

To gobble the default heading lines put by DocStrip:

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

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

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

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

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

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

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

\declarepreamble\gmdLeaf
\preamBeginningLeaf

\copyRightLeaf␣\gmOutYears
by␣Grzegorz␣'Natror'␣Murzynowski
natror␣(at)␣o2␣(dot)␣pl

\licenseNoteLeaf

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

\keepsilent
```

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

```
\errorcontextlines=1000

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

`\gmfile`

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

`\pack`

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

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

\gmBundleFile

```
  \def\gmBundleFile{gmutils}
```

```
  \generate{
```

```
    \usepreamble\gmdLeaf
```

\gmFileKind

```
    \def\gmFileKind{␣␣␣␣Package␣␣␣␣}
```

```
    \writeto{␣␣␣␣gmutils␣␣␣␣}
```

```
    \pack{␣␣␣␣base␣␣␣␣␣␣␣␣}% gmbase
    \pack{␣␣␣␣utils␣␣␣␣␣␣␣␣}% gmutils
    \pack{␣␣␣␣command␣␣␣␣}% gmcommand
    \pack{␣␣␣␣ampulex␣␣␣␣}% gmampulex
    \pack{␣␣␣␣envir␣␣␣␣␣␣␣␣}% gmenvir
    \pack{␣␣␣␣relsize␣␣␣␣␣␣␣␣}% gmrelsize
    \pack{␣␣␣␣meta␣␣␣␣␣␣␣␣}% gmmeta
    \pack{␣␣␣␣logos␣␣␣␣␣␣␣␣}% gmlogos
    \pack{␣␣␣␣notonlypream}% gmnotonlypream
    \pack{␣␣␣␣mw␣␣␣␣␣␣␣␣}% gmmw
    \pack{␣␣␣␣typos␣␣␣␣␣␣␣␣}% gmtypos
    \pack{␣␣␣␣parts␣␣␣␣␣␣␣␣}% gmparts
    \pack{␣␣␣␣url␣␣␣␣␣␣␣␣␣␣}% gmurl
    \pack{␣␣␣␣RCS␣␣␣␣␣␣␣␣␣␣}% gmRCS
    }
```

```
}% of changed catcodes' \firstofone
```

```
\Msg{%
     ************************************************************}
\Msg{␣}
\Msg{␣␣To␣finish␣the␣installation␣you␣have␣to␣move}
\Msg{␣␣␣the␣generated␣files␣into␣a␣directory␣searched␣by␣TeX.}
\Msg{␣}
\Msg{␣␣To␣type-set␣the␣documentation,␣run␣the␣file␣'\NOO'}
\Msg{␣␣twice␣through␣LaTeX␣and␣maybe␣MakeIndex␣it.␣␣}
\Msg{␣}
\Msg{%
     ************************************************************}
```

```
\csname␣fi\endcsname␣% probably for the directive's clause
\csname␣endinput\expandafter\endcsname␣%
\fi␣% of unless job name other than name of this file, which indicates the DocStrip
     pass.
```

```
⟨/ ins⟩
```

## Contents

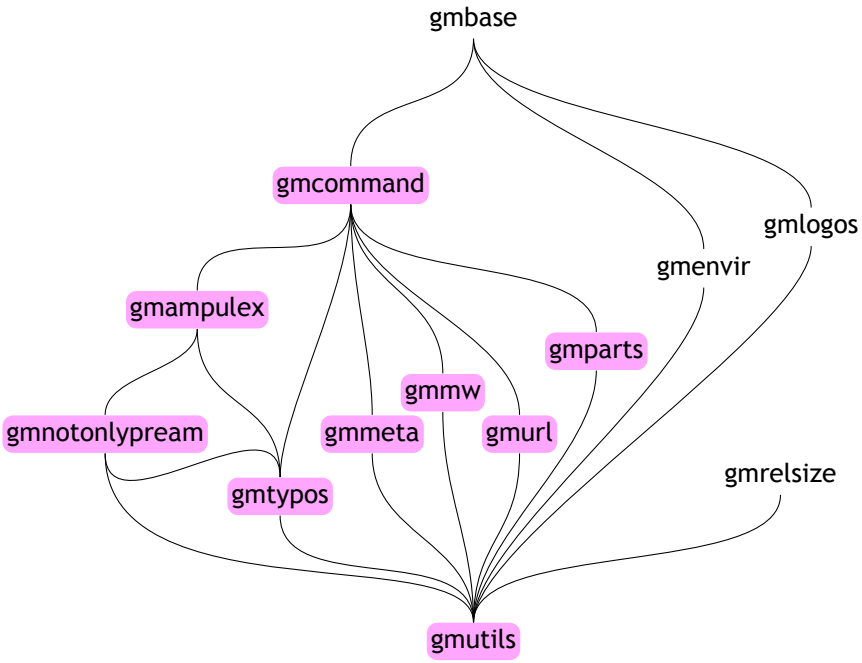
## Intro

The gmutils package bundle 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.

### Brave New gmutils and its inner dependencies

For details just read the code part (where you'll find some comments) or intros to the particular packages. Here let's give a short description of what you could expect of them.




— gmbase: basic, low-level macros such as `\@ifnextcat` and other tests for peeping next token, including such that respect blank space; conditionals in an argument form robust to open `\if`'s and `\fi`'s. Also some expandable tests comparing strings of detokenised or not detokenised tokens (use of `\strcmp`); test whether a token is of a given kind (`\dimen`, `\skip`, `\count` &c.).

— gmenvir: a modification of `\begin` and `\end` to fully expand and detokenise environment's name, so that the comparisons are fully compatible with `\csname`...`\endcsname`.

— gmcommand: probably the most important package of mine, providing a brand new implementation of the ancient (pre-expl3) idea of a command to declare LaTeX commands with many different optional arguments. This package implements `\DeclareCommand`, `\DeclareEnvironment` (you can use `#1`…`#9` also in the enddefs!). For the details see the package's intro, where all the arg. specifiers are described.
— gmlogos: a couple of TeX-related logos, with a trial of improving the position of »A« in LaTeX. (Presented at BachoTeX 2007).
— gmrelsize: some macros taken from the relsize package not to load all of them, and some new added, namely `\largerr` and `\smallerr`.
— gmampulex: modification of macros including those having parameters without total redefinition of them: you give the start tokens, the end tokens and the replacement. Use with care.
— gmnotonlypream: a modification of the `\@onlypreamble` declaration to provide a bit more informative error message and removal of many comands from the "only preamble" list that are useful also in the document.
— gmurl: some fixes to the url package not to use math mode but `\scantokens` which allows to get proper kerning.
— gmparts: in/exclude parts of one and the same file just as if you'd do with `\include` and `\includeonly` (in fact, you use *the* `\includeonly` to get some parts excluded).
— gmtypos: macros written while typesetting real books for money. Most of them do conform Polish typesetting standards of the 1960's, which I like best, or refined advice of leading (contemporary) Polish typographers (e.g. `\ATfootnotes`).
— gmmw: compatibilising the sectioning commands of my favourite MWCLS classes with the standard ones.
— gmmeta: a couple of macros for description of macros.

#### Installation

Unpack the \jobname-tds.zip archive (this is an archive that conforms the TDS standard, see CTAN/tds/tds.pdf) in some texmf directory or just put the gmutils.sty somewhere in the texmf/\:tex/\:latex branch. Creating a texmf/\:tex/\:latex/\:gm directory may be advisable if you consider using other packages written by me.

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

#### Contents of the gmutils.zip archive

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

gmutils.gmd
README
gmutils.pdf
gmutils.tds.zip

#### Compiling of the documentation

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

MakeIndex shell commands:

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

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

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

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

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

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

739 ⟨*base⟩

`%␣&utils&command&envir` doesn't make sense since gmbase is loaded by them all and sets it properly.

```
743 \ifx\XeTeXversion\relax
744   \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
        to \@ifundefined, but it's a macro and it eats its arguments, freezing their
        catcodes, which is not what we want in line 12025.
751 \fi
754 \ifdefined\XeTeXversion
755 \XeTeXinputencoding␣utf-8␣% we use Unicode dashes later in this file.
756 \fi% and if we are not in XeTeX, we skip them thanks to XeTeX-test.
```

758 ⟨/ base⟩

759 ⟨*utils⟩

### Options

```
763 \unless\ifdefined\Name
\Name 764   \def\Name#1#2{\expandafter#1\csname#2\endcsname}
765 \fi
768 \unless\ifcsname␣ifgmu@quiet\endcsname
769 \Name\newif␣{ifgmu@quiet}% it has to be at least (at highest) in gmcommand
        since is used by it and not always entire gmutils is loaded.
772 \fi
774 \RequirePackage{xkeyval}
776 \RequirePackage{gmbase}
```

777 ⟨/ utils⟩

### The gmbase packageThis file has version number v0.993 dated 2010/10/24.

This is the lowest-level package that defines such things as `\gmu@ifempty`, a handful of "if next" tests &c.

785 ⟨*base⟩

### A couple of abbreviations

```
\unless\ifdefined\strcmp
  \let\strcmp\pdfstrcmp
\fi
```

```
\let\@xa\expandafter
\let\@nx\noexpand
```

```
\def\@xanx{\@xa\@nx}
```

```
\long\def\@xadef#1{\@xa\def\@xa#1\@xa}
```

```
\long\def\@csn#1{\csname␣#1\endcsname}
```

Note that it differs from LaTeX's `\@nameuse` in `\long`ness.

```
\long\def\Name#1#2{\@xa#1\csname␣#2\endcsname}
```

```
\long\def\@xau#1{\unexpanded\@xa{#1}}
```

Note that there's only one `\expandafter`: after `\unexpanded`. It's because `\unexpanded` expands expandable tokens and gobbles `\relax`es while looking for an opening brace or `\bgroup`.

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*⟩.

```
\def\gmu@firstandspace#1{#1␣}
```

```
\long\def\strip@bslash#1{%
  \gmu@ifempty{#1}{}{%
    \gmu@if␣{cat}{\@nx~\@nx#1}% this is specially for an active backslash
        (have you ever met it?). Thanks to this special case the inner macro
        declared for an active \ by \DeclareCommand is \\\ not \\csname%
        \endcsname\.
    {\string#1}% if #1 is active
    {%
%%    \@xa\ifnum\@xa\escapechar\@xa=\@xa` % looks great, but what if #1
is 92 (while normal \escapechar)? or \1?
%%    \if\bslash
      \@xa\@xa\@xa\ifnum\@xa\@xa\@xa\escapechar
        \@xa\@xa\@xa=\@xa\@xa\@xa`\@xa\gmu@firstandspace
        \string#1\@xa\@gobble
      \else\@xa\@firstofone␣\fi
      {\string#1}%
    }% if #1 is not active
  }% of if #1 not empty
}
```

```
\long\def\bslash@or@ac#1{%
```

If `#1` is a CS or a name, we make it beginning with a backslash. Otherwise we keep it only `\string`ed.

```
  \ifcat\@nx~\@nx#1%
  \else
    \bslash
  \fi
```

```
  \strip@bslash{#1}%
}
```

\@xanxcs
```
\long\def\@xanxcs#1{%
```

\noexpand indefferent to whether the argument is a name or an active char.

```
  \ifcat\@nx~\@nx#1%
    \@nx#1%
  \else
    \@xa\@nx\csname␣#1\endcsname
  \fi
}
```

Meaning of a csname protected with \unexpanded

\@xaucs
```
\long\def\@xaucs#1{%
  \unexpanded\@xa\@xa\@xa{\csname␣#1\endcsname}%
}
```

\@xanxtri
```
\long\def\@xanxtri#1{%
```

\noexpand indifferent to whether the argument is a name, an active char or a CS.

Warning. It applies \string to the first token of #1 so doesn't expand it if it's a macro.

```
  \ifcat\@nx~\@nx#1%
    \@nx#1%
  \else
    \@xa\@nx\csname␣\strip@bslash{#1}\endcsname
  \fi
}
```

\pdef
```
\def\pdef{\protected\def}
```

\lpdef
```
\def\lpdef{\long\protected\def}
\let\pldef\lpdef
```

\pedef
```
\def\pedef{\protected\edef}
```

\pxdef
```
\def\pxdef{\protected\xdef}
```

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

\@gobble
```
\let\@gobble\gobble
```

\gobbletwo
```
\let\gobbletwo\@gobbletwo␣% it's a LaTeX's \long macro (in File d: ltdefns.dtx,
      Date: 2004/09/18 Version v1.3g l. 939)
```

\@gobbleeight
```
\long\def\@gobbleeight#1#2#3#4#5#6#7#8{}
```

```
\long\pdef\provide#1{%
  \ifdefined#1%
    \ifx\relax#1\afterfifi{\def#1}%
    \else\afterfifi{\gmu@gobdef}%
    \fi
  \else\afterfi{\def#1}%
  \fi}
```

```
\long\def\gmu@gobdef#1#{%
```

```
  \def\gmu@tempa{}% it's a junk \def-assignment to absorb possible prefixes.
  \@gobble}

\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`).

```
\long\def\@nameedef#1#2{%
  \@xa\edef\csname#1\endcsname{#2}}
```

#### `\ifs` as a four-argument LaTeX command robust to unbalanced `\ifs` and `\fis`

```
\pdef\gmu@DefSymbol#1{%
  \unless\ifdefined#1%
    \def#1{#1}%
  \fi
}
```

`%% |\pdef\globalize#1{\global#1=#1} %` doesn't make sense general enough (2010/11/03)

```
\long\def\gmu@if#1#2{%
  \ifnum\strcmp{incsname}{\detokenize{#1}}=\z@
    \@xa\@xa\@xa\ifincsname
    \@gobble\fi␣% this \fi balances the conditional when false
  \else
    \csname␣if#1\@xa\endcsname
  \fi
  #2\@xa\@firstoftwo\else\@xa\@secondoftwo\fi
}
```

A little `\expandafter` tip: to get the effect of `\expandafter\ifx`⟨*stuff*⟩ write `\gmu@if␣{x\expandafter\expandafter}`, where `x` stands for any conditional primitive suffix.

```
\long\def\gmu@notif#1#2{%
```

We leave the name `\gmu@unlessif` for analogon of `\gmu@if` prefixed with `\unless`, which works slightly different than reversion of `#3` and `#4` (if the tail of `#2` remains after test, it becomes part of true branch for unprefixed and part of false branch for `\unless`-prefixed version). (2010/7/25)

```
  \gmu@if␣{#1}{#2}%
  \@secondoftwo\@firstoftwo
}
```

And simplified versions of the testing macros, for the switches, because I many times forgot to add the empty `#2` (which of course lead to a disaster at best (at worst—to a perfidious bug that remains hidden for years)).

```
\def\gmu@ifsw␣#1{\gmu@if␣{#1}{}}
\def\gmu@notsw␣#1{\gmu@notif␣{#1}{}}

\long\def\gmu@unless␣#1#2{%
```

```
  \ifnum\strcmp{incsname}{\detokenize{#1}}=\z@
    \@xa\@xa\@xa\unless␣\@xa\@xa\@xa\ifincsname
    \@gobble\fi␣% this \fi balances the conditional when it's not \ifinc¦
        sname
  \else
    \@xa\unless␣\csname␣if#1\@xa\endcsname
  \fi
  #2\@xa\@firstoftwo\else\@xa\@secondoftwo\fi
}
```

For a special case of downright nesting of conditionals let's provide a shorthand. But wait a minute. This special case, which from TEXnical point of view is a tree growing downright, is just a `cases` special form from usual languages. Therefore let's name it `case(s)`.

It has one inconvenience: the last (innermost) false branch (the last case) also has to be preceded with `\gmu@EatDownright` (or just *be* this macro).

```
\lpdef\@iwru@EC#1#2#3{%
  \@iwrum{#1»{#2}«␣»#3«}%
  \gmu@passbraced{#1{#2}}{#3}%
}

\long\def\gmu@generalCASE
#1% Testing macro (\gmu@if etc.
#2% #1 of the testing macro
#3% #2 of the testing macro
#4% #3 of the testing macro (what if test satisfied)
      (#4 of the testing macro will always be empty)
{%
  #1{#2}{#3}%
  {%
    \gmu@EatCases{#4}}{}%
}
```

If the condition is satisfied, `#3` is executed after eating all the stuff succeding it up to a delimiter CS`\gmu@ESAC`.

```
\long\def\gmu@EatCases
#1%
#2\gmu@ESAC
{#1}

\let\gmu@lastCASE\gmu@EatCases

\def\gmu@CASE␣{\gmu@generalCASE␣\gmu@if␣}
\def\gmu@CASEnot␣{\gmu@generalCASE␣\gmu@notif␣}
\def\gmu@CASExany␣{\gmu@generalCASE␣\gmu@ifxany␣}
\def\gmu@CASExnone␣{\gmu@generalCASE␣\gmu@ifxnone␣}

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

An expandable test whether argument is a single token or not. Beware: it expands to an open if.

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

`%% \gmu &nil %` such a strange delimiter to make it work both in letter and other `@` scopes etc. (`&` seems to me recatcoded rather seldom)

```
  \ifnum␣\strcmp{\unexpanded{#1}}%
    {\@xau{\gmu@reserved@firstofmany␣#1\blekotnizza@␣\di␣%
         \broccoli
        \gmu&nil}%
    }% of right text of \strcmp
    =\z@
}
```

The conditional of this macro turns true if `#1` was a single token (of catcode $\neq 1, 2$) or such a token in curly braces (remember that the braces are stripped also from delimited argument if it consists only of braced text).

Note that single (unbalanced) tokens of catcodes 1 or 2 cannot be passed this macro.

The conditional turns false when `#1` (possibly after stripping braces) is of at least 2 tokens or is empty or is a blank space.

The last segment of last disjunction may be a bit confusing. But this macro is intended for determining whether we should pass `#1` in braces or not and passing a blank in braces seems reasonable.

A macro that applies `\string` to its argument if it's not braced. To be expanded by `\detokenize`.

```
\long\def\gmu@predetokstring␣#1{%
  \gmu@if␣{num}
  {%
```

After a couple of hours of debug I reached the proper test which is given below. The goal is to (expandably!) check whether `#1` is braced and/or begins with a blank space. In any of those cases we don't hit it('s first token) with `\string`.

`\@firstofone` strips outermost pair of braces if any and gobbles the lading blank(s) if any so the detokenised strings will differ.

```
    \strcmp
    {\detokenize{#1x}}%
    {\detokenize\@xa{\@firstofone␣#1x}}=\z@
  }% of test
  {\@xa{\string#1}}
  {{#1}}%
}
```

A test for comparison of CS es as macro delimiters (stronger than `\ifx`). Beware: it expands to an open if. And it has to, to be usable in `\gmu@if`.

```
\long\def\ifstrings#1#2{%
  \ifnum\strcmp
  {\detokenize\gmu@predetokstring{#1x}}%
  {\detokenize\gmu@predetokstring{#2x}}=\z@
```

We hit both texts with `\unexpanded` in case they are more than one token. Note `\string` is not the same as `\detokenize` because the latter adds a space after a letter CS.

Moreover, a char is added to both texts to assure that `\string` has sth. to hit not the closing brace (which would lead to a disaster).

```
}
```

As you see, it expands to an open if, but that's OK as far as we use it only via macros, e.g.

```
\gmu@if␣{strings}{\while\until}...
```

Thanks to this test defining CSes that are intended only as atoms (symbols) ceases to be necessary. Which is quite an advantage, if we wish to use symbol CSes such as `\while`, `\until` or `\SameAs` (in `\DeclareCommand`) that with this test may still be available for `\newcommand`.

And another, slightly different: turns true iff the arguments are equal after (stringing and) possible stripping bslash(es), e.g. `par` and `\par` (well, *any* esccape char that is currently in charge).

```
\long\def\ifstribs#1#2{%
  \ifnum\strcmp{\strip@bslash{#1}}{\strip@bslash{#2}}=\z@
}
```

And a test with a database flavour: for tokens that are defined it's `\ifx`. For tokens `\ifx`-equal `\@undefined`—it's `\ifstrings` (in relational databases any usual comparison of two NULLs returns null so there's a distant resemblance):

```
\long\def\ifStrX#1#2{%
  \gmu@CASEnot␣{x}{#1#2}%
  {\iffalse}% if tokens are x-unequal, all is clear.

  \gmu@CASEnot␣x␣{\@undefined#1}
  {\iftrue}%
```

some (i.e. *both*) tokens are-x `\@undefined` or `\relax`, then

```
  \gmu@CASE␣{strings}{#1#2}%
    {\iftrue}%
  \gmu@lastCASE
  {\iffalse}%
  \gmu@ESAC
}
```

### `\firstofone` and the queer `\catcodes`

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

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

\long\def\bracefirstofone#1{{#1}}

\long\pdef\scantwo#1#2{
  \begingroup\endlinechar\m@ne
  \@xa\endgroup\scantokens{#1#2}%
}

\long\def\@firstofthree#1#2#3{#1}
\long\def\@secondofthree#1#2#3{#2}
\long\def\@thirdofthree#1#2#3{#3}
\long\def\@twoofthree#1#2#3{#1#2}
\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.

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

```
\long\def\@secondofmany#1#2#3\@nil{#2}

\long\def\@allbutfirstof#1#2\@nil{#2}

\long\def\@firstthensecond␣#1#2{#1#2}␣% Note this macro strips braces if
      present.
\long\def\@secondthenfirst␣#1#2{#2#1}␣% Note as above.
```

### `\afterfi` and pals

It happens from time to time that you have some sequence of macros in an `\if`… and you would like to expand `\fi` before expanding them (e.g., when the macros should take some tokens next to `\fi`… as their arguments. If you know how many macros are there, you may type a couple of `\expandafter`s and not to care how terrible it looks. But if you don't know how many tokens will there be, you seem to be in a real trouble. There's the Knuthian trick with `\next`. And here another, revealed to me by my TEX Guru.

I think the situations when the Knuthian (the former) trick is not available are rather seldom, but they are imaginable at least: the `\next` trick involves an assignment so it won't work e.g. in `\edef`.

But `\afterfi` and pals are sensitive to `\fi`s that may occur in macros' arguments so probably the safest and expandable way is the `\expandafter\@(first|second)oftwo` trick.

```
\def\longafterfi{%
  \long\def\afterfi##1##2\fi{\fi##1}}
\longafterfi
```

And two more of that family:

```
\long\def\afterfifi#1#2\fi#3\fi{\fi\fi#1}
\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.

```
\long\def\afterififfififi#1#2\fi#3\fi#4\fi{\fi#1}
\long\def\afteriffififi#1#2\fi#3\fi#4\fi{\fi\fi#1}
\long\def\afterfififi#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.

```
\long\def\foone#1{\begingroup#1\relax\egfirstofone}
\long\def\egfirstofone#1{\endgroup#1}

\def\fooatletter{\foone\makeatletter}
```

`\@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}
```

### \@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.

```
\long\def\gmu@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.
     %%  \def\gmu@ifempty@resa{#1}%
     %%  \ifx\gmu@ifempty@resa\@empty
  \ifnum\strcmp{\detokenize{#1}}{}=\z@
```

Note that now (2010/6/23) it's expandable thanks to \strcmp and therefore it's no longer \protected. Another argument for strcmp is that it detokenizes its text so its robust to # es. But it expands all the macros which is not what we intended so we \detokenize{#1} anyway.

```
    \@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi
}

\long\pdef\@ifnonempty#1#2#3{\gmu@ifempty{#1}{#3}{#2}}

\long\def\gmu@ifexempty␣#1{%
  \ifnum␣\strcmp{#1}{}=\z@
    \@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi
}

\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.
```

Note this command has to be used with care since it recognizes the token(s) due to fitting a delimited macro, so it distinguishes any two different tokens even if they are \ifx-equal.

```
  \long\def\gmu@among@##1#1##2\gmu@among@{%
    \gmu@ifempty{##2}\@secondoftwo\@firstoftwo}%
  \gmu@among@#2#1\gmu@among@}
```

\ifgmu@ifquant
```
\newif\ifgmu@ifquant
```

```
\long\pdef\gmu@ifxany
#1% a single token to be \ifx ed with each of #2
#2% counterpart to the above—a sequence of tokens to check #1 against. It may contain anything including groups since checking is preceded by an assignment.
```

```
{%
  %%  \gmu@ifempty{#1}{\PackageError{gmbase}{We don't consider this
  %%  case}{}}%
   \gmu@ifempty{#2}{\@secondoftwo}{%
```

we wrap the iteration over `#2`'s tokens in `\gmu@ifempty` because we expect many empty `#2`'s in `\DeclareCommand`'s `\loop` arguments (such as `Q` and `U`)

```
    \gmu@ifquantfalse
    \let\gmu@ifxa@aasiter\@@gmu@ifxa@aasiter
    \edef\gmu@ifxa@aas{% edef and unexpanded to protect agains #6 token(s)
         in #1.
      \unexpanded{%
        \ifx␣#1\gmu@ifxa@token
          \gmu@ifquanttrue
```

We are happy we found what we looked for but anyway we have to iterate to let all the possible groups to the drain.

```
          \let\gmu@ifxa@aas\gmu@ifxa@drainer
        \else
          \ifx␣\gmu@ifxa@Limit\gmu@ifxa@token
            \emptify␣\gmu@ifxa@aasiter
          \fi
        \fi
        \gmu@ifxa@aasiter
      }% of \unexpanded
    }% of \gmu@ifxa@aas
    \gmu@ifxa@aasiter␣#2\gmu@ifa@PreLimit\gmu@ifxa@Limit
    \gmu@if␣{gmu@ifquant}{}%
  }% of if #2 nonempty
}
\def\gmu@ifxa@drainer{%
  \ifx\gmu@ifxa@Limit\gmu@ifxa@token
    \emptify\gmu@ifxa@aasiter
  \fi
  \gmu@ifxa@aasiter
}
\gmu@DefSymbol␣\gmu@ifa@PreLimit
\gmu@DefSymbol␣\gmu@ifxa@Limit
\def\@@gmu@ifxa@aasiter{%
```

It's a pattern CS to restore `\gmu@ifxa@aasiter` in each `\gmu@ifxany`.

```
  \afterassignment\gmu@ifxa@aas
  \let\gmu@ifxa@token=␣}% thanks to this space it'll look also at spaces (cat 10)
       and = chars.
```

`\gmu@ifxnone`

```
\long\pdef\gmu@ifxnone
#1% token to be checked against
#2% list of tokens to not find #1 at
{%
  \gmu@ifxany{#1}{#2}\@secondoftwo\@firstoftwo
```

```
}
%% \long\pdef\gmu@ifNXany
%% #1% a single token to be noexpanded and \if ed with each of
%% % #2 noexpanded
%% #2% counterpart to the above—a sequence of tokens to check #1
%% % against. It may contain anything including groups since checking is
%% % preceded by an assignment.
%% %
%% {%
%%  \gmu@ifempty{#2}{\@secondoftwo}{%
%%   % we wrap the iteration over #2's tokens in \gmu@ifempty because
%%   % we expect many empty #2's in \DeclareCommand's \loop
%%   % arguments (such as Q and U)
%%   \gmu@ifquantfalse
%%   \let\gmu@ifNXa@aasiter\@@gmu@ifNXa@aasiter
%%   %
%%   \edef\gmu@ifNXa@aas{% edef and unexpanded to protect agains
%%    % #6 token(s) in #1.
%%    \unexpanded{%
%%     \if \@nx#1\@nx\gmu@ifNXa@token
%%       % an „honest" char token, i.e., not an active char neither a
%%       % CS, when \let to a CS, keeps its catcode for
%%       % \ ifcat. An active char however, when let to a CS,
%%       % loses its 13 and ifcat-is \relax (a CS). Therefore for
%%       % active chars and CS es we use \ ifx.
%%       %
%%       % This is useless: we cannot determine that a CS let
%%       \ifnum
%%        \numexpr
%%         \ifcat \@nx#1\relax 0\else 1\fi *%
%%         \ifcat\@nx#1~0\else1\fi
%%         +\ifcat \@nx\gmu@ifNXa@token\relax 0\else 1\fi
%%        >\z@
%%        % if at least one of compared tokens is not a CS neither
%%        % an active char
%%
%%
%%      \fi
%%      \gmu@ifquanttrue
%% % We are happy we found what we looked for but anyway we have to
%% % iterate to let all the possible groups to the drain.
%%      \let\gmu@ifNXa@aas\gmu@ifNXa@drainer
%%    \else
%%      \ifx \gmu@ifNXa@Limit\gmu@ifNXa@token
%%       \emptify \gmu@ifNXa@aasiter
%%      \fi
%%    \fi
%%    \gmu@ifNXa@aasiter
%%   }% of \unexpanded
%%  }% of \gmu@ifxa@aas
%%  %
%%  \gmu@ifNXa@aasiter #2\gmu@ifa@PreLimit\gmu@ifNXa@Limit
```

```
%%    \gmu@if {gmu@ifquant}{}%
%%   }% of if #2 nonempty
%%  }
%%
%%
%%  \def\gmu@ifNXa@drainer {%
%%   \ifx\gmu@ifNXa@Limit\gmu@ifNXa@token
%%    \emptify\gmu@ifNXa@aasiter
%%   \fi
%%   \gmu@ifNXa@aasiter
%%  }
%%
%%
%%  \gmu@DefSymbol\gmu@ifNXa@Limit
%%
%%
%%  \def\@@gmu@ifNXa@aasiter{%
%%  % It's a pattern CS to restore \gmu@ifNXa@aasiter in each \gmu@ifNXany.
%%   \afterassignment\gmu@ifNXa@aas
%%   \let\gmu@ifNXa@token= }% thanks to this space it'll look also at
%%  % spaces (cat 10) and = chars.
%%
%%
%%  %
%%  \long\pdef\gmu@ifNXnone
%%  #1% token to be checked against
%%  #2% list of tokens to not find #1 at
%%  {%
%%   \gmu@ifNXany{#1}{#2}\@secondoftwo\@firstoftwo
%%  }
%%
```

We need a macro that iterates over every token/text on a list. LaTeX's `\@for` iterates over a list separated with commas so it's not the case. Our macro is much simpler.

```
\long\def\gmu@foreach#1\gmu@foreach@delim#2{%
```

We define the iterator that takes one item from the list, checks it against

```
  \long\def\gmu@forer##1{%
    \gmu@if␣{strings}␣{\gmu@foreach@delim␣{##1}}%
    {}% if we've met the delimiter, we stop.
    {% otherwise we wrap #1 in a macro to make it available to #2
      \edefU\gmu@forarg{##1}%
      #2% we execute #2 and probably continue iteration (unless the loop isn't
          broken with the next macro).
      \gmu@forer
    }%
  }% of forer.
```

So we apply the defined iterator

```
  \gmu@forer#1\gmu@foreach@delim
}
```

A macro to break the loop:

```
\long\def\gmu@foreach@break
#1\gmu@foreach@delim{}
```

The "if strings" or "if stribs" test performed with small quantifier. It's not as robust and all-purpose as `\gmu@ifxany` because for obvious reasons we cannot use `\futurelet`.

Note however that thanks to the nature of the test we don't need the sentinel CS to be defined.

And this is expandable

```
\long\def\gmu@ifsXXany
#1% kind of test (strings or stribs so far)
#2% token to be checked against #3
#3% the list of tokens to be iterated over
      #4 (implicit) what if found
      #5 (implicit) what if not found
{%
  \gmu@ifsXXany@{#1}{#2}#3\gmu@ifsXXany@end
  \@firstoftwo\@secondoftwo
}
```

```
\long\def\gmu@ifsXXany@
#1% kind of test
#2% left side of comparison
#3% right side of comparison
{%
  \gmu@if␣{#1}{#2#3}%
  \gmu@ifsXXany@found
  {% else
    \gmu@if␣{#1}{#3\gmu@ifsXXany@end}%
    \@secondoftwo
    {\gmu@ifsXXany@{#1}{#2}}% if we didn't meet the sentinel, we iterate
  }%
}
```

```
\long\def\gmu@ifsXXany@found
#1\gmu@ifsXXany@end
{\@firstoftwo}
```

`\gmu@ifstrany`

```
\def\gmu@ifstrany{\gmu@ifsXXany␣{strings}}
```

```
\def\gmu@ifsbany{\gmu@ifsXXany␣{stribs}}
```

```
\def\gmu@ifStrXany{\gmu@ifsXXany␣{StrX}}
```

And the counterparts:

```
\long\def\gmu@ifstrnone#1#2#3#4{%
  \gmu@ifstrany{#1}{#2}{#4}{#3}%
}
```

```
\long\def\gmu@ifsbnone#1#2#3#4{%
  \gmu@ifsbany{#1}{#2}{#4}{#3}%
}
```

```
\long\def\gmu@ifStrXnone#1#2#3#4{%
  \gmu@ifStrXany{#1}{#2}{#4}{#3}%
}
```

And their downright versions:

```
\lpdef\gmu@CASEstrany␣{\gmu@generalCASE␣\gmu@ifstrany␣}
\lpdef\gmu@CASEstrnone␣{\gmu@generalCASE␣\gmu@ifstrnone␣}
\lpdef\gmu@CASEsbany␣{\gmu@generalCASE␣\gmu@ifsbany␣}
\lpdef\gmu@CASEsbnone␣{\gmu@generalCASE␣\gmu@ifsbnone␣}
```

Note that `\gmu@ifsXXany` iterate hash by hash, so don't distinguish a CS`\par` from a `\string`ed sequence of other chars `\par` passed them in braces. To avoid such a case we provide a degrouper with which we may prepare the text for token-by-token `\gmu@ifsXXany` test:

`\degroup@toks`

```
\newtoks\degroup@toks
\long\pdef\gmu@degroup
#1% text to be degrouped
{\degroup@toks={}%
  \gmu@degroup@iter#1\gmu@degroup@end
}
\long\def\gmu@degroup@afterlet{%
  \gmu@if␣x{\degroup@lettoken\bgroup}%
  {\degroup@drainanditer}%
  {\gmu@if␣x{\degroup@lettoken\egroup}%
    {\degroup@drainanditer}%
    {\degroup@addanditer}%
  }%
}
\def\gmu@degroup@iter{%
  \futurelet\degroup@lettoken\gmu@degroup@afterlet}
\def\degroup@drainanditer{%
  \afterassignment\gmu@degroup@iter
  \let\gmu@drain=
}
\def\degroup@addanditer{%
  \gmu@if␣x{\degroup@lettoken\gmu@letspace}%
  {\addtotoks\degroup@toks{␣}%
    \degroup@drainanditer
  }{%
    \degroup@addanditer@i
  }%
}
\long\def\degroup@addanditer@i
#1{%
  \gmu@if␣{strings}{#1\gmu@degroup@end}%
  {}% we've reached the end of iteration
  {% it's sth. to add
    \addtotoks\degroup@toks{#1}%
    \gmu@degroup@iter
  }%
}
```

```
\pdef\IfIntersect
```

This is a 4-argument command (not expandable) that checks whether the list of tokens `#1` and `#2` have nonempty intersection in the sense of `\ifx` and if so it executes `#3` or `#4` otherwise:
`#1` first list to match,
`#2` second list to match,
`#3` if match (nonempty intersection),
`#4` if not match (empty intersection).

```
{\gmu@ifintersect␣\gmu@ifxany}

\lpdef\gmu@ifintersect
#1% an iterating test (\gmu@ifxany, \gmu@ifstrany or \gmu@ifsbany so far)
#2% one list to match
#3% another list to match #4 (implicit) what if intersect
     #5 (implicit) what if don't
{%
  \let\IfIntersect@next\@secondoftwo
  \gmu@foreach␣#2\gmu@foreach@delim{%
    \@xa␣#1\gmu@forarg{#3}%
    {\let\IfIntersect@next\@firstoftwo
      \gmu@foreach@break
    }{}%
  }% of \gmu@foreach's #2.
  \IfIntersect@next
}

\pdef\gmu@ifstrintersect
{\gmu@ifintersect\gmu@ifstrany}

\pdef\gmu@ifsbintersect
{\gmu@ifintersect\gmu@ifsbany}
```

A somewhat generalised `\expandafter`:

\@XAtoks
```
\newtoks\@XAtoks

\long\pdef\@XA#1{%
  \@XAtoks={#1}%
  \@xa\the\@xa\@XAtoks}

\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.
  \@XA{\IfAmong#1\among}\@xa{#2}}

\gmu@DefSymbol\defNoHash
\gmu@DefSymbol\defHashy
\gmu@DefSymbol\boolean

\def\gmu@geteschar{%
```

A macro that edefines detokenised char of the charcode `\escapechar`

```
  \edef\gmu@xiieschar{%
    \gmu@CASE␣{num}␣{\escapechar␣<\z@}
    {}%
```

```
    \gmu@CASE␣{num}{\escapechar␣<32␣}
    {\@xa␣\@firstofmany␣\string␣\blekotnizza\@nil}% not firstthreeof-
        many!!!!
    \gmu@CASE␣{num}␣{\escapechar=32␣}
    {␣}% space cannot be "first of…".
    \gmu@lastCASE
    {\@xa␣\@firstofmany␣\string␣\blekotnizza␣\@nil}%
    \gmu@ESAC
  }%
}
\long\def\gmu@IfBooleanMacro#1{%
```

this macro should provide a yes-no answer (`\@firstoftwo`/`\@secondoftwo`).

```
  \gmu@ifedetokens{\meaning#1}{macro:->true}%
  {\@firstoftwo}%
  {\gmu@ifedetokens{\meaning#1}{macro:->false}%
    {\@firstoftwo}%
    {\@secondoftwo}%
  }%
}
\pdef\IfIs
#1% a CS
#2% \dimen, \long, \toks, \skip, \count, \dimexpr, \numexpr, \glueexpr,
      \newif for \iffalse and \iftrue, \if or \conditional for any Boolean
      test/switch, \def for a macro.
```

This test tells us in particular what kind of assignment may be applied to `#1`. Therefore it turns true for `#1` being e.g. primitive TEX's skip registers and `#2==\skip`.

```
{%
  \PackageInfo{gmbase}{^^J%
    Checking␣whether␣***\string#1***␣is␣a␣***\string␣#2***^^J
    If␣the␣next␣message␣you␣get␣is␣an␣error^^J%
    ***missing␣number␣treated␣as␣␣0***,^^J
    then␣it's␣probably␣not^^J}%
  \@tempswafalse
  \gmu@CASE␣x{\defNoHash#2}%
  {% case "def no hash" (hashless macro)
    \@tempswatrue
    \edef\gmu@IfIs@resa{%
      \pdef\@nx\gmu@IfIs@resa
      ####1\detokenize{macro:->}%
      ####2\@nx\@nil{\@ifnonempty{####2}}%
```

And we prepare applying thus defined macro to `#1`:

```
      \unexpanded{\@xa\gmu@IfIs@resa\meaning#1}%
      \detokenize{macro:->}\@nx\@nil
    }% of \edef
  }% of case hasless macro ("def no hash")
```

if not hashless

```
  \gmu@CASE␣x␣{\defHashy#2}%
```

```
{%
```

case hashy macro

```
    \@tempswatrue
    \edef\gmu@IfIs@resa{%
      \pdef\@nx\gmu@IfIs@resa
      ####1\detokenize{macro:}####2\xiihash1####3->%
      ####4\@nx\@nil{\@ifnonempty{####4}}%
```

And we prepare applying thus defined macro to `#1`:

```
      \unexpanded{\@xa\gmu@IfIs@resa\meaning#1}%
      \detokenize{macro:}\xiihash1->\@nx\@nil
    }% of \edef
  }% of case hashy macro

  \gmu@CASE␣x␣{#2\boolean}
  {% case Boolean
    \@tempswatrue
    \def\gmu@IfIs@resa{\gmu@IfBooleanMacro#1}%
  }% of case Boolean

  \gmu@CASEstrany␣#2{\dimexpr␣\numexpr␣\glueexpr␣}%
  {% case ε-TeX expression
    \@tempswatrue
    \def\gmu@IfIs@resa{% if should be a macro expanding to expression contents
      \IfIsExpression␣#1#2}%
  }% of case expression
```

The subsequent part of this huge test refers to `\meaning`s. We want to make it robust to possible (although rather seldom) changes of `\escapechar`. Therefore define an aux macro that expands to detokenised escape char.

Assume that letters, icluding »e«, will not be used as the escape char:

```
  \gmu@geteschar
```

Now `\gmu@xiieschar` carries the detokenised escape char (is empty if `\escapechar <` 0).

```
  \gmu@CASE␣x␣{\dimen#2}%
  {% Case dimen. Let's check if it's special (inner TeX's) or normal.
    \gmu@ifxany#1{\prevdepth␣\pagegoal␣\pagetotal␣\pagestretch
      \pagefilstretch␣\pagefillstretch␣\pagefilllstretch
      \pageshrink␣\pagedepth
      \hfuzz␣\vfuzz␣\overfullrule␣\emergencystretch␣\hsize␣%
          \vsize
      \maxdepth␣\splitmaxdepth␣\boxmaxdepth␣\lineskiplimit
      \delimitershortfall␣\nulldelimiterspace␣\scriptspace
      \mathsurround␣\predisplaysize␣\displaywidth
      \displayindent␣\parindent␣\hangindent␣\hoffset␣\voffset
    }%
    {\@tempswatrue\let\gmu@IfIs@resa\@firstoftwo}%
    {% subcase normal dimen
      \def\gmu@IfIs@resa{%
        \gmu@ifexempty␣{\gmu@xiieschar}%
        {%
```

```
      \pdef\gmu@IfIs@resa####1####2####3####4####5####6%
           \@nil{%
        \gmu@if␣{}
        {1%
          \if␣d####1\elseo\fi
          \if␣i####2\elseo\fi
          \if␣m####3\elseo\fi
          \if␣e####4\elseo\fi
          \if␣n####5\elseo\fi
          1}%
      }% of inner \gmu@IfIs@resa...
    }% ...if eschar negative
    {%
      \pdef%
           \gmu@IfIs@resa####1####2####3####4####5####6####7%
           \@nil{%
        \gmu@if␣{}␣{1%
          \if␣\gmu@xiieschar␣####1\elseo\fi
          \if␣d####2\elseo\fi
          \if␣i####3\elseo\fi
          \if␣m####4\elseo\fi
          \if␣e####5\elseo\fi
          \if␣n####6\elseo\fi
          1%
        }%
      }% of inner \gmu@IfIs@resa...
    }% ...when eschar nonnegative
  }% of outer \gmu@IfIs@resa
 }% of if special dimen or not
}% of case dimen

\gmu@CASE␣x␣{\count#2}%
{% case count
  \gmu@ifxany#1{\spacefactor␣\prevgraf␣\deadcycles
    \insertpenalties
    \pretolerance␣\tolerance␣\hbadness␣\vbadness␣%
         \linepenalty
    \hyphenpenalty␣\exhyphenpenalty␣\binoppenalty␣%
         \relpenalty
    \clubpenalty␣\widowpenalty␣\displaywidowpenalty
    \brokenpenalty␣\predisplaypenalty␣\postdisplaypenalty
    \interlinepenalty␣\floatingpenalty␣\outputpenalty
    \doublehyphendemerits␣\finalhyphendemerits␣\adjdemerits
    \looseness␣\pausing␣\holdinginserts␣\tracingonline
    \tracingmacros␣\tracingstats␣\tracingparagraphs
    \tracingpages␣\tracingoutput␣\tracinglostchars
    \tracingcommands␣\tracingrestores␣\language␣\uchyph
    \lefthyphenmin␣\righthyphenmin␣\globaldefs
    \defaulthyphenchar␣\defaultskewchar␣\escapechar␣%
         \endlinechar
    \newlinechar␣\maxdeadcycles␣\hangafter␣\fam␣\mag
    \delimiterfactor␣\time␣\day␣\month␣\year␣\showboxbreadth
    \showboxdepth␣\errorcontextlines
```

```
      \lastlinefit
    }%
    {% if special count register then:
      \@tempswatrue\let\gmu@IfIs@resa\@firstoftwo}%
    {%
```

if normal count register then

```
      \def\gmu@IfIs@resa{%
        \gmu@ifexempty␣\gmu@xiieschar
        {\pdef\gmu@IfIs@resa####1####2####3####4####5####6%
             \@nil{%
          \gmu@if␣{}␣{1%
            \if␣c####1\elseo\fi
            \if␣o####2\elseo\fi
            \if␣u####3\elseo\fi
            \if␣n####4\elseo\fi
            \if␣t####5\elseo\fi
            1%
          }% of test
        }% of inner \gmu@IfIs@resa...
        }% ... when eschar is negative
        {% eschar not empty (nonnegative)
          \pdef%
               \gmu@IfIs@resa####1####2####3####4####5####6####7%
               \@nil{%
          \gmu@if␣{}␣{1%
            \if␣\gmu@xiieschar####1\elseo\fi
            \if␣c####2\elseo\fi
            \if␣o####3\elseo\fi
            \if␣u####4\elseo\fi
            \if␣n####5\elseo\fi
            \if␣t####6\elseo\fi
            1%
          }% of test
        }% of inner \gmu@IfIs@resa...
        }% ... when eschar nonnegative
      }% of outer \gmu@IfIs@resa
    }% of if normal count register
  }% of case count
  \gmu@CASE␣x␣{\skip#2}%
  {% case skip
    \gmu@ifxany#1{%
      \baselineskip␣\lineskip␣\parskip␣\abovedisplayskip
      \abovedisplayshortskip␣\belowdisplayskip
      \belowdisplayshortskip␣\leftskip␣\rightskip␣\topskip
      \splittopskip␣\tabskip␣\spaceskip␣\xspaceskip␣%
           \parfillskip
    }%
    {% if special skip
      \@tempswatrue\let\gmu@IfIs@resa\@firstoftwo}%
    {% not special skip
      \def\gmu@IfIs@resa{%
```

```
        \gmu@ifexempty␣\gmu@xiieschar
        {\pdef\gmu@IfIs@resa####1####2####3####4####5\@nil{%
            \gmu@if␣{}␣{1%
              \if␣s####1\elseo\fi
              \if␣k####2\elseo\fi
              \if␣i####3\elseo\fi
              \if␣p####4\elseo\fi
              1%
            }%
          }% of inner \gmu@IfIs@resa...
        }% ...when eschar nonnegative
        {\pdef\gmu@IfIs@resa####1####2####3####4####5####6%
             \@nil{%
            \gmu@if␣{}␣{1%
              \if␣\gmu@xiieschar␣####1\elseo\fi
              \if␣s####2\elseo\fi
              \if␣k####3\elseo\fi
              \if␣i####4\elseo\fi
              \if␣p####5\elseo\fi
              1%
            }%
          }% of inner \gmu@IfIs@resa...
        }% ...when eschar nonnegative
      }% of outer \gmu@IfIs@resa
    }% of if skip but not special
  }% of case skip

  \gmu@CASE␣x␣{\toks#2}
  {% case toks
    \gmu@ifxany#1{%
      \output␣\everypar␣\everymath␣\everydisplay␣\everyhbox
      \everyvbox␣\everyjob␣\everycr␣\errhelp␣\everyeof}%
    {% subcase special toks
      \@tempswatrue\let\gmu@IfIs@resa\@firstoftwo}%
    {% subcase normal toks
      \def\gmu@IfIs@resa{%
        \gmu@ifexempty␣\gmu@xiieschar
        {\pdef\gmu@IfIs@resa####1####2####3####4####5\@nil{%
            \gmu@if␣{}␣{1%
              \if␣t####1\elseo\fi
              \if␣o####2\elseo\fi
              \if␣k####3\elseo\fi
              \if␣s####4\elseo\fi
              1%
              }%
            }% of inner \gmu@IfIs@resa...
          }% ...when eschar negative
        {\pdef\gmu@IfIs@resa####1####2####3####4####5####6%
             \@nil{%
            \gmu@if␣{}␣{1%
              \if␣\gmu@xiieschar␣####1\elseo\fi
              \if␣t####2\elseo\fi
              \if␣o####3\elseo\fi
```

```
                \if␣k####4\elseo\fi
                \if␣s####5\elseo\fi
                1%
                }%
              }% of inner \gmu@IfIs@resa...
            }% ...when eschar nonnegative
        }% of outer \gmu@IfIs@resa
    }% of if toks but not special
}% of case toks

\gmu@CASE␣x␣{\long#2}%
```

if a macro is `\long`, then these detokens open its meaning despite of possible `\outer`.

```
{% case long macro
  \def\gmu@IfIs@resa{%
    \gmu@ifexempty␣\gmu@xiieschar
    {\pdef\gmu@IfIs@resa####1####2####3####4####5\@nil{%
        \gmu@if␣{}␣{1%
          \if␣l####1\elseo\fi
          \if␣o####2\elseo\fi
          \if␣n####3\elseo\fi
          \if␣g####4\elseo\fi
          1%
        }%
      }% of inner \gmu@IfIs@resa...
    }% ...when eschar negative
    {\pdef\gmu@IfIs@resa####1####2####3####4####5####6%
         \@nil{%
        \gmu@if␣{}␣{1%
          \ifnum␣\gmu@xiieschar␣####1\elseo\fi
          \if␣l####2\elseo\fi
          \if␣o####3\elseo\fi
          \if␣n####4\elseo\fi
          \if␣g####5\elseo\fi
          1%
        }%
      }% of inner \gmu@IfIs@resa...
    }% ...when eschar nonnegative
  }% of outer \gmu@IfIs@resa
}% of case \long Note that we also can put here a test whether the meaning begins
     with macro which would mean that a macro is short.
\gmu@CASE␣x␣{\newif#2}%
{% case \newif (Boolean switch)
  \def\gmu@IfIs@resa{%
    \gmu@ifexempty␣\gmu@xiieschar
    {\pdef\gmu@IfIs@resa####1####2####3%
      ####4####5####6####7####8\@nil{%
        \gmu@unless␣{}␣{0% lazy disjunction
          \gmu@if␣{}␣{1% lazy conjunction
            \if␣i####1\elseo\fi
            \if␣f####2\elseo\fi
            \if␣f####3\elseo\fi
```

```
      \if␣a####4\elseo\fi
      \if␣l####5\elseo\fi
      \if␣s####6\elseo\fi
      \if␣e####7\elseo\fi
      1%
    }%
    {}{0}%
    \gmu@if␣{}␣{1% lazy conjunction
      \if␣i####1\elseo\fi
      \if␣f####2\elseo\fi
      \if␣t####3\elseo\fi
      \if␣r####4\elseo\fi
      \if␣u####5\elseo\fi
      \if␣e####6\elseo\fi
      \if␣\relax####7\elseo\fi
      1%
    }%
    {}{0}%
    0%
  }%
 }% of inner \gmu@IfIs@resa...
}% ...when eschar nonnegative

{\pdef\gmu@IfIs@resa####1####2####3%
  ####4####5####6####7####8####9\@nil{%
  \gmu@unless␣{}␣{0%
    \gmu@if␣{}␣{1%
      \if␣\gmu@xiieschar␣####1\elseo\fi
      \if␣i####2\elseo\fi
      \if␣f####3\elseo\fi
      \if␣f####4\elseo\fi
      \if␣a####5\elseo\fi
      \if␣l####6\elseo\fi
      \if␣s####7\elseo\fi
      \if␣e####8\elseo\fi
      1%
    }%
    {}{0}%
    \gmu@if␣{}␣{1%
      \if␣\gmu@xiieschar␣####1\elseo\fi
      \if␣i####2\elseo\fi
      \if␣f####3\elseo\fi
      \if␣t####4\elseo\fi
      \if␣r####5\elseo\fi
      \if␣u####6\elseo\fi
      \if␣e####7\elseo\fi
      \if␣\relax####8\elseo\fi
      1%
    }%
    {}{0}%
    0%
  }%
 }% of inner \gmu@IfIs@resa...
```

```
      }% …when eschar nonnegative
    }% of outer \gmu@IfIs@resa
  }% of case \newif (Boolean switch)

  \gmu@CASE␣{x\@xa\@xa}␣{\csname␣if\endcsname#2}%
  {% case \if test
    \def\gmu@IfIs@resa{%
      \gmu@ifexempty␣\gmu@xiieschar
      {\pdef\gmu@IfIs@resa####1####2####3\@nil{%
          \gmu@if␣{}␣{1%
            \if␣i####1\elseo\fi
            \if␣f####2\elseo\fi
            1%
          }%
        }% of inner \gmu@IfIs@resa…
      }% …when eschar negative

      {\pdef\gmu@IfIs@resa####1####2####3####4\@nil{%
          \gmu@if␣{}␣{1%
            \if␣\gmu@xiieschar␣####1\elseo\fi
            \if␣i####2\elseo\fi
            \if␣f####3\elseo\fi
            1%
          }%
        }% of inner \gmu@IfIs@resa…
      }% …when eschar nonnegative
    }% of outer \gmu@IfIs@resa
  }% of case \if

  \gmu@lastCASE␣%
  {false}{}{}%
  \gmu@ESAC
```

For the cases 1–*n* we just launch their auxiliary macro:

```
  \if@tempswa
    \@xa\gmu@IfIs@resa
  \else
```

For the other cases their auxiliary macro defines "inner `@resa`" (protected) which perform the test on the meaning of `#1`.

```
      \gmu@IfIs@resa
```

now (new) `\gmu@IfIs@resa` it's defined as delimited with `#2` and `\@nil` (if `#1` may be one of kinds checked letter by letter).

```
      \edef\gmu@IfIs@resb{%
      \gmu@IfIs@resa\meaning#1%
      \relax\relax\relax\relax\relax\relax\relax\relax␣% to provide
            something for gobbling tests.
      \@xa\detokenize\@xa{\string#2}\@nx\@nil}%
    \@xa\gmu@IfIs@resb
  \fi
}% of \IfIs

\unless\ifdefined\@tempskipa\newskip\@tempskipa\fi
\unless\ifdefined\@tempmuskipa\newmuskip\@tempmuskipa\fi
```

```
\long\def\IfIsExpression
#1% the stuff to be examined
#2% \dimexpr, \glueexpr, \numexpr or \muexpr
{%
  \ifx#2\numexpr\let\next\@tempcnta\fi
  \ifx#2\glueexpr\let\next\@tempskipa\fi
  \ifx#2\dimexpr\let\next\@tempdima\fi
  \ifx#2\muexpr\let\next\@tempmuskipa\fi
  \afterassignment\gmu@testtopenalty
  \next=#2#1\penalty
}

\def\gmu@testtopenalty#1\penalty{%
  \gmu@ifempty{#1}}
```

#### Global Boolean switches

The `\newgif` declaration's effect is used even in the LaTeX $2_\varepsilon$ 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`

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

'Almost' is also in the detail that in this case, which deals with `\global` assignments, we don't have to bother with storing and restoring the value of `\escapechar`: we can do all the work inside a group.

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

\pdef\newif#1{% We not only make \newif \protected but also make it to de-
      fine \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@}

\def\@if#1#2{%
  \protected␣\@xa\def\csname\@xa\@gobbletwo\string#1%
  \@xa\@gobbletwo\string#2\endcsname
  {\let#1#2}}

\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:

```
\pdef\hgrefstepcounter#1{%
  \ifhmode\leavevmode\fi\grefstepcounter{#1}}
```

By the way I read some lines from *The TeX book* 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

```
\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 \expandafters e.g.

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

And for use in the very document,

\addtomacro
```
\lpdef\addtomacro
#1% macro (has to be parameterless)
#2% stuff to be added
{\edef␣#1{\@xau␣#1\unexpanded{#2}}}
```

We use unexpanded edef to allow # tokens in #2. Note that LaTeX's \g@addto@macro uses analogous trick from the pre-ε-TeX era (scratch toks register and edef).

Note that \addtomacro loses the possible prefixes of the previous version of #1 but may be prefixed on its own (anyway, what's the use of \long for a macro with no parameters? And who on Earth uses \outer?).

Note moreover it works fine also for #1's that are \protected because \expandafter hits first and always works.

A \global version:

```
\pdef\gaddtomacro{\global\addtomacro}
```

2008/08/09 I need to prepend something not add at the end—so

```
\long\def\prependtomacro#1#2{%
  \edef#1{\unexpanded{#2}\@xa\unexpanded\@xa{#1}}}
```

Note that `\prependtomacro` can be prefixed.

`\addtotoks`

```
\lpdef\addtotoks#1#2{%
  #1=\@xa{\the#1#2}}
```

`\gmu@prependtoks@aux`

```
\newtoks\gmu@prependtoks@aux
```

```
\lpdef\gmu@prependtotoks@ambig
#1% scope
#2% toks register
#3% text of prependement
{%
  \iffalse␣{\fi␣% hack to balance braces in definition
    \@XA{%
      #1#2=\bgroup#3%
    }% during execution, this brace closes the \@XA's argument…
    \the#2}% and this one closes the text opened by \bgroup, i.e., the text of
          \toks assignment.
}
```

`\prependtotoks`

```
\lpdef\prependtotoks
```

the "looocal" version (2010/11/10)

```
#1% toks register
#2% text to be prepended
{\gmu@prependtotoks@ambig␣{}{#1}{#2}}%
```

```
\lpdef\gprependtotoks
```

the "global" version (2010/11/10)

```
#1% toks register
#2% text to be prepended
{\gmu@prependtotoks@ambig␣\global{#1}{#2}}%
```

Adding an element to a list iterated by by `\@for`

`\addto@forlist`

```
\long\def\addto@forlist
#1% a comma-separated list
#2% the element(s) added
{\ifx#1\@empty
    \@xa\@gobble
  \else\@xa\@firstofone
  \fi
  {\addtomacro#1{,}}%
  \addtomacro#1{#2}%
}
```

```
\lpdef\eaddtomacro
#1% a macro
#2% stuff to be added (will be fully expanded)
{\edef#1{\@xau#1#2}}
```

### \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.

```
\long\def\gmu@ifundefined␣#1{% not \protected because expandable.
  \gmu@CASEnot␣{csname}␣{#1\endcsname}% defined…
  {\@firstoftwo}%
  \gmu@CASE␣␣{x\@xa\@xa}␣{\csname␣#1\endcsname\relax}% but only as
       \relax—then 2nd argument.
  {\@firstoftwo}%
```

defined and not \relax—then 3rd argument.

```
  \gmu@lastCASE
  {\@secondoftwo}%
  \gmu@ESAC
}
\long\def\gmu@ifdefined␣#1#2#3{%
  \gmu@ifundefined␣{#1}{#3}{#2}}
```

While \gmu@if(un)defined are intended for csnames and any macros present in their #1 are expanded, the next two are intended for #1 being a single CS or an active char.

It's the active char's case why we write all *da capo*.

```
\long\def␣\gmu@ifCSdefined#1{%
  \gmu@CASEnot␣{defined}␣{#1}
  {\@secondoftwo}%
  \gmu@CASE␣x␣{\relax␣#1}
  {\@secondoftwo}%
  \gmu@EatCases
  {\@firstoftwo}%
  \gmu@ESAC
}
\long\def\CSNameIf
#1% the name to check and probably execute
 2 stuff when the CS#1 defined
 3 stuff when the CS#1 not defined
{%
  \ifcsname␣#1\endcsname
    \@xa\@twoofthree
  \else\@xa\@thirdofthree
  \fi
  {\csname␣#1\endcsname}%
}
```

### 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␣}
{\let\sups=^}

\foone{\@makeother\_}
{\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.

```
\foone{\@makeother\^^M}{\def\xiiM{
  }}%

\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.

```
\def\all@spaces{␣}
\foone{\@makeother\␣}{%
\addtomacro\all@spaces{␣}%
}
\foone{\obeyspaces}{%
\addtomacro\all@spaces{␣}%
}

\foone\obeylines{%
  \def\two@Ms{
  }}
\foone{\@makeother\^^M}{%
  \addtomacro\two@Ms{
  }}

\edef\spaces@and@Ms{\@xau{\all@spaces}\@xau{\two@Ms}}

\ifdefined\XeTeXversion
  \chardef\_="005F
\fi

\foone{\@makeother\`}%
{\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}
{%
\smartunder  \pdef\smartunder{%
    \catcode`\_=\active
    \def_{%
      \ifincsname\xiiunder
      \else
        \ifmmode\subs
        \else\_%
        \fi
      \fi}}}% We define it as \_ not just as \xiiunder because some font encodings don't have _ at the \char`\_ position.

\foone{\catcode`\!=0
  \@makeother\\}
\xiibackslash {!newcommand*!xiibackslash{\}}

\bslash \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
    \def\xxt@visiblespace@fallback{{%
        \fontspec{Latin␣Modern␣Mono}\textvisiblespace}}%
  \else
    \let\visiblespace\xiispace
  \fi}

\foone\obeyspaces{\def\gmu@activespace{␣}}

\foone\obeylines{\def\activeM{^^M}}

\pdef\makeblanksignored{%
  \catcode`\^^M=9\relax
  \catcode`\^^I=9\relax
  \catcode`\␣=9\relax}

\pdef\fooblanksignored{%
  \foone{\makeblanksignored}%
}
```

### \@ifnextcat, \@ifnextac, catcode-independent \gm@ifstar, \@ifnextnotgroup, \@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 TeX book*, 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….

```
\long\pdef\edefU#1#2{%
```

This is to allow passing the hashes.

```
  \edef#1{\unexpanded{#2}}%
}

\long\pdef\@ifnextcat#1#2#3{%
  \edefU\reserved@d{#1}%
  \edefU\reserved@a{#2}%
  \edefU\reserved@b{#3}%
  \futurelet\@let@token\@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.

```
\pdef\spifletter{\@ifnextcat␣a{\space}{}}

\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.

```
\long\pdef\@ifnextac#1#2{%
  \@ifnextnotgroup
  {\gmu@ifnac{#1}{#2}}%
  {#2}}

\long\def\gmu@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 `\gmu@ifstar` to make it work with whatever catcode $\star$ may be. make a version of `\gmu@ifstar` that would work with $\star_{11}$.

```
\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 mis-
    leading (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 12606 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\gmu@ifstar##1##2{%
    \@nx\@ifnextif\gmu@tempa%
    {\@nx\@firstoftwo{##1}}% it's a bit hacky but O.K.: if the condition is not
        satisfied, the following brace is taken
    {%
      \@nx\@ifnextchar\@xa\@nx\gmu@tempb
      {\@nx\@firstoftwo{##1}}% and again, if the condition is not satisfied,
          the second brace is taken.
      {##2}%
    }% of if not ⋆₁₂
  }% of \gmu@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 }.

```
\long\pdef\@ifnextnotgroup#1#2{% This macro checks whether the next to-
    ken 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}}}

\long\pdef\@ifnextgroup#1#2{% Note this macro turns true both before a group
    opener and before a group closer.
  \@ifnextnotgroup{#2}{#1}}
```

now let's apply this to sth. useful (used in gmverse and in some typesetting, e.g. for prof. JSB.

```
\pdef\ignoreactiveM{%
  \@ifnextgroup{}{\gmu@checkM}}

\foone\obeylines{% we know it's a single token since we use this macro only
    in \@ ifnextgroup's 'else'.
  \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 TEX book* appendix D) since `\let`'s syntax includes one optional space after `=`.

```
\let\gmu@reserveda\*%
\def\*{%
  \let\*\gmu@reserveda
  \let\gmu@letspace=␣}%
\*␣%
```

*The TEX book* chapter 8, 10th double bend says, if we read thoroughly (or meet this perversity in our daily TEXing), that a blank line (`^^M^^M` of catcode 5, the two subsequent chars of catcode 5 (preceded with sth. else)) are transformed in `␣\par`—a blank space (cat. 10) followed by `\par`. This strange case we *don't* want to treat as 'next is space', as TEX itself doesn't (a letter CS gobbles such a space).

```
\lpdef\gmu@peep@next␣#1{%
```

This macro performs `\futurelet` to `\@let@token`, checks if we are in this strange case of a blank space before `\par` (and fixes `\@let@token` if so), moreover, if `\@let@token` turns out to be undefined or `\relax`, it grabs the next token (thus surely single and not `\outer`) and passes as the contents of `\@def@token` (which is un-defined each time).

Therefore `#1` for it may be branched with `\ifdefined\@def@token`.

```
  \let\@def@token\@undefined
  \edefU\gmu@peepnext@inner␣{%
    \gmu@CASE␣x␣{\@let@token\gmu@letspace}%
    {\@ifnextchar\par
      {#1}
      {\let\@let@token=␣\gmu@letspace␣% note = and blank space
        \@XA{#1}\space
      }%
    }% of if \futurelet detected a blank space

    \gmu@CASE␣{AnyClause}␣% if any of the conditions below:
    {{␣x␣{\@let@token\@undefined}
        x␣{\@let@token\relax}}␣% it's arg of disjunction then we grab the
            next token into the \@def@token macro.
    {\gmu@peep@hash{#1}}%
    \gmu@lastCASE
    {#1}%
    \gmu@ESAC
  }%
  \futurelet\@let@token\gmu@peepnext@inner
}

\lpdef\gmu@peep@hash
#1% stuff that probably tests #2 somehow
#2% a single token, which we are sure is undefined or \relax (and thus not \outer
     neither a blank space)
{%
  \def\@def@token{#2}%
  #1%
  #2%
}

\lpdef\gmu@ifpeeped
```

```
#1% kind of comparison
#2% stuff to compare
{%
  \gmu@if␣{defined}␣{\@def@token}
  {\@XA{\gmu@if␣{#1}}\@xa{\@def@token␣#2}}
  {\gmu@if␣{#1}␣{\@let@token␣#2}}%
}
```

\@ifnextspace

```
\long\pdef\@ifnextspace
#1% if yes
#2% if not
{%
```

Note that this macro doesn't gobble space(s)

```
  \gmu@peep@next
  {\gmu@if␣x␣{\@let@token\gmu@letspace}{#1}{#2}}%
}
```

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.

The next—in the gmdoc bundle not to gobble spaces following %, which is crucial for determining whether there is a DocStrip directive or not. But this is done with a wrapper for both \@ifnextspace and \@ifnextchar:

"if next char" that respects spaces

\@ifnextcharRS

```
\long\pdef\@ifnextcharRS
#1% a single token (will be \ifxed)
#2% what if yes
#3% what if not
{%
  \gmu@peep@next
  {\gmu@if␣{defined}␣{\@def@token}
    {\@XA{\gmu@if␣{StrX}}\@xa␣{\@def@token␣#1}}% of the special "null"
          case
    {\gmu@if␣x␣{\@let@token␣#1}}%
    {#2}{#3}%
  }% of \gmu@peep@next
}
```

"if next any" that respects spaces

\@ifnextanyRS

```
\long\pdef\@ifnextanyRS
#1% list of tokens (any balanced text, will be \let token after token
#2% what if next is one of listed #1
#3% what if not
{\gmu@peep@next
  {\gmu@if␣{defined}␣{\@def@token}
    {\@xa␣\gmu@ifStrXany␣\@def@token␣}
```

otherwise we are next to a defined and not \relax token so

```
    {\gmu@ifxany␣{\@let@token}}%
    {#1}{#2}{#3}%
  }% of peep
```

```
}
```

Some (quite large) chunks of commented out code were here till v0.240.

```
\long\def\gmu@ifnextStrXany
```

We call this macro only in the true branch of `\ifx\@let@token#1`

```
#1% outer macro's tested list of tokens
#2% "if found" branch
#3% "if not found" branch
{\gmu@notif␣{AnyClause}
  {{␣x␣{\@let@token\@undefined}␣x␣{\@let@token\relax}}}%
  {#2}%
```

Some (i.e. *both*) `\@undefined` or `\relax`, then we compare strings

```
  {\gmu@futureifany␣{#2}{#3}{#1}}%
}
```

```
\long\def\gmu@futureifany
```

As above.

```
#1% what if OK
#2% what if not OK
#3% list of tokens
#4% token ot be checked against #3 (and put back on the input) Note we use this
      macro only where we know #4 is either undefined or \relax.
{\gmu@ifstrany␣#4␣{#3}{#1}{#2}#4}
```

"if next any" that ignores space(s)

`\@ifnextanyIS`

```
\long\pdef\@ifnextanyIS
#1% list of tokens (any balanced text, will be \let token after token
#2% what if next is one of listed in #1
#3% what if not
{%
  \edefU\gmu@ifna@afterlet{%
    \gmu@if␣x{\@let@token\gmu@letspace}%
    {% if next is blank space, we drop it and retry:
      \edefU\gmu@ifna@resa{\@ifnextanyIS{#1}{#2}{#3}}%
      \afterassignment\gmu@ifna@resa
      \let\gmu@drain=
    }%
    {% else we perform \gmu@ifnextany
      \gmu@ifxany{\@let@token}{#1}{%
        \gmu@ifnextStrXany␣{#1}{#2}{#3}%
      }%
      {#3}%
    }%
  }% of the after-let macro
  \futurelet\@let@token\gmu@ifna@afterlet
}% of \@ifnextanyIS
```

`\@ifnextnoneRS`

```
\long\pdef\@ifnextnoneRS
#1% list of tokens (any balanced text, will be \let token after token
#2% what if next is one of listed #1
#3% what if not
```

```
{%
  \@ifnextanyRS{#1}{#3}{#2}%
}
```

If-next-group respecting spaces

```
\long\pdef\@ifnextgroupRS␣#1#2{%
```

Note that this macro doesn't gobble space(s)

```
  \gmu@peep@next
  {\gmu@if␣x{\@let@token\bgroup}{#1}%
    {\gmu@if␣x{\@let@token\egroup}{#1}{#2}}%
  }%
}
```

```
\long\pdef\@ifnextnotgroupRS#1#2{%
  \@ifnextgroupRS{#2}{#1}}
```

What seems worth noticing is that my if-nexts don't use `\reserved@`(a|…|d) and set `\@let@token` properly.

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.

```
\foone\obeylines{%
  \long\pdef\@ifnextMac#1#2{%
    \@ifnextchar^^M{#1}{#2}}}
```

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'.

```
\def\@xiispaces#1␣#2\@nil{%
  #1%
  \ifx\@xiispaces#2\@xiispaces
  \else
  \xiispace
  \afterfi{\@xiispaces#2\@nil}%
  \fi}
```

```
\long\pdef\xiiEdetoke
#1% a scratch CS
#2% stuff to be fully expanded and turned to catcode 12 including spaces.
{%
  \edef#1{#2}%
  \edef#1{%
    \@xa\@xa\@xa\@xiispaces
    \@xa\detokenize\@xa{#1}␣\@nil}%
```

```
}
\long\def\@ifEUnextchar#1#2#3{%
```

'if Edefed Unexpanded next char'

```
  \let\reserved@d=#1%
 \edefU\reserved@a{#2}%
 \edefU\reserved@b{#3}%
 \futurelet\@let@token\@ifnch}
```

#### Storing and restoring the catcodes of specials

\gmu@storespecials
\gmu@restorespecials

```
\newcommand\gmu@storespecials[1][]{%
```

we provide a possibility of adding stuff. For usage see line **??**.

```
  \def\do##1{\catcode`\@nx##1=\the\catcode`##1\relax}%
  \edef\gmu@restorespecials{%
    \dospecials\do\^^M}#1}
```

```
\pdef\gmu@septify{%
```

restoring the standard catcodes of specials. The name is the opposite of 'sanitize' :-) . It restores also the original catcode of `^^M`.

```
  \def\do{\relax\catcode`}%
  \do\␣10\do\\0\do\{1\do\}2\do\$3\do\&4%
  \do\#6\do\^7\do\_8\do\%14\do\~13\do\^^M5\relax
 %%␣\let\do\@makeother
 %%␣\do0\do1\do2\do3\do4\do5\do6\do7\do8\do9\relax
}
```

#### Storing and restoring the meanings of CSes

First a Boolean switch of globalness of assignments and its verifier.

\ifgmu@SMglobal

```
\newif\ifgmu@SMglobal

\pdef\SMglobal{\gmu@SMglobaltrue}

\def\MakePrivateLetters{\makeatletter}
```

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\MakePrivateLetters
  \gmu@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 3350.

```
\lpdef\egStore@Macro#1{\endgroup\Store@Macro{#1}}
\lpdef\egStore@MacroSt#1{\endgroup\Store@MacroSt{#1}}

\pdef\StoreMacro@nocat
```

It's version of `\StoreMacro` to be used in arguments of other macros, where recatcoding wouldn't change anything anyway.

```
{%
  \gmu@ifstar␣\Store@MacroSt␣\Store@Macro
}
```

```
\def\gmu@storeprefix{/gmu/store}
```

```
\lpdef\Store@Macro#1{%
  \escapechar92
  \ifgmu@SMglobal\afterfi\global\fi
  \@xa\let\csname␣\gmu@storeprefix/\bslash@or@ac{#1}%
      \endcsname#1%
  \global\gmu@SMglobalfalse
}
```

```
\lpdef\Store@MacroSt#1{%
  \edef\gmu@smtempa{%
    \ifgmu@SMglobal\@xa\global\fi
    \let\@xa\@nx\csname\gmu@storeprefix/\bslash@or@ac{#1}\@xa%
        \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`.

```
    \ifcsname␣#1\endcsname␣%
```

If the argument CS is undefined, we undefine the storage macro too.

```
      \@xa\@nx\csname#1\endcsname
    \else\@nx\@undefined
    \fi
    \global\gmu@SMglobalfalse}%
```

we wish the globality to be just once.

```
  \gmu@smtempa
}
```

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

The starred version, `\StoreMacro*` works with csnames (without the backslash). It's first used to store the meanings of robust commands, when you may need to store not only `\foo`, but also `\csname␣foo␣\endcsname`.

The next command iterates over a list of CSes and stores each of them. The CS'es may be separated with commas but they don't have to.

`\StoreMacros`

```
\lpdef\StoreMacros{\begingroup\MakePrivateLetters%
    \egStore@Macros}
```

```
\lpdef\egStore@Macros#1{\endgroup
  \Store@Macros{#1}%
}
```

```
\lpdef\Store@Macros␣#1{%
  \gmu@setsetSMglobal
  \let\gml@StoreCS\Store@Macro
  \gml@storemacros#1.}
```

```
\def\gmu@setsetSMglobal{%
  \ifgmu@SMglobal
    \let\gmu@setSMglobal\gmu@SMglobaltrue
  \else
    \let\gmu@setSMglobal\gmu@SMglobalfalse
```

```
  \fi}
```

And the inner iterating macro:

```
\lpdef\gml@storemacros#1{%
  \def\gmu@storemacros@resa{\@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.
  \if\gmu@storemacros@resa.% a dot finishes storing.
    \global\gmu@SMglobalfalse
  \else
    \if\gmu@storemacros@resa,% The list this macro is put before may con-
         tain commas and that's O.K., we just continue the work.
      \afterfifi\gml@storemacros
    \else% what is else this shall be stored.
      \gml@StoreCS{#1}% we use a particular CS to may \let it both to the stor-
           ing macro as above and to the restoring one as below.
      \afterfifi{\gmu@setSMglobal\gml@storemacros}%
    \fi
  \fi
}
```

And for the restoring

\RestoreMacro

```
\lpdef\RestoreMacro{%
  \begingroup\MakePrivateLetters\gmu@ifstar\egRestore@MacroSt%
       \egRestore@Macro}
\lpdef\egRestore@Macro#1{\endgroup\Restore@Macro{#1}}
\lpdef\egRestore@MacroSt#1{\endgroup\Restore@MacroSt{#1}}

\lpdef\Restore@Macro#1{%
  \escapechar92
  \gmu@ifstored#1{%
    \ifgmu@SMglobal\afterfi\global\fi
    \@xa\let\@xa#1\csname␣\gmu@storeprefix/\bslash@or@ac{#1}%
         \endcsname
    \global\gmu@SMglobalfalse}%
  {\unless\ifgmu@quiet
     \PackageWarning{gmutils}{\@nx#1␣is␣not␣stored,␣I␣do␣
          nothing␣with
        it}%
    \fi
  }%
}

\long\def\gmu@ifstored#1#2#3{%
  \gmu@ifundefined{\gmu@storeprefix/%
    \bslash@or@ac{#1}}{#3}{#2}%
}

\lpdef\gmu@storeifnotyet#1{%
  \gmu@if␣{}␣{\relax\@nx#1}% we check if it's a CS
  {\gmu@ifstored{#1}{}{\StoreMacro@nocat#1}}%
  {}%
}

\lpdef\Restore@MacroSt#1{%
```

```
  \gmu@ifundefined{\gmu@storeprefix/\bslash@or@ac{#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@storeprefix/\bslash@or@ac{#1}\endcsname}% cf.
          the commentary in line 3151.
    \gmu@smtempa}%
  \global\gmu@SMglobalfalse
}
```

\RestoreMacros

```
\lpdef\RestoreMacros{%
  \begingroup\MakePrivateLetters
  \egRestore@Macros}

\lpdef\egRestore@Macros#1{\endgroup
  \Restore@Macros{#1}%
}

\lpdef\Restore@Macros #1{%
  \gmu@setsetSMglobal
  \let\gml@StoreCS\Restore@Macro% we direct the core CS towards restoring
      and call the same iterating macro as in line 3192.
  \gml@storemacros#1.}
```

As you see, the \RestoreMacros command uses the same iterating macro inside, it only changes the meaning of the core macro.

```
\pdef\ResetMacro{% restore possibly to \@undefined
  \begingroup\MakePrivateLetters\gmu@ifstar\egReset@MacroSt%
      \egReset@Macro}

\lpdef\egReset@Macro#1{\endgroup\Reset@Macro{#1}}
\lpdef\egReset@MacroSt#1{\endgroup\Reset@MacroSt{#1}}

\lpdef\Reset@Macro#1{%
  \escapechar92
  \ifgmu@SMglobal\@xa\global\fi
  \ifcsname  \gmu@storeprefix/\bslash@or@ac{#1}\endcsname
    \@xa\let\@xa#1\csname \gmu@storeprefix/\bslash@or@ac{#1}%
        \endcsname
  \else
    \let#1\@undefined
  \fi
  \global\gmu@SMglobalfalse
}%

\lpdef\Reset@MacroSt#1{%
  \ifcsname \gmu@storeprefix/\bslash@or@ac{#1}\endcsname
    \ifgmu@SMglobal\@xa\global\fi
    \@xa\let\csname#1\@xa\endcsname
    \csname\gmu@storeprefix/\bslash@or@ac{#1}\endcsname % cf. the
        commentary in line 3151.
  \else
```

```
    \@xa\let\csname#1\endcsname\@undefined
  \fi
  \global\gmu@SMglobalfalse
}
```

\ResetMacros
```
\lpdef\ResetMacros{\begingroup\MakePrivateLetters%
    \Reset@Macros}
\lpdef\Reset@Macros#1{\endgroup
  \gmu@setsetSMglobal
  \let\gml@StoreCS\Reset@Macro% we direct the core CS towards restoring
      and call the same iterating macro as in line 3192.
  \gml@storemacros#1.}
```

As you see, the \ResetMacros command uses the same iterating macro inside, it only changes the meaning of the core macro.

And to restore *and* use immediately:

```
\pdef\StoredMacro{\begingroup\MakePrivateLetters\Stored@Macro}
\lpdef\Stored@Macro#1{\endgroup\Restore@Macro#1#1}
```

To be able to call a stored CS without restoring it.

```
\long\def\storedcsname#1{%
  \ifcsname␣\gmu@storeprefix/\bslash@or@ac{#1}\endcsname
    \afterfi{%
      \csname␣\gmu@storeprefix/\bslash@or@ac{#1}\endcsname}%
  \else␣\@xa␣\@undefined
  \fi
}
```

2008/08/03 we need to store also an environment.

```
\pdef\StoreEnvironment#1{%
  \Store@MacroSt{#1}\Store@MacroSt{end#1}}
\pdef\RestoreEnvironment#1{%
  \Restore@MacroSt{#1}\Restore@MacroSt{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
```
\pdef\StoringAndRelaxingDo{%
  \gmu@SMdo@setscope
  \long\def\do##1{%
    \gmu@SMdo@scope
    \@xa\let\csname␣\gmu@storeprefix/\bslash@or@ac{##1}%
        \endcsname##1%
    \gmu@SMdo@scope\let##1\relax}}
\pdef\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
```
\lpdef\RestoringDo{%
  \gmu@SMdo@setscope
  \long\def\do##1{%
    \gmu@SMdo@scope
    \@xa\let\@xa##1\csname
    \gmu@storeprefix/\bslash@or@ac{##1}\endcsname}%
}
```

Note that both \StoringAndRelaxingDo and \RestoringDo are sensitive to the \SMglobal 'prefix'.

(Preliminary:)

```
\pdef\gmu@MakeScopePrefix
#1% CS to be let \global or \relax
#2% a sequence of tokens
{%
  \let#1\relax
  \gmu@ifxany{\global}{#2}%
  {\let#1\global}{}%
}
```

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):

```
\lpdef\gmu@namelet
#1% scope prefix (to be honest, any sequence of tokens that may be passed as an
      argument: \gmu@ifxany will parse it)
#2% left side of the assignment
#3% right side of the assignment
{%
  \gmu@MakeScopePrefix\gmu@namelet@scpref{#1}%
  \gmu@if␣{csname}␣{#3\endcsname}%
  {%
    \@xa\gmu@namelet@scpref\@xa\let\csname#2\@xa\endcsname
    \csname#3\endcsname}%
  {%
    \@xa\gmu@namelet@scpref
    \@xa\let\csname#2\endcsname\@undefined}%
}

\pdef\n@melet{\gmu@namelet\relax}

\pdef\gn@melet{\gmu@namelet\global}

\long\pdef\tri@let
#1% scope prefix(es)
#2% left side
#3% right side of the assignment
{%
  \gmu@MakeScopePrefix\gmu@tri@let@scpref{#1}%
```

both s.o.a. can be names, CS es or active chars.

```
  \ifcat\@nx~\@nx#2%
    \def\next{\gmu@tri@let@scpref\let#2}%
  \else␣\edef\next{\gmu@tri@let@scpref\let\@xanxtri{#2}}%
  \fi
```

Thus the left argument of the assignment is handled and of the assignment prepared.

```
  \ifcat\@nx~\@nx#3%
    \next#3%
  \else
    \edef\next{%
      \@xau\next
      \ifcsname␣\strip@bslash{#3}\endcsname
      \@xanxtri{#3}%
      \else\@nx\@undefined
      \fi
    }% of next's edef
    \next
  \fi
}%
```

```
\long\pdef\envirlet#1#2{% for \letting environments.
  \n@melet{#1}{#2}%
  \n@melet{end#1}{end#2}%
}
```

```
\long\pdef\glenvirlet#1#2{% for \letting environments.
  \gn@melet{#1}{#2}%
  \gn@melet{end#1}{end#2}%
}
```

`\@ifprevenvir` are defined in gmenvir

### Setting for XeTeX

```
\def\@ifXeTeX{% two-argument command
  \ifdefined\XeTeXversion
  \unless\ifx\XeTeXversion\relax\afterfifi\@firstoftwo\else%
       \afterfifi\@secondoftwo\fi
  \else\afterfi\@secondoftwo\fi}
```

```
\def\XeTeXifprefix{% to be used as prefix to an \if... test.
  \@ifXeTeX{}{\unless}}
```

`\XeTeXthree` is defined with `\DeclareCommand` so occurs yet in gmcommand.

```
\@ifXeTeX{%
  \pdef\textbullet{%
    \iffontchar\font"2022␣\char"2022␣\else\ensuremath{%
         \bullet}\fi}%
  \pprovide\glyphname#1{%
    \XeTeXglyph␣\numexpr\XeTeXglyphindex␣"#1"\relax\relax}% since
         XeTeX ... \numexpr is redundant.
}
{\def\textbullet{\ensuremath{\bullet}}}
```

```
\def\if@XeTeX␣{\@ifXeTeX␣{\iftrue}{\iffalse}}
```

#### Expandable turning stuff all into 'other'

A shorthand. Note that it takes an undelimited argument not requires ⟨*balanced text*⟩.

```
\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
```
\long\def\all@other#1{\@xa\gmu@gobmacro\meaning#1}
```

The `\gmu@gobmacro` macro above is applied to gobble the `\meaning`'s beginning, `long␣macro:->` all 'other' that is.

\gmu@gobmacro
```
\edef\gmu@tempa{%
  \def\@nx\gmu@gobmacro##1\@xa\@gobble\string\macro:##2->{}}
\gmu@tempa
```

#### Show must go on

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,

```
\def\gmshowlists{%
  \tracingonline=1
  \showboxdepth=1␣\showboxbreadth=1000000␣\showlists}

\def\gmshowbox{%
  \tracingonline=1
  \showboxdepth=10000␣\showboxbreadth=10000␣\showbox}

\def\gmtracingoutput{%
  \tracingoutput\@ne
  \tracingonline=\@ne
  \showboxdepth=1
  \showboxbreadth=1000000
}
```

\ifgmu@debug@msgs
```
\newif\ifgmu@debug@msgs

\def\gmtron{%
  \tracingonline=\@M
  \tracingmacros=\@M
  \tracingassigns=\@M
  \tracingcommands=\@m
  \gmu@debug@msgstrue
  \let\let\let
}

\def\gmtroff{%
  \tracingonline=\m@ne
  \tracingmacros\m@ne
  \tracingassigns=\m@ne
  \tracingcommands=\m@ne
```

```
  \tracingoutput=\m@ne
  \gmu@debug@msgsfalse
  \let\let\let
}
```

\nameshow
```
\newcommand\nameshow[1]{%
  \ifcsname␣#1\endcsname
    \@xa\show\csname#1\endcsname
  \else␣\show\@undefined
  \fi}
```

\nameshowthe
```
\newcommand\nameshowthe[1]{%
  \ifcsname␣#1\endcsname
    \@xa\showthe\csname#1\endcsname
  \else␣\showthe\@undefined
  \fi}
```

Note that to get proper `\showthe\my@dimen14` in the 'other' `@`'s scope you write `\nameshowthe{my@dimen}14`.

\namemeaning
```
\newcommand\namemeaning[1]{%
  \ifcsname␣#1\endcsname
    \@xa\typeout{\@xa\meaning\csname#1\endcsname}%
    \show\relax
  \else␣\typeout{\meaning\@undefined}\show\relax
  \fi}
```

Note that to get proper `\showthe\my@dimen14` in the 'other' `@`'s scope you write `\nameshowthe{my@dimen}14`.

#### Second class document class

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.

```
\def\secondclass{%
```
\ifSecondClass
```
  \newif\ifSecondClass
  \SecondClasstrue
  \@fileswithoptions\@clsextension}% [outeroff,gmeometric]{gmdocc}
```
it's loading gmdocc.cls with all the bells and whistles except the error message.

```
\AtBeginDocument{%
  \unless\ifdefined\@parindent
```
\@parindent
```
    \newskip\@parindent
    \@parindent=\parindent
  \fi
}
```

`\def\balsmiley#1␣{}%` to balance parentheses and brackets in smileys. ;-) `\balsmiley(␣ % ;-)`.

```
\long\def\scantnoline#1{% 'rescan tokens without adding line at the end'
  {\endlinechar\m@ne\scantokens{#1}}}
```

`\pdef\getprevdepth{%` to pass last depth through a group (e.g. `\end{envir.}`)

```
  \endgraf
```

```
  \xdef\setprevdepth{\prevdepth=\the\prevdepth\relax}%
}

\pdef\getprevdepthlocal{%
  \endgraf
  \edef\setprevdepth{\prevdepth=\the\prevdepth\relax}%
}
```

### Storing the catcode of line end

```
\def\StoreCatM{%
  \protected\edef\RestoreCatM{%
    \catcode`\@nx\^^M=\the\catcode`\^^M\relax}%
}

\pdef\RestoreCatM{\PackageE{gmutils}{first␣store␣the␣catcode␣
      of
    ^\empty^\empty␣M␣with␣\string\StoreCatM.}%
}
```

### `\resizegraphics`

```
\RequirePackage{graphicx}

\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
```

Paragraph with last or first line centered: `\lastcentered` declaration and `lastcentered` environment

```
\def\lastcentered{%
  \lastlinefit\z@
  \parindentosp\relax
  \leftskip\dimexpr1\leftskip\relax␣plus␣1fil\relax
```

```
  \rightskip\dimexpr1\rightskip\relax␣plus␣-1fil\relax
  \parfillskipposp␣plus␣2fil\relax
}

\def\endlastcentered{\par\@endpetrue}

\def\firstcentered{%
  \lastlinefit\z@
  \parfillskip␣osp\relax
  \rightskipposp␣plus␣1fil\relax
  \leftskipposp␣plus␣-1fil\relax
  \parindent␣osp
  \addtotoks\everypar{\hskipposp␣plus␣2fil\relax}%
}

\let\endfirstcentered\endlastcentered

\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}%
}

\def\@xa@three#1#2{% reverses expansion of three tokens, two given as arguments. (third may be {)
  \@xa\@xa\@xa␣\@xa\@xa\@xa␣\@xa␣#1%
  \@xa\@xa\@xa␣#2%
  \@xa␣}

\def\@xa@four#1#2#3{% reverses expansion of four tokens, three given as arguments. (fourth may be {)
  \@xa\@xa\@xa\@xa␣\@xa\@xa␣\@xa
  \@xa\@xa\@xa\@xa␣\@xa\@xa\@xa\@xa␣#1%
  \@xa\@xa\@xa\@xa␣\@xa\@xa␣\@xa␣#2
  \@xa\@xa\@xa␣#3%
  \@xa␣}
```

#### Comparison of detokenised strings

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`.

```
\long\pdef\@ifenvir#1{%
    % #1 enquired environment name which will be confronted with \@currenvir
    % #2 what if true (if the names are equivalent[1])
    % #3 what if false
  \gmu@ifedetokens{\@currenvir}{#1}%
}
```

1 The names are checked whether they produce the same `\csname`. They don't have to have the same catcodes.

```
\pdef\@ifjobname#1{\gmu@ifedetokens{\jobname}{#1}}
```

(2010/09/23, v0.993:) since `\gmu@ifedetokens` turned to be expandable, we make this macro `\protected`

2010/09/23 introduced as a common part of the three then four then… of the below.

```
\long\def\gmu@ifstrcmp
#1% wrapper for left side of comparison
#2% wrapper for right side of comparison
#3% left side of comparison (list of tokens)
#4% right —"—
{%
  \ifnum\strcmp{#1{#3}}{#2{#4}}=\z@
    \@xa\@firstoftwo
  \else
    \@xa\@secondoftwo
  \fi
}
```

```
\def\gmu@ifdetokens
```

test if two list of tokens agree when decategorised (without expansion)
implicit `#1`: left tokens
implicit `#2`: right tokens
(implicit `#3`: if agree)
(implicit `#4`: if disagree

```
{%
  \gmu@ifstrcmp\detokenize\detokenize
}% of \gmu@ifdetokens
```

```
\def\gmu@ifutokens
```

test if two list of tokens agree when decategorised (without expansion)

```
{%
  \gmu@ifstrcmp\unexpanded\unexpanded
}
```

```
\def\gmu@ifedetokens{%
```

basically a wrapper for the basic IMHO use of `\strcmp`. Won't work the same in some extremely perverse cases I can imagine. But with `\@firstofone`s won't work either, only another way.

```
  \gmu@ifstrcmp␣\@firstofone␣\@firstofone
}
```

And the `\protected` versions of those macros.

```
\@XA{\pdef\gmu@pifdetokens}\@xa{\gmu@ifdetokens}
```

```
\@XA{\pdef\gmu@pifutokens}\@xa{\gmu@ifutokens}
```

```
\@XA{\pdef\gmu@pifedetokens}\@xa{\gmu@ifedetokens}
```

(2010/09/23, v0.993:) redefined deeply and made expandable thanks to `\strcmp`. Also renamed from `\gmu@ifedetokens`(the `\gmu` prefix added)

```
%% \lpdef\@ifedetokens#1#2{%
%%   %
```

```
%%   % #1 first list of tokens to be expanded and detokenized
%%   % #2 second list
%%   % #3 if agree
%%   % #4 else
%%   %
%%   \edef\gmu@edetoka{#1}% to get #1 fully expanded.
%%     \edef\gmu@edetokb{\@xa\detokenize\@xa{\gmu@edetoka}}% with
our
%%   % brave new \begin, \@currenvir is fully expanded,
%%   % remember?
%%   \edef\gmu@edetoka{#2}% to get #2 fully expanded.
%%   \edef\gmu@edetokc{\@xa\detokenize\@xa{\gmu@edetoka}}%
%%   \ifx\gmu@edetokb\gmu@edetokc\@xa\@firstoftwo
%%   \else\@xa\@secondoftwo
%%   \fi
%% }
```

#### Hashes for meta-defining macros

In this section we use an expandable loop described in The ε-TEX Manual p. 9. In the gmampulex package we construct a more general definer for such loops.

```
\def\gmu@HHashes#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@HHashes
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}% of \gmu@HHashes.
```

```
\def\gmu@Hashes#1#2{% this is a fully expandable loop analogous to that of The
                       ε-TEX Manual p. 9. expanding in an \edef to ####1...####⟨h.$2-1$⟩ (quadru-
                       ple hashes' sequence)
  \ifnum#1<#2␣%
  ########\number#1
  \expandafter\gmu@HHashes
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}% of \gmu@hashes.
```

```
\def\gmu@hashes#1#2{% this is a loop analogous to that of The ε-TEX Manual p. 9.,
                       that expands to a sequence of double hashes ⟨#1⟩–⟨#2 − 1⟩, useful in edefing
                       a definition of macros.
  \ifnum#1<#2%
  ####\number#1
  \expandafter\gmu@hashes
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi
}% of \gmu@hashes.
```

```
\def\gmu@HHashesbraced#1#2{%
  \ifnum#1<#2%
  {################\number#1}%
```

```
  \expandafter\gmu@HHashesbraced
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}% of \gmu@hashesbraced.

\def\gmu@Hashesbraced#1#2{%
  \ifnum#1<#2%
  {########\number#1}%
  \expandafter\gmu@HHashesbraced
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}% of \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.

\def\gmu@hashesOut#1#2{%
  \ifnum#1<#2%
  \space\space\space\space
  »\@nx\unexpanded{####\number#1}«%
  \expandafter\gmu@hashesOut
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi
}% of \gmu@hashesbraced.

\def\gmu@hashesOutU#1#2{%
  \ifnum#1<#2%
  \space\space\space\space
  »\@nx\unexpanded{####\number#1}«%
  \expandafter\gmu@hashesOut
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi
}% of \gmu@hashesbraced.

\@tempcnta=1
\@whilenum\@tempcnta<11\do{% 2010/4/15
  \@nameedef{gmu@hashes@\the\numexpr\@tempcnta-1\relax}%
  {\gmu@hashes1\@tempcnta}%
  \@nameedef{gmu@Hashes@\the\numexpr\@tempcnta-1\relax}%
  {\gmu@Hashes1\@tempcnta}%
  \@nameedef{gmu@HHashes@\the\numexpr\@tempcnta-1\relax}%
  {\gmu@HHashes1\@tempcnta}%
  \@nameedef{gmu@hashesbraced@\the\numexpr\@tempcnta-1\relax}%
  {\gmu@hashesbraced1\@tempcnta}%
  \@nameedef{gmu@Hashesbraced@\the\numexpr\@tempcnta-1\relax}%
  {\gmu@Hashesbraced1\@tempcnta}%
  \@nameedef{gmu@HHashesbraced@\the\numexpr\@tempcnta-1%
      \relax}%
```

```
  {\gmu@HHashesbraced1\@tempcnta}%
  \edef\gmu@eloops@resa{%
    \long\def\@xanxcs{%
      gmu@TOhashes@\romannumeral\numexpr\@tempcnta-1\relax}%
      \gmu@hashes1\@tempcnta{%
        \@nx\TypeOut{%
          \gmu@hashesOut␣%
          1␣\@tempcnta}%
      }% of \gmu@TOhashes@viii etc.

    }% of temporary macro

    \gmu@eloops@resa
```

Generates nine pairs of macros `\gmu@TOhashes@i`, `\gmu@TOUhashes@i`, `\gmu@TOhashes@ii`, `\gmu@TOUhashes@ii` etc. that print (type out) their arguments on the terminal, unexpanded

```
    \advance\@tempcnta\@ne
}% of \@whilenum
```

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.)

```
\foone{\catcode`\␣\active}%
\gmobeyspaces {\newcommand*\gmobeyspaces{\let␣\␣\catcode`\␣\active}}
```

And a macro to forbid hyphenation of the next word:

```
\nohy  \newcommand*\nohy{\leavevmode\kernosp\relax}
\yeshy \newcommand*\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'.

```
\long\pdef\gmu@EdefCurrnames#1{%
  \xiiEdetoke\gmu@EdefCurrnames@resa{#1}%
  \@xa␣␣\gmu@EdefCurrnames@␣\gmu@EdefCurrnames@resa.tex.\@nil
```

if no extension is present, `tex` is assumed. This couple of macros won't work well for files with dots in their names.

```
}

\pdef\gmu@EdefCurrnames@␣#1.#2.#3\@nil{%
  \def\@currname{#1}%
  \def\@currext{#2}%
}

\def\NamedInput@prepare#1{% we wrap in a macro to use also in \DocInput
  \@pushfilename
  \gmu@EdefCurrnames␣{#1}%
}

\let\NamedInput@finish=\@popfilename

\pdef\NamedInput#1{% useful e.g. in error handling
```

```
  \NamedInput@prepare␣{#1}%
  \@@input␣#1\relax
  \NamedInput@finish
}

\def\gmu@ifdim
#1% dimen specification
#2% comparison
#3% dimen specification
{%
  \ifnumo\ifx#2≤1\fi\ifx#2≥1\fi\ifx#2≠1\fi=\@ne
    \afterfi\unless
  \fi
  \ifdim#1\ifx#2≤>\fi\ifx#2≥<\fi\ifx#2≠=\fi
    \ifx#2>>\fi\ifx#2<<\fi\ifx#2==\fi
    \dimexpr(#3)*1\relax% parentheses are for closing all possible ε-TEXpressions
        not to gobble that \relax by them but only by the outermost \dim|
        expr to avoid premature expansion of the following \expandafter.
        (2010/6/14)
    \@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi
}

\def\gmu@ifskip
#1% glue specification
#2% comparison for natural part
#3% comparison for stretch part
#4% comparison for shrink part
#5% glue specification
{\ifnum
    o\gmu@ifdim{1\glueexpr#1}#2{1\glueexpr#5}10%
    \gmu@ifdim{\gluestretch\glueexpr#1}#3{\gluestretch%
        \glueexpr#5}10%
    \gmu@ifdim{\glueshrink\glueexpr#1}#4{\glueshrink%
        \glueexpr#5}10=111
    \@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi
}

\def\gmu@ifbox
```

We provide an expandable macro for comparing boxes' dimensions. We handle the registers' numbers not any boxes just to allow expandability.

```
#1% a box register number (e.g. \copy\z@)
#2% comparison for heights
#3% comparison for depths
#4% comparison for widths
#5% a box register number
{%
  \gmu@ifskip
  {\glueexpr␣(\ht#1␣plus␣\dp#1␣minus\wd#1␣)*1\relax}%
  #2#3#4%
```

```
  {\glueexpr (\ht#5 plus \dp#5 minus\wd#5 )*1\relax}%
}
```

```
\def\greater@dim#1#2{%
  \ifdim\dimexpr#1>\dimexpr(#2)*1\relax
    #1%
  \else #2%
  \fi
}
```

Removal of an element from a comma-separated list macro (such as used in the LaTeX's `\@for` loops). The removed element becomes macro-meaning of `#3`.

```
\long\pdef\gmu@removeelement
#1% element to be removed
#2% macro carrying a comma-separated list
#3% CS to carry removed element.
{%
  \let#3\@undefined%
  \def\gmu@removeelement@resa##1,#1,##2\@nil{%
    \gmu@ifempty{##2}%
    {\edefU#2{##1}}%
    {\edefU#2{##1,##2}%
      \def#3{#1}%
      \@xa\gmu@removeelement@resa#2\@nil
    }%
```

If `##2` is not empty, then we know `#1` was in the list so we have to remove its copy from the end of the list.

```
  }% of \gmu@removeelement@resa
  \@xa\gmu@removeelement@resa\@xa,#2,#1,\@nil
```

We gobble the beginning comma

```
  \@xa\@xa\@xa\ifx\@xa\@firstofmany#2\relax\@nil,%
    \edef#2{\unexpanded
      \@xa\@xa\@xa{\@xa\@gobble#2}}%
  \fi
}
```

A macro for edefs of csnames: expands to a csname hit by `\noexpand`.

```
\long\def\@nxcsn#1{%
  \@xa\@nx\csname #1\endcsname}
```

```
\def\@listbegvskipping{%
  \@topsepadd=\topsep
  \ifvmode
    \advance\@topsepadd by\partopsep
  \fi
  \par
  \addvspace\@topsepadd
}
```

Remember that proper vskips at the end of an environment will be put by `\@endparenv`

```
\def\foolc#1#2{%
```

```
  \begingroup\lccode`#1=`#2\relax
  \lcfirstofone
}

\long\def\lcfirstofone#1{%
  \lowercase{\endgroup#1}%
}
```

Just to make it `\long`

```
\long\def\PackageWarning#1#2{%
  \GenericWarning{%
    (#1)\@spaces\@spaces\@spaces\@spaces
  }{%
    Package␣#1␣Warning:␣#2%
  }%
}
```

long version of `\typeout` (`\par` occurs quite often)

```
\long\def\TypeOut#1{%
  \edef\TO@resa{#1}%
  \edef\TO@resa{\@xa\detokenize\@xa{\TO@resa}}%
  \typeout{\TO@resa}}

\long\def\ShowOut#1{%
  \TypeOut{#1}%
  \show\TO@resa
}
```

A definition that makes its `#1` a stringed version of itself if in `\csname`...`\endcsname`

```
\long\pdef\incsdef
#1% a CS or active char
#2% parameters string
#3% definition's body
{%
  \def#1#2{%
  %%   \ifincsname \@xa\@firstoftwo \else \@xa\@secondoftwo \fi
    \gmu@if␣{incsname}{}%
    {\string#1}{#3}%
  }%
}
```

#### Deducing whether hash was braced

Since not built-in TEX's test can distinguish {$_1$ from `\bgroup`, there seemed not to be a way to deduce whether the stuff we are passed as an argument was braced or not.

In general it still seems so to me, but in the case of undelimited arguments a partial solution seems to exist: count the tokens and assume they were braced if they are they and don't bother if they are just one.

In fact, this partial solution seems quite satisfactory to avoid bracing single tokens when passing them further as arguments.

```
\gmu@DefSymbol\gmu@CountTokens@end
\newcount\c@gmu@TokensCount
```

`\c@gmu@TokensCount`

```
\lpdef\gmu@CountTokens
```

`%%  #1%` upper bound? No.

Imposing numeric upper bound doesn't make sense since anyway we have to `\let` each token to balance the braces and/or not to bother with possibly unbalanced ifs:

If we'd stop at reaching the bound, we could throw the tail of tokens to nonexistence by putting a macro with a parameter delimited with the counting's sentinel. But that wouldn't work for unbalanced braces. On the other hand, wrapping the tail in an `\iffalse`, would release us from bothering with unbalanced braces, but in such case unbalanced ifs could occur so both ways are not satisfactory.

Note moreover that $\{_1$ is indistinguishable from `\bgroup` so we can't count braces' nesting to put as many as needed.

Counting groups and opening a junk box with this many groups is absurd because would lead to execution of all those tokens and that's what we cannot allow. BTW it'd be prone to unbalanced ifs, too.

```
#1% the tokens to be counted.
{%
  \c@gmu@TokensCount=\m@ne
  \let\gmu@CountToken@token\@undefined
  \gmu@CountToken@iter
  #1\gmu@CountTokens@end
}
\def\gmu@CountToken@iter{%
  \gmu@if␣x{\gmu@CountToken@token\gmu@CountTokens@end}%
  {}% we've reached the end of iteration
  {%
```

We increase the counter and throw the iterator after next assignment

```
    \advance\c@gmu@TokensCount\@ne
    \afterassignment\gmu@CountToken@iter
    \let\gmu@CountToken@token=
  }%
}%
```

Now we are ready to define a brace-wrapper:

```
\long\def\gmu@passbraced
```

This macro checks whether its `#2` was a balanced text (in explicit braces) or a `\bgroup` token. Well, actually it checks whether `#2` consists of not one token (0 or $> 1$) and if so, wraps it in braces before putting it right next to `#1`.

```
#1% the stuff to be put before #2
#2% the stuff we check and pass unbraced if single or braced otherwise
{%
  \gmu@CountTokens{#2}%
  \gmu@if␣{num}{\c@gmu@TokensCount=\@ne}%
```

If we have only one token, we have to check whether it's space or not: the 'blank space' token could never become a hash without braces.

```
  {\gmu@if␣x{#2␣}%
    \@secondoftwo\@firstoftwo}%
```

Otherwise (not one token, incl. the possibility of 0 tokens)

```
  \@secondoftwo
```

```
  {#1#2}{#1{{#2}}}%
}% of \gmu@passbraced Note that this works also for #2 being empty: it will be
    considered not single and passed in braces.

\long\def\MeaningOrUnex#1{%
  \gmu@if␣{singletoken}{{#1}}%
  {\meaning#1}{\unexpanded{#1}}%
}

\pldef\@iwru#1{%
```

as simple as possible not to put much output on the terminal if tracing is on.

```
  \immediate␣\write␣\@unused␣{l.\the\inputlineno:\space␣#1}%
}

\pldef\@iwruJ{\@iwru{^^J}}

\pldef\@iwrum#1{%
  \@iwru{»\unexpanded{#1}«␣is␣»\MeaningOrUnex{#1}«}%
}

\pldef\@iwruif#1{%
  \gmu@if␣{gmu@debug@msgs}{}
  {\@iwru{#1}}{}%
}

\long\pdef\IgnInfo
#1% package name
#2% description
#3% stuff we announce as ignored
{%
  \PackageInfo{#1}{Item␣»\unexpanded{#3}«␣(\MeaningOrUnex{%
       #3})␣ignored^^J%
      #2}%
}

\pldef\@iwma{%
```

(Added 2010/10/19)
Note it takes a *text* not an argument.

```
  \immediate␣\write␣\@mainaux
}

\def\stepnummacro
#1% a macro that expands to some numerical stuff
#2%
{\edef#1{\the\numexpr␣#1+#2}}
```

An expandable macro that expands to `\numexpr` containing conversion of given sequence of switches to a binary number, presented in its Horner's schema.

\boolstobin
```
\def\boolstobin
#1% a sequence of Boolean switches' names without »if«
{%
  \numexpr␣\boolstobin@iter␣0␣#1␣{}\gmu@delim␣%
}

\def\boolstobin@iter
```

```
#1% expression so far
#2% the name of current switch (without »if«)
#3% tail of switches
\gmu@delim
{%
  \gmu@ifempty{#3}%
  {#1\relax}% \relax to close the num expression
  {\boolstobin@iter
    {(#1)*2+\csname␣if#2\endcsname␣1\else␣0\fi}%
    #3\gmu@delim
  }%
}
```

```
\long\def\condstobin
#1% a sequence of conditionals' names without »if« followed by the condition
{%
  \numexpr␣\condstobin@iter␣0␣#1␣{}{}\gmu@delim␣%
}
```

```
\long\def\condstobin@iter
#1% expression so far
#2% the name of current conditional (without »if«)
#3% the condition for #2
#4% tail of switches
\gmu@delim
{%
  \gmu@ifempty{#3}%
  {#1\relax}% \relax to close the num expression
  {\condstobin@iter
    {(#1)*2+\csname␣if#2\endcsname␣#31\else␣0\fi}%
    #4\gmu@delim
  }%
}
```

```
\long\def\gmu@ifQUANT@iter
```

(it's left-to-right and lazy)

```
#1% the (binary) value that terminates calculation: 0 for AND and 1 for OR (it has to
      be a single token due to \expandafter in line 4456)
#2% the (binary) value so far;
#3% the name of current conditional (without »if«)
#4% the condition for #2
#5% tail of condition(al)s
\gmu@delim
{%
  \gmu@ifempty{#5}%
```

if #5 is empty, we expand to the most recent (previous) value.

```
  {#2}%
  {% or else we check whether #2 is terminating
    \gmu@if␣{num}{#2=#1␣}% with a space and expand to it and finish calcula-
          tion if so…
    {#1}%
```

or iterate. This case happens where the value so far is `#1`, so the future of the conjunction doesn't depend on it.

```
      {\@xa\gmu@ifQUANT@iter␣\@xa␣#1%
        \the\numexpr␣%
```

`%% 1 * %` `\numexpr` is used here to get the full expansion in one step. No need of superposing with identity.

```
        (\gmu@if␣{#3}{#4␣}␣{1}␣{0})␣+\z@\relax
        #5\gmu@delim
      }%
    }%
}% of \gmu@ifQUANT@iter.
```

Disjunction of conditions (finite Existential Quantifier). First as a pseudo-conditional. It expands to an open `\ if` but that's OK if we use it only inside macros or with `\gmu@if`.

```
\long\def␣\ifAnyClause
#1{%
    \ifnum
        \gmu@ifQUANT@iter
        1% for the Existential Quantifier 1 terminates calculation (an(y) example has
              just been found)
        0% To make any calculation sense we assume 0 at the beginning (i.e., "we
              haven't found an example yet")
        #1
        {false}{}␣{false}␣{}% the sentinel(s)
        \gmu@delim␣% the delimiter
        =\@ne␣% right side of \ ifnum
}
```

```
\long\def\gmu@OR
#1% as above
```

```
%% #2% (impl.) True branch
%% #3% (impl.) False branch
{%
    \ifAnyClause␣{#1}%
        \@xa\@firstoftwo
    \else
        \@xa\@secondoftwo
    \fi
}
```

Now the dual, i.e., conjunction of conditions. First as a pseudo-conditional (as above)

```
\long\def\ifAllClauses
#1% a sequence of pairs {⟨conditionals' name without »if«⟩}{⟨the condition⟩}. (Don't
      delimit the conditions, it's been taken care of!) (it has to be braced).
{%
  \ifnum
      \gmu@ifQUANT@iter
      0% for the General Quantifier 0 terminates calculation (a counter-example has
            just been found)
```

```
      1% To make any calculation sense we assume 1 at the beginning
      #1
      {true}{}␣{true}␣{}% the sentinel(s)
      \gmu@delim␣% the delimiter
      =\@ne␣% right side of \ ifnum
}
\long\def\gmu@AND
#1%
{%
   \ifAllClauses␣{#1}%
      \@xa\@firstoftwo
   \else
      \@xa\@secondoftwo
   \fi
}
```

A shorthand: as `\Name` but with the second parameter delimited with a space (for less tokens)

```
\long\def\sName␣␣␣␣#1#2␣␣␣␣{\@xa#1\csname␣#2\endcsname}
```

and a version that detokenises its `#2`

```
\long\def\sdName␣␣␣␣#1#2␣␣␣␣{\@xa#1\csname␣\detokenize{#2}%
    \endcsname}
\long\def\dName␣#1#2{\@xa␣#1\csname␣\detokenize{#2}\endcsname}
\long\def\@sN␣␣␣␣#1␣␣␣␣{\csname␣#1\endcsname}
\long\def\@sdN␣␣␣␣#1␣␣␣␣{\csname␣\detokenize{#1}\endcsname}
\long\def\gmu@extreme
```

Note it's expandable and expands to the smallest (for `#2` being `<`) or largest (for `#2` being `>`) number/dimen of `#3` and `#4`. May be used recursively (tree-way).

```
#1% kind of test (num or dim)
#2% inequality sign: < for minimum, > for maximum.
#3% left side of comparison
#4% right side of comparison
```

But *in fact* this macro eats a sequence of numexprs or dimexprs that must be terminated by `\relax` (or sth. `\ifx`-equal) and expands to the extreme value of that sequence.

```
{%
  \gmu@if␣{x\@xa\@xa}{\@firstofmany#4\@nil\relax}␣% this complicated
       test is to allow arguments beginning with some \if⟨...⟩ as in \possvfil
       e.g.
  {#3}%
  {%
    \gmu@if␣{#1}␣{\csname␣#1expr\endcsname#3\relax
      #2\csname␣#1expr\endcsname␣#4\relax␣}
    {\gmu@extreme␣{#1}#2{#3}}%
    {\gmu@extreme␣{#1}#2{#4}}%
  }%
}% of \gmu@extreme
```

Two expandable macros that expect a sequence of undelimited num(expr)s terminated with `\relax` and expand to the biggest or the smallest one.

```
\def\gmu@maxnum{\gmu@extreme␣{num}>}
\def\gmu@minnum{\gmu@extreme␣{num}<}
```

And the same for dim(expr)s:

```
\def\gmu@maxdim{\gmu@extreme␣{dim}>}
\def\gmu@mindim{\gmu@extreme␣{dim}<}
```

And if we wish to assign the larger/smaller value in an iteration:

```
\pdef\gmu@fitto
#1% scope (nothing, \relax or \global.
#2% left side of the assignment (must be a dimen able to be "passive")
#3% the comparison (if #2#3#4 then reassign #2)
#4% right side of the assignment—any correct text for \dimexpr.
{%
  \gmu@if␣{dim}{#2#3\dimexpr␣(#4)+\z@\relax\relax}%
  {#1#2=\dimexpr␣(#4)+\z@\relax}%
  {}%
}
```

```
\pdef\gmu@g@enlargeto
#1% #2 of the above
#2% #4 of the above
{\gmu@fitto\global{#1}<{#2}}
```

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\kern\z@
  \discretionary{#1}{#2}{#3}\penalty\@M\hskip\z@skip}
```

\discret
```
\pdef\discret#1{\discre{#1}{#1}{#1}}
```

A discretionary hyphen that allows other (automatic) break-points in the word.

```
\pdef\gmu@flexhyphen{%
  \discre{% before break
    \ifnum\hyphenchar\font>\z@
      \char\hyphenchar\font
    \fi
  }% end of before break
  {}% after break
  {}% without break
}
```

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
}
\lpdef\hboxreflected#1{%
  \hbox{%
    \reflectbox{#1}%
  }%
}
\pdef\gmu@ifSystemX{% the 'If file exists" test is NOT expandable since it involves opening some streams. Therefore we define this macro as protected.
  \IfFileExists{/etc/passwd}%
}
\def\hrule@zero{\hrule␣height\z@␣width\z@␣depth\z@}
\def\vrule@zero{\vrule␣height\z@␣width\z@␣depth\z@}
\def\do#1#2{% (2010/10/11) Define \gmu@iflast⟨sth.⟩ as a two-argument expandable macro-conditional.
  \Name␣␣\def{gmu@iflast#1}{%
    \ifnum␣#2=\lastnodetype
      \@xa\@firstoftwo
    \else
      \@xa\@secondoftwo
    \fi
  }%
}
\do␣{glue}{11}␣␣␣␣\do{skip}{11}
\do{kern}{12}
\do␣{penalty}{13}
\def\gmu@dimratio
#1% numerator dim(en/expr)
#2% denominator dim(en/expr)
{\strip@pt
  \dimexpr␣␣1pt␣*
  \numexpr\dimexpr␣(#1)*1\relax\relax␣/
  \numexpr␣\dimexpr␣(#2)*1\relax␣\relax
  \relax
}
```

2010/10/21

```
\def\MakeFalseIfDefined␣#1{%
  \ifcsname␣if#1\endcsname
    \csn{#1false}%
  \fi
}
\def\MakeTrueIfDefined␣#1{%
  \ifcsname␣if#1\endcsname
    \csn{#1true}%
  \fi
}
```

2010/10/28

```
\def\gmu@pageremain{\dimexpr
```

```
  \ifdim\pagegoal=\maxdimen␣\textheight␣\else␣\pagegoal␣\fi
  -\pagetotal
  \relax
}
```

#### Absolute values of dimens & nums

Strictly for use in the "absoluters"

```
\def\gmu@GobbleMinus␣#1{\if-#1\else#1\fi}
```

```
\def\gmu@absExpr
#1% ε-TEX's expression primitive, num or dim so far (2010/12/17, 11.46)
#2% expression of respective kind
{%
  \@xa\gmu@GobbleMinus\the␣#1#2\endexpr
}
```

```
\def\gmu@absdim␣{\gmu@absExpr␣\dimexpr␣}
\def\gmu@absnum␣{\gmu@absExpr␣\numexpr␣}
```

2010/10/29

```
\pdef\gmu@SetPagegoal␣#1{%
```

This macro is LATEX-specific in the sense that it assumes resetting of `\pagegoal` by `\output` at the beginning of a new page and excludes that case.

```
  \gmu@if␣{dim}␣{\pagegoal=\maxdimen}%
  {}% in this case we do nothing, as explained above
  {\pagegoal␣=␣\dimexpr␣(#1)\relax}%
}
```

2010/11/10

```
\pdef\gmu@SetPagegoalGlobal␣#1{%
```

This macro is LATEX-specific in the sense that it assumes resetting of `\pagegoal` by `\output` at the beginning of a new page and excludes that case.

```
  \gmu@if␣{dim}␣{\pagegoal=\maxdimen}%
  {}% in this case we do nothing, as explained above
  {\global\pagegoal␣=␣\dimexpr␣(#1)\relax}%
}
```

```
\pdef\gmu@AdvancePagegoal␣#1{% the arg. should be a proper stuff for a sub-
      dimexpr (i.e., no spurious \relaxes are allowed).
```

```
  \gmu@SetPagegoal␣{\pagegoal␣+#1}%
}
```

```
\def\pagegoalortextheight{% as in the name:
```

(Note that „it's alive, it's alive!!!" but behaves as a dimen (except left side of assignments)). In particular, it's a subject to `\the` and thus can fully expand to a "dead" dimen spec. in one step.

```
  \dimexpr
    \ifdim␣\pagegoal=\maxdimen
      \textheight
    \else␣\pagegoal
```

```
    \fi
  \relax
}

\def\gmu@totalht␣#1{% the arg. should be a box register.
  \dimexpr␣\ht␣#1+\dp#1\relax
}

\def\vbadness@M␣{%
   \unless\ifnum\vbadness=\@M
      \edef\gmu@VeryBadBadness␣{\the\vbadness␣}%
      \vbadness\@M
   \fi
}

\def\vbadness@Restore␣{%
   \ifdefined␣\gmu@VeryBadBadness
      \vbadness␣\gmu@VeryBadBadness␣\relax
   \else
      \PackageError{gmbase}{You␣try␣to␣gm-restore␣\vbadness%
           \space␣where
         it␣is␣not␣gm-stored}{}%
   \fi
}
```

For proper indentation of ε-TEX's expressions let's introduce an alias of `\relax`

```
\relaxen\endexpr
```

2010/11/22 the same for TEX's (primitive) rules:

```
\relaxen\endrule
```

2010/11/22

```
\def\gmu@ifpageodd␣{%
```

`%% #1%` (implicit) what if page number is odd
`%% #2%` (implicit) what if page number is even

```
  \ifodd␣\c@page
    \@xa\@firstoftwo
  \else
    \@xa\@secondoftwo
  \fi
}
```

2010/12/08, 15.14

```
\gmu@DefSymbol\gmu@Fake

\def\gmu@FakeLoaded
#1% extension
#2% name
{\ifcsname␣␣ver@#2.#1\endcsname
   \@xa\ifx\csname␣ver@#2.#1\endcsname␣\relax
     \@xa\@xa\@xa␣\@firstofone
   \else
      \@xa\@xa\@xa␣\@gobble
   \fi
```

```
  \else
    \@xa\@firstofone
  \fi
  {\Name\def␣{ver@#2.#1}{\gmu@Fake␣}}%
}
\def\gmu@FakeUnloaded
#1% extension
#2% name
{\ifcsname␣␣ver@#2.#1\endcsname
    \@xa\ifx\csname␣ver@#2.#1\endcsname\gmu@Fake
      \Name\let␣{ver@#2.#1}\@undefined
    \fi
  \fi
}
```

And an immediate use of the above. Because the polski package recatcodes a bunch of chars that are Polish diacritics in some ancient TEX encoding but in Unicode include the guillemots &al. that should rather remain "other".

```
\pdef\LoadPackagePolski␣{%
  \gmu@FakeLoaded␣\@pkgextension␣{inputenc}%
  \RequirePackage␣{polski}%
  \gmu@FakeUnloaded␣\@pkgextension␣{inputenc}%
}
\def\hrule@zero{\hrule␣height\z@␣width\z@␣depth\z@}
```

```
⟨/ base⟩
⟨⋆utils⟩
```

### The (gmutils package) options

quiet
```
\DeclareOptionX{quiet}{\gmu@quiettrue
  \PassOptionsToPackage{quiet}{gmtypos}%
}
```

The packages of this bundle may be loaded as options of the gmutils package. Here is how we provide it.

We define a requirer:

```
\def\gmu@PackOptionX
#1% name of a package with or without leading "gm".
{%
```

So we declare an OptionX that by default loads this package thanks to a special CS having been defined to load it or do nothing.

#1
```
  \DeclareOptionX{#1}[on]{%
    %%   \ifcsname gmu@Require@#1\endcsname
    %%     \PackageError{gmutils}{Value clash for the ***#1*** package option}{}%
    %%   \fi
    \lowercase{\@xa\if\@gobble␣##1\relax}% "off" given as the value
    \@namedef{gmu@Require@#1}{}%
    \else␣% "on"
```

```
4886      \afterfi{%
4887        \@namedef{gmu@Require@#1}{%
4888          \IfFileExists{gm#1.sty}%
4889          {\RequirePackage{gm#1}}% if there's a gm package, we load it, else
                we load
4891          {\RequirePackage{#1}}%
4892        }% of namedef
4893      }% of afterfi
4894      \fi
4895    }% of \DeclareOptionX
4896    \IfFileExists{gm#1.sty}%
gm#1 4897    {\DeclareOptionX{gm#1}[on]{%
4898        \ExecuteOptionsX{#1=####1}%
4899      }%
4900    }% of if yes. Else:
4901    {}%
4903 }
```

4905 ⟨/ utils⟩

### \DeclareCommand and \DeclareEnvironment—the gmcommand package[2]

4910 ⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{command}
4912 ⟨⋆command⟩

(2010/07/26, v0.993:) Incompatibilities with earlier versions: because of making all the specifiers "case-insensitive", i.e. aliasing the uppercases as lowercase, the `S` spec. ceased to be a **S**ingle-token-catcher and became a **S**tar-token-catcher.

Moreover, the parameters for *all* the one-parameter specifiers became optional (not only for the lowercase as earlier).

Moreover, catcher names of CS specifiers are now created by `\string`ing the CS and stripping its backslash (earlier: by crude detokenising it).

4929 `\RequirePackage{gmbase}␣%` we require the tricky low-level macros.

4931 `\unless\ifcsname␣ifgmu@quiet\endcsname`
4932 `\Name\newif␣{ifgmu@quiet}%` it has to be at least (at highest) in gmcommand since is used by it and not always entire gmutils is loaded.

(2010/09/06, v0.993:) moved here from gmutils.sty (a bug fix)

4937 `\fi`

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.

"mimetic" specifier (cf. Réné Girard, James Alison: »mimetic desire«;-) )).

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 tool for 'document' commands.

`\DeclareCommand`⟨*a CS*⟩⟨*prefs.*⟩{⟨*args' spec*⟩}{⟨*body, using #1…#2*⟩}

2 This file has version number v0.993 dated 2010/10/24.

The ⟨*prefs.*⟩ may be any (incl. empty) subset of

```
\global\protected\outer\long\relax!lL.qQiIwW\sphack
```

first for for effect analogous to that of prefixing `\def` of a macro, `\relax` for a tricky case when `\DeclareCommand` is used automatically, `!Ll` aliases for `\long`, `.qQ` to suppress placing of the diagnostic message in the definition `iIwW\sphack` to make the command Invisible (with `\@bsphack`—`\@esphack`).

(One more possible all-command prefix, `\envhack`, used to inform the machine the command will be used as an environment, is placed automatically when `\DeclareEn|vironment` is used. You may use it however at your own risk if you read the code of this package and want to hack it.)

In the ⟨*body*⟩ you use single `#`es as a refernce to the parameters specified in ⟨*args' spec*⟩, `##` (double hashes) as the parameters of macros defined by your command and so on.

The ⟨*args' spec*⟩ consists of the specifier tokens optionally preceded by a `>{%` ⟨*prefixex*⟩`}` sign and prefixes. Anything else is ignored, so you can write e.g., `#1`…`#9` for better readability if you like.

Before we go further with details, a word about general idea. `\DeclareCommand` defines a `\protected` macro named ⟨*a CS*⟩ that takes no arguments and expands to a bunch of *argument catchers*. The notion of an "argument catcher" seems crucial for understanding how does this machinery work. Each specifier, that is a possibly-prefixed-specifier-token, is translated to proper argument catcher in ⟨*a CS*⟩'es meaning. Then each of those catchers (postpones remaining list of catchers and) tries to catch the bunny, i.e., for the optional arguments performs respective `\@ifnext`… test and takes an argument if present and adds it to a dedicated toks (or adds the default value if suitable argument is absent (except for mandatory arguments that are scanned for anyway and whose absence results with an error)).

The ⟨*body*⟩ becomes body of a macro with undelimited parameters of the number corresponding to the number of non-ignored ⟨*args' spec*⟩'s specifiers.

Moreover, `\DeclareCommand` defines a macro carrying as its meaning the (parsed and simplified) specifiers' sequence to allow e.g. repeating it in another `\DeclareCom|mand` thanks to `\SameAs` special specifier.

The ⟨*prefixes*⟩ are:

— `P` or any of `p!lL\long\par` to make subsequent argument `\long` (some arguments are *always* `\long` which will be remarked);
— `\@xa`, `\expandafter` for one-level expansion of (the first token of) the first and all possible further specifier's argument(s) before actual declaring;
— `\@nx@xa`, `\@nxxa`, `\nxxa` to leave first specifier's argument intact and one-level expand the second;
— `\@xa@nx`, `\@xanx`, `\xanx` reverse of the above;
— `\@xaeacher`, `\xaeacher`, `\xaEacher`, `\xaE` for one-level expansion of the "eacher" token of the interating specifiers `\loop`, `U|u` and/or `W|w` (see the description of those specifiers);
— `\GroteNegere`, `\GrossIgnore`, `\GroteN`, `\GrossI` to ignore *all* the specifiers starting from this prefix and stopping at (any of) the following prefixes:
— `\GroteNegereStop`, `\GrossIgnoreStop`, `\GroteNStop`, `\GrossIStop`
— `\GroteLang`, `\GrossLong`, `\GroteL`, `\GrossL` to make all the specifiers following this prefix `\long` up to the (any of) following prefixes:
— `\GroteLangStop`, `\GrossLongStop`, `\GroteLStop`, `\GrossLStop`

The accepted argument specifiers are (case of the single letters doesn't matter):

— `m|M` for mandatory argument (braced or not) (undelimited macro parameter in the sense of Chapter 20 of *The TEX book*),

— (o|O|\NoValue)[⟨*default*⟩] for a LaTeX's optional argument (in square brackets) with default value ⟨*default*⟩ (which is passed as #⟨*n*⟩ if the argument is missing),
— (s|S|\NoValue)[⟨*default*⟩] for optional star of the catcode any of {11, 12, 13} (if is present, it becomes respective #⟨*n*⟩, or else \NoValue is at the #⟨*n*⟩ (unlike in xparse!).
— (t|T|\NoValue){⟨*list*⟩}[⟨*default*⟩] for a single **T**oken, 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.
Almost like in xparse, s is almost a shorthand for T{*}, but *unlike* in xparse, in the parsed arguments you get * or \NoValue as #⟨*n*⟩. You can now declare

```
\DeclareCommand\mimbla{T{+-}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 $n+k+1$.
— (ť|Ť|\NoValue){⟨*list*⟩}[⟨*default*⟩] as above only respecting blank spaces
— (q|Q|\NoValue){⟨*list*⟩}[⟨*default*⟩] as 'se**Q**uence', a sequence of symbols from ⟨*list*⟩; for the fundamental usage see line 7470. More clearly, the catcher specified with Q{⟨*tokens*⟩} catches a word over the alphabet ⟨*tokens*⟩ ∪ ␣ where ␣ is ignored and shall not be passed in the respective #⟨*n*⟩.
— (d|D|\NoValue)[⟨*default*⟩] the **D**ecimal (expansion of an integer) argument, equivalent to Q{+-0123456789}. If no ⟨*default*⟩ is given, 0 is assumed.
— (ď|Ď|\drs)[⟨*default*⟩] (d/D-haček): decimal integer respecting space (ď and Ď are made available only in XeTeX). Works as the above only doesn't ignore/gobble spaces.:

```
\DeclareCommand \foo{ D }{\romannumeral #1\relax}
\DeclareCommand \fóó{ Ď }{\romannumeral #1\relax}

:\foo 17 17 :\quad :\fóó 17 17 :
```

results in:

:mdccxvii: :xvii 17 :

— (c|C|\NoValue)[⟨*default*⟩] for an optional argument in parentheses. Historically it comes from the 'coordinate' argument and may serve as such: if you declare \DCcoordinate \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 \DCnocoordinate is executed that defines the c type argument as just another optional in parentheses.
— (b|B|\NoValue)[⟨*default*⟩] 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.

— `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 TEX book*. 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|G|\NoValue){`⟨*pair of tokens*⟩`}[`⟨*default*⟩`]` a **G**eneral catcher of an optional. For instance, to declare an optional in angles (used e.g. in Till Tantau's `beamer` class), declare `G{<>}`. Again, if the second argument is absent, `\NoValue` is assumed.

— `(a|A|\NoValue)[`⟨*default*⟩`]` (for 'angles'): a shorthand for `G{<>}`

— `=[{`⟨*assignment-stuff*⟩`}]` a pseudo-argument that in fact *interrupts* parsing arguments to execute an assignment to ⟨*assignment-stuff*⟩. The default ⟨*assignment-stuff*⟩ is `\@tempcnta`. For example,

```
\DeclareCommand\llf{=}{{\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, ⟨*assignment-stuff*⟩ 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.
I call it "pseudo-argument" because it doesn't add anything to the arguments' list (toks).
*This pseudo-argument type should be considered experimental.*

— `\loop␣`⟨*"direction"*⟩`␣{`⟨*list to match for/against iteration*⟩`}␣␣`
`[\drop](\any|\none){`⟨*list of dropped*⟩`}␣`
`[(\count|\ubound)][`⟨*decimal*⟩`]␣`
`[(\default|\Default)][{`⟨*default*⟩`}]␣`
`[(\eacher|\Eacher)][{`⟨*eacher*⟩`}]` for a catcher that iterates *while* (for ⟨*"direction"*⟩ being one of `\any`, `W`, `w`, `\W`, `\w`) or *until* (for ⟨*"direction"*⟩ being one of `\none`, `U`, `u`, `\U`, `\u`) it meets tokens listed on ⟨*list to match for/against iteration*⟩.
During this iteration the catcher drops the tokens listed on ⟨*list of dropped*⟩ if `\any` precedes that list or adds *only* those if the list is preceded with `\none`.
The number of iterations may be upper-bound with ⟨*decimal*⟩, therefore you may write `\count` or `\ubound` before it for readability.
⟨*default*⟩ is almost self-explaining: we have only explain when applies and what if absent. So, it applies when the parsed sequence of tokens/texts is empty—then ⟨*default*⟩ substutites that emptiness. By default, i.e., when you specify no ⟨*default*⟩, `\NoValue` is assumed.
⟨*eacher*⟩ is a token (or a list of tokens) added before each token/text caught. An example is given at the `u|U` specifier's description.

— `w|W` a shorthand for `\loop␣w`: a catcher iterating **W**hile.

— `u|U` a shorthand for `\loop␣u`: a catcher iterating **U**ntil. Let's

```
\def\EmergencyFont#1{%
   \ifcat a\noexpand#1%
     \iffontchar\font`#1
        #1%
     \else{\fontspec{FreeSerif}#1}%
     \fi
   \fi
}

\DeclareCommand\foo{
```

```
    u{\par\<\bgroup} \eacher{\EmergencyFont}
    w{\<} \count 1
  }{#1}
```

and then

```
\fontspec{HerculanumLTStd}␣\foo␣Pójdź,␣kińże␣tę␣chmurość␣w~głąb␣flaszy{}
```

(the Herculanum STD font hasn't Polish diacritics) results in

P Ó J D Ź K I Ń Ż E T Ę C H M U R O Ś Ć W G Ł Ą B F L A S Z Y

— `\lostwax{`⟨*list to match against iteration*⟩`}␣␣`
`[\drop](\any|\none){`⟨*list of dropped*⟩`}␣`
`[[(\count|\ubound)]`⟨*decimal*⟩`]␣`
`[(\default|\Default)][{`⟨*default*⟩`}]␣`
`[(\eacher|\Eacher)][{`⟨*eacher*⟩`}]␣`
`[\lost|\Lost]{`⟨*lost-list*⟩`}[[\count|\ubound]`⟨*decimal*⟩`]␣`
`[\endlost|\EndLost]` for a catcher that iterates *until* as the one specified with `u|U` and *loses* (drops) its delimiter(s) as specified in the `\lost` part of specification, i.e., the ones present on ⟨*lost-list*⟩, or *any* delimiter (cf. `\grab@lostwax`).

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 1337.

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 uses the `\gmu@ifxany` test that iterates token by token not hash by hash.

`\global\outer` If you wish to define your command `\globally`, 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': if you type `W` or `w` (for 'white') or `I` or `i` (for 'invisible') or, if you prefer a prefix-like, `\sphack`[3] between the CS to be defined and arguments spec, `\@bsphack` and `\@esphack` will be added in proper places.

`iIwW\sphack`

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 `\ArgumentCatcher@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 `\Declarecommand` with all other 'prefixes', `.` or `q` or `Q` for 'quiet'.

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

3 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.

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.

```
5444 \unless\ifdefined\@temptokenb
\@temptokenb 5445 \newtoks\@temptokenb
5446 \fi
```

```
\if@dc@alllong@ 5450 \newif\if@dc@alllong@␣%
```
for an option of all arguments long (it's stronger than `GroteLang` defined later (the latter is `\let` it).

```
\if@dc@quiet@ 5453 \newif\if@dc@quiet@␣%
```
for suppressing the message of using of declared command.

```
\ifDCMessages 5456 \newif\ifDCMessages␣%
```
a global switch to suppress the message about parsing.

```
\@dc@arguments 5461 \newtoks\@dc@arguments␣%
```
the register for storing parsed `\\command␣{#1}…{%`⟨*n*⟩`}`.

```
\c@dc@catchernum 5466 \newcount\c@dc@catchernum
  \c@dc@argnum 5467 \newcount\c@dc@argnum

\if@dc@ignore@ 5469 \newif\if@dc@ignore@
\if@dc@GroteNegere@ 5470 \newif\if@dc@GroteNegere@␣%
\if@dc@GroteLang@ 5471 \newif\if@dc@GroteLang@␣%
```
for "gross" ignoring

for "gross" longness

```
\if@dc@long@ 5474 \newif\if@dc@long@

\if@dc@BareSpace@ 5476 \newif\if@dc@BareSpace@

%% \gmu@DefSymbol\BareSpace% no need: the test checks bebackslashed strings.

5481 \def\@dc@long@letter{%
5483 \if@dc@long@␣P\fi
5484 }

\if@dc@xadefault 5486 \newif\if@dc@xadefault
```

This is a switch whether a default value of an argument should be one-expanded (`\expandafter`ed) before adding it to the catchers toks. This switch may be set true in a prefix and to make it safe it has to be set false after parsing of each argument. That's why it's set false in so many places. Maybe in the future we'll add analogous switch whether to `\edef` default value, then they both will have to be switched in the same places.

And three swith macros for the specifiers with two defaults.

```
\def\@nx@xa{\@nx\@xa}
\def\@xa@nx{\@xa\@nx}
\def\@xa@xa{\@xa\@xa}
```

```
\if@dc@xadefault@i@
\if@dc@xadefault@ii@
\if@dc@xaeacher@
```

```
\newif\if@dc@xadefault@i@
\newif\if@dc@xadefault@ii@
\newif\if@dc@xaeacher@
```

```
\def\@dc@falsify@xadefaults{%
  \@dc@xadefaultfalse
  \@dc@xadefault@i@false
  \@dc@xadefault@ii@false
}
```

```
\lpdef\DeclareCommand#1#2#3{%
```
7278.

`#1` command to be defined,
`#2` arguments specification,
`#3` definition body.

```
\@dc@alllong@true
```

```
\@dc@alllong@true
  \@dc@ResetParseAuxilia
  \@dc@ParseNextSpecifier␣#2%
  \@dc@buckstopshere% it's the sentinel of parsing. Doesn't have to be defined
      since the test performs a strings comparison.
  \protected\edef#1{%
    \@nx\@dc@␣␣␣␣{\the\toks@}%
    \@xanxcs{\@dc@InnerName{#1}}%
    \@nx␣#1%
  }%
  \edef\gmu@DeclareCommand@resa{%
    \long\def\@xanxcs{\@dc@InnerName{#1}}%
    \@xau\@dc@innerhashes{%
      \unexpanded{#3}}%
  }% of resa
  \gmu@DeclareCommand@resa
}% of the 'draft' \DeclareCommand.
```

```
\long\def\@dc@
  #1% the argument catchers in a pair of braces (their arguments may contain \par
      so the macro is long).
  #2% \\thecommand\
  #3% \thecommand
  {%
  \ifx\protect\@typeset@protect
    \@xa\@firstofone
  \else
    \protect#3\@xa\@gobble
  \fi
  {\@dc@arguments{#2}#1\the\@dc@arguments}%
}
```

```
\def\@dc@ResetParseAuxilia{%
  \emptify\@dc@ParsedSpecs{}%
  \c@dc@catchernum\z@␣% the count of catchers
```

```
    \c@dc@argnum\z@␣% the count of arguments (inner arity) (it may be smaller
                      than the above if we ignore some arguments)

    \toks@{}% in this register we will store the created sequence of argument catch-
                      ers.
```

And for the `M` specifiers:

```
    \@dc@Mmode@false
    \emptify\@dc@innerhashes␣% in this macro we will store the sequence of
                      % #1#2....

    \@dc@GroteNegere@false
    \let\if@dc@GroteLang@\if@dc@alllong@
}
```

`\if@dc@Mmode@`

```
\newif␣\if@dc@Mmode@
\def\@dc@Ms@num␣{\z@}%

\def\@dc@Ms@init{%
    \@dc@Mmode@true
    \def\@dc@Ms@num␣{\z@}%
```

We memorise the state of longness and ignorance at the beginning of `M|m`'s collection (encoded as a numexpr consisting of binary vector of the (standard conversion to $\{0,1\}$) of the switches `\if@dc@long@` and `\if@dc@ignore@`).

```
    \edef\@dc@Ms@initial@listate{%
      \@dc@Ms@listate
    }% of edef
}

\def\@dc@Ms@listate{%
    \boolstobin{{@dc@long@}{@dc@ignore@}}%
}

\def\@dc@Ms@shipout␣{%
```

This macro adds the number of collected `M|m`'s after the catcher and ends the `M`-mode:

```
    \gmu@if␣{num}␣{\@dc@Ms@num>\z@}%
    {\@xa\dc@addtoParsed@\@xa{\@dc@Ms@num}%
      \def\@dc@Ms@num{\z@}%
      \@dc@Mmode@false
     }%
     {}%
}

\def\@dc@ResetStepAuxilia␣{%
```

We set the local ignorance switch to its "gross" value

```
    \let\if@dc@ignore@␣=\if@dc@GroteNegere@
    \let\if@dc@long@␣=␣\if@dc@GroteLang@
```

and reset the expandaftering switches

```
    \@dc@falsify@xadefaults
    \@dc@BareSpace@false
}

\def\@dc@argtypes{%
```

```
  =\gobblespace␣\loop␣\lostwax␣\SameAs␣\Scope
  AaBbCcDdGgKkMmOo%
%% Ôô
  QqSs*TtŤťUu Ww%
  \drs
}
\def\@dc@drses{\drs}
\@ifXeTeX
{\addtomacro\@dc@argtypes{dď}%
  \addtomacro\@dc@drses{dď}%
}
{}
\def\@dc@ParseSpecifier␣#1{% The main inner macro parsing argument specifiers in \DeclareCommand.
%%  \@dc@ResetStepAuxilia % putting it here is wrong andd leads to disaster
  \gmu@CASE␣{stribs}{{#1}\@dc@buckstopshere}%
```

If we meet the sentinel we stop.

```
  {%
```

There could have been caught some `M|m`'s in the previous steps and they are not put immediately, so now we shipout their number, if any:

```
  \@dc@Ms@shipout
```

And edefine the macro carrying the inner hashes:

```
    \edef\@dc@innerhashes␣{%
      \gmu@hashes␣1{\numexpr\c@dc@argnum␣+\@ne}%
    }%
  }%
```

Else we check if we've met the prefixer

```
  \gmu@CASE␣{stribs}{{#1}>}%
  {%
%% \@dc@ResetStepAuxilia % No! the prefix-switches are reset at \@dc@ParseNextSpecif
```

and here we may be after previous prefix.

```
    \grab@prefix␣}%
```

Else we check if `#1` is a known arg specifier…

```
  \@XA{%
    \gmu@CASEsbnone{#1}}\@xa{%
```

Here is the list of arg specifiers (arg types) recognised by `\DeclareCommand`.

Anything else (if not parsed as a specifier's parameter or prefix) is just ignored. For instance, you may write `#1`…`#9` to make your code more readable.

```
    \@dc@argtypes
  }%
```

…and ignore it and iterate further if not…

```
  {\IgnInfo{gmcommand}{while␣parsing␣arg␣specifiers}{#1}%
```

```
5707      \@dc@ParseNextSpecifier␣}%
```

…or else (`#1` is known and not stop and not prefixer) we check if it's the `\SameAs` special specifier.

If so, then we apply special parser `\@dc@SameAs` (quite simple in fact).

```
5715    \gmu@CASE␣{stribs}{{#1}\SameAs}%
```

In this case we just expand the specifiers of "`\SameAs` arguments"

```
5718    {\@dc@SameAs}%
```

Here `#1` is not stop neither prefixer nor the special specifier `\SameAs`, so we add its catcher possibly with »P« for longness, and possibly prefixed with ignorance.

If we parsed sth. else than `M|m`, we possibly ship the `m`'s collected formerly:

```
5726    \gmu@ifsbnone␣{#1}{Mm}%
5727    {\@dc@Ms@shipout␣}%
5728    {}%
```

But only if not in the `M`-mode (about special treatment of `M|m`'s see line 5793).

```
5733    \gmu@if␣{@dc@Mmode@}{}%
5734    {}%
5735    {%
```

Now. If we parse the assignment pseudo-argument, we don't want to add anything to the arguments' toks register so we apply general mechanism of ignoring an argument. The same for the pseudo-argument that causes ignoring spaces.

```
5742      \gmu@ifsbany{#1}{=\gobblespace}{\@dc@ignore@true}{}%
```

and add the catcher of `#1` type.

The catcher's CS is `\ArgumentCatcher@[P]`⟨*arg. type*⟩:

```
5748      \gmu@ifsw␣{@dc@BareSpace@}%
5749      {\@dc@addtoparsed@BareSpace␣}
5750      {}%
5752      \gmu@ifsw␣{@dc@long@}%
5753      {\@dc@addtospecs@bare{>P}}
5754      {}%
5756      \gmu@ifsw␣{@dc@ignore@}%
5757      {\@dc@addtoparsed@Ignore␣␣␣}%
5758      {}%
5760      \@dc@addtospecs@bare␣{#1}%
5762      \@xa\addtotoks\@xa\toks@\@xa{%
5763        \csname␣ArgumentCatcher@\@dc@long@letter
5764        \strip@bslash{#1}%
5765        \endcsname
5766      }% of toks' text.
```

Now we step the counters of catchers and argums (they'll be unequal if some argument is ignored or if there are multiple subsequent `m`'s) and probably add `#`⟨*n*⟩ tokens to respective toks.

```
5772      \advance\c@dc@catchernum\@ne
5773    }% of if not M-mode
```

But we step the number of arguments (of the inner macro) also in the `M`-mode:

```
5777    \gmu@notif␣{@dc@ignore@}{}%
```

```
5778    {\@dc@addinnerhash␣}%
5779    {}% (empty "not-else" branch of ignoring)
```

(the ignorance switch will be reset at the beginning of next step of parsing, just like other step-local switches).

Now we parse the parameter(s) of `#1` where we should (usually the default value)

The `M|m` catcher(s) are treated specially, for legacy and performance reasons: when we meet an `m` or `M`, we set our parser to the `M`-mode if not yet, to look if it's not a beginning of a longer sequence of such specifiers.

```
5793    \gmu@CASEsbany␣{#1}{Mm}%
```

If so, we process it appropriately:

```
5796    {\@dc@process@ms}%
```

The `K` catcher also requires special treatment since the parameters are not just passed to it but a particular macro is defined with use of them.

```
5802    \gmu@CASEsbany␣{#1}{Kk}%
5803    {\@dc@define@K}%
```

Else (not mandatory(s) neither Knuthian) we check if parameterless, one- or two-parameter.

```
5808    \gmu@CASEsbany␣{#1}{\gobblespace}%
```

For the parameterless specifiers we did all and we just continue parsing.

```
5813    {\@dc@ParseNextSpecifier}%
```

Now the one-parameter specifiers (the parameter is optional, although (necessarily) in curly braces). For these we look for the parameter and provide the default if not provided by the user.

```
5819    \gmu@CASEsbany{#1}{=\Scope␣AaBbCcDdOoSs*}%
5820    {\grab@optparam{#1}}%
5823    \@XA{\gmu@CASEsbany{#1}}\@xa{\@dc@drses}%
5824    {\grab@optparam{#1}}%
```

For the loop catchers, `Uu` so far, maybe `Qq` in the future:

```
5828    \gmu@CASEsbany{#1}{Uu Ww\lostwax}%
5829    {\@dc@grab@Loop␣#1}% putting the specifier's letter as #1 for the grabber suppresses adding it to the parsed specifiers' list.
```

Else we've met a specifier with first parameter mandatory and second optional (the latter has to be in curly braces):

```
5836    \gmu@lastCASE
5837    {%
5838      \grab@twoparams{#1}}%
5839    \gmu@ESAC
5841 }% of \@dc@ParseSpecifier
```

```
5844 \def\@dc@addtoparsed@Ignore{%
5845    \addtotoks\toks@{\@dc@ignorearg}%
5846    \@dc@addtospecs@bare{>i}%
5847 }
```

```
5850 \def\@dc@addinnerhash{%
```

```
5851   \advance\c@dc@argnum\@ne
5852 }
```

```
5855 \def\@dc@addtoparsed@BareSpace{%
5856   \addtotoks\toks@{\@dc@BareSpaceArg␣}%
5857   \@dc@addtospecs@bare{>\BareSpace␣}%
5858 }
```

```
5863 \def\@dc@process@ms{%
5864   \gmu@if␣{@dc@Mmode@}{}%
```

If in M-mode, we compare current state of ignorance and longness with their initial values

```
5868   {\gmu@if␣{num}{\@dc@Ms@initial@listate=\@dc@Ms@listate}%
```

If the state hasn't changed, we look if the optional number of m's is provided and proceed in any case appropriately.

```
5872       {\@dc@process@ms@grabnum␣}% of if the state has'nt been changed
```

Else, i.e. when we have an M|m specifier but of another longness or ignorance, we ship the m's collected so far and *close* the M-mode…

```
5877       {\@dc@Ms@shipout
```

…and *reparse* this M|m without changing current ignorance and longness, to let the new instance of parser add now closed number of m's and the new catcher.

Remember however that we've increased number of arguments so now we set it back:

```
5885         \advance\c@dc@argnum\m@ne
5886       \@dc@ParseSpecifier␣m%
```

Note it's not Next, which means we don't reset the switches. And first of all, it's not process@ms, because we have to add new catcher etc.

```
5890       }% of if the state has been changed
```

```
5892   }% of if M-mode
```

If we are not in M-mode but have parsed an M|m, now we set us to be in the mode and look for (optional) number of m's.

```
5896   {\@dc@Ms@init
```

(This macro sets the number of m's found to 1 and turns the switch \if@dc@Mmode@ true).

```
5901   \@dc@process@ms@grabnum
5902   }% of not if M-mode.
5903 }% of dc@process@ms
```

We peep the next char and if it looks as an argument

```
5909 \def␣\@dc@process@ms@grabnum␣{%
5910   \@ifnextanyIS␣{\bgroup␣\@ne␣\tw@␣\thr@@␣123456789}%
5911   {%
5912     \@dc@process@ms@fin}%
5913   {%
5914     \@dc@process@ms@fin␣\@ne}%
5915 }
```

```
5917 \def␣\@dc@process@ms@fin␣#1{%
```

```
  \edef\@dc@processms@hash␣{\the\numexpr#1}%
  \edef\@dc@processms@left␣{\the\numexpr␣9-\c@dc@argnum+\@ne}%
```

+1 because we've increased the #es number in line 5778.

```
  \gmu@AND␣{%
    {num}␣{\@dc@processms@hash␣>\z@}
    {num}␣{\@dc@processms@hash␣<\numexpr
      \@dc@processms@left+1␣\relax␣}%
  }%
  {\stepnummacro␣\@dc@Ms@num␣\@dc@processms@hash
```

And we add proper number of #es

```
    \edef\@dc@hashgoal␣{%
      \the\numexpr\c@dc@argnum␣+%
      \@dc@processms@hash-1}% minus 1 because one has (one has already
                                been added in line 5778).
    \@whilenum␣\c@dc@argnum<\@dc@hashgoal␣\do{%
      \@dc@addinnerhash
    }%
    \@dc@ParseNextSpecifier
  }% of if all conds. satisfied
  {% some of the conds. not satisfied
    \PackageError{gmcommand}{%
      The␣argument␣to␣the␣M␣or␣m␣specifier,␣if␣any,␣has␣to␣be␣
           ^^J%
      a␣number␣between␣1␣and␣\@dc@processms@left␣^^J%
      (there␣is/are␣\the\c@dc@argnum\space␣arg(s)␣declared
      already).^^J%
      I␣ignore␣this␣m/M.}%
    {}%
    \gmu@passbraced␣␣\@dc@ParseNextSpecifier␣{#1}%
  }%
}% of \@dc@process@ms
```

```
\def\@dc@ParseNextSpecifier{%
  \@dc@ResetStepAuxilia
  \@dc@ParseSpecifier
}
```

```
\lpdef\@dc@AddAndParse#1{%
  \dc@addtoParsed@{#1}%
  \@dc@falsify@xadefaults
  \@dc@ParseNextSpecifier
}
```

The eating M's uses the same concept of a "while" iterator as the `\loop` catcher but we prefer to define here a simplified version for this particular purpose.

```
\def\grab@optparam␣#1{% arg specifier
  \@ifnextchar\bgroup{\@dc@AddAndParse}%
  {%
```

If the default is not provided in the command's declaration, then we check whether it's provided particularly for this specifier or put `\NoValue`.

```
    \edef\@dc@tempa{%
```

```
      \gmu@if␣{csname}%
      {@dc@optparam@deft@\strip@bslash{#1}\endcsname}%
      {\@xaucs␣{@dc@optparam@deft@\strip@bslash{#1}}}%
      {\@nx␣\NoValue}%
    }%
    \@xa\@dc@AddAndParse\@xa{\@dc@tempa}%
  }%
}
```

Default default values for some argument types:

The declaring macro may be used with any-case version of the specifier, because both the upper- and lowercase defaults are defined, however this works only for letter specifiers. The caseness of CS specifiers is not modified.

```
\long\def\@dc@DeclareDefault␣#1#2{%
  \uppercase{%
    \Name\edefU{\strip@bslash{\@dc@optparam@deft@}%
        \strip@bslash{#1}}{#2}}%
  \lowercase{%
    \Name\edefU{@dc@optparam@deft@\strip@bslash{#1}}{#2}}%
}
```

The default default of "**d**ecimal" argument is 0.

```
\@dc@DeclareDefault␣D{0}
\@dc@DeclareDefault␣d{0}

\@ifXeTeX␣{%
  \@dc@DeclareDefault␣Ď{0}%
}
{}
```

The default default delimiter of the "Scope" argument's parsing is a star of any catcode of {11, 12, 13}

```
\@xa\@dc@DeclareDefault␣\@xa\Scope\@xa{\all@stars}

\@dc@DeclareDefault␣=␣{\@tempcnta}
```

The "se**Q**uence" and "**U**ntil" arguments have `\NoValue` as the default default values (which hasn't to be announced so explicitly since it's the default setting):

```
\@dc@DeclareDefault␣Q␣{\NoValue}

\@dc@DeclareDefault␣U␣{\NoValue}

\long\def\@dc@maybe@expandafter␣#1{%
  \gmu@if␣{@dc@xadefault}{}%
  {\gmu@ifutokens{#1}{\NoValue}%
    {\@secondoftwo}{\@firstoftwo}%
  }% if we are to expandafter #1.
  {\@secondoftwo}% if we don't expandafter #1
}

\long\def\dc@addtotoks@␣#1{%
  \@dc@maybe@expandafter␣{#1}%
  {\toks@\@xa\@xa\@xa{\@xa\the\@xa\toks@\@xa{#1}}}%
  {\toks@\@xa{\the\toks@{#1}}}%
}
```

```
\long\def\@dc@addtospecs␣#1{%
  \@dc@maybe@expandafter␣{#1}%
  {\@xa␣\addtomacro␣\@xa\@dc@ParsedSpecs\@xa{\@xa{#1}}%
  }%
  {\addtomacro\@dc@ParsedSpecs{{#1}}}%
}
\long\def\dc@addtoParsed@␣#1{%
  \dc@addtotoks@{#1}%
  \@dc@addtospecs{#1}%
}
\long\def\@dc@addtospecs@bare␣#1{%
```

note it also doesn't respect `xadefault` switch.

```
  \addtomacro\@dc@ParsedSpecs{#1}%
}
\long\def\@dc@addtotoks@bare␣#1{%
  \addtotoks\toks@␣{#1}%
}
\long\def\@dc@addtoParsed@bare␣#1{%
```

note it also doesn't respect `xadefault` switch.

```
  \@dc@addtotoks@bare␣{#1}%
  \@dc@addtospecs@bare␣{#1}%
}
\long\def\grab@twoparams
#1% the specifier
#2% first parameter, which is mandatory
{% default default (when default is absent)
  \if@dc@xadefault@i@\@dc@xadefaulttrue\fi
  \dc@addtoParsed@{#2}%
  \if@dc@xadefault@ii@
    \@dc@falsify@xadefaults\@dc@xadefaulttrue
  \fi
  \grab@optparam{#1}%
}
\long\def\@dc@addargum#1{%
  \addtotoks\@dc@arguments{#1}%
\@iwruif␣{@@@@@@␣@dc@arguments:␣»\the\@dc@arguments«}%
}
\Store@Macro\@dc@addargum
\pdef\@dc@ignorearg{%
  \long\def\@dc@addargum##1{%
    \Restore@Macro\@dc@addargum}%
}
\pdef\@dc@BareSpaceArg{%
  \let␣\@dc@Loop@CareOfSpace␣\@dc@Loop@CareOfSpace@NoBraces
  \def\@dc@addargum␣{%
    \Restore@Macro\@dc@addargum
```

```
    \Restore@Macro\@dc@Loop@CareOfSpace
    \@dc@addargum
  }%
}
```

Parsing of Knuthian parameters string is easier to implement with full-feature `\DeclareCommand` so we postpone it till line 7541

```
\def\grab@prefix@CASE{% just a shorthand
  \@xa\gmu@CASEsbany\gmu@forarg
}

\long\def\grab@prefix␣#1{%
```

(2010/07/26, v0.993:) made for-eaching to make it behave as expected, i.e. that latter settings prævalent over the former

`%% \@dc@ResetStepAuxilia %` No! We want allow many subsequent prefixes, they make no harm.

```
  \gmu@foreach#1\gmu@foreach@delim␣{%
    \grab@prefix@CASE
    {\long!lL\par␣Pp}%
    {\@dc@long@true}%
    \grab@prefix@CASE
    {Ii}%
    {\@dc@ignore@true}%
    \grab@prefix@CASE
    {\@xa\expandafter}%
    {\@dc@xadefaulttrue}%
    \grab@prefix@CASE
    {\@xa␣\@xa@nx␣\@xa@xa␣\expandafter␣\@xanx␣\xanx␣\xaxa}%
    {\@dc@xadefault@i@true}%
    \grab@prefix@CASE
    {\@xa␣\@nx@xa␣\@xa@xa␣\expandafter␣\@nxxa␣\nxxa}%
    {\@dc@xadefault@ii@true}%
    \grab@prefix@CASE
    {\@xaeacher␣\xaeacher␣\xaEacher␣\xaE␣}%
    {\@dc@xadefault@ii@true}%
    \grab@prefix@CASE
    {\GroteNegere␣\GrossIgnore␣\GroteN␣\GrossI}
    {\@dc@GroteNegere@true
      \@dc@ignore@true}%
    \grab@prefix@CASE
    {\GroteNegereStop\GrossIgnoreStop
      \GroteNStop\GrossIStop}
    {\@dc@GroteNegere@false
      \@dc@ignore@false}%
    \grab@prefix@CASE
    {\GroteLang\GrossLong\GroteL\GrossL}
    {\@dc@GroteLang@true
      \@dc@long@true}%
    \grab@prefix@CASE
    {\GroteLangStop\GrossLangStop
      \GroteLStop\GrossLStop}%
    {%
```

```
      \let\if@dc@GroteLang@=\if@dc@alllong@
      \let\if@dc@long@=\if@dc@alllong@
    }%
    \grab@prefix@CASE
    {\BareSpace}
    {\@dc@BareSpace@true}
    \gmu@EatCases
    {\IgnInfo{gmcommand}{while␣parsing␣arg␣specifiers}{#1}}%
    \gmu@ESAC
  }% of for each
    \@dc@ParseSpecifier␣%
```

Note that here (unlike in all other places) is `\@dc@ParseSpecifier` not `\@dc@ParseNextSpeci` The difference between them is in that the former doesn't reset the switches (which we've just set here).

```
}% of \grab@prefix
```

```
\long\def\dc@DefParseNext#1{%
```

This is to allow hashes in the defaults.

```
  \edefU\@dc@parse@next{#1}%
}
```

```
\def\@dc@Alias
#1% \lowercase or \firstofone
#2% left side of the assignment
#3% right side of the assignment
#4% Lower or empty
{%
  \gmu@if␣{csname}%
  {ArgumentCatcher@\strip@bslash␣#3\endcsname}%
  {#1{%
      \@xa\let\csname\strip@bslash\ArgumentCatcher@
      \strip@bslash␣#2\@xa\endcsname}%
    \csname␣ArgumentCatcher@\strip@bslash␣#3\endcsname
  }{\PackageError{gmutils/gmcommand}{%
      You're␣trying␣to␣#4Alias␣the␣»\unexpanded{#3}«␣
           catcher^^J%
      but␣it␣is␣not␣defined!}{}%
  }%
}
```

To provide lowercase parallels of a specifier

```
\def\@dc@LowerAliases␣#1{%
```

To provide lowercase aliases of the catchers:

```
  \@dc@Alias␣\lowercase␣{#1}{#1}{Lower}%
  \@dc@Alias␣\lowercase␣{\P#1}{\P#1}{Lower}% we provide »P« as a CS
       to avoid lowercase'ing it.
}
```

```
\def\@dc@AliasCatcher
#1% left side of the assignment
#2% right side of the assignment
```

```
{%
  \@dc@Alias␣\firstofone␣{#1}{#2}{}%
}

\def\@dc@AliasPLower
#1% specifier to alias
{\@dc@AliasCatcher␣{P#1}{#1}%
  \@dc@LowerAliases{#1}%
}
```

Parsing of a braced optional. Note the test is `\ifcat` so *any* begin-group character will turn it true.

Note that although this parser issues an error when brace contains `\par`, its default may still contain `\par`.

```
\long\def\ArgumentCatcher@B
#1% default value
#2% tail of args catchers.
\@dc@arguments␣% delimiter
{%
  \@ifnextcat\bgroup
```

If we are before a begin-group token, we store the tail of parsers in an auxiliary macro and launch inner catcher of a mandatory argument.

```
  {\dc@DefParseNext{#2}\ArgumentCatcher@m@i␣}%
  {\@dc@addargum␣{{#1}}#2\@dc@arguments␣}%
}
```

Analogously to the previous catcher only the mandatory catcher is long.

```
\long\def\ArgumentCatcher@PB
#1% default value
#2% tail of args catchers.
\@dc@arguments␣% delimiter
{\@ifnextcat\bgroup
  {\dc@DefParseNext{#2}\ArgumentCatcher@Pm@i␣}%
  {\@dc@addargum␣{{#1}}#2\@dc@arguments␣}%
}

\@dc@LowerAliases␣B

\long\def\ArgumentCatcher@T@␣% parsing of a single Token continued:
#1% kind of test (strings or StrX so far (2010/12/28, 7.24))
#2% the list to search #4 on,
#3% the default value,
#4% the tail of args parsers,
#5% the token we search in #1 (it's an unbraced single token as we checked in line
      6314).
{%
  #1␣#5{#2}%
  {\@dc@addargum␣{{#5}}#4\@dc@arguments}%
  {\@dc@addargum␣{{#3}}#4\@dc@arguments#5}%
}

\long\def\ArgumentCatcher@T␣% parsing of a single Token. It's always long
      since we look for a single unbraced token so there is no danger of "runaway
      argument".
```

This catcher uses the StrX test so is very subtle: it applies \ifx first and if the compared tokens are both CS es and \ifx-equal then \ifstrings is applied.

```
#1% the list we search optional token on,
#2% the default value,
#3% the tail of args parser.
\@dc@arguments{%
  \@ifnextnotgroup
  {\ArgumentCatcher@T@␣\gmu@ifStrXany␣{#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.
}

\@dc@AliasPLower␣T

\long\def\ArgumentCatcher@Ť␣#1#2#3% as in ...@T
{%
  \ArgumentCatcher@Loop
  {StrX}{\any}{#1}{\any}{}{1}{#2}{% empty eacher
  }{#3}%
}

\@dc@AliasPLower␣Ť
```

This is incompatibility of this version (v0.993) with other versions.

```
%% \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.
%% \let\ArgumentCatcher@PT\ArgumentCatcher@S

\edef\ArgumentCatcher@S
#1% default value
{%
  \@nx\ArgumentCatcher@T␣{\@xa\@nx\all@stars}{#1}}% the s arg spec
      is a shorthand for T{*}. Except the star may be of any category from
      {11, 12, 13}.

\@dc@AliasPLower␣S
\@dc@AliasCatcher␣*␣S
```

\if@debugacro@ `\newif\if@debugacro@`

\ArgumentCatcher@Loop `\long\def\ArgumentCatcher@Loop␣␣% Now we introduce a general iterating catcher. The two parsers: "while" (Q) and "until" (U) are special cases of it.`

Moreover, clever tripling it generalises also GBbOoCc specifiers (with some inner macros relaxed for the first instance).

`#1%` kind of test (StrX or str or anything else for what \gmu@if⟨??⟩any/none is defined)

The catcher has four parameters that interact with one another:

`#2%` "sign" of match for iteration: \any or \none token

`#3%` list of tokens \@let@token will be matched against with #1 test and sustain iteration or stop catching.

We assure in line **??** that when we condition iteration on presence of next token on `#2` list, i.e. `#1` is `\any`, we appended the "drainers" list to `#2` so we won't stop at them.

`#4%` test for "drainers" (allowed values as for `#1`)

`#5%` list of "drainers": if `\@let@token` satisfies the `#3` test, we throw it down the drain, otherwise we add it to the argument.

The "iterate or stop" matching precedes the "add or throw" matching so if we meet a stopping token, surely it will not be added to the argument.

`#6%` upper bound of number of items (may be empty or negative to turn counting of items off).

For a moment I was tempted to try implement *any* loop condition, but that seems not to make sense, since during argument catching tokens are not executed so hardly any tests except matching items against a list or counting them seem to be reasonable.

This catcher is always `\long` but it'd be not the only danger of iterating beyond end of file: another would be allowing all three: `#1`, `#2`, `#4` empty at the same time, but we check that in the defaults' parser (line **??**).

`#7%` default value (if empty sequence is parsed), `\NoValue` by default.

`#8%` an "eacher"—stuff to be put before each token/text added to the argument. Empty by default. If `#6` is empty and used (empty sequence parsed), `#7` is not put before it.

`#9%` the tail of args parser

```
\@dc@arguments␣%
{%
  \dc@DefParseNext{#9}%
  \emptify\@dc@seQuence
  \c@dc@Loop@bound=\numexpr(\gmu@ifempty{#6}{-1}{#6})*1\relax
```

Since `#6` is empty by default, the counter is set to −1 by default (l. **??**)

```
  \gmu@if␣{num}{\c@dc@Loop@bound<\z@}%
```

We look at the sign of the bound and if it's − we oppose the bound and make the decreasement step = 0

```
  {\c@dc@Loop@bound=-\c@dc@Loop@bound
    \let\@dc@Loop@countby\z@
  }%
```

Otherwise we make the decreasement step −1.

```
  {\let\@dc@Loop@countby\m@ne}%
  \gmu@ifempty{#3}%
  {\gmu@if␣{strings}{\none␣#2}%
    {\@dc@Loop@iteralltrue}% it's an additional switch for better performance
        in a special case: when we wish only count the items and iterate over
        anything
    {%
```

Here we'd look for presence of the next token on an empty list, what cannot be. Therefore we stop. In this case the value of switch is irrelevant.

```
      \Store@Macro\ArgumentCatcher@Loop@defcheck
      \def\ArgumentCatcher@Loop@defcheck{%
        \Restore@Macro\ArgumentCatcher@Loop@defcheck
```

```
6442          \ArgumentCatcher@Loop@shipout}%
6443      }% of if #2 str-is \any
6444    }% of if #3 empty
6445    {\@dc@Loop@iterallfalse}% when #2 nonempty, we have to perform
            matching.
```

We prepare shortcuts for adding/rejecting all analogously:

```
6449    \gmu@ifempty{#5}%
6450    {\gmu@if␣{strings}␣{\any␣#4}%
6451      {\@dc@Loop@addallfalse\@dc@Loop@drainalltrue}%
6452      {\@dc@Loop@addalltrue\@dc@Loop@drainallfalse}%
6453    }% of if #5 empty
6454    {\@dc@Loop@addallfalse\@dc@Loop@drainallfalse}% when #4
            nonempty, we have to perform matching.
6457    \edef\ArgumentCatcher@Loop@afterpeep{%
```

This is the macro that'll be executed after the main `\futurelet`, We define it here since nothing in this sequence of tokens changes during catching, only the meaning of `\@let@token`.

```
6463      \gmu@if␣{@dc@Loop@iterall}{}%
```

If we iterate over anything, we just unbrace further stuff (that determines what is to be added and what discarded) and discard the "else" branch of iteration test.

```
6469      {\firstoftwo}%
```

Otherwise (iteration over a proper subset of all possible tokens) we prepare proper test:

```
6474      {\unexpanded{%
6475          \@dc@Loop@test
6476          {#1}#2{#3}% if the next token StrX-is/is not on #2, then…
6478        }% of \unexpanded
6479      }% of if we don't iterate over all (of preparation of iteration test)
```

Now what to do in the iteration step:

```
6483      {\gmu@if␣{@dc@Loop@addall}{}%
```

If we add anything then we prepare bare adder:

```
6487        {\@nx␣\ArgumentCatcher@Loop@add}%
```

If we add not everything, then maybe we should *discard* everything?

```
6492        {\gmu@if␣{@dc@Loop@drainall}{}%
```

If indeed, we prepare a bare discarder

```
6496          {\@nx␣\ArgumentCatcher@Loop@drain}%
```

Otherwise (we neither add anything nor discard anything) we prepare proper test. It has to be braced since it's multitoken

```
6501          {\unexpanded{%
6502              \@dc@Loop@test
6503              {#1}#4{#5}% if the next token satisfies #4-direction StrX/str match
                      against #5, then we discard it, otherwise we add it.
6506              \ArgumentCatcher@Loop@drain
6507              \ArgumentCatcher@Loop@add
```

```
6508            }% of \unexpanded
6509          }% of not drain all
6510        }% of if not add all
6511      }%
```

And what we do if we iterate over a proper subset of items and the test test of line 6476 turned false. We wrap it in curly braces to let it be gobbled if we iterate over all.

```
6517      {\gmu@if␣{@dc@Loop@iterall}{}%
6518        {}%
6519        {\@nx␣\ArgumentCatcher@Loop@shipout}%
6520      }%
6521      \unexpanded{{#7}{#8}}%
6522    }% of \ArgumentCatcher@Loop@afterpeep
```

```
6525    \ArgumentCatcher@Loop@defcheck␣{#7}{#8}%
6526    \ArgumentCatcher@Loop@check
6527 }% of \ArgumentCatcher@Loop
```

\c@dc@Loop@bound

```
6529 \newcount\c@dc@Loop@bound
```

\if@dc@Loop@iterall
\if@dc@Loop@addall
\if@dc@Loop@drainall

```
6531 \newif\if@dc@Loop@iterall
6532 \newif\if@dc@Loop@addall
6533 \newif\if@dc@Loop@drainall
```

```
6536 \long\def\@dc@Loop@test
6537 #1% kind of test (StrX or str so far (2010/12/28, 7.51))
6538 #2% \any or \none
6539 #3% list of tokens to match against
6540 #4% what if OK
6541 #5% what if not OK.
6542 {% remember we are in the argument of \gmu@peep@next so we are after peep.
```

```
6545  \gmu@if␣{defined}{\@def@token}
6546  {\csname␣gmu@if#1\strip@bslash␣#2\@xa\endcsname
6547    \@def@token␣}%
6548  {\csname
```

we construct csname of the quantifier-test. Depending on `#1` it will apply `\ifx` or `\if` with `\noexpand`s.

```
6551       gmu@if%
6552       \gmu@ifdetokens␣{#1}{str}%
6553       {NX}%
6554       {x}%
6555       \strip@bslash␣#2%
6556     \endcsname
6557     \@let@token␣}%
6558  {#3}{#4}{#5}%
6559 }
```

```
6564 \@dc@AliasCatcher␣{PLoop}␣{Loop}
```

```
6566 \long\def\ArgumentCatcher@Loop@defcheck
6567 #1% default,
6568 #2% “eacher”
6569 {%
6570   \edefU\ArgumentCatcher@Loop@check{%
```

```
      \gmu@if␣{num}{\c@dc@Loop@bound>\z@}
```

If we haven't reached maximum allowed number of parsed items, we peep the next token, i.e. we perform `\futurelet` and if `\@let@token` is undefined or `\relax` then we pass it inside `\@def@token` so that string comparisons may look at it.

```
      {\advance␣\c@dc@Loop@bound␣\@dc@Loop@countby
        \gmu@peep@next
        \ArgumentCatcher@Loop@afterpeep␣}
```

Otherwise we finish catching and send the argument to the arguments' toks.

```
      {\ArgumentCatcher@Loop@shipout␣{#1}{#2}}% of if num bound reached
    }% of \ArgumentCatcher@Loop@check
}% of \ArgumentCatcher@Loop@defcheck

\lpdef\@dc@Loop@CareOfSpace
#1% a macro with all the arguments except last
{\gmu@ifpeeped␣x␣\gmu@letspace
  {\edef\@dc@Loop@careofspace@aas{%
      \unexpanded{#1{␣}}%
    }% edef unexpanded to allow # tokens in #1
    \afterassignment\@dc@Loop@careofspace@aas
    \let\gmu@drain=␣}% after = is a space.
  {#1}% next not space, so just #1 and let it process the next token its way.
}

\StoreMacro@nocat\@dc@Loop@CareOfSpace

\lpdef\@dc@Loop@CareOfSpace@NoBraces
#1% a macro with all the arguments except last
{%
\gmu@ifpeeped␣x␣\gmu@letspace
  {\edef\@dc@Loop@careofspace@aas{%
      \unexpanded{#1␣}% note the space
    }% edef unexpanded to allow # tokens in #1
    \afterassignment\@dc@Loop@careofspace@aas
    \let\gmu@drain=␣}% after = is a space.
  {#1}% next not space, so just #1 and let it process the next token its way.
}

  %% \lpdef\@dc@Loop@CareOfGroup
  %% #1% as for the previous macro
  %% {\gmu@if x{\@let@token\bgroup}%
  %%  {\gmu@passbraced{#1}}% if next bgroup
  %%  {#1}% not \bgroup, so we get it normally
  %% }% of \@dc@Loop@CareOfGroup No need of the above (commented out): \gmu@passbraced does that.

\lpdef\@dc@Loop@CareOfSpaceAndGroup
#1% as in the previous macro
{\@iwruif␣{Care␣of␣both:␣@let@token:␣»\meaning\@let@token«}%
  \@dc@Loop@CareOfSpace{%
    \gmu@passbraced{#1}%
  }%
}% of "take care of both"

\long\def\ArgumentCatcher@Loop@add
```

```
#1% default
#2% eacher
```
At this level we pass both default and eacher for symmetry of the macro(s) at higher level.
```
{\@dc@Loop@CareOfSpaceAndGroup
  {\ArgumentCatcher@Loop@add@{#2}}%
}
\long\def\ArgumentCatcher@Loop@add@
```
But here we can limit ourselves to what we really need.
```
#1% eacher
#2% token/text to be added
{\addtomacro\@dc@seQuence{#1#2}%
  \@iwruif␣{@@@@@@␣Q-add:␣»\the\@dc@arguments«}%
  \ArgumentCatcher@Loop@check
}% of \ArgumentCatcher@Loop@add@
\long\def\ArgumentCatcher@Loop@drain
#1% default
#2% eacher
```
At this level we pass both default and eacher for symmetry of the macro(s) at higher level.
```
{%
  \@iwruif{@@@@@@␣Q-drain:␣»\the\@dc@arguments«}%
  \@dc@Loop@CareOfSpaceAndGroup
  {\@xa\ArgumentCatcher@Loop@check\@gobble}%
}
\long\def\ArgumentCatcher@Loop@shipout
#1% default
#2% eacher
```
This macro needs both of these parameters.
```
{%
  \gmu@if␣x{\@dc@seQuence\@empty}%
  {\def\@dc@seQuence{#1}%
```
Thus we put the default value to the temporary macro. Now, it's nonempty, we prepend to it the "eacher":
```
    \gmu@if␣x{\@dc@seQuence\@empty}%
    {}{\prependtomacro\@dc@seQuence{#2}}%
  }%
  {}% if \@dc@seQuence is nonempty, we don't modify it.
  \@xa\@dc@addargum\@xa{\@xa{\@dc@seQuence}}%
  \@iwruif␣{@@@@@@␣Q-finish:␣»\the\@dc@arguments«}%
  \@dc@parse@next\@dc@arguments
}% of \ArgumentCatcher@Loop@shipout
\long\def\ArgumentCatcher@Q
#1% list
#2% default (\NoValue by default, as in previous versions).
{%
  \ArgumentCatcher@Loop
```

```
  {StrX}%
  \any
  {␣#1}% note the space before #1—settheory sum of #1 and drainers.
  \any
  {␣}% we ignore spaces—it's a legacy setting, maybe we'll optionise it in the future
  \m@ne␣% we allow any number of tokens
  {#2}% default, by default \NoValue, as defined in line ??
  {}% empty eacher
}%
\@dc@AliasPLower␣Q
  %% \@ifXeTeX{%
  %%  \long\def\ArgumentCatcher@Ô
  %%  #1% list
  %%  #2% default (\NoValue by default, as in previous versions).
  %%  {%
  %%   \@dc@BareSpaceArg
  %%   \ArgumentCatcher@Loop
  %%   {str}%
  %%   \any
  %%   { #1}% note the space before #1—settheory sum of #1 and drainers.
  %%   \any
  %%   { }% we ignore spaces—it's a legacy setting, maybe we'll optionise
  %%   % it in the future
  %%   \m@ne % we allow any number of tokens
  %%   {#2}% default, by default \NoValue, as defined in line ??
  %%   {}% empty eacher
  %%  }%
  %%
  %%  \@dc@AliasPLower Ô
  %%
  %% }{}% of if XeTeX
\def\ArgumentCatcher@D{% decimal argument (useful for dates e.g.)
  \ArgumentCatcher@Q
  {-+0123456789}%
```

the default value will be delivered by the defaults' catcher

```
}
\@dc@AliasPLower␣D
\@ifXeTeX
{%
  \def\ArgumentCatcher@Ď
```

The difference between D and Ď catchers is that the latter stops at a blank while the former doesn't.

```
  #1% default value, 0 by default
  {% decimal argument
    \ArgumentCatcher@Loop
    {StrX}%
    \any
    {-+0123456789}%
```

```
      \none
      {}%
      \m@ne
      {#1}%
      {}%
    }%
\@dc@AliasPLower␣Ď
}{}% of if XeTeX
\def\ArgumentCatcher@U␣{%
  \ArgumentCatcher@Loop␣{StrX}%
}
```

Yes, because *de facto* we pass it *all* the arguments of catcher `Loop`.

```
\@dc@AliasPLower␣U
\@dc@AliasCatcher␣W␣{U}
\@dc@AliasPLower␣W
\@dc@AliasCatcher␣{lostwax}␣U␣% The \lostwax catcher is a normal "until"
      catcher. It's parsing of the specifier's parameters that makes it different.
\long\def\ArgumentCatcher@item
#1% "sign" of matching
#2% list to match against
#3% default value
#4% eacher (which in this case will be applied to the (one) item)
{%
  \ArgumentCatcher@Loop
  {StrX}%
  #1%
  {#2}%
  \none␣{}% we add all that matches
  1␣% at most one item
  {#3}%
  {#4}%
}
\if@XeTeX
  \def\ArgumentCatcher@ {%
    \@dc@BareSpaceArg
    \ArgumentCatcher@Loop␣{StrX}%
  }
  \@dc@AliasPLower␣
\fi
```

TODO: edef prefixing.

```
\long\def\ArgumentCatcher@genM
#1% the letter »P« (for long catcher) or nothing (for the short version)
#2% number of undelimited parametrs to catch (arabic)
#3% tail of parsers
\@dc@arguments{%
```

```
  \dc@DefParseNext{#3}%
  \csname␣ArgumentCatcher@#1m@%
  \romannumeral#2%
  \endcsname
}
```

They both have to be long for the tail of parser that can contain `\par`. It's the *inner* catchers …`@(P)m@ii`…that are short or long.

```
\def\ArgumentCatcher@M{\ArgumentCatcher@genM␣{}}
\def\ArgumentCatcher@PM{\ArgumentCatcher@genM␣P}

\@dc@AliasCatcher␣{m}␣{M}
\@dc@AliasCatcher␣{Pm}␣{PM}
```

We allow specifying mandatory arguments also as `(M|m){`⟨*num*⟩`}` and such a specifier is passed to the carrier macro.

```
\@dc@AliasCatcher␣{ms}␣{M}
\@dc@AliasCatcher␣{Pms}␣{PM}

\@tempcnta=\@ne
\@whilenum\@tempcnta<9\do{%
  \edef\gmu@tempa{%
```

The short catchers of mandatories:

```
    \def\@xanxcs{%
      ArgumentCatcher@m@\romannumeral\@tempcnta}%
    \gmu@hashes1{\numexpr\@tempcnta+1}%
    {\@nx\@dc@addargum{%
        \gmu@hashesbraced␣1{\numexpr\@tempcnta+1}}%
      \@nx\@dc@parse@next\@nx\@dc@arguments
    }% of \ArgumentCatcher@m@⟨rom.num⟩.
```

And the long ones:

```
    \long␣\def\@xanxcs{%
      ArgumentCatcher@Pm@\romannumeral\@tempcnta}%
    \gmu@hashes1{\numexpr\@tempcnta+1}%
    {\@nx\@dc@addargum{%
        \gmu@hashesbraced␣1{\numexpr\@tempcnta+1}}%
      \@nx\@dc@parse@next\@nx\@dc@arguments
    }% of \ArgumentCatcher@Pm@⟨rom.num⟩.

  }% 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{%
  \typeout{\bslash␣␣ArgumentCatcher@m@\romannumeral%
      \@tempcnta:␣␣␣␣^^J%
    \Name\meaning{ArgumentCatcher@m@\romannumeral%
        \@tempcnta}^^J}%
```

```
  \typeout{\bslash␣␣ArgumentCatcher@Pm@\romannumeral%
      \@tempcnta␣:␣^^J%
    \Name\meaning
    {ArgumentCatcher@Pm@\romannumeral\@tempcnta}^^J^^J^^J%
  }%
  \advance\@tempcnta\@ne
}
  %% \show\ArgumentCatcher@Pm@viii
\fi
```

The code above produces on the terminal (without the blank between 1's of course):

```
\ArgumentCatcher@m@i:
macro:#1->\@dc@addargum {{#1}}\@dc@parse@next \@dc@arguments

\ArgumentCatcher@Pm@i:
macro:#1->\@dc@addargum {{#1}}\@dc@parse@next \@dc@arguments

\ArgumentCatcher@m@ii:
macro:#1#2->\@dc@addargum {{#1}{#2}}\@dc@parse@next
     \@dc@arguments

\ArgumentCatcher@Pm@ii:
macro:#1#2->\@dc@addargum {{#1}{#2}}\@dc@parse@next
     \@dc@arguments

\ArgumentCatcher@m@iii:
macro:#1#2#3->\@dc@addargum {{#1}{#2}{#3}}\@dc@parse@next
     \@dc@arguments

\ArgumentCatcher@Pm@iii:
macro:#1#2#3->\@dc@addargum {{#1}{#2}{#3}}\@dc@parse@next
     \@dc@arguments

\ArgumentCatcher@m@iv:
macro:#1#2#3#4->\@dc@addargum
     {{#1}{#2}{#3}{#4}}\@dc@parse@next \@dc@arguments

\ArgumentCatcher@Pm@iv:
macro:#1#2#3#4->\@dc@addargum
     {{#1}{#2}{#3}{#4}}\@dc@parse@next \@dc@arguments

\ArgumentCatcher@m@v:
macro:#1#2#3#4#5->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}}\@dc@parse@next \@dc@argu
ments

\ArgumentCatcher@Pm@v:
macro:#1#2#3#4#5->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}}\@dc@parse@next \@dc@argu
ments

\ArgumentCatcher@m@vi:
macro:#1#2#3#4#5#6->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}{#6}}\@dc@parse@next \@d
c@arguments

\ArgumentCatcher@Pm@vi:
```

```
macro:#1#2#3#4#5#6->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}{#6}}\@dc@parse@next \@d
c@arguments

\ArgumentCatcher@m@vii:
macro:#1#2#3#4#5#6#7->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}{#6}{#7}}\@dc@parse@ne
xt \@dc@arguments

\ArgumentCatcher@Pm@vii:
macro:#1#2#3#4#5#6#7->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}{#6}{#7}}\@dc@parse@ne
xt \@dc@arguments

\ArgumentCatcher@m@viii:
macro:#1#2#3#4#5#6#7#8->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}}\@dc@pa
rse@next \@dc@arguments

\ArgumentCatcher@Pm@viii:
macro:#1#2#3#4#5#6#7#8->\@dc@addargum
     {{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}}\@dc@pa
rse@next \@dc@arguments
```

```
\def\ArgumentCatcher@m@ix
```

And it doesn't need to be long and launch inner short macro because there's nothing more to catch.

```
#1#2#3#4#5#6#7#8#9{%
  \@dc@addargum{%
    {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
  \the␣\@dc@arguments
}

\long\def\ArgumentCatcher@Pm@ix
#1#2#3#4#5#6#7#8#9{%
  \@dc@addargum{%
    {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
  \the␣\@dc@arguments
}

\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@DefParseNext{#2}#1}

\@dc@AliasPLower␣K

\long\def\gmu@twostring#1#2{\string#1\string#2}

\long\def\ArgumentCatcher@G@longorshort
#1% longness prefix (\long or nothing)
#2% left and right delimiters
#3% default value
```

```
#4% the tail of args parser.
\@dc@arguments␣% tail delimiter
{%
  \edef\@dc@Gname{ArgumentCatcher@G\gmu@twostring#2}%
  \gmu@if␣{csname}{\@dc@Gname␣\endcsname}%
```

If catcher for this particular pair of delimiters is defined, we don't repeat definition (we assume nobody else would define such a csname).

```
  {}%
```

If it's undefined, we define: `\ArgumentCatcher@G`⟨*stringed delimiters*⟩

```
  {#1\Name\def{\@dc@Gname␣\@xa}%
    \@dc@Gparams#2{\@dc@addargum{{##2}}##1\@dc@arguments}%
  }%
  \@xa\@ifEUnextchar\@firstoftwo#2%
  {\def\dc@parse@next{#4}% we hide possible \pars.
    \csname␣\@dc@Gname␣\endcsname\dc@parse@next}%
  {\@dc@addargum{{#3}}#4\@dc@arguments}%
}

\long\def\@dc@Gparams#1#2{##1#1##2#2}
```

This macro 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.

```
\lpdef\ArgumentCatcher@G␣% short general catcher
#1% left and right delimiters
#2% default value
{\ArgumentCatcher@G@longorshort{}{#1}{#2}}

\lpdef\ArgumentCatcher@PG␣% long general catcher
#1% left and right delimiter
#2% default value
{\ArgumentCatcher@G@longorshort{\long}{#1}{#2}}
```

Now the "canonical" catchers as special cases of `G`:

```
\def\@dc@DefAsGeneral
#1% specifier
#2% delimiters
{\Name\lpdef{ArgumentCatcher@\strip@bslash{#1}}%
  ##1% default value
  {\ArgumentCatcher@G{#2}{##1}}%
```

And the long version:

```
  \Name\lpdef{ArgumentCatcher@P\strip@bslash{#1}}%
  ##1% default value
  {\ArgumentCatcher@G{#2}{##1}}%
  \gmu@if␣{cat}{a\@nx#1}%
  {\@dc@LowerAliases␣#1}{}%
}

\@dc@DefAsGeneral␣A{<>}

\@dc@DefAsGeneral␣O{[]}
```

For legacy reasons we treat the `()` argument differently: the ancient xparse processed a parenthesised argument to a pair of braces splitting it on a comma. We leave this as a possibility with the `\DCcoordinate` declaration:

```
\def\DCnocoordinate{\@dc@DefAsGeneral␣C{()}}
\DCnocoordinate

\edefU\DCcoordinate{%
  \long\def\ArgumentCatcher@C
  #1% default value
  #2% tail of parsers
  \@dc@arguments␣% tail's delimiter
  {%
    \@ifEUnextchar(%
    {\dc@DefParseNext{#2}\ArgumentCatcher@c@}%
    {\@dc@addargum{{#1}}#2\@dc@arguments}%
  }%
  \let\ArgumentCatcher@c\ArgumentCatcher@C
  \let\ArgumentCatcher@Pc\ArgumentCatcher@PC
  \long\def\ArgumentCatcher@PC#1#2\@dc@arguments{%
    \@ifEUnextchar(%
    {\dc@DefParseNext{#2}\ArgumentCatcher@Pc@}%
    {\@dc@addargum{{#1}}#2\@dc@arguments}%
  }%
  \def\ArgumentCatcher@c@(#1){%
    \gmu@ifxany,{#1}%
    {\@xa\@dc@addargum\@dc@commasep#1\@dc@commasep}%
    {\@dc@addargum{{#1}}}\@dc@parse@next\@dc@arguments
  }%

  \def\@dc@commasep#1,#2\@dc@commasep{{{{#1}{#2}}}}%

  \long\def\ArgumentCatcher@Pc@(#1){%
    \gmu@ifxany,{#1}%
    {\@xa\@dc@addargum\@dc@commasep#1\@dc@commasep}%
    {\@dc@addargum{{#1}}}\@dc@parse@next\@dc@arguments
  }%
}
```

2010/7/21 A scope prefix argument

```
\def\@dc@Scope@shipout#1{%
```

The arg. is the default value (if argument not provided).

```
  \ifx\@dc@seQuence\@empty\def\@dc@seQuence{#1}\fi
  \@xa␣␣\gmu@ifxany\@xa␣\global\@xa{\@dc@seQuence}%
  {\@dc@addargum{{\global}}}%
  {\@dc@addargum{{\relax}}}%
  \@dc@parse@next\@dc@arguments
}

\def\ArgumentCatcher@Scope
#1% the separator(s)
{\let\@dc@seQuence@shipout@@\@dc@seQuence@shipout
  \let\@dc@seQuence@shipout\@dc@Scope@shipout
  \ArgumentCatcher@Q{\global\relax}{\relax}%
```

```
  \let\@dc@seQuence@shipout\@dc@seQuence@shipout@@
  \@dc@ignorearg
  \ArgumentCatcher@T{#1}{\NoValue}%
}
```

```
\@dc@AliasCatcher␣{PScope}␣\Scope
```

[End of Argument Specifiers' Definitions]

```
\def\NoValue{-NoValue-}
```

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

```
  \@xa\ifx\@xa\NoValue\@firstofmany#1\@empty\@nil␣\afterfi%
      \@firstoftwo
  \else\afterfi\@secondoftwo
  \fi}
```

```
\long\def\IfNoValueT#1#2{\IfNoValueTF{#1}{#2}\@empty}
```

```
\long\def\IfNoValueF#1#2{\IfNoValueTF{#1}\@empty{#2}}
```

```
\long\def\IfValueTF#1#2#3{\IfNoValueTF{#1}{#3}{#2}}
```

```
\long\def\PutIfValue#1{\IfValueT{#1}{#1}}
```

\IfValueT
```
\let\IfValueT\IfNoValueF
```

\IfValueF
```
\let\IfValueF\IfNoValueT
```

```
\long\pdef\IfLong#1{%
      % #1 the argument to check for \par,
      % #2 what if \par found,
      % #3 what if \par not found.
  \edef\gmu@IfLong@resa{\detokenize{#1}}%
  \edef\gmu@IfLong@resa{%
    \IfAmong\string\par\@nx\among{\gmu@IfLong@resa}}%
  \gmu@IfLong@resa}
```

= (\afterassignment) pseudo-argument

```
\long\@namedef{ArgumentCatcher@=}%
#1% register used in the assignment
#2% the tail of args parser.
\@dc@arguments{%
  \dc@DefParseNext{%
    \@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=}{ArgumentCatcher@=}
```

\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\def\ArgumentCatcher@gobblespace
```

```
{\ArgumentCatcher@T␣{\dc@gobblespace@dummy}{\NoValue}}
\let\ArgumentCatcher@Pgobblespace\ArgumentCatcher@gobblespace
\def\dc@gobblespace@dummy{\dc@gobblespace@dummy}
\let\gobblespace\dc@gobblespace@dummy
```

end of catchers' definitions

```
\errorcontextlines=100
\long\def\@dc@InnerName#1{%
  \bslash@or@ac{#1}%
  \bslash
}%
\long\def\@dc@InnerEndName␣#1{% version for the end-macros of environments
  \bslash␣end\strip@bslash{#1}%
}%
```

Name of the CS carrying the command's specifiers sequence and arity of the inner macro.

(2010/07/26, v0.993:) This strikingly simple idea of storing the specifiers' sequence in a macro came to my head only after some three years of developing `\DeclareCommand`

```
\long\edef\@dc@SpecsArName#1{%
  Specs␣\string&␣arity␣of
  \@nx\bslash@or@ac{#1}%
}%
\pdef\@dc@messages␣{%
  \gmu@notif␣{@dc@quiet@}{}%
  {%
    \gmu@if␣{DCMessages}{}%
    {\PackageInfo{gmcommand}{^^J%
        Parsing␣arguments␣for␣\dc@innername^^J%
        [specified␣as:
        \unexpanded
        \@xa\@xa\@xa␣\@xa\@xa\@xa␣\@xa{%
          \@xa\@xa\@xa\@secondofmany
```

not `\@secondofthree` because first use of this definition is for `\DeclareCommand` not having its specifiers' carrier.

```
          \csname␣\@dc@specsname\endcsname␣{}{}\@nil␣}%
        \space␣]^^J%
        [in␣file:␣\@currname.\@currext\space␣]}% of info message
    }% of if DC messages
    {}% of if not DCMessages
  }% of if not @dc@quiet@
  {}% of if not not @dc@quiet@
}% of \@dc@messages
```

(2010/07/15, v0.993:) added to modify naming of the inner macro of a `\DeclareCommand`ed commands: preceding the names with a backslash is OK for the letter CS'es, but it may cause a collision for the active chars. Therefore we both precede the name *and* succede it with a bslash

```
\edefU\@dc@DCbody␣{%
{%
  \edef\dc@innername␣{\@dc@InnerName␣#1}% we'll use it in K-type arg. catcher
       and to allow #1 == \par ( \PackageWarning is short and \typeout
       too!). And in \lostwax.
```

We define the name of the macro carrying specs and inner arity:

```
  \edef\@dc@specsname␣␣{\@dc@SpecsArName{#1}}%
  \gmu@ifCSdefined␣{#1}%
  {\gmu@if{DCMessages}{}%
    {\PackageWarning{gmcommand}{%
        ^^J%
        Redefining␣command␣\dc@innername\space
        with␣\string\DeclareCommand}%
    }%
    {}%
  }%
\gmu@if  {\gmu@if␣{DCMessages}{}%
    {\PackageInfo{gmcommand}{^^J%
\dc@innername        \string\DeclareCommand-ing␣\dc@innername\space␣
                 (new)^^J}%
    }%
    {}%
  }%
```

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
```

Now we collect the prefixes.

```
  \emptify\@dc@Specs@tempa
  \gmu@ifsbintersect{#2}{\long!lL}{%
    \addtomacro\@dc@Specs@tempa{\long}%
    \@dc@alllong@true}%
  {}%
  \gmu@ifsbintersect{#2}{.qQ\quiet}{%
    \addtomacro\@dc@Specs@tempa{\quiet}%
    \@dc@quiet@true}%
  {}%
  \gmu@ifsbintersect{#2}{iIwW\sphack}%
  {% if 'invisibility' is specified:
    \addtomacro\@dc@Specs@tempa{\sphack}%
    \def\@dc@bsphack@{\@nx\@bsphack}%
    \def\@dc@esphack@{\@nx\@esphack}%
  }%
  {\emptify\@dc@bsphack@
    \emptify\@dc@esphack@}% if 'invisibility' is not specified.
  \gmu@ifsbany\envhack{#2}%
  {% but if we define an environment:
    \addtomacro\@dc@Specs@tempa{\envhack}%
    \emptify\@dc@bsphack@
    \emptify\@dc@esphack@
```

```
    \edef\@dc@setThrowArgs{%
      \dc@ThrowArgs{\string#1}}% if our command is used as an environ-
                                  ment, we throw the args to the end in a toks register (primary version
                                  defined a macro but that failed for #es).
  }%
  {% and if we don't define an environment:
    \emptify\@dc@setThrowArgs
  }%
  \relaxen\@dc@global@
  \relaxen\@dc@outer@
  \gmu@ifsbany\outer{#2}%
  {\addtomacro\@dc@Specs@tempa{\outer}%
    \let\@dc@outer@\outer}%
  {}%
  \gmu@ifsbany\global{#2}{%
```

We store the information about globalness of definition for the future but in an inactive form (globalness seems to me something very local)

```
    \addtomacro\@dc@Specs@tempa␣{\IAmDeclaredGlobally}%
    \let\@dc@global@\global}%
  {}%
```

Now the possible prefixes are processed.
We add the (distilled version of) them to the storage macro as its first item:

```
  \@xa\Name␣\@xa\edefU
  \@xa\@dc@specsname
  \@xa{\@xa{\@dc@Specs@tempa}}%
```

We initialise the scratch count and toks registersa and the Boolean switches before parsing of the arguments specifiers.

```
  \@dc@ResetParseAuxilia
```

We parse the arguments' specifiers:

```
  \@dc@ParseNextSpecifier␣#3%
  \@dc@buckstopshere␣% it's the sentinel of parsing. Since the test performed by
                       \@dc@ParseNextSpecifier is a comparison of (bslash-stripped) strings,
                       this CS doesn't have to be defined or have any particular meaning.
  \@xa␣\Name␣\@xa␣\addtomacro␣\@xa␣\@dc@specsname
  \@xa␣{\@xa{\@dc@ParsedSpecs}}%
```

We add the inner arity (braced) to the specs-carrying macro

```
  \@xa\Name\@xa\addtomacro\@xa\@dc@specsname
  \@xa{\@xa{\the␣\c@dc@argnum}}%
```

Now we define the basic macro:

```
  \@dc@outer@\@dc@global@␣% possibly the prefixes,
```

Here comes the definition of the coating macro, named the same as the command:

```
  \protected\edef#1{% always \protected ;-)
    \@dc@bsphack@␣% perhaps \@bsphack
    \@dc@messages
    \@nx\@dc@{\the\toks@}% in the braces appear the argument catchers.
```

```
      \@xanxcs{\dc@innername}%
      \ifx\@dc@outer@\outer
        \@xanxcs{\@xa\gobble\string#1␣}% 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
    }% of main coating macro's \edef.
    \edef\gmu@DeclareCommand@resa{%
      \@dc@global@␣% perhaps \global
      \long\def\@xanxcs{\@dc@InnerName{#1}}% for
            a command \command, define \\command\
      \@xau\@dc@innerhashes␣% it's #1#2#3...
      {% the body of the inner '\\⟨name⟩\' macro:
        \@dc@setThrowArgs
        \unexpanded{#4}%
        \@dc@esphack@}%
    }% of \edef.
    \gmu@DeclareCommand@resa
  }% of \DeclareCommand's body.
}

\def\@dc@DCprefixQlist{\global\protected\outer\long
    \relax␣% for the very tricky case when \@dc@global may be relaxed (in \De¦
          clareEnvironment). If someone might want to redefine a cs let to relax, it
          anyway comes as the first argument. And \relax is not a proper value for
          the third argument (args. spec.) so it's all right at this point. (2010/6/10)
    !lL.qQiIwW\sphack\envhack}

\long\def\ShowCommand␣#1{%
    \@iwruJ
    \@iwru{»\string#1«␣is␣»\meaning#1«}%
    \@iwruJ
    \@iwru{»\Name\string{\@dc@InnerName{#1}}«␣is
      »\Name\meaning{\@dc@InnerName{#1}}«}%
    \@iwruJ
    \@iwru{»\Name\string{\@dc@SpecsArName{#1}}«␣is
      »\Name\meaning{\@dc@SpecsArName{#1}}«}%
    \@iwruJ
    \show\show
}

\DeclareCommand \@XA{\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@innername macro.

{
    #1␣>Pm␣␣␣␣␣␣␣% command to be defined (may be even \par if you really wish),
    #2␣>\@xa␣Q{\@dc@DCprefixQlist}{}␣% an optional seQuence 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 'in-
          visible' or W or w as 'white' or \sphack to make the command 'invisible'
```

in the leading with the \@bsphack… \@esphack. To deal with an 'invisible' environment, there is one more acceptable value of this argument: \envhack, which suppresses placing \@bsphack—\@esphack and places \@ignore@true instead in the end macro,

```
7484    #3␣>Pm␣%  arguments specification (may contain \par),
7485    #4␣>Pm␣%  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⟩.
7489 }}\@dc@DCbody
```

Now the inner body of \DeclareCommand is full-featured but its specifiers' carrier is not yet defined. Therefore we repeat its definition with the almost-fullfeatured version. Now it's the fixed point.

\DeclareCommand

```
7501 \@XA{\DeclareCommand␣\DeclareCommand␣\long
7502   {m␣>\@xa␣Q{\@dc@DCprefixQlist}{}␣mm}}%
7503 \@dc@DCbody
```

```
7512 \long\def\dc@EAName#1{%  Env. arguments' toks' name It will be passed a stringed
                               or bslashless name of a command
7514   \if\bslash#1\else#1\fi's_args}
```

```
7516 \pdef\dc@ThrowArgs#1{%
7517   \gmu@if{csname}{\dc@EAName{#1}\endcsname}
7518   {}%  if it's defined, we do nothing (we assume it's defined by us and is toks regis-
             ter.)
7522   {%  else we define it as a new toks
7524     \@xa\newtoks\csname\dc@EAName{#1}\endcsname
7525   }%
7526   \csname\dc@EAName{#1}\endcsname=%
7527   \@xa\@xa\@xa{\@xa\@gobble\the\@dc@arguments}%
7528 }%
```

Now we have a convenient tool to implement parsing of a Knuthian argument(s).

Knuthian argument's default replacement

```
7536 \def\@dc@K@defaultrepl{##1}
```

2009/11/22 inner pair of braces removed (as leading to additional group causing errors).

\@dc@define@K

```
7541 \DeclareCommand\@dc@define@K␣\long␣{mb}{%
```

```
%%   \@dc@xadefaultfalse First we add the particular CS to the toks register:
7544   \edef\gmu@tempa{%
7545     \unexpanded{\toks@\@xa}%
7546     {\@nx\the\toks@{%
7547         \@xanxcs{\dc@innername␣@K\the\c@dc@catchernum}%
7548       }%
7549     }%
```

here we add to the parsers toks list a CS named \⟨*our command*⟩@K⟨*num. of catcher*⟩ (the particular Knuthian catcher)

```
7552   }\gmu@tempa
```

and define this macro:

```
  \edef\gmu@tempa{%
```

And now we define that particular Knuthian catcher.

```
    \def\@xanxcs{\dc@innername␣@K\the\c@dc@catchernum}%
    \gmu@if␣{@dc@xadefault@i@}{}%
    {\@xau{#1}}%
    {\unexpanded{#1}}%
    {%
      \@nx\@dc@addargum{{%
          \IfValueTF{#2}{%
            \gmu@if{@dc@xadefault@ii@}{}%
            {\@xau{#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@ParseNextSpecifier␣#3⟨buck⟩.
  \if␣P\@dc@long@letter\relax\long\fi
  \gmu@tempa
```

Now we clean up and continue construction of arguments parser.

```
 \@dc@ParseNextSpecifier
}% of \@dc@define@K
```

\@dc@grab@Loop

```
\DeclareCommand\@dc@grab@Loop␣\long␣{%
  #1␣T{\any␣\none␣Uu\U␣\u␣ \␣\ ␣Ww\W␣\w␣\lostwax}␣% the "sign" of matching for adding; use of the letters suppresses adding them to the parsed specifiers' list, which is crucial for \SameAs copying of specifiers: the "until" and "while" loops don't shift their arguments.
  #2␣>Pm␣% list of tokens to match for/against iteration
  >i␣T{\drain␣\drop␣\Drain␣\Drop}
  #3␣T{\any␣\none␣}{\none}␣% the "sign" of matching for draining
  #4␣b{}␣% list of drainers, empty by default
  >i␣T{\count␣\ubound}
  #5␣D{\m@ne}␣% upper bound of iterations
  >i␣T{\default␣\Default}
  #6␣b␣% default value (\NoValue by default)
  #7␣T{\eacher␣\Eacher␣\defEacher}␣% default is \NoValue and that's no harm since we test if #7 == \defEacher.
  #8␣B{}␣% eacher, empty by default
}{%
  \@dc@process@matchsign␣{#1}␣{##1}%
  \dc@addtoParsed@{#2}%
  \gmu@if␣{strings}{\lostwax#1}%
  {\gmu@if␣{@dc@xadefault}{}%
    {\@xa\edefU\@xa\@dc@LostWaxList\@xa{#2}}%
    {\edefU\@dc@LostWaxList{#2}}%
  }%
  {}%
```

```
  \@dc@process@matchsign {#3} {##3}%
  \dc@addtoParsed@{#4}%
  \@xa \@dc@addtospecs@bare \@xa{\the\numexpr #5}%
  \@dc@addtotoks@bare{{#5}}% to ignore possible expandafter.
  \dc@addtoParsed@ {#6}%
  \gmu@if {strings}{\defEacher#7}
  {%
    \Name\def{\dc@innername/defdEacher/\the\c@dc@argnum}%
    ##1{#8}%
    \Name \dc@addtoParsed@
    {\dc@innername/defdEacher/\the\c@dc@argnum}%
  }
  {\gmu@if {@dc@xaeacher@}{}
    {\@dc@xadefaulttrue}{\@dc@xadefaultfalse}%
    \dc@addtoParsed@ {#8}%
  }%
  \gmu@if {strings}{\lostwax#1}
  {\grab@LostWax}
  {\@dc@ParseNextSpecifier}%
}
```

`\grab@LostWax`

```
\DeclareCommand\grab@LostWax {
  >iT{\lost \Lost}
  #1 b{\lostwax@NoValue}
  >iT {\count \ubound}
  #2 D{1}
  >iT{\endlost \EndLost}
}{%
```

We build an ignored “while” catcher.

```
  \def\@dc@LostWaxA{>iw}%
  \gmu@ifdetokens {\lostwax@NoValue}{#1}
  {%
```

if no particular value of the “lost wax” is provided, we assume the same as the “until” delimiter(s).

```
    \prependtomacro\@dc@LostWaxA{>\@xa}%
    \addtomacro\@dc@LostWaxA{\@dc@LostWaxList}%
  }%
```

Otherwise first we check whether “lost wax” provided has nonempty intersection with the “until” delimiters.

```
  {%
    \@xa\gmu@ifstrintersect
    \@xa{\@dc@LostWaxList}{#1}
    {\addtomacro\@dc@LostWaxA{{#1}}}%
    {\PackageError{gmutils/gmcommand}{^^J%
       You try to declare a "Lost Wax" catcher^^J%
       with the "Lost" list disjoint with the "Until"
            list.^^J%
       I assume sum of them}%
     {}%
     \@xa \addtomacro\@xa\@dc@LostWaxA
```

```
      \@xa{\@xa{\@dc@LostWaxList␣#1}}%
    }%
 }%
 \addtomacro\@dc@LostWaxA{␣\count␣#2␣\relax}%
 \@xa\@dc@ParseNextSpecifier␣\@dc@LostWaxA
}% of \grab@LostWax.

\def\@dc@process@matchsign
#1% the argument
#2% its numeral in human language (for the error message)
{%
  \gmu@CASE␣{strings}␣{{#1}\any␣}
  {\@dc@addtoParsed@bare␣{\any␣}}%
  \gmu@CASEsbany␣{#1}␣{Ww}
  {\@dc@addtotoks@bare␣{\any}}%
  \gmu@CASE␣{strings}␣{{#1}\none}
  {\@dc@addtoParsed@bare␣{\none␣}}%
  \gmu@CASEsbany␣{#1}␣{Uu \lostwax␣}
  {\@dc@addtotoks@bare␣{\none␣}}%
  \gmu@lastCASE
  {\PackageError{gmcommand}{%
      You␣*have*␣to␣declare␣»\string\any«␣or␣»\string\none«␣
           ^^J%
      as␣the␣#2␣argument␣for␣the␣»\string\loop«␣catcher}{}%
  }%
  \gmu@ESAC
}

\long\pdef\UnDeclareCommand#1{%
  \let#1\@undefined
  \ifcsname\@dc@InnerName{#1}\endcsname
    \n@melet{\@dc@InnerName{#1}}{@undefined}%
  \fi
  \ifcsname\@dc@SpecsArName{#1}\endcsname
    \n@melet{\@dc@SpecsArName{#1}}{@undefined}%
  \fi
}
```

### The \SameAs parsing

Implemented 2010/4/14–16. Lets you not to repeat all the complicated specifiers but just write \SameAs\⟨*another command*⟩

```
\edef\gmu@tempa{%
  \def\@nx\gmu@ifHashless@##1%
  \detokenize{macro:}##2->##3\@nx\@nil}%
\gmu@tempa
```

This produces \gmu@ifHashless@#1macro:#2->#3\@nil with the middle delimiter detokenized

```
{\gmu@if␣{}{\gmu@ifempty{#2}{1}{0}\gmu@ifempty{#3}{2}{1}}}
```

This expands to 1 if #1 is a hashless macro (to be honest, if its meaning contains string macro:->)

```
\long\edef\gmu@ifHashlessMacro
```

```
#1% a CS, name or active char
  %% #2% (implicit) what if hashless
  %% #3% (implicit) what if hashy
{%
  \edef\@nx\gmu@WHL@arg{\@nx\@xanxtri{#1}}%
  %% \unexpanded{\typeout{@@@@@ »\meaning\gmu@WHL@arg«}}%
  \unexpanded{\@xa\@xa\@xa\gmu@ifHashless@
    \@xa\meaning\gmu@WHL@arg}%
  \relax␣% to ensure that #3 of \gmu@ifHashless@ is nonempty if match for the
        delimiters is found earlier
  \detokenize{macro:***->}\@nx\@nil␣% the sentinels
}

\long\def\gmu@ifDeclared#1{%
% #1 a CS or csname or an active char,
% #2 true branch,
% #3 false branch
  \let\gmu@ifDe@next\@secondoftwo
```

The test is three-level: first we check whether the specifiers' carrier is defined, then whether the inner macro is defined, then we check if the outer CS is a hashless macro (it could have been that the outer CS was redefined with sth. else)

```
  \gmu@CASEnot␣{csname}␣{\@dc@SpecsArName{#1}\endcsname}%
  {}%
  \gmu@CASEnot␣{csname}{\@dc@InnerName{#1}\endcsname}%
  {}%
  \gmu@EatCases
  {\gmu@ifHashlessMacro{#1}%
    {\edef\gmu@ifDe@resa{%
        \@nx\gmu@ifxany\@xanxcs{\@dc@InnerName{#1}}%
        {\unexpanded\@xa\@xa\@xa{\gmu@WHL@arg}}%
```

Above: to get the contents of the outer macro we use the macro carrying conversion of possible name into a CS and expand it twice. Then we freeze it with `\unexpanded`.

```
        {\let\@nx\gmu@ifDe@next\@nx\@firstoftwo}% if the inner macro
              is present in the contents of #1, we take the first implicit argument
        {}% otherwise we do nothing (taking the second implicit is already set)
      }% of edef
      \gmu@ifDe@resa
    }% of if a hashless macro
    {}% of if not a hasless macro
  }%
  \gmu@ESAC
  \gmu@ifDe@next
}% of \gmu@ifDeclared

\long\def\@dc@SameAs#1{%
```

As you see, it's very simple when we took care of storing the specifiers in a special macro: we only take the middle brace of it.

```
  \gmu@ifDeclared␣{#1}%
```

```
    {%
```

7 \expandafters before \@dc@ParseNextSpecifier and 3 before \@secondofthree to expand the specifiers carrier twice, then get its middle piece of data, which is the specifiers' sequence, and then parse as a part of "our" specifiers.

```
      \@xa\@xa\@xa␣\@xa\@xa\@xa␣\@xa
      \@dc@ParseNextSpecifier
      \@xa\@xa\@xa␣\@secondofthree
      \csname␣\@dc@SpecsArName{#1}\endcsname
    }%
    {%
      \PackageError{gmcommand}{%
        Command␣\@nx#1␣^^J%
        is␣not␣declared␣with␣\DeclareCommand.␣^^J%
        I␣can't␣repeat␣its␣parameters␣in␣current␣command␣
            declaration}%
      {}%
    }%
}% of \@dc@SameAs
```

\DeclareCommand \DCUse

```
\DeclareCommand\DCUse{
  #1␣␣>Pm␣% the command
}{%
```

TODO: handling longness of the arguments properly. (Now it assumes all the inner arguments are \long).

```
  \gmu@ifDeclared{#1}%
```

If #1 is a Declared command, we put its inner macro to the \@dc@arguments toks and launch catcher of proper number of undelimited arguments.

```
  {%
    \@dc@arguments\@xa{\@xa
      \csname␣\@dc@InnerName{#1}\endcsname
    }%
    \csname␣ArgumentCatcher@Pm@\romannumeral
    \@xa␣\@xa␣\@xa␣\@thirdofthree
    \csname␣\@dc@SpecsArName{#1}\endcsname␣% of arity carrier
    \endcsname␣% of the catcher
  }%
  {\DC@EndNotDeclaredErr{#1}}%
}%

\pdef␣\DC@EndNotDeclaredErr␣#1{%
  \PackageError{gmcommand}{%
    Command␣\bslash␣end\strip@bslash{#1}␣^^J%
    is␣not␣declared␣with␣\DeclareCommand.␣^^J%
    Therefore␣I␣can't␣use␣its␣DC-inner␣macro␣(because␣of␣its␣
        nonexistence).^^J}%
  {}%
}
```

\DeclareCommand \DCUseEnd

```
\DeclareCommand\DCUseEnd{
  #1␣␣>Pm␣% the command (without "end")
}{%
```

```
    \gmu@ifDeclared{end\strip@bslash{#1}}%
    {%
      \@dc@arguments\@xa{\@xa
        \csname␣\@dc@InnerEndName{#1}\endcsname
      }%
      \csname␣ArgumentCatcher@Pm@\romannumeral
      \@xa␣\@xa␣\@xa␣\@thirdofthree
      \csname␣\@dc@SpecsArName{#1}\endcsname␣% of arity carrier
      \endcsname␣% of the catcher
    }%
    {\DC@EndNotDeclaredErr{#1}}%
}%
```

end of `\SameAs` parsing

#### Immediate uses

`\DeclareEnvironment`

```
\DeclareCommand\DeclareEnvironment␣\long
{
  #1--#4␣␣\SameAs␣\DeclareCommand
```

We repeat the specifiers, but the role of `#4` is slightly different here: here it's the `\begin` part definition.

```
  #5␣m␣% the end definition; it can contain any #⟨n⟩—the arguments of the environment are passed to it, too.
}{%
  \def\gmu@DeclareEnvironment@resa{\envhack}% we add the information we define an environment.
  \IfValueT{#2}%
  {\addtomacro\gmu@DeclareEnvironment@resa{#2}}% we pass all the 'prefixes' to \DeclareCommand

  \@xa\DeclareCommand\csname#1\@xa\endcsname
  \gmu@DeclareEnvironment@resa{#3}{#4}%
```

Now the begin definition is done. Let's get down to the end definition.

```
 \let\@dc@env@endglobal=\@dc@global@␣% note this CS was for sure redefined by the last \DeclareCommand (and by nothing else) and that's exactly what we want.
```

(2010/07/15, v0.993:) we make the `\end…` command `\DeclareCommand`ed for the symmetry, for simplifying definition of `\envirlet` in particular

```
  \edef\gmu@DeclareEnvironment@resb{%
    \@nx\@xa
    \@xanxcs{\bslash␣end#1}%
    \@nx\the
    \@xanxcs{\dc@EAName{#1}}%
  }% of \edef. It reuslts in
```

`\@xa␣\\end⟨name⟩␣\the\\⟨name⟩'s_args`

which does not collide with the inner macro of the `\end…` command, since the latter is `\\end…\`.

```
\gmu@if{csname}{\dc@EAName{#1}\endcsname}{\gmu@namelet%
      \global{\dc@EAName{#1}}{@undefined}}{}%
```

We postpone the definition of the \end... command after of the end inner macro not to spoil the \@dc@innerhashes.

Now the end inner macro

```
  \edef\gmu@DeclareEnvironment@resa{%
    \@dc@global@\long\def␣% the inner end macro is always \long and perhaps defined \global ly.
    \@xanxcs{\bslash␣end#1}%
    \@xau\@dc@innerhashes␣% 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...@resa.
  \gmu@DeclareEnvironment@resa
```

And now the postponed from above definition of \end...

```
  \edef\gmu@DeclareEnvironment@resc{%
    \DeclareCommand\@xanxcs{end#1}%
    \@dc@env@endglobal
    {}% no parameters
    {\@xau␣␣␣␣{\gmu@DeclareEnvironment@resb
        \@ignoretrue}%
    }%
  }%
  \gmu@DeclareEnvironment@resc
}% of \DeclareEnvironment
```

\@xanxcs

```
\DeclareCommand\NewCommand
{\SameAs\DeclareCommand}
{%
  \gmu@ifCSdefined␣{#1}
  {\DCUse\DeclareCommand#1{#2}{#3}{#4}}%
  {\PackageError{gmcommand}%
    {Command␣\@nx#1␣already␣defined␣\on@line!}%
    {}%
  }%
}
```

\NewCommand \gmu@ifCSdefined \PackageError

```
\DeclareCommand\RenewCommand
{\SameAs\DeclareCommand}
{%
  \gmu@ifCSdefined␣{#1}
  {\DCUse\DeclareCommand#1{#2}{#3}{#4}}%
  {\PackageError{gmcommand}%
    {Command␣\@nx#1␣not␣yet␣defined␣\on@line!}%
    {}%
  }%
}
```

\RenewCommand \gmu@ifCSdefined \PackageError

```
\DeclareCommand\ProvideCommand
{\SameAs\DeclareCommand}
{%
  \gmu@ifCSdefined␣{#1}
  {}
  {\DCUse\DeclareCommand#1{#2}{#3}{#4}}%
```

\ProvideCommand \gmu@ifCSdefined

```
}
```

\DeclareCommand
\NewEnvironment

```
\DeclareCommand\NewEnvironment
{\SameAs\DeclareEnvironment}
{%
  \gmu@ifdefined{#1}
  {\DCUse\DeclareEnvironment{#1}{#2}{#3}{#4}{#5}}%
  {\PackageError{gmcommand}
    {Environment␣#1␣already␣defined␣\on@line!}{}%
  }%
}
```

\RenewEnvironment

```
\DeclareCommand\RenewEnvironment
{\SameAs\DeclareEnvironment}
{%
  \gmu@ifdefined␣{#1}
  {\DCUse\DeclareEnvironment{#1}{#2}{#3}{#4}{#5}}%
  {\PackageError{gmcommand}
    {Environment␣#1␣not␣yet␣defined␣\on@line!}{}%
  }%
}
```

### Setting for XeTeX

which could not be defined earlier (in gmbase because of lack of \DeclareCommand.

\XeTeXthree

```
\DeclareCommand\XeTeXthree{o}{%
  \@ifXeTeX{%
    \IfValueT{#1}{\PassOptionsToPackage{#1}{fontspec}}%
    \@ifpackageloaded{gmverb}%
    {%
      \Store@Macro\verb
      \StoreEnvironment{verbatim}%
      \StoreEnvironment{verbatim*}%
    }{}%
    \RequirePackage{xltxtra}% since v 0.4 (2008/07/29) this package redefines \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.
    \@ifpackageloaded{gmverb}%
    {%
      \Restore@Macro\verb
      \RestoreEnvironment{verbatim}%
      \RestoreEnvironment{verbatim*}%
    }{}%
    \AtBeginDocument{%
      \@ifpackageloaded{gmlogos}{%
        \Restore@Macro\LaTeX\Restore@MacroSt{LaTeX␣}% my version of the LaTeX logo has been stored just after defining, in line 9657.
        \Restore@Macro\eTeX}%
      {}%
    }%

    \pdef\adddefaultfontfeatures##1{%
      \addtomacro\zf@default@options{#1,}}
```

```
  }% of \@ifXeTeX's first argument,
  {}% \@ifXeTeX's second argument,
}% of \XeTeXthree's body.
```

\setspaceskip

```
\DeclareCommand\setspaceskip{%
  A{1}% optional factor for all three components
  O{\fontdimen2\font}%
 >is
 O{\fontdimen3\font}%
 >is
 O{\fontdimen4\font}}
{\spaceskip=\glueexpr\dimexpr#2*#1\relax␣plus\dimexpr␣#3*#1%
     \relax
  minus\dimexpr#4*#1\relax\relax}
```

```
\def\makestarlow{%
  \begingroup\lccode`\~=`\*\lowercase{%
    \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
}%
{%
  \def\gmu@lowstarfake{%
    \leavevmode\vbox{\hbox{*}\kern#1\fontchardp\font`*}%
  }%
}
```

```
\def\gmu@lowstarfake{*}␣% useful for next command where normal star is
    low.
```

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{%
  #1␣␣D{\NoValue}␣% (optional short version (number of \baselineskips))
       Before 2010/9/10 the default value was the D-default, i.e., 0, which lead to
       improper working of this command.
  #2␣␣B{2\baselineskip}% (2) optional (formerly mandatory) long version of
       required room on a page
  >is
  #3␣␣B{}␣% (3) what if the room is enough
  >is
  #4␣B{\newpage}␣% (4) what if there's to little room on a page
}{%
  \gmu@if␣{num}
  {\numexpr
      \ifdim
```

if the space left is less than we wish then 1, otherwise 0.

```
        \gmu@pageremain
        <%
```

```
        \dimexpr
          (\IfValueTF{#1}{#1\baselineskip}{#2})*1%
        \endexpr
```

(2010/06/28, v0.993:) I added |(›…)*1| to assure that | | really delimits the dimexpr

```
        1\else␣0%
      \fi
      *%
      \ifdim␣\gmu@pageremain␣>-1sp
        1\else␣0%
      \fi
      >\z@
     }% of condition
    {\typeout{@@@@@␣not␣enough␣page␣\on@line}%
      #4%
    }%
    {#3}%
}
```

```
\long\def\gmu@LC@LetInners
```

Macro letting the inner (executing) and the specs. carrying macros of a `Declared` command.

```
#1% scope prefix (\global or \relax
#2% left side
#3% right side of the assignment
{%
  \edef\gmu@LC@InnerLeft{\@dc@InnerName{#2}}%
  \edef\gmu@LC@InnerRight{\@dc@InnerName{#3}}%
  \gmu@namelet{#1}{\gmu@LC@InnerLeft}{\gmu@LC@InnerRight}%
```

`\tri@let` put here originally was disastrous since it gobbled the leading bslash

```
  \edef\gmu@LC@InnerLeft{\@dc@SpecsArName{#2}}%
  \edef\gmu@LC@InnerRight{\@dc@SpecsArName{#3}}%
  \gmu@namelet{#1}{\gmu@LC@InnerLeft}{\gmu@LC@InnerRight}%
}
```

```
\long\def\gmu@CL@PrepareArg
#1% left/right
#2% \@xa or \edef, maybe sth. more in the future
#3% a CS, text of a name or an active char
{%
  \Name\let{gmu@CL@#1}\@undefined
  \ifx\@xa#2\Name\edef{gmu@CL@#1}{\@xau{#3}}\fi
  \ifx\edef#2\Name\edef{gmu@CL@#1}{#3}\fi
  \unless\ifcsname␣gmu@CL@#1\endcsname
    \Name\edef{gmu@CL@#1}{%
      \gmu@if{cat}{\@nx~\@nx#3}%
      {\@nx#3}{\strip@bslash{#3}}%
    }%
  \fi
}
```

\CommandLet

```
\DeclareCommand\CommandLet␣\long␣{%
```

```
  \Scope␣% (1)
  T{\@xa\edef}␣% (2) whether left side should be \expandaftered
  m␣% (3) left side of assignment
  >iT{=}␣% pro forma
  T{\@xa\edef}␣% (4) whether right side should be \expandaftered
  m␣% (5) right side of the asssignment
```

Both arguments may be names. Warning! The first token of a csname will not be expanded (will be `\string` ed).

```
}{%
  \gmu@CL@PrepareArg{left}#2{#3}%
  \gmu@CL@PrepareArg{right}#4{#5}%
  \@xa\gmu@ifDeclared\@xa{\gmu@CL@right}{%
    \edef\gmu@LC@resa{%
      \DeclareCommand␣%
      \@xa\@xanxtri\@xa{\gmu@CL@left}%
      #1% scope prefix
      {\@nx\SameAs\@xa\@xanxtri\@xa{\gmu@CL@right}}% args. spec
      {}% the body of the command will be \let via letting inners.
    }%
    \gmu@LC@resa
```

Now we have a command `#2` Declared with emty body and proper args' parsers and inner macro properly named.

```
    \@XA{\@xa\gmu@LC@LetInners\@xa#1%
    \@xa{\gmu@CL@left}}\@xa{\gmu@CL@right}%
```

No Ampulex for the outer macro is necessary.

```
  }% of if Declared
  {% if not Declared, we just \let:
\if\@nx#3ϵ\typeout{@@@@@␣»\@xa\@dc@InnerName\@xa{%
      \gmu@CL@right}«␣is␣not
  Declared?:␣\Name\meaning{\@xa\@dc@InnerName\@xa{%
       \gmu@CL@right}}%
}%
\show\NotDeclared
\fi
    \@XA{\@xa\tri@let\@xa#1%
      \@xa{\gmu@CL@left}}\@xa{\gmu@CL@right}%
  }%
}
```

`\envirlet`

```
\DeclareCommand\envirlet␣\long{\Scope␣mm}{% for \letting environments.
  \CommandLet␣#1{#2}{#3}%
  \CommandLet␣#1{end#2}{end#3}%
}
```

A numeric `for` loop as in decent languages:

`\@fornum`

```
\DeclareCommand\@fornum{%
  m% (1) initial (least) num. value
  m% (2) limit (last iterated over) num. value
  O{1}% (3) step of iteration
  m% (4) body of the loop
```

```
}%
{\edef\gmu@fornum@min{\the\numexpr␣#1}%
  \edef\gmu@fornum@lim{\the\numexpr␣#2}%
  \let\gmu@fornum@curr\gmu@fornum@min
  \@whilenum\gmu@fornum@curr<\gmu@fornum@lim
  \do{#4%
    \edef\gmu@fornum@curr{%
      \the\numexpr\gmu@fornum@curr+#3}%
  }% of \@whilenum's body.
}% of \@fornum.

\pdef\StoreCommand{%
  \begingroup
  \MakePrivateLetters
  \@StoreCommand
}

\long\def\gmu@SC@StorageName
#1% a CS, name or active char
#2% group level
{%
  \gmu@storeprefix/%
  \ifnum\numexpr#2>-1␣\the\numexpr(#2)*1\relax\fi
  \bslash@or@ac␣{#1}%
}%
```

The storing csnames with `#2` $\leq -1$ don't contain the number.

```
\def\gmu@SC@setgrouplevel
#1% +|-|\NoValue
#2% a decimal number
{\edef\gmu@SC@grouplevel{%
    \the\numexpr
    \IfValueTF{#1}%
    {\currentgrouplevel#1#2+1}{#2}%
  }% of the grouplevel-carrying macro
}
```

`\@StoreCommand`

```
\DeclareCommand\@StoreCommand
{%
  #1␣T{+-}% grouplevel shift indicator
  #2␣␣d␣% group level or shift (the latter relative to the current g.l.)
  #3␣␣>Pm% name or CS
}{%
  \endgroup␣% we close the group opened in line 8251.
```

Now. We define a special csname prefixed with `\gmu@storeprefix` and containing current group level. Later, when we refer to the stored command, we'll check subsequent levels, from current upwards.

```
  \gmu@SC@setgrouplevel{#1}{#2}%
  \edef\gmu@SC@resa{%
    \gmu@SC@StorageName{#3}{\gmu@SC@grouplevel}}%
  \let\gmu@SC@scope\relax
  \ifnum\gmu@SC@grouplevel>\currentgrouplevel
    \let\gmu@SC@scope\global
```

```
  \fi
  \@xa\CommandLet\@xa\gmu@SC@scope\@xa{\gmu@SC@resa}{#3}%
}

\pdef\StoreEnvironment{%
  \begingroup
  \MakePrivateLetters
  \@StoreEnvironment
}

\def\@StoreEnvironment#1{%
  \@StoreCommand{#1}%
  \begingroup
  \@StoreCommand{end#1}%
}

\pdef\RestoreEnvironment{%
  \begingroup
  \MakePrivateLetters
  \@RestoreEnvironment
}

\def\@RestoreEnvironment#1{%
  \endgroup
  \RestoreCommand{#1}%
  \RestoreCommand{end#1}%
}

\pdef\UseStored{%
  \begingroup
  \MakePrivateLetters
  \@UseStored}
```

\@UseStored

```
\DeclareCommand\@UseStored␣\long{%
  #1␣␣T{+-}% (1) indicator of grouplevel shift (in distiction from an absolute group level)
  #2␣d␣% (2) optional grouplevel or shift (0 by default)
  #3␣␣0{\@gobble}% (3) stuff to be put before the #4 CS, e.g., an assignment. The deafult value \@gobble makes the default use of a stored CS to be just an execution/expansion of its meaning. Another use of this command, perhaps the most useful, is restoring of the previous meaning, as we'll see later. Note that #4 after #3 will be put in braces.
  #4␣␣m␣% (4) a CS or csname to be taken from the storage liquid
}{%
  \endgroup
  \gmu@SC@setgrouplevel{#1}{#2}%
  \edef\gmu@SC@grouplevel{%
    \the\numexpr\gmu@SC@grouplevel+1}% for the first turn of loop
  \loop
    \edef\gmu@SC@grouplevel{%
```

we decrease the group level number

```
      \the\numexpr\gmu@SC@grouplevel-1}%
    \edef\gmu@SC@currname{%
```

define the cs' storage name for that level

```
      \gmu@SC@StorageName{#4}{\gmu@SC@grouplevel}}%
  \ifnum
```

and check the conjunction of conditions: the group level is >=-1…

```
    \unless\ifnum\gmu@SC@grouplevel<-1␣1\else␣0\fi
```

and the csname remains not defined.

```
    \unless\ifcsname␣\gmu@SC@currname␣\endcsname␣1\else␣0\fi
    =11
  \repeat
```

We repeat until we find defined csname or reach the top level.

```
  \gmu@if{csname}{\gmu@SC@currname\endcsname}%
  {%
    \@XA{#3{#4}}\csname␣\gmu@SC@currname\endcsname}% first
  {%
    #3{#4}\@undefined}% second
}% of \@UseStored
```

\RestoreCommand

```
\DeclareCommand\RestoreCommand{\Scope␣d}{%
  \UseStored␣#2[\CommandLet␣#1*]%
}
```

\gmu@smashbox

```
\newbox\gmu@smashbox
```

\set@smashbox

```
\DeclareCommand\set@smashbox{%
  \Scope
  T{hv}{h}
}{#1\setbox\gmu@smashbox␣=\csname␣#2box\endcsname␣}%

\def\smash@box{\smash{\box\gmu@smashbox}}
```

Now, using the iterating catchers, we define a finder of max/min dimen of a given sequence of arguments.

(There's also the \gmu@comparenum macro defined in gmbase that expands to the smaller or larger numbers (numexprs) of given two.)

\gmu@boxA \gmu@boxB \gmu@dima \gmu@dimb

```
\newbox\gmu@boxA
\newbox\gmu@boxB
\newdimen\gmu@dima
\newdimen\gmu@dimb
```

We call this macro \gmu@extremebox because in the gmbase package we defined \gmu@maxdim and \gmu@mindim as expandable taking a sequence of dim(expr)s and expanding to the biggest/smallest of them.

\gmu@extremebox

```
\DeclareCommand\gmu@extremebox␣{
  #1␣␣m␣% a dimen register (will be assigned the extr. value) or a CS(will be defined as \the extreme value)
  #2␣␣T␣{\min\max}{\max}␣% kind of extreme
  #3␣␣T␣{\wd\ht\dp\totalheight}{\wd}␣% dimension to compare, the last two for total \ht+\dp.
  #4␣␣\lostwax␣{\global\relax}␣\eacher{\gmu@dimextreme@step}␣%
      \lost{\relax}
  #5␣␣T{\global}{}␣% scope of assignment
}{%
  \gmu@if␣x␣{\max#2}
```

```
  {\gmu@dimb=-\maxdimen
    \def\gmu@dimextreme@comp{<}%
  }
  {\gmu@dimb=\maxdimen
    \def\gmu@dimextreme@comp{>}%
  }%
  \long␣␣\def\gmu@dimextreme@step␣##1{%
    \setbox\gmu@boxA=\hbox{##1}%
    \gmu@dima=␣\gmu@if␣x␣{\totalheight#3}
    {\dimexpr␣\ht\gmu@boxA␣+\dp\gmu@boxA\relax}
    {#3\gmu@boxA}%
    \relax
    \gmu@if␣{dim}
    {\gmu@dimb\gmu@dimextreme@comp␣\gmu@dima␣}
    {\gmu@if␣{dim}␣{\gmu@dima>\z@}
      {\gmu@dimb=\gmu@dima␣}
      {}%
    }
    {}%
  }%
  #4%
  \IfIs␣#1\dimen
  {#5#1=\gmu@dimb}
  {#5\edef␣#1{\the\gmu@dimb␣}}%
}%
```

#### Getting height ;-)

The `\gmu@getht` macro is intended for very high vboxes, i.e., whose height may exceed `\maxdimen`. (In such case TEX's dimens cycle which is not what we want in < and > comparisons.) So, this macro checks whether a vbox got "too high" and returns `\maxdimen` as substitute of its height. If given box fits within `\maxdimen` or is not a vbox, then its original height is returned.

```
\pdef\gmu@getht
#1% box register
#2% dimen (or glue) register to assign the (substitute) height to
{%
  \setbox\gmu@boxA=\copy#1\relax
  \ifvbox\gmu@boxA
    \vbadness@M
    \setbox\gmu@boxB=\vsplit\gmu@boxA␣to␣\maxdimen
    \vbadness@Restore
```

chcemy rzeczywiście pomierzyć, a nie być może odrzucić odstępy, kerny i grzywny.

```
    \setbox\gmu@boxB=\vbox{\splitdiscards␣}%
    \ifdim␣\ht\gmu@boxB␣>\z@
      \setbox␣\gmu@boxA=\vbox{
        \unvbox\gmu@boxB
        \unvbox\gmu@boxA
      }%
%% \else % otherwise we leave box A intact for the Test of Śunyata.
    \fi
```

```
  \else
    \setbox\gmu@boxA=\box\voidb@x
  \fi
  \ifvoid\gmu@boxA
    #2=\ht#1\relax
  \else
    #2=\maxdimen
  \fi
}% of \gmu@getht
```

TODO: if one wraps such a very high vbox in an hbox, then attempt of assignment of its height results with an error "Dimension too large". Can we handle it? *Should* we handle it?

#### Enlarging and scaling of the font size

\enlargefsize
```
\DeclareCommand\enlargefsize{
  #1␣␣m␣% enlargement (dim(expr)) for font size
  #2␣␣b␣% enlargement (dim(expr)) for baselineskip (if absent, #1 is used)
}{\edef\gmu@tempa{%
    \@nx\fontsize{\the\dimexpr\f@size␣pt+#1}%
    {\the\dimexpr␣1\baselineskip+␣\IfValueTF{#2}{#2}{#1}}%
  }\gmu@tempa\selectfont
}
```

\scalefsize
```
\DeclareCommand\scalefsize␣{
  #1␣m␣% scale for font size
  #2␣b␣% scale for baselineskip (if absent, #1 is used)
  \gobblespace
}{\edef\gmu@tempa{%
    \@nx\fontsize{\the\dimexpr␣#1\dimexpr\f@size␣pt\relax}%
    {\the\dimexpr␣␣\IfValueTF{#2}{#2}{#1}\baselineskip}%
  }\gmu@tempa\selectfont
}
```

```
\let\gmu@discretionaryhyphen\-␣% for the cases when we don't redefine \
      but use \
```

A redefinition of \- that makes it optional-argument to allow further hyphenation

\bihyphen
```
\DeclareCommand\bihyphen{
  O{*}␣% token that'll make the discretionary hyphen \- allow other break-points
}{%
```

\gmu@discretionaryhyphen
```
  \DeclareCommand\gmu@discretionaryhyphen{T{#1}ca}{%
    \IfValueT{##2}{%
      \gmu@ifempty{##2}{}{%
        \def\gmu@bihyphen@char{##2}}%
    }%
    \IfValueT{##3}{%
      \gmu@ifempty{##3}{}{%
        \def\gmu@bihyphen@corr{##3}}%
    }%
```

Depending on #1 we allow (if present, then we take \discre) hyphenation of the word's before- and after-parts or forbid it (if absent, then we take \discretionary).

```
    \IfValueTF{##1}\discre\discretionary
```

```
      {% before break
        \IfValueTF{##2}
        {%
          \gmu@ifempty{##2}{\gmu@bihyphen@char}{##2}%
        }%
        {%
          \ifnum\hyphenchar\font>\z@
            \char\hyphenchar\font
          \fi}%
      }% end of before break
      {% after break
        \IfValueT{##3}{%
          \gmu@ifempty{##3}{\gmu@bihyphen@corr}{##3}%
        }%
      }%
      {% without break
      }% almost as in The TeX book: unlike The TeX book, we allow hyphenchars ≥ 255
          as we are XeTeX.
    }% of \DeclareCommand\-
    \gmu@storeifnotyet\-% original \- is a TeX's primitive, therefore we should
        store it.
    \CommandLet\-\gmu@discretionaryhyphen
    \CommandLet\@dischyph\gmu@discretionaryhyphen␣% to override framed.sty
    \pdef\gmu@flexhyphen{\gmu@discretionaryhyphen#1\relax}%
}% of \bihyphen

\relaxen\gmu@bihyphen@corr

\@ifXeTeX{%
  \DeclareCommand\gmshowlists␣{
    >iT{\depth}
    Ď{1}%
    >iT{\breadth}
    Ď{10000000}
  }%
}{% not in XeTeX we use legacy D arg. specifiers:
  \DeclareCommand\gmshowlists␣{
    >iT{\depth}
    Ď{1}%
    >iT{\breadth}
    Ď{10000000}
  }%
}
{\tracingonline=\@ne
  \showboxdepth=#1\relax␣% how many levels of box nesting should be shown
  \showboxbreadth=#2\relax␣% after how many elements »etc.« will be put
  \showlists
}

\DeclareCommand\gmtracingoutput␣{
  \SameAs\gmshowlists
}
{\tracingonline=\@ne
  \tracingoutput=\@ne
```

\gmshowlists

\gmshowlists

\gmtracingoutput

```
    \showboxdepth=#1\relax␣%  how many levels of box nesting should be shown
    \showboxbreadth=#2\relax␣%  after how many elements »etc.« will be put
}
```

`\c@FiBreakPenalty`
```
\newcount\c@FiBreakPenalty
```

`\AllowFiBreak`
```
\DeclareCommand␣\AllowFiBreak␣{%
```

The defaults are as in DEK's `\filbreak`.

```
    Q{l}{l}
    =␣{\c@FiBreakPenalty␣}
}
{%
    \penalty\@M
    \csname␣vfi#1\endcsname
    \penalty␣\c@FiBreakPenalty
    \csname␣vfi#1neg\endcsname
}
```

`\IgnorePars`
```
\DeclareCommand␣\IgnorePars␣{m␣Q{\par}}␣{#1}
```

This command gobbles a sequence of epty lines and explicit `\par` CS es. Unlike primitive `\ignorespaces`, it doesn't expand macros.

```
⟨/ command⟩
```

## Ampulex Compressa-like modifications of macros—the gmampulex packageThis file has version number v0.993 dated 2010/10/24.

```
⟨utils⟩ ␣␣␣\gmu@PackOptionX{ampulex}
⟨*ampulex⟩
```

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.

```
\RequirePackage{gmcommand}
```

`\ampulexlet`
```
\DeclareCommand\ampulexlet\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.
   T{\def\edef\gdef\xdef\pdef}{\def}␣%  (2) (optional) kind of definition; if
      not specified, \def will be used.
   m␣%  (3) macro to be let to,
   m␣%  (4) macro to provide the definition,
   O{}␣%  (5) \def's parameters string; empty by default,
   O{}␣%  (6) definition body's parameters to be taken in a one-step expansion of the
      redefined macro; empty by default; the undelimited parameters should be
      double-braced here.
   m␣%  (7) start token(s),
   m␣%  (8) end token(s)
   m␣%  (9) the replacement of #7, #8 and whatever between them.
}{%  For the example of usage see 9870.
```

```
  \long\def\gmu@ampulexlet@resa
  ##1#7% we put #7 as a delimiter
  ##2#8% we put #8 as a delimiter
  ##3\gmu@AmpulexDelimiter{%
```

We use a special (undefined) CS `\gmu@AmpulexDelimiter` as the final delimiter because standard LaTeX's `\@nil` isn't probably a good idea since we want to ampulex deep LaTeX's macros and other `\gmu@`... macros too.

```
    \gmu@ifempty{##3}%
  }%
```

Now `\gmu@ampulexlet@resa` is redefined to produce an open `\gmu@ifempty` depending on whether the start and end token(s) are found in the meaning of `#4`.

Before we proceed, we deal with a difficulty with a special case when `#6` is "`#1`", which occurs because of stripping braces of a single-brace argument.

```
  \gmu@ifutokens{#6}{##1}%
  {\def\ampulex@Args{{%
          ####1}}%
  }%
  {\edef\ampulex@Args{\@nx\unexpanded{%
        \unexpanded{#6}}}}%
  \long␣␣\def\gmu@ampulexlet@resc##1{%
    \@xa\@xa\@xa\@xa␣␣␣␣␣␣\@xa\@xa\@xa\gmu@ampulexlet@resa
    \@xa\@xa\@xa#4%
    \ampulex@Args
    ##1% this parameter will be substituted with #7#8 in line 8775 and with emptiness in line 8826.
    \gmu@AmpulexDelimiter
  }%
  \gmu@ampulexlet@resc{#7#8}%
```

`%%` `\gmu@ampulexlet@resb` We've just applied the checker and it produces an open `\gmu@ifempty{`⟨*some tokens*⟩`}` if the delimiters are found in the meaning of `#4` so, if ⟨*some tokens*⟩ are none, we issue a warning

```
  {%
    \PackageWarning{gmampulex}{%
      \@nx#4␣doesn't␣contain␣tokens
      \detokenize{#7}␣nor␣\detokenize{#8}.␣You␣better␣check␣
          if␣this␣is
      what␣you␣want␣to␣redefine.^^J%
      \@nx#4␣is^^J%
      \meaning#4^^J%
    }}%
```

and we proceed if they are really some

```
  {%
```

We define a temporary macro with the parameters delimited with the 'start' and 'end' parameters of `\ampulexdef`. It has to stand a double `\edef`.

```
  \edef\gmu@ampulexlet@resa{%
    \long\def\@nx\gmu@ampulexlet@resa
    ####1\unexpanded{#7}%
    ####2\unexpanded{#8}%
```

```
      ####3\@nx\gmu@AmpulexDelimiter{%
        \@nx\unexpanded{####1}%
```

we drop the part between the #7 and #8 delimiters (including delimiters)

```
        \unexpanded{\unexpanded{#9}}% we replace the part of the redefined
              macro's meaning with the replacement text.
        \@nx\unexpanded{####3}%
      }% of inner \gmu@ampulexlet@resa
    }% of outer \gmu@ampulexlet@resa
    \gmu@ampulexlet@resa
```

Now \gmu@ampulexlet@resa carries the modifier of #4's definition.

```
\unless\ifx\czat#4%
    \edef\gmu@ampulexlet@resb{% double definition for double hashes of ex-
          panded \unexpanded{#1...}
      #1#2%
      \@nx#3\unexpanded{#5}{%
        \gmu@ampulexlet@resc{}% Here we are sure the tokens sequences #7
              and #8 are in the one-level expansion of #4 so we don't pass them as
              sentinels (which BTW would totally spoil the redefinition, what it did
              indeed 2010/6/23).
      }% of #3's definition body
    }% of inner \gmu@ampulexlet@resb
    \gmu@ampulexlet@resb
\else
     \gmu@ifxany#2{\gdef\xdef}{\global}{}%
     #1\edef#3#5{\gmu@ampulexlet@resc{}}%
\fi
  }% of if the delimiters were found in the meaning.
}% of \ampulexlet
```

\ampulexdef

```
\DeclareCommand\ampulexdef\long{%
#1␣␣Q{\outer\long\global\protected}{}␣% (1) as \ampulexlet
#2␣␣T{\def\edef\gdef\xdef\pdef}{\def}␣% (2) ad \ampulexlet
#3␣␣m␣% (3) macro to be redefined,
#4␣␣O{}␣% (4) as \ampulexlet's #5, i.e., \def's parameters string; empty by de-
      fault,
#5␣␣O{}␣% (5) as \ampulexlet's #6, i.e., definition body's parameters to be taken
      in a one-step expansion of the redefined macro; empty by default; the unde-
      limited parameters should be double-braced here (but not doubled).
#6␣␣m␣% (6) start token(s),
#7␣␣m␣% (7) end token(s),
#8␣␣m␣% (8) the replacement
}{%
  \DCUse\ampulexlet{#1}{#2}{#3}{#3}{#4}{#5}{#6}{#7}{#8}%
}
```

#### A definer for expandable loops

Now, as an example of use of \ampulexlet, we'll build a definer for expandable numerical loops.

```
\def\gmu@ENumLoop#1#2{% this is a fully expandable loop generating #2 − #1
      space tokens (cf. The ε-TEX Manual p. 9).
```

```
  \ifnum#1<#2␣%
    \gmu@tempa
    \@xa\gmu@ENumLoop
    \@xa{\number\numexpr#1+1\@xa}%
    \@xa{\number#2\@xa}%
  \fi}% of \gmu@hashes.
\long\def\defENumLoop
#1% the loop macro's name
#2% the replacement of \gmu@tempa
{%
  \ampulexlet#1\gmu@ENumLoop
  [##1##2][{##1}{##2}]%
  \gmu@tempa\@xa{#2\@xa}%
  \ampulexdef#1%
  [##1##2][{##1}{##2}]%
  \gmu@ENumLoop\@xa{#1\@xa}%
}
```

Let `\GenericInfo` write also to the terminal when `\tracingonline>0`.

```
\edef\GenericInfoToTerminal{%
  \unexpanded{%
    \@XA{\ampulexlet\protected\long\GenericInfo}\csname
    GenericInfo␣\endcsname[#1#2][{#1}{#2}]%
    \write\m@ne␣% we replace the token between these with:
    {\write\ifnum\tracingonline>\z@␣\@unused\else\m@ne\fi}%
  }%
}
\ampulexdef\@starttoc[#1][#1]\makeatletter\@input{%
      \makeatletter\NamedInput}
⟨/ ampulex⟩
```

**The gmenvir PackageThis file has version number v0.993 dated 2010/10/24.**

```
⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{envir}
⟨⋆envir⟩
```

The gmenvir.sty package provides some improvements of the LaTeX's environments machinery. It provides a starred version of begin with which the CS respective to the argument doesn't have to be defined. This package also improves `\end` by detokenising environment's name (which is equivalent to comparing the names of CS'es not strings of tokens). The package also provides some tests such as `\@ifenvir`.

For details just read the code part.

#### Contents of the gmenvir.zip archive

```
\RequirePackage{gmbase,␣gmampulex}␣% the low-level macros
```

### 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}
```

We provide a stack of environments consisting of triads {⟨*env. name*⟩}{⟨*group level*⟩}{⟨*beg. line*⟩} (2010/6/9)

```
\emptify\@envirstack

\def\@pushenvir{%

  %% \edef\@currenvir{\@currenvir}% is already expanded.

  \xdef\@envirstack{%
    {\@xa\detokenize\@xa{\@currenvir}}%
    {\the\currentgrouplevel}%
    {\@currenvline}%
    \@envirstack
  }%
}

\def\@popenvir#1#2#3{%
  \@XA{\@popenvir@#1#2#3}\@envirstack\@nil
}

\def\@popenvir@#1#2#3#4#5#6#7\@nil{%
  \gdef#1{#4}% #1 carries last envir name
  \gdef#2{#5}% #2 carries last envir level
  \gdef#3{#6}% #3 carries last envir beginnig line
  \gdef\@envirstack{#7}% and we update the stack
}
```

\@begnamedgroup

```
\long\def\@begnamedgroup#1{%
  \edef\@prevgrouplevel{\the\currentgrouplevel}% added 2009/03/24
      to handle special pseudo-environments that don't increase \currentgrou¦
      plevel(such as document). Note it's \edefed 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
      9041.
\edef\@currenvline{\on@line}%
\@pushenvir␣% we put current envir to \@envirstack.
\csname␣#1\endcsname}% if the argument is a command's name (an environ-
      ment's e.g.), this command will now be executed. (If the corresponding con-
      trol sequence hasn't been known to TEX, this line will act as \relax.)
```

Let us make it the starred version of `\begin`.

```
\begin*  \def\begin{\gmu@ifstar{\@begnamedgroup}{%
\begin         \@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'.

`\gmu@ifedetokens` and `\@ifenvir` are defined in gmbase.

```
\long\def\@fourthofmany#1#2#3#4#5\@nil{#4}%

\lpdef\@ifprevenvir#1{%
      % #1 enquired environment name which will be confronted with \@preven¦
        vir
      % #2 what if true (if the names are equivalent[4])
      % #3 what if false
```

(2010/09/23, v0.993:) to be precise, it's not a change but rather a staus quo action: `\gmu@ifedetokens` suddenly turned to be expandable and un`\protected` so we make *this* macro `\protected`

```
  \gmu@ifedetokens
  {\@xa\@fourthofmany\@envirstack\relax\relax\relax\relax%
       \@nil}%
  {#1}%
}
```

Note that `\@ifjobname` and `\@ifenvir` are expandable and in an `\edef` they expand to

`\gmu@ifedetokens{`⟨*current jobname*⟩`}{`⟨*arg.1*⟩`}`

and

`\gmu@ifedetokens{`⟨*current envir*⟩`}{`⟨*arg.1*⟩`}`

4 The names are checked whether they produce the same `\csname`. They don't have to have the same catcodes.

resp. which may be useful for some TEXvert.

```
\def\@checkend#1{%
  \@ifenvir{#1}%
  {}%
  {\@badend{#1}}%
}
```

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.

```
\ampulexdef\end[#1][#1]\endcsname\@checkend{%
\endcsname
\@popenvir\gmu@drain\gmu@drain\gmu@drain
\@checkend}
\pdef\@endif#1{\@ifenvir{#1}{\end{#1}}{}}
\pdef\@endifprev#1{\@ifprevenvir{#1}{\end{#1}}{}}
```

`\c@EnvirInterruption`

```
\newcount\c@EnvirInterruption
\lpdef\gmu@InterruptEnvir
#1% the contents of interruption.
{%
  \global␣␣\advance\c@EnvirInterruption\@ne
  \Name␣\@popenvir
  {gmu@InterruptCurrenv␣\the\c@EnvirInterruption}\gmu@drain%
      \gmu@drain
  \endgroup
  #1%
  \begingroup
  \@XA␣{\let\@currenvir}%
  \csname␣gmu@InterruptCurrenv␣\the\c@EnvirInterruption␣%
      \endcsname
  \@pushenvir
  \global␣␣\advance\c@EnvirInterruption\m@ne
}
```

⟨/ envir⟩

### From relsize

```
⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{relsize}
⟨⋆relsize⟩
```

As file relsize.sty, v3.1 dated July 4, 2003 states, LaTeX 2ε version of these macros was written by Donald Arseneau asnd@triumf.ca and Matt Swift swift@bu.edu after the LaTeX 2.09 smaller.sty style file written by Bernie Cosell cosell@WILMA.BBN.COM.

I take only the basic, non-math mode commands with the assumption that there are the predefined font sizes.

\relsize \smaller \larger \smallerr \largerr

You declare the font size with `\relsize{`⟨*n*⟩`}` where ⟨*n*⟩ gives the number of steps ("mag-step" = factor of 1.2) to change the size by. E.g., $n = 3$ changes from `\normalsize` to `\LARGE` size. Negative *n* selects smaller fonts. `\smaller == \relsize{-1}`; `\larger == \relsize{1}`. `\smallerr`(my addition) `== \relsize{-2}`; `\largerr` guess yourself.

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

\relsize

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

Change the number by the given increment:

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

watch out for size underflow:

```
     \ifnum\@tempcnta<\z@␣\rs@size@warning{small}{\string%
          \tiny}\@tempcnta\z@␣\fi
     \@xa\endgroup
     \ifcase\@tempcnta␣␣% set new size based on altered number
        \tiny␣\or␣\scriptsize␣\or␣\footnotesize␣\or␣\small␣%
            \or␣\normalsize␣\or
        \large␣\or␣\Large␣\or␣\LARGE␣\or␣\huge␣\or␣\Huge␣%
            \else
         \rs@size@warning{large}{\string\Huge}\Huge
\fi\fi}% end of \relsize.
```

\rs@size@warning

```
\providecommand*\rs@size@warning[2]{\PackageWarning{gmutils␣
    (relsize)}{%
 Size␣requested␣is␣too␣#1.\MessageBreak␣Using␣#2␣instead}}
```

\rs@unknown@warning

```
\providecommand*\rs@unknown@warning{\PackageWarning{gmutils␣
    (relsize)}{Current␣font␣size
 is␣unknown!␣(Why?!?)\MessageBreak␣Assuming␣\string%
     \normalsize}}
```

And a handful of shorthands:

\larger

```
\DeclareRobustCommand*\larger[1][\@ne]{\relsize{+#1}}
```

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

We could implement continuous growth: `\larger[2.567]` means predefined fontsize 2 steps up plus .567 of the difference between step 2 and step 3.

```
⟨/ relsize⟩
```

### The gmmeta package for meta-symbols

```
⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{meta}␣% provides \bihyphen, \discre, \discret
⟨*meta⟩
\RequirePackage{gmcommand}
```

I fancy also another Knuthian trick for typesetting ⟨*meta-symbols*⟩ in *The TEX book*. 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."

```
\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.

\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.
```

```
}

\def\meta@fontsetting{\color{red!85!black}}

\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[5] is a version of *The TEX book*'s one.

`\<..>`

```
\def\<#1>{\meta{#1}}

\pdef\metachar#1{\begingroup\metacharfont␣#1\endgroup}
\def\metacharfont{\meta@fontsetting\rm}
```

### Macros for printing macros and filenames

The `\discre` macro is defined in gmbase. It works like `\discretionary` but allows hyphenation before and after itself.

```
\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`\catcode`ing has no effect).

```
\def\printspaces#1{{\let~=\vs␣\let\␣=\vs␣\gmu@pswords#1␣%
    \@nil}}
\def\gmu@pswords#1␣#2\@nil{%
  \ifx\relax#1\relax\else#1\fi
  \ifx\relax#2\relax\else\vs\penalty\hyphenpenalty%
      \gmu@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.

\pdef\sfname#1{\textsf{\printspaces{#1}}}

\def\gmu@discretionaryslash{\discre{/}{\hbox{}}{/}}% the
    second pseudo-argument nonempty to get \hyphenpenalty
    not \exhyphenpenalty.

\pdef\file#1{\gmu@printslashes#1/\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.

```
\pdef\pk#1{\textsf{#1}}
```

Some (if not all) of the below macros are copied from doc and/or ltxdoc.

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

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}%
}% of \cs

\pdef\cs@inner#1#2{%
  #1#2%
```

%% \hyphenchar\font=\narrativett@storedhyphenchar\relax % we don't restore the value of \hyphenchar since this restores it back for the entire paragraph.

```
  \endgroup}

\def\narrativett{\ttfamily}% such name because I introduce it to distin-
        guish the narrative verbatims from the code in gmdoc.

\long\pdef\env{\cs[]}
```

And for the special sequences like ^^A:

```
\foone{\@makeother\^}
  {\pdef\hathat{\cs[^^]}}

\AtBeginDocument{%
  \@ifpackageloaded{gmdoc}{\def\hash{\cs[\#]}}{}}
```

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

```
\def\possfil{\hfil\penalty1000\hfilneg}

\def\possvfil{\vfil
  \penalty\numexpr
  \gmu@minnum{\clubpenalty+\widowpenalty}{9999}%
  \relax % eaten by \gmu@maxnum
  \relax % eaten by \numexpr
  \vfilneg}
```

#### 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`)."

```
\long\def\cmd#1{\@xa\cs\@xa{\@xa\cmd@to@cs\string#1}\spifletter}% it has to be un \protected! It has so many \expandafter s to allow \cmd % \par and still keep the \cs command 'short'.

\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}\arg@wrap{#1}{%
    \narrativett\char`\}}}
```

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

\oarg
```
\pdef\oarg{\@ifnextchar[\@oargsq\@oarg}
\pdef\@oarg#1{{\narrativett[}\arg@wrap{#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(}\arg@wrap{#1}{\narrativett)}}
\def\@pargp(#1){\@parg{#1}}

\pdef\aarg{\@ifnextchar<\@aarga\@aarg}
\def\@aarg#1{{\narrativett<}\arg@wrap{#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
```

Now provide the default `\arg@wrap`, for meta-arguments that is.

```
\def\arg@wrap{\meta}
```

`\arg@dc`

```
\DeclareCommand\arg@dc!{%
  s␣% (1)
  o␣% (2)
  c␣% (3)
  b␣% (4)
  a␣% (5)
  T{\arg}␣% (6) just gobbled (for backwards compatibility)
}{% This command iterates while it has arguments and typesets them in brackets,
    parentheses or curly braces. Note it gobbles subsequent \args and just iterates.
  \def\next{o}%
  \IfValueT{#1}%
  {\metachar[\scanverb{*}\metachar]\def\next{1}}%
  \IfValueT{#2}{\@oarg{#2}\def\next{1}}%
  \IfValueT{#3}{\@parg{#3}\def\next{1}}%
  \IfValueT{#4}{\marg{#4}\def\next{1}}%
  \IfValueT{#5}{\aarg{#5}\def\next{1}}%
  \@ifnextchar\egroup{\endgroup}{%
    \if1\next\@xa\arg@dc
    \else␣% it's crucial that we look for verbatim braces after we checked there
        were no #4, otherwise there would be an error.
      \def\next{%
        \@ifnextif\xiilbrace{\@verbmargm}%
        {% not active or other lbrace
          \@ifnextif<{% then we look for angles
            \ifnum\catcode`>=\active
              \@xa\@verbaargact
            \else\@xa\@verbaarga
            \fi}%
          {% and if not angles neither verbatim braces, then
            \endgroup␣␣% if we have no more arguments to typeset, we close
                the group opened in lines 9525 and 9542.
            \spifletter
          }%
        }%
      }%
      \@xa␣\next
    \fi
  }% of not egroup
}% of \arg@dc
```

Now define the front-end macro of the \arg command:

```
\foone{\obeylines\@makeother\^^C}{%
  \AtBeginDocument{%
    \let\math@arg\arg␣%
    \pdef\arg{% This is \arg for meta-arguments.
      \ifmmode\math@arg␣%
      \else\afterfi{%
        \begingroup␣%
        \ifdefined\@ifQueerEOL\@ifQueerEOL{%
          \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 ␣, gobbles the
                space it turns into so the comment layer would 'leak' to the code
                layer.
        }{}\fi␣%
        \arg@dc}%
      \fi}% of \arg,
  }% of \AtBeginDocument,
```

And this is arg-typesetting command for verbatim arguments.

```
  \pdef\argv{%
    \begingroup␣%
    \@makeother\^^C%
    \pdef\arg@wrap##1{%
      \narrativett{##1}%
    }%
    \ifdefined\@ifQueerEOL\@ifQueerEOL{%
        \def^^M{\unskip\space}% in the 'queer' EOLs scope we keep line end
              active… as above
      }{}%
    \fi␣% of \ifdefined
    \arg@dc}%
}% 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
```
\DeclareCommand\cat{Q{'"0123456789ABCDEF}{0}}{%
  ${}_{\the\numexpr#1}\m@th$\spifletter
}
```

A shorthand for \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.

\pdef\CSs{\CS{}es\@ifnextcat␣a{␣}{}}% for pluralis.
\pdef\CSes{\CS{}es\@ifnextcat␣a{␣}{}}% for pluralis.
```

⟨/ meta⟩

### The gmlogos package—a couple of TEX-related logos

```
9589 ⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{logos}
9590 ⟨*logos⟩

9592 \RequirePackage{gmbase}
```

We'll modify The LaTeX logo now to make it fit better to various fonts.

```
9598 \let\oldLaTeX\LaTeX
9599 \let\oldLaTeXe\LaTeXe

9601 \pdef\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
9602 \Store@Macro\TeX
9603 \AtBeginDocument{\Restore@Macro\TeX}
```

\DeclareLogo

```
9605 \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.
9613   \ifx\relax#1\def\gmu@DeclareLogo@resa{\@xa\@gobble%
           \string#2}%
9614   \else
9615     \def\gmu@DeclareLogo@resa{#1}%
9616   \fi
9617   \edef\gmu@DeclareLogo@resa{%
9618     \@nx\DeclareTextCommand\@nx#2{PD1}{\gmu@DeclareLogo@resa}}
9619   \gmu@DeclareLogo@resa
9620   \DeclareTextCommandDefault#2{#3}%
```

\pdef

```
9621   \pdef#2{#3}% added for XeTeX.
9622 }

9625 \DeclareLogo\LaTeX{%
9626   {%
9627     L%
9628     \setbox\z@\hbox{\check@mathfonts
9629       \fontsize\sf@size\z@
9630       \math@fontsfalse\selectfont
9631       A}%
9632     \kern-.57\wd\z@
9633     \sbox\tw@␣T%
9634     \vbox␣to\ht\tw@{\copy\z@␣\vss}%
9635     \kern-.2\wd\z@% originally −, 15 em for T.
9636   }%
9637   {%
9638     \ifdim\fontdimen1\font=\z@
9639     \else
9640       \count\z@=\fontdimen5\font
9641       \multiply\count\z@␣by␣64\relax
9642       \divide\count\z@␣by\p@
9643       \count\tw@=\fontdimen1\font
9644       \multiply\count\tw@␣by\count\z@
9645       \divide\count\tw@␣by␣64\relax
9646       \divide\count\tw@␣by\tw@
```

```
      \kern-\the\count\tw@␣sp\relax
    \fi}%
  \gmlogos@hyphen
  \TeX}
```

\LaTeXe
```
\DeclareLogo\LaTeXe{\mbox{\m@th␣\if
    b\expandafter\@car\f@series\@nil\boldmath\fi
    \LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}
```

```
\Store@Macro\LaTeX
\Store@MacroSt{LaTeX␣}
```

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

\LaTeXpar
```
\DeclareLogo[(La)TeX]{\LaTeXpar}{%
  {%
    \setbox\z@\hbox{(}% )
    \leavevmode␣%
    \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@}\gmlogos@hyphen\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."

```
\gmu@ifundefined{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 TEX book*'s logo:

```
\TeXbook 9709 \DeclareLogo[The␣TeX␣book]\TeXbook{\textsl{The␣\TeX\space␣
              book}}
         9710 \let\TB\TeXbook% TUG Boat uses this.

  \eTeX  9712 \DeclareLogo[e-TeX]\eTeX{%
         9713   \iffontchar\font"03B5{\itshape␣ε}\else
         9714   \ensuremath{\varepsilon}\fi-\kern-.125em\TeX}% definition sent by
                    Karl Berry from TUG Boat itself.

         9717 \Store@Macro\eTeX

\pdfeTeX 9719 \DeclareLogo[pdfe-TeX]\pdfeTeX{pdf\gmlogos@hyphen\eTeX}

 \pdfTeX 9721 \DeclareLogo\pdfTeX{pdf\gmlogos@hyphen\TeX}
\pdfLaTeX 9722 \DeclareLogo\pdfLaTeX{pdf\gmlogos@hyphen\LaTeX}

         9725 \gmu@ifundefined{XeTeX}{%
  \XeTeX 9726   \DeclareLogo\XeTeX{X\kern-.125em\relax
         9727     \gmu@ifundefined{reflectbox}{%
         9728       \lower.5ex\hbox{E}\kern-.1667em\relax}{%
         9729       \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
         9730     \TeX}}{}

         9732 \gmu@ifundefined{XeLaTeX}{%
 \XeLaTeX 9733   \DeclareLogo\XeLaTeX{X\kern-.125em\relax
         9734     \gmu@ifundefined{reflectbox}{%
         9735       \lower.5ex\hbox{E}\kern-.1667em\relax}{%
         9736       \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
         9737     \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-XƎTEX usage. With XƎTEX, 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.

```
\XeTeXpar 9745 \DeclareLogo\XeTeXpar{%
          9746   \setbox\z@\hbox{(}% )
          9747   \leavevmode
          9748   \copy\z@
          9749   \kern-.2\wd\z@
          9750   \smash{% the "Xe" part is copied from xltxtra
          9751     X\lower0.5ex
          9752    \hbox{\kern-0.15em
          9753      \gmu@ifundefined{XeTeXversion}%
          9754      {\setbox0=\hbox{E}\dimen0=\ht0\advance\dimen0by\dp0%
          9755        \raise\dimen0\hbox{\rotatebox{180}{\box0}}%
          9756      }% of if not in XƎTEX, then in XƎTEX:
          9757      {\ifnum\XeTeXfonttype\font>0
          9758        \ifnum\XeTeXcharglyph"018E>0
          9759          \char"018E\relax
          9760        \else
          9761          \ifdim\fontdimen1\font=0pt
          9762            \reflectbox{E}%
          9763          \else
```

```
            \XeTeXuseglyphmetrics=1%
            \setboxo=\hbox{E}\dimeno=\hto\advance\dimenoby\dpo%
            \raise\dimeno\hbox{\rotatebox{180}{\boxo}}%
          \fi
        \fi
      \else
        \setboxo=\hbox{E}\dimeno=\hto\advance\dimenoby\dpo%
        \raise\dimeno\hbox{\rotatebox{180}{\boxo}}%
      \fi}% of reversed E when in XeTeX
    }% of hbox
  }% of smash
  \setbox\z@\hbox{)}%
  \kern-.2\wd\z@
  \copy\z@
  \kern-0.15em
  \TeX}%
```

\LuaTeX
```
\DeclareLogo[LuaTeX]\LuaTeX{\textsc{Lua}\gmlogos@hyphen\TeX}
```

```
\emptify\gmlogos@hyphen
```

```
\def\HyphenateLogo#1{%
  {\let\gmlogos@hyphen\-%
    #1}%
}
```

```
⟨/ logos⟩
```

### The gmnotonlypream—modification of the 'only preamble' clause

```
⟨utils⟩ ␣␣␣␣␣\gmu@PackOptionX{notonlypream}
⟨*notonlypream⟩
```

```
\RequirePackage{gmampulex}
```

### 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
```
\newcommand\not@onlypreamble[1]{{%
  \def\do##1{\ifx#1##1\else\@nx\do\@nx##1\fi}%
  \xdef\@preamblecmds{\@preamblecmds}}}
```

```
\not@onlypreamble\@preamblecmds
\not@onlypreamble\@ifpackageloaded
\not@onlypreamble\@ifclassloaded
\not@onlypreamble\@ifl@aded
\not@onlypreamble\@pkgextension
```

We use the two below e.g. in `\NamedInput` which we surely want to allow also within `document`.

```
\not@onlypreamble\@pushfilename
```

```
\not@onlypreamble\@popfilename
\not@onlypreamble\@currnamestack
```

And let's make the message of only preamble command's forbidden use informative a bit:

```
\def\gmu@notprerr{␣can␣be␣used␣only␣in␣preamble␣(\on@line)}
\AtBeginDocument{%
  \def\do#1{\@nx\do\@nx#1}%
  \edef\@preamblecmds{%
    \def\@nx\do##1{%
      \def##1{\@nx\gmno@NotprerrMessage##1}\@nx\@eha}}%
    \@preamblecmds}
\def\gmno@NotprerrMessage#1{%
  \PackageError{gmutils/LaTeX}%
  {\@nx\string#1␣\@nx\gmu@notprerr}{}%
}
```

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.

```
\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`.

```
  \ampulexdef\nocite[##1][##1]
  \ifx
  {\@onlypreamble\document}%
  \iftrue}
\AtBeginDocument{\gmu@nocite@ampulex}%
```

⟨/ notonlypream⟩

### Improvements to mwcls sectioning commands

```
⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{mw}
⟨⋆mw⟩
\RequirePackage{gmcommand}
```

That is, 'Expe-ri-mente'[6] mit MW's 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`.

6 A. Berg, *Wozzeck*.

```
\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?).
    \def\gmu@hyperrefstepcounter{\refstepcounter{NoNumSecs}}%
    \pdef\gmu@targetheading#1{%
      \hypertarget{#1}{#1}}}% end of then
  {\def\gmu@hyperrefstepcounter{}%
    \def\gmu@targetheading#1{#1}}% end of else
}% of \AtBeginDocument
```

Auxiliary macros for the kernel sectioning macro:

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

To avoid `\def`ing of `\mw@sectionxx` if it's undefined, we redefine `\def` to gobble the definition and restore the original meaning of itself.

Why shouldn't we change the ontological status of `\mw@sectionxx` (not define if undefined)? Because some macros (in gmdocc e.g.) check it to learn whether they are in an mwcls or not.

But let's make a shorthand for this test since we'll use it three times in this package and maybe also somewhere else.

```
\@ifnotmw  \long\def\@ifnotmw#1#2{\gmu@ifundefined{mw@sectionxx}{#1}{#2}}
```

The kernel of MW's sectioning commands:

```
\@ifnotmw{}{%
\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 `\gmu@ifundefined` to make it work in classes other than mwbk

```
      \gmu@ifundefined{if@mainmatter}{}{%
           \gmu@dontnumbersectionsoutofmainmatter}
  \fi
```

```
%   \ifHeadingNumbered
%     \refstepcounter{#1}%
%     \protected@edef\HeadingNumber{\csname
     the#1\endcsname\relax}%
%   \else
%     \let\HeadingNumber\@empty
%   \fi
```

```
  \def\HeadingRHeadText{#2}%
  \def\HeadingTOCText{#3}%
  \def\HeadingText{#4}%
  \def\mw@HeadingType{#1}%
  \if\mw@HeadingBreakBefore
    \if@specialpage\else\thispagestyle{closing}\fi
```

```
    \gmu@ifundefined{if@openright}{}{%
        \gmu@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
    \gmu@hyperrefstepcounter␣% we step an auxiliary counter to make a hyperref's label/target.
  \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

```
\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}%
    \gmu@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
  \gmu@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}
```

```
\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.
  \gmu@ifempty{#3}
    {\mw@secundef{#1#2}{#4}}
    {\@namedef{#1#2}{#3}}}
```

```
\def\mw@secundef#1#2{%
  \gmu@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.

```
\def\addtoheading#1#2{%
    \n@melet{gmu@addtoheading@resa}{#1@head}%
    \edef\gmu@addtoheading@resa{\unexpanded{#2}\@xa%
        \unexpanded{\gmu@addtoheading@resa}}%
    \n@melet{#1@head}{gmu@addtoheading@resa}%
}
```

```
}% 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{%
  \gmu@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 10154.
    {\@xa\mw@getflags\@xa{\gmu@prevsec}%
      \glet\gmu@checkaftersec@resa\mw@HeadingBreakAfter}%
    \if\mw@HeadingBreakBefore\def\gmu@checkaftersec@resa{11}\fi% if
        the current heading inserts page break before itself, all the play with
        vskips is irrelevant.
    \if\gmu@checkaftersec@resa\else
    \penalty10000\relax
    \skip\z@=\csname\gmu@prevsec␣@postskip\endcsname\relax
    \skip\tw@=\csname\mw@HeadingType␣@preskip\endcsname\relax
    \gmu@ifundefined{\mw@HeadingType␣@twoheadskip}{%
      \ifdim\skip\z@>\skip\tw@
      \vskip-\skip\z@% we strip off the post-skip of previous header if it's big-
          ger 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 \gmu@ifundefined{gmu@prevsec} 'else'.
}% of \def\gmu@checkaftersec.

\def\ParanoidPostsec{% this version of \ifgmu@postsec is intended for the
                        special case of sections may contain no normal text, as while gmdocing.
  \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

\def\gmu@getaddvs#1\addvspace#2\gmu@getaddvs{%
  \toks\z@={#1}%
  \toks\tw@={#2}}
```

And the modification of the inner macros at last:

```
\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

\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.
\DeclareCommand\WPheadings{T{\chapter}}{%
  \SetSectionFormatting[breakbefore,wholewidth]
     {part}{\z@\@plus1fill}{}{\z@\@plus3fill}%
```

\WPheadings

```
  \IfValueF{#1}{%
    \gmu@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@\@plus0,4\p@\@minus6\p@}
      {\FormatHangHeading␣{\large}}% 12/14 pt
      {6\p@\@plus0,3\p@}% after-skip 6 pt due to p.12, not to squeeze the
            before-skip too much.

  \SetTwoheadSkip{subsubsection}{10\p@\@plus1,75\p@\@minus1%
       \p@}%
  \SetSectionFormatting{subsubsection}
      {10\p@\@plus0,2\p@\@minus1\p@}
      {\FormatHangHeading␣{\normalsize}}
      {3\p@\@plus0,1\p@}% those little skips should be smaller than you calcu-
            late out of a geometric progression, because the interline skip enlarges
            them.

  \SetSectionFormatting[runin]{paragraph}
      {7\p@\@plus0,15\p@\@minus1\p@}
      {\FormatRunInHeading{\normalsize}}
      {2\p@}%
  \SetSectionFormatting[runin]{subparagraph}
      {4\p@\@plus1\p@\@minus0,5\p@}
      {\FormatRunInHeading{\normalsize}}
      {\z@}%
}% of \WPheadings
}% of \@ifnotmw
```

#### Compatibilising standard and mwcls sectionings

If you use Marcin Woliński's document classes (mwcls), you might have met their little queerness: the sectioning commands take two optional arguments instead of standard one. It's reasonable since one may wish one text to be put into the running head, another to the toc and yet else to the page. But the order of optionalities causes an incompatibility with the standard classes: MW section's first optional argument goes to the running head not to toc and if you've got a source file written with the standard classes in mind and use the first (and only) optional argument, the effect with mwcls would be different if not error.

Therefore I counter-assign the commands and arguments to reverse the order of optional arguments for sectioning commands when mwcls are in use and reverse, to make mwcls-like sectioning optionals usable in the standard classes.

With the following in force, you may both in the standard classes and in mwcls give a sectioning command one or two optional arguments (and mandatory the last, of course). If you give just one optional, it goes to the running head and to toc as in scls (which is unlike in mwcls). If you give two optionals, the first goes to the running head and the other to toc (like in mwcls and unlike in scls).

(In both cases the mandatory last argument goes only to the page.)

What more is unlike in scls, it's that even with them the starred versions of sectioning commands allow optionals (but they still send them to the Gobbled Tokens' Paradise).

(In mwcls, the only difference between starred and non-starred sec commands is (not) numbering the titles, both versions make a contents line and a mark and that's not changed with my redefinitions.)

```
\@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those written with two optionals.
  \def\gmu@secini{gm@la}%
  \Store@Macros{%
    \partmark␣\chaptermark␣\sectionmark␣\subsectionmark
    \subsubsectionmark␣\paragraphmark}%
\gmu@secxx  \def\gmu@secxx#1#2[#3]#4{%
    \ifx\gmu@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}%
        \Restore@MacroSt{#1mark}% after we've done what we wanted we restore original \#1mark.
      }%
      \def\gmu@secstar{[#3]}% if \gmu@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\gmu@secini#1\endcsname
    \gmu@secstar{#4}}%
}{% we are in mwcls and want to reverse MW's optionals order i.e., if there's just one optional, it should go both to toc and to running head.
  \def\gmu@secini{gm@mw}%
  \let\gmu@secmarkh\@gobble% in mwcls there's no need to make tricks for special version to running headings.
\gmu@secxx  \def\gmu@secxx#1#2[#3]#4{%
    \@xa\@xa\csname\gmu@secini#1\endcsname
    \gmu@secstar[#2][#3]{#4}}%
}

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

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

\let\do\gmu@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?
```

⟨/ mw⟩

`enumerate*` and `itemize*` moved to gmbase.

### The gmtypos package—some typographical tricks

Mostly according to Polish 20th Century typesetting standards.

```
⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{typos}
⟨*typos⟩
\RequirePackage{gmcommand,␣gmnotonlypream}

\unless\ifcsname␣ifgmu@quiet\endcsname
  \@xa\newif\csname␣ifgmu@quiet\endcsname
\fi
```

quiet
```
\DeclareOption{quiet}{\gmu@quiettrue}

\ProcessOptions
```

### Brave New World of XeTeX

`\@ifXeTeX` moved to gmbase (2010/04/10)

The `\udigits` declaration causes the digits to be typeset uppercase. I provide it since by default I prefer the lowercase (nautical) digits.

```
\AtBeginDocument{%
  \@ifpackageloaded{fontspec}{%
    \pdef\udigits{%
      \addfontfeature{Numbers=Uppercase}}%
  }{%
    \emptify\udigits}}
```

#### Fractions

```
\def\Xedekfracc{\gmu@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.

```
\def\gmu@xedekfraccstar{%
  \def\gmu@xefraccdef##1##2{%
      \iffontchar\font␣##2
        \@namedef{gmu@xefracc##1}{\char##2␣}%
      \else
        \n@melet{gmu@xefracc##1}{relax}%
      \fi}%
    \def\gmu@dekfracc##1/##2{%
      {\addfontfeature{VerticalPosition=Numerator}##1}%
           \gmu@numeratorkern
      \char"2044␣\gmu@denominatorkern
      {\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.

```
    \gmu@xefraccdef{1/4}{"BC}%
    \gmu@xefraccdef{1/2}{"BD}%
    \gmu@xefraccdef{3/4}{"BE}%
    \gmu@xefraccdef{1/3}{"2153}%
    \gmu@xefraccdef{2/3}{"2154}%
    \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}%
    \pdef\dekfracc@args##1/##2{%
      \gmu@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.
    \gmu@ifstar{\let\gmu@dekfracc\gmu@dekfraccsimple}{}%
  }
\def\gmu@xedekfraccplain{% 'else' of the main \gmu@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).

```
\pdef\dekfracc{%
  \ifmmode\hbox\bgroup\@gmu@mmhboxtrue\fi
  \dekfracc@args}

\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 \gmu@ifstar we wrote 8 (eight) #s to get the correct definition and in the second argument 'only' 4. (The LaTeX $2_\varepsilon$ Source claims that that is changed in the 'new implementation' of \gmu@ifstar so maybe it's subject to change.)

A simpler version of \dekfracc is provided in line 11757.

```
\pdef\textsuperscript@@#1{%
  \@textsuperscript{\selectfont#1}}%
\def\@textsuperscript#1{%
  {\m@th\ensuremath{^{\mbox{\fontsize\sf@size\z@#1}}}}}

\let\textsuperscript@@\textsuperscript

\def\GMtextsuperscript{%
  \@ifXeTeX{%
```

\textsuperscript

```
    \DeclareCommand\textsuperscript{sm}{%
      \IfValueTF{##1}{\textsuperscript@@{##2}}%
      {%
        \begingroup
        \addfontfeature{VerticalPosition=Numerator}##2%
        \endgroup}}%
  }{\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 \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.

```
\def\gmu@getfontstring{%
  \xdef\gmu@fontstring{%
    \gmu@fontstring@}}

\def\gmu@fontstring@{%
  \@xa\@xa\@xa\gmu@fontstring@@\@xa\meaning\the\font\@nil}

\def\gmu@fontstring@@#1"#2"#3\@nil{"#2"}

\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}%
        \afterfi\@firstoftwo
        \else\afterfi\@secondoftwo
      \fi
    \fi
    {\xdef\gmu@fontscalebr{[\gmu@fontscale] }}%
    {\xdef\gmu@fontscalebr{[#3] }}%
    \afterfi\gmu@getfontscale
  \fi
}

\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}

\def\gmu@stripchar#1"{"}

\gmath@getfamnum \DeclareCommand\gmath@getfamnum{C{\gmath@fam}}{%
  \edef\gmath@famnum{\@xa\gmu@stripchar\meaning#1}%
```

```
10583 }
```

`\XeTeXmathcode`⟨*char slot*⟩ [⟨=⟩]⟨*type*⟩ ⟨*family*⟩ ⟨*char slot*⟩

\gmathFams
```
10587 \DeclareCommand\gmathFams{o␣␣% the name of math version
10588   C{\NoValue}␣% 'rescue' font. Will be accessible via \symgmathRoman/<version>
                      (math family) and \gmath@fontt/<version> (font).
10591 }{%
10592   \IfValueT{#1}{%
10593     \DeclareMathVersion{#1}% this sets the defaults so no need to define them
                                  explicitly
10595     }% of if #1 given
10597   \gmu@getfontdata{\rmfamily\itshape}%
10599   \edef\gmu@tempa{%
10600     \IfValueTF{#1}{\@nx\SetSymbolFont{letters}{#1}}%
10601     {\@nx\DeclareSymbolFont{letters}}%
10602     {\encodingdefault}{gmathit\PutIfValue{#1}}{m}{it}%
10603     \IfValueT{#1}{%
10604       \@nx\DeclareSymbolFont{letters#1}%
10605       {\encodingdefault}{gmathit\PutIfValue{#1}}{m}{it}%
10606       }%
10607     \@nx\DeclareFontFamily{\encodingdefault}{gmathit%
                               \PutIfValue{#1}}{%
10608       \skewchar\font\gmu@theskewchar\space}%
10609     \@nx\DeclareFontShape{\encodingdefault}{gmathit%
                               \PutIfValue{#1}}{m}{it}{%
10610       <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
10611     \IfValueT{#1}{%
10612       \@nx\SetMathAlphabet\@nx\mathit{#1}{\encodingdefault}{%
                                  gmathit#1}{m}{it}}%
10613   }\gmu@tempa
10615   \IfValueT{#2}{%
10616     \gmu@getfontdata{#2\gmu@ifstored{\upshape}{\storedcsname{%
                                  upshape}}{\upshape}}%
10617     \edef\gmu@tempa{%
10618       \@nx\DeclareSymbolFont{gmathRoman\PutIfValue{#1}}%
10619       {\encodingdefault}{gmathRm\PutIfValue{#1}}{m}{n}%
10620       \@nx\DeclareFontFamily{\encodingdefault}{gmathRm%
                                  \PutIfValue{#1}}{%
10621         \skewchar\font\gmu@theskewchar\space}%
10622       \@nx\DeclareFontShape{\encodingdefault}{gmathRm%
                                  \PutIfValue{#1}}{m}{n}{%
10623         <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
10624     }\gmu@tempa
10625     \@xa\font\csname␣gmath@fontt\PutIfValue{#1}\endcsname
10626     =\gmu@fontstring\relax
10628     \gmu@getfontdata{#2\gmu@ifstored{\itshape}{\storedcsname{%
                                  itshape}}{\itshape}}%
10629     \edef\gmu@tempa{%
10630       \@nx\DeclareSymbolFont{gmathItalic\PutIfValue{#1}}%
10631       {\encodingdefault}{gmathIt\PutIfValue{#1}}{m}{n}%
10632       \@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}{}{\Store@Macro##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␣=
      \mathchar@type##3\space
      \gmath@famnum\space
      `##1\relax}%
  }%
  \gmu@tempa
}% of \gmath@do
\DeclareCommand\gmath@doif{%
  m␣␣␣␣␣␣␣␣% (1) the Unicode hex of char enquired,
  m␣␣␣␣␣␣% (2) the char or CS to be declared,
  m␣␣␣␣␣␣% (3) math type CS(\mathord etc.),
  o␣␣␣␣␣␣% (4) second-choice Unicode (taken if first-choice is absent),
  o␣␣␣␣␣␣% (5) third-choice Unicode (as above if second-choice is absent from
       font).
  B{\gmath@fam}␣% (6) never used in this package. Why?
  C{\NoValue}
}{%
  \gmu@storeifnotyet{##2}%
  \@xa\let\@xa\gmath@ft\csname
    gmath@font%
    \ifx\gmath@fam##6\else␣t\fi
    \gmath@version
  \endcsname
  \iffontchar\gmath@ft"##1␣\gmath@do##2[##1]##3(##6)%
  \else
    \IfValueTF{##4}{%
      \iffontchar\gmath@ft"##4␣\gmath@do##2[##4]##3(##6)%
      \else
        \IfValueTF{##5}{%
          \iffontchar\gmath@ft"##5␣%
               \gmath@do##2[##5]##3(##6)%
          \else
            \gmath@restore{##1}{##2}{##3}{##4}{##5}{##7}%
          \fi}%
        {\gmath@restore{##1}{##2}{##3}{##4}{##5}{##7}}%
      \fi}%
    {\gmath@restore{##1}{##2}{##3}{##4}{##5}{##7}}%
  \fi
}% 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.
\def\gmath@restore##1##2##3##4##5##6{%
  \IfValueT{##6}%
  {\ifcsname␣##6\endcsname
      \edef\gmu@tempa{%
        \unexpanded{\gmath@doif{##1}{##2}{##3}[##4][##5]}%
        {\@xanxcs{##6}}\relax
      }\gmu@tempa
      \@xa\@gobbletwo␣% if family ##6 is defined, we gobble the other
           branch
    \fi
  }%
```

```
      \firstofone
      {\if\relax\@nx##2%
          \Restore@Macro##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

  \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

  \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.

  \pdef\rmopname##1##2##3{%
    \mathop {##1\kern \z@ \mathrm{##3}}\csname n##2limits@%
         \endcsname
  }%
\gmu@dogmathbase
  \DeclareCommand\gmu@dogmathbase{oC{\NoValue}}{%
    \Restore@Macro\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)%
\gmath@doif{221E}\infty\mathord(\gmath@famm)%
```

```
\gmath@doif{2248}\approx\mathrel(\gmath@famm)%
\gmath@doif{2260}\neq\mathrel(\gmath@famm)%
\let\ne\neq
\gmath@doif{00AC}\neg\mathbin(\gmath@famm)%
\gmath@doif{00AC}\nego\mathord(\gmath@famm)%
\gmath@do/\mathop
\gmath@do<\mathrel
\gmath@do>\mathrel
\gmath@doif{2329}\langle\mathopen(\gmath@famm)%
\gmath@doif{232A}\rangle\mathclose(\gmath@famm)%
\gmath@doif{2202}\partial\mathord(\gmath@famm)%
\gmath@doif{00B1}\pm\mathbin(\gmath@famm)%
\gmath@doif{007E}\sim\mathrel(\gmath@famm)%
\gmath@doif{2190}\leftarrow\mathrel(\gmath@famm)%
\gmath@doif{2192}\rightarrow\mathrel(\gmath@famm)%
\gmath@doif{2194}\leftrightarrow\mathrel(\gmath@famm)% if not
    present, \gmathfurther will take care of it if left and right arrows are
    present.
\gmath@doif{2191}\uparrow\mathrel(\gmath@famm)% it should be a de-
    limiter (declared with \gmath@delimif) but in a non-math font the de-
    limiters don't work (2008/11/19) and I don't think I'll ever need up- and
    down- arrows as delimiters.
\gmath@doif{2193}\downarrow\mathrel(\gmath@famm)%
\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.

```
\if\iffontchar\gmath@font"25CA␣0\else␣1\fi
   \iffontchar\gmath@font"25FB␣0\else\iffontchar%
        \gmath@font"25A1␣0\else␣2\fi\fi
  \gmath@do\lozenge[25CA]\mathord
  \gmath@doif{25FB}\square\mathord[25A1](\gmath@famm)% 'medium
      white square (modal operator)' of just 'white square'.
\fi
\gmath@doif{EB08}\bigcircle\mathbin(\gmath@famm)%
\gmath@doif{2227}\wedge\mathbin(\gmath@famm)%
\gmath@doif{2228}\vee\mathbin(\gmath@famm)%
\gmath@doif{0393}\Gamma\mathalpha(\gmath@famm)%
\gmath@doif{0394}\Delta\mathalpha(\gmath@famm)%
\gmath@doif{0398}\Theta\mathalpha(\gmath@famm)%
\gmath@doif{039B}\Lambda\mathalpha(\gmath@famm)%
\gmath@doif{039E}\Xi\mathalpha(\gmath@famm)%
\gmath@doif{03A3}\Sigma\mathalpha(\gmath@famm)%
\gmath@doif{03A5}\Upsilon\mathalpha(\gmath@famm)%
\gmath@doif{03A6}\Phi\mathalpha(\gmath@famm)%
\gmath@doif{03A8}\Psi\mathalpha(\gmath@famm)%
\gmath@doif{03A9}\Omega\mathalpha(\gmath@famm)%
\@xa\let\@xa\gmath@fam\csname␣symletters\gmath@version%
\endcsname
\edef\gmath@famm{symgmathItalic\gmath@version}%
\gmath@doif{03B1}\alpha\mathalpha(\gmath@famm)%
\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.
    \def\max{\rmopname␣\relax␣m{max}}%
    \def\min{\rmopname␣\relax␣m{min}}%
    \def\lim{\rmopname␣\relax␣m{lim}}%
    \def\sin{\rmopname␣\relax␣o{sin}}%
    \def\cos{\rmopname␣\relax␣o{cos}}%
    \def\tg{\rmopname␣\relax␣o{tg}}%
    \def\ctg{\rmopname␣\relax␣o{ctg}}%
    \def\tan{\rmopname␣\relax␣o{tan}}%
    \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.

```
\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}% to be able to redefine it
           when the big quantifier is Bauhaus-like.
    \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}%
```

```
11027             \lower\dimexpr(\ht0␣+␣\dp0)/2-\dp0␣-((\gma@tempa+%
                      \gma@tempb)/2-\gma@tempb)␣%
11028             \box0}}}}%
11029     \do\bigl\mathopen\larger
11030     \do\bigr\mathclose\larger
11031     \do\Bigl\mathopen\largerr
11032     \do\Bigr\mathclose\largerr
11033     \do\biggl\mathopen{\larger[3]}%
11034     \do\biggr\mathclose{\larger[3]}%
11035     \do\Biggl\mathopen{\larger[4]}%
11036     \do\Biggr\mathclose{\larger[4]}%
11039     \addtotoks\everymath{%
11042       \def\do##1##2{\gmu@storeifnotyet##1%
11043         \def##1{\ifmmode##2{\mathchoice
11044             {\hbox{\rm\char`##1}}{\hbox{\rm\char`##1}}%
11045             {\hbox{\rm\scriptsize\char`##1}}{\hbox{\rm\tiny%
                      \char`##1}}}%
11046           \else\char`##1\fi}}%
11047       \do\{\mathopen
11048       \do\}\mathclose
11050       \def\={\mathbin{=}}%
11051       \def\neqb{\mathbin{\neq}}%
11052       \let\neb\neqb
11053       \def\do##1{\gmu@storeifnotyet##1%
11054         \edef\gma@tempa{%
11055           \def\@xanxcs{\@xa\gobble\string##1r}{%
11056             \@nx\mathrel{\@nx##1}}}%
11057         \gma@tempa}%
11058       \do\vee␣\do\wedge␣\do\neg
11059       \def\fakern{\mkern-3mu}%
11060       \thickmuskip=8mu␣plus␣4mu\relax
11062       \gma@gmathhook
11063     }% of \everymath.
11064     \everydisplay\everymath
11065     \ifdefined\Url
11066       \ampulexdef\Url{\let\do}\@makeother
11067       {\everymath{}\let\do\@makeother}% I don't know why but the url
              package's \url typesets the argument inside a math which caused dig-
              its not to be typewriter but Roman and lowercase.
11071   \fi% of \ifdefined\Url.
11072 }% of \def\gmathfurther.

11074 \Store@Macro\mathchar@type

\gmath 11076 \DeclareCommand\gmath{oC{\NoValue}}{%
11077   \gmathbase[#1](#2)%
11078   \gmathfurther
11079   \IfValueT{#1}{\csname␣gmathhook#1\endcsname}% this allows adding version-
            specific stuff (I first used this for Fell fonts rescued with Garamond Premier)
11082 }

11084 \pdef\gmathscripts{%
11085   \addtotoks\everymath{\catcode`\^=7\relax␣\catcode`\_=8%
            \relax␣}%
```

```
    \everydisplay\everymath}

\pdef\gmathcats{%
    \addtotoks\everymath{\gmu@septify}%
    \everydisplay\everymath}

\emptify\gma@quantifierhook
\def\quantifierhook#1{%
    \def\gma@quantifierhook{#1}}

\emptify\gma@gmathhook
\def\gmathhook#1{\addtomacro\gma@gmathhook{#1}}

\def\gma@dollar$#1${{\gmath$#1$}}%
\def\gma@bare#1{\gma@dollar$#1$}%
\def\gma@checkbracket{\@ifnextchar\[%
    \gma@bracket\gma@bare}
\def\gma@bracket\[#1\]{{\gmath\[#1\]}\@ifnextchar\par{}{%
        \noindent}}
\def\gma{\@ifnextchar$%
    \gma@dollar\gma@checkbracket}

\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}}%
    \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}}%
    \def\gma@gmathhook{%
      \def\do####1####2####3{\gmu@storeifnotyet####1%
        \def####1{####3{%
            \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}{%
```

```
11141           \glyphname{u03A5.a}}}\mathrel␣% added 2009/9/11
11142   }% of \gma@gmathhook
11143 }% of \garamath.
```

#### Minion and Garamond Premier kerning and ligature fixes

»Ws« shall not make long »s« because long »s« looks ugly next to »W«.

```
\gmu@tempa 11152 \def\gmu@tempa{\kern-0,08em\penalty10000\hskiposp\relax
11153   s\penalty10000\hskiposp\relax}
11155 \protected\edef\Vs{V\gmu@tempa}
11157 \protected\edef\Ws{W\gmu@tempa}
11159 \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.

```
11180 \pdef\leftslanting@{%
\litdimen 11181   \def\litdimen{\strip@pt\fontdimen1\font␣ex}%
\litcorrection 11182   \def\litcorrection{%
11183     \ifhmode\null\nobreak\hskip\litdimen\relax\fi}%
\litkern 11184   \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.
11187     \leavevmode\null
11188     \kern-\litdimen\relax}%
\dilitkern 11189   \def\dilitkern{\kern\litdimen\litkern}%
11191   \pdef\litshape{%
11193     \litcorrection
11194     \itshape
11195     \@tempdima=-2\fontdimen1\font
11196     \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.
11200     \litcorrection
11201     \edef\gmu@tempa{%
11202       \@nx\addfontfeature{FakeSlant=\strip@pt\@tempdima}}% when not \edefed, it caused an error, which is perfectly understandable.
11205     \gmu@tempa}%
11208   \pdef\textlit##1{%
11209     {\litshape##1}}%
11211   \pdef\lit{\rm\litshape}%
```

```
  \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 it-
         shape 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.

```
  \pdef\em{%
    \ifdim\fontdimen1\font=\z@␣␣\itshape
    \else
      \ifdim\fontdimen1\font>\z@␣\litshape
      \else␣\upshape
      \fi
    \fi}%
  \pdef\emph##1{%
    {\em##1}}%
}% of \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{%
  o␣% 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 uncom-
     fortable, 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}}

    \gmu@tempa \def\gmu@tempa#1{%
    \do#1{\leavevmode
      \gmu@calculateslant{#1}% uses \gmu@tempa and \gmu@tempb, therefore 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@tempa0␣% \fake@onum@zero
\gmu@tempa1␣% \fake@onum@i
\gmu@tempa2␣% \fake@onum@ii

    \gmu@tempa \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

    \gmu@tempa \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}`.

```
\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@cepstnof@resc
```
  \def\gmu@cepstnof@resc##1:##2:\@nil{%
    \ifx:##2:\else␣RawFeature={\gmu@maybestripcomma##2,,%
      \@nil}␣\fi}%
```

\gmu@cepstnof@resd
```
  \def\gmu@cepstnof@resd##1/##2/\@nil{% to check whether font name contains / (which may not be true!)
```

```
    \ifx&##2&\@xa\@firstoftwo\else\@xa\@secondoftwo\fi}%
\gmu@cepstnof@rese  \def\gmu@cepstnof@rese##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@cepstnof@resa{%
```

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@cepstnof@resd\@xa\meaning\the\font//\@nil
    {% font name doesn't contain a slash
      \@xa\@xa\@xa\gmu@cepstnof@rese\@xa\meaning\the\font::%
           \@nil
      {% nor does it contain a colon
        \def\@nx\gmu@cepstnof@resb\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@cepstnof@resb\detokenize{select␣font
          "}####1:####2\detokenize{"}####3\@nx\@nil{%
          \ifx\fontspec#3%
            \@nx\@nx\@nx\fontspec[\@nx%
                 \gmu@cepstnof@resc####2::\@nx\@nil#4]{####1}%
          \else
            #3Font={####1},␣#3Features={\@nx%
                 \gmu@cepstnof@resc####2::\@nx\@nil#4}%
          \fi
        }%
      }%
    }% of 'no slash' case
    {% font name contains a slash
      \def\@nx\gmu@cepstnof@resb\detokenize{select␣font
        "}####1/####2\detokenize{"}####3\@nx\@nil{%
        \ifx\fontspec#3%
          \@nx\@nx\@nx\fontspec[\@nx\gmu@cepstnof@resc####2::%
               \@nx\@nil#4]{####1}%
        \else
          #3Font={####1},␣#3Features={\@nx%
               \gmu@cepstnof@resc####2::\@nx\@nil#4}%
        \fi
      }% of \gmu@cepstnof@resb
    }% of 'slash present' case
  }\gmu@cepstnof@resa
  \xdef#1{\@xa\@xa\@xa\gmu@cepstnof@resb\@xa\meaning\the\font%
       \@nil}%
}%
```

```
\gmu@stripcomma 11445 \def\gmu@stripcomma#1,{#1}
\gmu@maybestripcomma 11447 \def\gmu@maybestripcomma#1,,#2\@nil{#1}
11449 \pdef\gmu@setbasefont{\@xa\let\@xa\gmu@basefont\the\font}
11451 \let\setbasefont\gmu@setbasefont
\gmu@calc@scale 11453 \DeclareCommand\gmu@calc@scale{%
11454   O{1}% a factor,
11455   m% number of the fontdimen
11456   }
11457 {\begingroup
```

We 'descale' the current font:

```
11459   \gmu@cepstnof\fontspec[,␣Scale=1]\gmu@tempa
11460   \@xa\let\@xa\gmu@currfont@descaled\the\font
11461   \gmu@basefont
11462   \gmu@cepstnof\fontspec[,␣Scale=1]\gmu@tempa
```

now also the base font is descaled.

```
11464   \xdef\gmu@fontscale{%
11465     \strip@pt
11466     \dimexpr␣␣1pt␣*
11467      \numexpr\dimexpr#1\fontdimen#2\font\relax\relax␣/
11468     \numexpr\fontdimen#2\gmu@currfont@descaled\relax
11469     \relax}%
11470   \endgroup}
```

### Varia

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

```
\gmu@tilde 11478 \def\gmu@tilde{%
11479   \leavevmode\lower.8ex\hbox{$\,\widetilde{\mbox{␣}}\,$}}
```

Originally there was just \␣ instead of \mbox{␣} but some commands of ours do redefine \␣.

```
11484 \AtBeginDocument{% to bypass redefinition of \~ as a text command with various
          encodings
11486   \pdef\texttilde{%
11493     \@ifnextchar/{\gmu@tilde\kern-0,1667em\relax}\gmu@tilde}}
```

We prepare the proper kerning for "~/".

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.

```
11502 \foone{\catcode`\^^M\active}% the comment signs here are crucial.
\defobeylines 11503 {\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 11512 \def\deksmallskip{\vskip\smallskipamount}
\undeksmallskip 11513 \def\undeksmallskip{\vskip-\smallskipamount}
\dekmedbigskip 11514 \def\dekmedbigskip{\vskip\glueexpr␣\medskipamount+%
          \smallskipamount}
```

```
\dekmedskip 11515 \def\dekmedskip{\vskip\medskipamount}
\dekbigskip 11516 \def\dekbigskip{\vskip\bigskipamount}

 \hfillneg 11519 \def\hfillneg{\hskip␣opt␣plus␣-1fill\relax}
```

A mark for the **TO-DO!**s:

```
     \TODO 11524 \newcommand*{\TODO}[1][]{{%
           11525     \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 11560 \newcommand*\twocoltoc{%
           11561   \RequirePackage{multicol}%
 \@starttoc 11562   \def\@starttoc##1{%
           11563     \begin{multicols}{2}\makeatletter
           11564       \NamedInput␣% changed from \@input 2010/8/13.
           11565       {\jobname␣.##1}%
           11566       \if@filesw␣\@xa␣\newwrite␣\csname␣tf@##1\endcsname
           11567         \immediate␣\openout␣\csname␣tf@##1\endcsname␣\jobname␣
                             .##1\relax
           11568       \fi
           11569       \@nobreakfalse\end{multicols}%
           11570   }% of \starttoc
           11571 }

           11573 \@onlypreamble\twocoltoc
```

An equality sign properly spaced:

```
           11578 \pdef\equals{\hunskip${}={}$\ignorespaces}
```

And for the LaTeX's pseudo-code statements:

```
           11580 \pdef\eequals{\hunskip${}=={}$\ignorespaces}

           11582 \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 11602 \def\freeze@actives{%
           11603   \count\z@\z@
           11605   \@whilenum\count\z@<\@cclvi\do{%
           11606     \ifnum\catcode\count\z@=\active
           11607       \uccode`\~=\count\z@
           11608       \uppercase{\let~\relax}%
           11609     \fi
           11610     \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 be-
      cause if we don't, then verse's \everypar would be executed in a group
      and thus its effect lost.
  \Store@Macro\*%
  \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.

```
\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{%
  \pdef\acro##1{%
    \begingroup
```

```
      \acropresetting
      \gmu@acrospaces##1␣\gmu@acrospaces
      \endgroup
    }%
}
\emptify\acropresetting
\iteracro
```

`\gmu@acrospaces`
```
\def\gmu@acrospaces#1␣#2\gmu@acrospaces{%
  \gmu@acroiter#1\gmu@acroiter
  \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@acroiter`
```
\def\gmu@acroiter#1{%
  \gmu@notif␣{x\@xa\@xa}␣{\@firstofmany#1\@undefined\@nil␣%
      \gmu@acroiter}
  {\gmu@acrokernel{#1}%
```
and iterate
```
    \gmu@acroiter
  }% of if sentinel or not
  {}%
}
```

`\gmu@acrokernel`
```
\def\gmu@acrokernel#1{%
  \gmu@if␣{cat}␣{a\@xanx\@firstofmany#1\@undefined\@nil}
  {\gmu@if␣{num}␣{`#1=\uccode`#1\space␣}% a space to delimit numer (with-
```
out it further macros were expanded which in this case are `\expandafter% \@firstoftwo\else\@secondoftwo` and this made the test didn't work.)
```
    {{\acrocore{#1}}}%
    {{#1}}% tu było \smallerr
  }%
  {#1}%
}
```

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.

```
\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}}
\pdef\usc#1{{\addfontfeature{Letters=UppercaseSmallCaps}#1}}
```

`\uscacro`
```
\def\uscacro{\let\acrocore\usc}
```

`\SecondClass` moved to gmbase.

Cf. *The TEX book* 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.

```
\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}
\def\gmu@fracfontsetup{%
  \smaller[3]\addfontfeature{FakeBold=1}}
\def\dekfraccsimple{%
  \let\dekfracc@args\gmu@dekfraccsimple
}
\@ifXeTeX{\def\dekfraccslash{\char"2044␣}}
{\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:

```
\newcommand*\ikern{\,\penalty\@M\hskip\z@skip\relax}
```

### Faked small caps

```
\def\gmu@scapLetters#1{%
  \ifx#1\relax\relax\else% two \relaxes to cover the case of empty #1.
    \ifcat␣a\@nx#1\relax
      \ifnum\the\lccode`#1=`#1\relax
        {\fakescapscore\MakeUppercase{#1}}% not Plain \uppercase because that works bad with inputenc.
      \else#1%
      \fi
    \else#1%
    \fi%
    \@xa\gmu@scapLetters
  \fi}%
\def\gmu@scapSpaces#1␣#2\@nil{%
  \ifx#1\relax\relax
  \else\gmu@scapLetters#1\relax
  \fi
  \ifx#2\relax\relax
  \else\afterfi{\␣\gmu@scapSpaces#2\@nil}%
  \fi}
\def\gmu@scapss#1\@nil{{\def~{{\nobreakspace}}%
    \gmu@scapSpaces#1␣\@nil}}% %␣\def\\{{\newline}}\relax adding redefinition of \\ caused stack overflow. Note it disallows hyphenation except at \-.
```

```
\pdef\fakescaps#1{{\gmu@scapss#1\@nil}}
\let\fakescapscore\gmu@scalematchX
```

Experimente z akcentami patrz no3.tex.

\tinycae
```
\def\tinycae{{\tiny\AE}}% to use in \fakescaps[\tiny]{…}
\RequirePackage{calc}
```

wg \zf@calc@scale pakietu fontspec.

```
\@ifpackageloaded{fontspec}{%
```
\gmu@scalar
```
  \def\gmu@scalar{1.0}%
```
\zf@scale
```
  \def\zf@scale{}%
```
\gmu@scalematchX
```
  \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}}%
      \gmu@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 \fakesc@extrascale
```
\def\fakescextrascale#1{\def\fakesc@extrascale{#1}}
```

**See above/see below**

To generate a phrase as in the header depending of whether the respective label is before of after.

\wyzejnizej
```
\newcommand*\wyzejnizej[1]{%
  \edef\gmu@tempa{\gmu@ifundefined{r@#1}{\arabic{page}}{%
      \@xa\@xa\@xa\@secondoftwo\csname␣r@#1\endcsname}}%
  \ifnum\gmu@tempa<\arabic{page}\relax␣wy\.zej\fi
  \ifnum\gmu@tempa>\arabic{page}\relax␣ni\.zej\fi
  \ifnum\gmu@tempa=\arabic{page}\relax␣\@xa\ignorespaces\fi
}
```

### `luzniej` and `napapierki`—environments used in page breaking for money

The name of first of them comes from Polish typesetters' phrase "rozbijać [skład] na papierki"—'to broaden [leading] with paper scratches'.

\napapierkistretch
```
\def\napapierkistretch{0,3pt}%
```
It's quite much for 11/13pt leading.

\napapierkicore
```
\def\napapierkicore{\advance\baselineskip%
  by␣optplus\napapierkistretch\relax}

\DeclareEnvironment{napapierki}{s}{%
  \par\IfValueT{#1}{\global}%
  \napapierkicore}
{%
  \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{\kerno,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{\hskipo.2em␣pluso.1em\relax
  \pauzacore
  \@ifnextchar,{% 2009/11/22 added a special case of a comma following pauza
  }{\hskip.2em␣pluso.1em\relax\ignorespaces}}%
```

\ppauza@skipcore
```
\def\ppauza@skipcore{\unskip\penalty10000\hskipo.2em␣
    pluso.1em\relax
      \ppauza@dash\hskip.2em␣pluso.1em\ignorespaces}

\AtBeginDocument{%
  \pdef\pauza{%
    \ifhmode
      \unskip\penalty10000
      \hskipo.2em␣pluso.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.

```
  \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:

```
  \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`…

```
   \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}}}%
      }%
    \fi
  }%
\emdash  \def\emdash{\char`\-}
}% of at begin document

\longpauza \def\longpauza{\def\pauzacore{-}}
\pauzacore \longpauza
\shortpauza \def\shortpauza{%
\pauzacore   \def\pauzacore{\hbox{-\kern,23em\relax\llap{-}}}}%
\ppauza@dash \def\ppauza@dash{-}%

\else␣% not XeTeX
\longpauza \def\longpauza{\def\pauzacore{---}}
\pauzacore \longpauza
\shortpauza \def\shortpauza{%
\pauzacore   \def\pauzacore{--\kern,23em\relax\llap{--}}}%
\ppauza@dash \def\ppauza@dash{--}%

\fi% of if XeTeX or not.

\ifdefined\XeTeXversion
```

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 –.

```
\foone{\catcode`-\active␣\catcode`-\active␣\catcode`-\active}{%
    %
\adashes   \def\adashes{\AtBeginDocument\adashes}% because \pauza is defined
             at begin document.
\adashes   \AtBeginDocument{\def\adashes{%
     \-      \catcode`-\active␣\def-{\-}%
     \-      \catcode`-\active␣\def-{\-}%
             \addtomacro\dospecials{\do\-\do\-}%
             \addtomacro\@sanitize{\@makeother\-\@makeother\-}%
             \addtomacro\gmu@septify{\do\-13\do\-13\relax}%
}}}
\else
\relaxen\adashes
\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 12046 \def\gmu@dywiz{\ifmmode-\else
           12047   \ifvmode-\else\afterfifi\dywiz\fi\fi}%
           12049 \foone{\catcode`-\active}{% aktivnyj diefis aktywny dywiz active hyphen
  \ahyphen 12050     \def\ahyphen{\let-\gmu@dywiz\catcode`\-\active}}
```

To get current time. Works in ε-TEXs, including XETEX. `\czas` typesets 19.22 and `\czas[:]` typesets 19:22.

```
    \czas 12055 \newcommand*\czas[1][.]{%
          12056   \the\numexpr(\time-30)/60\relax#1%
          12057   \@tempcnta=\numexpr\time-(\time-30)/60*60\relax
          12058   \ifnum\@tempcnta<10␣0\fi\the\@tempcnta}
 tytulowa 12060 \newenvironment*{tytulowa}{\newpage}{\par\thispagestyle{%
              empty}\newpage}
```

To typeset peoples' names on page 4 (the editorial page):

```
\nazwired 12063 \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).

```
           12077 \pdef\polskadata{%
\gmu@datefsl 12078   \DeclareCommand\gmu@datefsl{%
           12079     ##1␣␣␣␣Q{0123456789\bgroup}>iT{/-}␣% year
           12080     ##2␣␣␣␣Q{0123456789\bgroup}>iT{/-}␣% month
           12081     ##3␣␣␣␣Q{0123456789\bgroup}␣% day
           12082     ##4␣␣␣␣T{,}
           12083     ##5␣␣␣␣K{##1\gmu@datefsl}␣% additional stuff after comma
           12084   }{%
           12085       \IfValueF{##2}{\PutIfValue{##3}}%
           12086       \IfValueT{##2}{%
           12087         \@tempcnta=0##3\relax\the\@tempcnta
           12088         \ifcase##2\relax\or␣stycznia\or␣lutego%
           12089         \or␣marca\or␣kwietnia\or␣maja\or␣czerwca\or␣
                             lipca\or␣sierpnia%
           12090         \or␣września\or␣października\or␣listopada\or␣
                             grudnia\else
           12091         {}%
           12092         \fi}%
           12093         \IfValueT{##1}{\space␣##1}%
           12094         \PutIfValue{##4}\IfValueT{##5}{␣##5}%
           12095       }% of \gmu@datefsl.
           12096   }% of \polskadata
           12098 \polskadata
```

For documentation in English:

```
\pdef\englishdate{%
  \DeclareCommand\gmu@datefsl{%
    Q{0123456789\bgroup}>iT{/-}␣% (1) year
    Q{0123456789\bgroup}>iT{/-}␣% (2) month
    Q{0123456789\bgroup}␣% (3) day
    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.

```
\newif\ifdate

\pdef\bidate#1{%
  \gmu@datefsl#1\gmu@datefsl
}

\pdef\linedate{\gmu@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{\DateFont␣\bidate{#1}}%
  \nopagebreak
  \else% %\ifnum\arabic{dateinsection}>0\dekbigskip\fi
    \addvspace{\bigskipamount}\possvfil
  \fi}% end of \linedate.

\newcommand*\DateFont{\footnotesize\itshape}

\let\linedate@hook\@gobble

\let\dateskipamount\medskipamount

\pdef\rdate{\let\date@line\rightline␣\linedate}

\def\date@left#1{\par{%
    \raggedright#1%
    \leftskip\z@skip
    \@@par}}%

\pdef\ldate{%
  \let\date@line\date@left
  \linedate}
```

```
\runindate 12162 \newcommand*\runindate[1]{%
           12163   \paragraph{\footnotesize\itshape␣\gmu@datef#1\gmu@datef}%
           12164   \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.

```
           12167 \let\thedate\ldate
```

```
           12170 \pdef\zwrobcy#1{\emph{#1}}␣% ostinato, allegro con moto, garden party etc.,
                       także kompliment
```

```
           12173 \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 12179 \newenvironment{maszynopis}[1][]{#1\ttfamily
           12180   \hyphenchar\font=45\relax% this assignment is global for the font.
           12181   \@tempskipa=\glueexpr\rightskip+\leftskip\relax
           12182   \ifdim\gluestretch\@tempskipa=\z@
           12183   \tolerance900
```

it worked well with tolerance = 900.

```
           12185   \advance\rightskip␣by\z@␣plus0,5em\relax\fi
           12186   \fontdimen3\font=\z@% we forbid stretching spaces…
            %␣␣\fontdimen4\font=\z@ but allow shrinking them.
           12188   \hyphenpenalty0␣% 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.
           12191   \Store@Macro\pauzacore
 \pauzacore 12192   \def\pauzacore{-\rlap{\kern-0,3em-}-}%
           12193 }{\par}
```

```
           12197 \pdef\justified{%
           12198   \leftskip=1\leftskip% to preserve the natural length and discard stretch
                       and shrink.
           12200   \rightskip=1\rightskip
           12201   \parfillskip=1\parfillskip
           12202   \advance\parfillskip␣by␣0sp␣plus␣1fil\relax
           12203   \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 12208 \DeclareCommand\fullpar{%
           12209   T{+-}%
           12210   Q{+-0123456789}␣% optional looseness (most probably negative)
           12211 }{%
           12212   \begingroup
           12213   \IfValueT{#1}{\looseness=#1\IfValueTF{#2}{#2}{1}\relax
           12214     \multiply\tolerance␣by␣\tw@
           12215   }%
           12216   \fullparcore
           12217   \par
           12218   \endgroup}
```

```
           12220 \pdef\fullparcore{%
           12221   \hunskip
```

```
\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`
```
\newcommand*\twoparinit{% the name stands for 'last paragraph line's length minimum 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}
      \gobblespace
    }{%
      \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}
      \gobblespace
    }{%
      \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{%
\restoreparindent
    \def\restoreparindent{\parindent\@parindent}%
  }% of \AtBeginDocument.
}% of \twoparinit.
```

#### For dati under poems

Or explanations under results of time.

```
\gmu@leftskipcorr \def\gmu@leftskipcorr{%
  \kern1\leftskip
  \ifcsname␣\@currenvir␣leftskip\endcsname
    \kern-1\csname␣\@currenvir␣leftskip\endcsname
  \fi
}

\wherncore \DeclareCommand\wherncore{om}{%
      % [#1] optional value of \hskip of (left) indent of the parbox. If absent,
        parbox is aligned right;
      % [#2] ot ext for the datum parbox.
  \IfValueTF{#1}{\leftline{%
      \kern1\leftskip
      \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}%
      \kern1\rightskip
    }% of \rightline,
    \setprevdepth
  }% of ValueF{#1},
}% of \wherncore.

\whern@parbox \DeclareCommand\whern@parbox{%
  T␣{\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 t he \parbox's contents.

  \parbox[#2]{#3}{%
    \IfValueTF{#1}{#1}{\leftskip}=osp␣plus␣\textwidth
    \parfillskiposp\relax
```

```
      \let\\\linebreak
      \disobeylines
      \whernfont␣#4\unskip\strut\endgraf
      \getprevdepth
    }% of \parbox,
}% of \whern@parbox.

\whern \def\whern{%
    \endgraf\nopagebreak
    \gmu@ifstar{\wherncore}%
    {\vskip\whernskip\wherncore}}

\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{%
      \gmu@leftskipcorr
      \IfValueTF{#2}{\whern@parbox\rightskip[b][#2]}%
      {\whern@parbox\rightskip[b]}%
      {#3}%
    }%
    \setprevdepth
    \nopagebreak\relax
    \@ifenvir{quote}{\noindent\ignorespaces}{}}
```

### Thousand separator

```
\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 \expandafters and \the. Let's do it only once for all:

```
\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{
    D␣{\NoValue}␣% decimal argument
  }{%
```

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).

#### 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
}

\pdef\gmu@AT@ampulex#1{%
  \typeout{@@@@@^^J^^J\@nx#1^^J^^J%
    @@@@␣␣␣␣meaning:^^J\meaning#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.
}

\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.

```
  \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
```

### Only this paragraph

```
\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}

\pdef\centerthis{% 2009/12/15
  \begingroup
  \rightskip=1\rightskip␣plus␣\hsize
  \leftskip=1\leftskip␣plus\hsize
  \parfillskip=\z@skip
  \endgraf\endgroup}
```

### 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\gmu@smarttilde
  \@xa{\@xa␣T\@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>\dimexpr10pt-1sp\relax
          168dd
        \else
          144dd
        \fi
        \nobreakspace␣{}%
      \else
        \␣%
      \fi
    }% of \gmu@ifstar's else,
  }% of \gmu@smarttilde,
  \let~\gmu@smarttilde
}% of \TrzaskaTilde.
```

### A really empty page

Copied from Marcin Woliński's macros.

\clearemptydoublepage

```
\newcommand{\clearemptydoublepage}{%
      \newpage{\pagestyle{empty}\cleardoublepage}}

\foone\obeylines{%
  \def\disobeylines{% for arguments in which line end is active to simulate
      normal behaviour
    \ifnum\catcode`\^^M=\active%
```

\disobeylines

```
      \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}%
}
```

The \⋆ CS and active ⋆ should be defined different to make them distinguishable by tests, especially with \gmu@ifstar in mind.

```
\DeclareCommand\⋆{Q{0123456789}{1}}
⟨⋆OperaOmnia⟩
```

Modified in gmmacros to accept optional brace and replace *its* tokens with stars.

```
⟨/ OperaOmnia⟩
{\gmu@flexhyphen
  \gmu@star@loopo{#1}\relax
}%
```

\gmu@star@loop

```
\def\gmu@star@loop#1#2{% this is an expandable loop as in The ε-TEX Manual p. 9.
  \ifnum#1<\numexpr#2\relax%
    \gmu@lowstar
    \gmu@flexhyphen
    \@xa\gmu@star@loop
    \@xa{\number\numexpr#1+1\@xa}%
    \@xa{\number#2\@xa}%
  \fi}
```

```
\ifdefined\XeTeXversion
```

```
\foone{%
  \catcode`„\active
  \catcode`”\active
  \catcode`’\active
  \catcode`-\active
  \catcode`-\active
}{%
```

\activequotes

```
  \def\activequotes{%
    \incsdef„{}{\@ifnextchar-%
      {\lv\llap{\string„}\pauzadial}{\string„}}%
    \incsdef”{}{\string”\@ifnextanyRS{.,}{\quotkern}{}}%
    \incsdef’{}{\string’\@ifnextanyRS{.,}{\quotkern}{}}%
    \catcode`„\active
    \catcode`”\active
    }%
```

\activepunctsQ

```
    \def\activepunctsQ␣{„”’--}%
}
```

(Cyrillic) iotified e. The special delimiter that will be lost.

```
\catcode`ѥ\active
```

defined later, here for proper catcode.

\ac

```
\DeclareCommand\ac␣{b}␣{%
```

```
  \IfValueTF{#1}
  {%
    \acro{#1}%
  }
  {\ac@u}%
}
```

`\ac@kernel` `\def\ac@kernel#1{{\gmu@acrokernel␣{#1}}}%` `#1` in braces because `\acro{core}` valid in *Dzienniczek* uses `\lowercase` (that requires braced text).

Delimiters of until-iterating arguments

They don't contain space(s)!

```
\@xa\def\@xa\@dc@basicdelims\@xa{%
  \two@Ms␣\par␣\relax
  \bgroup
  \egroup␣\begingroup␣\endgroup
  \begin␣\end
  $%
  \(\)\[\]% other math delims
  &\\% tabular delims
}
```

Now *they* do:

```
\let\@dc@basicdelimsp\@dc@basicdelims
\@xa\addtomacro\@xa\@dc@basicdelimsp
\@xa{\all@spaces}
\@xa\def\@xa\@dc@punctanddelims
\@xa{\@dc@basicdelims␣:,.;?!„"«»‹ ›"'`'}%
\let␣\@dc@acpunctanddelims␣\@dc@punctanddelims
\@xa\addtomacro\@xa\@dc@acpunctanddelims
\@xa{\activepunctsQ}
```

now, it's the list containing spaces

```
\let␣\@dc@acpunctanddelimsp␣\@dc@punctanddelims
\@xa\addtomacro\@xa\@dc@acpunctanddelimsp
\@xa{\all@spaces}
\addtomacro\@dc@acpunctanddelimsp{~}
```

acro iterating until

`\ac@u`

```
\DeclareCommand\ac@u{
  >\@xa␣U{\@dc@acpunctanddelimsp␣()[]<>ϵ}␣\default{}
  \eacher␣{\ac@kernel}
  >iW{ϵ}
}
{\text{#1}}
\fi␣% of if XeTeX of l. 12624.
```

### 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}
```

⟨/ typos⟩

### The gmparts package—in/exclusion of parts of one file analogous to `\include`

⟨utils⟩ `␣␣␣␣\gmu@PackOptionX{parts}`
⟨*parts⟩

```
\RequirePackage{gmcommand}
```

### `\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}

\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`
```
\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.

```
\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.

```
\def\filepart#1{\relax
  \ifnum\@auxout=\@partaux
  \@latex@error{\string\filepart\space␣cannot␣be␣nested}\@eha
  \else\afterfi{\@filepart#1␣}\fi}

\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\gmu@filepart@resa:=\@partlist\do{%
      \ifx\gmu@filepart@resa\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{\Restore@MacroSt␣{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 12864.
    \let\@auxout\@mainaux
}

\Store@Macro\endfilepart

\def\nofileparts{%
    \let\filepart\@gobble
\endfilepart \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.

```
\def\includecountfix{%
  \def\@wckptelt##1{%
    \immediate\write\@partaux{%
      \providecounter{##1}% to provide the font counters defined in parts of
                           the document.
      \string\setcounter{##1}{\the\@nameuse{c@##1}}}}%
}

\pdef\providecounter#1{%
#1  \unless\ifcsname␣c@#1\endcsname\newcounter{#1}\fi}

⟨/ parts⟩
```

### The gmurl package

```
⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{url}
⟨*url⟩

\RequirePackage{gmcommand}
```

#### hyperref's \nolinkurl into \url*

```
\def\urladdstar{%
  \AtBeginDocument{%
    \@ifpackageloaded{hyperref}{%
      \Store@Macro\url
      \pdef\url{\gmu@ifstar{\nolinkurl}{\storedcsname{url}}}%
    }{}}}
```

```
\@onlypreamble\urladdstar
```

### 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{%
  \pdef\UrlFix{\AtBeginDocument{%
      \@ifpackageloaded{url}{\gmu@UrlFix}{}}%
    \relaxen\UrlFix}%
  \AtBeginDocument{%
    \pdef\UrlFix{%
      \@ifpackageloaded{url}{\gmu@UrlFix}{}%
      \relaxen\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␣}
```

```
\def\gmu@UrlFix{%
```

default style assignments

```
  \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\$}%
  \def\UrlBigBreaks{\do\:}%
  \def\UrlNoBreaks{\do\(\do\[\do\{}%
  \def\UrlSpecials{%
    \do\␣{\hbox{\visiblespace}}\do\^^M{\hbox{\visiblespace}}}%
```

```
  \def\Url@Format##1{%
    \UrlFont
    \ifdefined\verbatim@specials
```

```
      \catcode`\>\active
      \verbatim@specials
      \verbatim@mathhack
    \fi␣% setting of the escape char, begin and end group and optionally math
        shift, defined in gmverb.
    \gmu@UrlSetup
    \UrlLeft
    \edef\gmu@theendlinechar{\the\endlinechar}%
    \endlinechar\m@ne
    \kern\z@% to forbid hyphenating the first word if the URL begins with a word
    \hyphenchar\font=\UrlHyphenchar\relax
    \let\-\gmu@discretionaryhyphen
    \scantokens{##1}%
    \endlinechar\gmu@theendlinechar\relax
    \UrlRight
}% of \Url@Format.

\edef\UrlHyphenchar{%
  \ifdefined\gmv@hyphenchar\gmv@hyphenchar
  \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.

\def\verbatim@mathhack{%
  \ifdefined\verbatim@specials@list
    \@xa\verbatim@mathhack@\verbatim@specials@list
  \fi
}%

\def\verbatim@mathhack@##1##2##3##4##5##6{%
  \IfValueT{##4}{%
    \edef\gmu@thinmuskip{\the\thinmuskip}%
    \edef\gmu@medmuskip{\the\medmuskip}%
    \edef\gmu@thickmuskip{\the\thickmuskip}%
    \begingroup
    \lccode`\~=`##4\lowercase{%
      \endgroup\def~####1~}%
    {$\thinmuskip\gmu@thinmuskip\relax
      \medmuskip\gmu@medmuskip\relax
      \thickmuskip\gmu@thickmuskip\relax
      ####1%
      $}%
    \catcode`##4\active
  }%
}%

\def\gmu@UrlSetup{%
  \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 3738.
      \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 13029
      }\gmu@tempa
    \fi
    \addfontfeature{Ligatures=NoCommon,␣Mapping=none}% instead of 'doing'
        % \verbatim@nolig@list.
  }% of \gmu@UrlSetup.

  \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
13121       {\ifmmode\char`##2% else
13122         \else\gmu@urlbreakable{##1}{##2}%
13123         \fi}%
13124     }% of \addtomacro's argument \ifnum true.
13125     \else
13126     {%
13127       {\char`##2\csname␣gmu@dbl\string##2kern\endcsname}{%
              \char`##2}%
13128     }% of \addtomacro's argument \ifnum false.
13129     \fi
13130       \catcode`##2=\active
13131     }% of \gmu@doUrlMath.
13133   \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.
13140     \begingroup
13141     \lccode`\~=`##2\lowercase{%
13142       \endgroup\def~{\@ifnextchar~}%
13143       \@xa\addtomacro\@xa~}% of \lowercase.
13144     \ifnum␣##1<\@M
13145     {%
13146       {\ifmmode\char`##2\else$##3\m@th$\fi}%
13147       {\ifmmode\char`##2%
13148         \else\discretionary{\hbox{$##3\m@th$}}{}{\hbox{$##3%
                \m@th$}}%
13149         \fi}%
13150     }% of \addtomacro's argument if num true.
13151     \else
13152     {%
13153       {\ifmmode\char`##2\else$##3\m@th$\fi}{\ifmmode\char`##2%
              \else$##3\m@th$\fi}%
13154     }% of \addtomacro's argument if num false.
13155     \fi
13156     \catcode`##2=\active
13157   }% of \gmu@doUrlMathAc.
13159   \pdef\gmu@url@rigidbreak##1##2{\discretionary{\char`##2}{}{%
          \char`##2}}%
13161   \pdef\gmu@url@flexbreak##1##2{\penalty\@M␣\hskip\z@␣
          pluso,o3em
13162     \char`##2\penalty##1\hskip\z@␣pluso,o3em\relax}%
13164   \let\gmu@urlbreakable\gmu@url@flexbreak
13166   \def\Url@z##1{%
```

Do any hyper referencing due to hyperref (or perform a url-def)

```
13168     \Url@HyperHook
```

Now do the formatting in a group (can also have `\Url@HyperHook` take this as an argument).

```
13171     {\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.
\DeclareCommand\UrlSlashKern{O{tt}m}%
{\AtBeginDocument{%
    \@nameedef{url@#1style}{\def\@nx\UrlFont{%
        \@xanxcs{#1family}%
        \def\@xanxcs{gmu@dbl\string\/kern}%
        {\kern#2\relax}%
      }% of \UrlFont
    }% of \url#1style
    \urlstyle{#1}%
  }% of \AtBeginDocument
}% of \UrlSlashKern
\def\DeclareUrlCommand#1#2{\pdef#1{\leavevmode\begingroup␣#2%
     \Url}}
  %% [#1][#1]{\def}{#1}{\pdef#1}
\foolc␣~␣:␣{%
  \@ifXeTeX{%
    \def\metaat~{%
      \penalty\@M␣\hskip\z@skip
      \meta{at}% it's a Cyrillic »a«!
      \penalty\exhyphenpenalty
      \hskip\z@skip
    }%
    \def\metadot~{%
      \penalty\@M␣\hskip\z@skip
      \meta{dot}% it's a Cyrillic »o«!
      \penalty\exhyphenpenalty
      \hskip\z@skip
    }%
  }% of if XeTeX
  {%
    \def\metaat~{\PackageError{gmurl}{Command␣\bslash␣metaat
        works␣only␣in␣XeTeX}@}%
    \def\metadot~{\PackageError{gmurl}{Command␣\bslash␣metaat
        works␣only␣in␣XeTeX}.}%
  }% of if not XeTeX
}% of \foolc
⟨/ url⟩
```

### The gmRCS package

```
⟨utils⟩ ␣␣␣␣\gmu@PackOptionX{RCS}
⟨*RCS⟩
```

```
\def\ParseRCSVersion$Id%
:␣#1.#2,v␣#3␣#4␣#5␣#6␣#7${%
  \def\RCSFileName{#1}%
  \def\RCSFileExt{#2}%
  \def\RCSFile{#1.#2}%
  \def\RCSVersion{#3}%
  \def\RCSDate{#4}%
  \def\RCSTime{#5}%
  \def\RCSAuthor{#6}%
}
```

And the expandable version of those above (when we only one datum at one place)

```
\def\defRCSeDatum
#1% datum name
#2% datum definition (basically #number)
{%
  \@namedef{eRCS#1}$Id%
  :␣##1.##2,v␣##3␣##4␣##5␣##6␣##7${#2}%
}
\defRCSeDatum{File}{#1.#2}␣% \eRCSFile
\defRCSeDatum{Version}{#3}% \eRCSVersion
\defRCSeDatum{Date}{#4}% \eRCSVDate
\defRCSeDatum{Time}{#5}% \eRCSVTime
\defRCSeDatum{Author}{#6}% \eRCSVAuthor
```

⟨/ RCS⟩

### Back to gmutils

```
⟨*utils⟩
\ExecuteOptionsX{command,␣envir,␣ampulex,␣relsize,␣meta,␣
      logos,
  notonlypream,␣%
  %% mw=off,
  typos,␣parts,␣url}
\ProcessOptionsX
\def\doifdefined#1{\ifdefined#1\@xa#1\fi}
\doifdefined\gmu@Require@command
\doifdefined\gmu@Require@envir
\doifdefined\gmu@Require@ampulex
\doifdefined\gmu@Require@relsize
\doifdefined\gmu@Require@meta
\doifdefined\gmu@Require@logos
\doifdefined\gmu@Require@notonlypream
\doifdefined\gmu@Require@mw
\doifdefined\gmu@Require@typos
\doifdefined\gmu@Require@parts
\doifdefined\gmu@Require@url
\doifdefined\gmu@Require@RCS
```

### 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 these days, '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.)

```
\def\heshe{they}
\def\hisher{their}
\def\himher{them}
\def\hishers{theirs}

\def\HeShe{They}
\def\HisHer{Their}
\def\HimHer{Them}
\def\HisHers{Theirs}
```

⟨/ utils⟩

```
\endinput
```

End of file 'gmutils.gmd'.

□

### Change History

gmampulex v0.93
  \ampulexlet:
    added, 8734
gmampulex v0.94
  \ampulexlet:
    made xparse-ish and \ampulexset
    removed, 8734
gmampulex v0.98
  \ampulexlet:
    first argument (the prefix) made of the
    Q type, 8734
gmampulex v0.994
  \ampulexdef:
    rewritten not to use the arguments 7–9
    explicitly not to wrap them in a
    temporary macros, and to accept
    single hashes (instead of quadruple)
    in arguments 5 and 6. The temporary
    macros renamed to
    \gmu@reserveda and
    \gmu@reservedb. The body moved
    to \ampulexlet with adding
    another CS argument to let let a
    macro not necessarily redefine it itself
    and this command made a particular
    case of that new one, 8832
  \ampulexlet:
    added as a more general version of
    \ampulexdef (which now is a
    particular use of this command), 8716
gmbase v0.75
  \@ifnif:
    added, 2707
  \@xifncat:
    \let for #1 changed to \def to allow
    things like \noexpand~, 2667
  \xiibackslash:
    \let for #1 changed to \def to allow
    things like \noexpand~, 2627
gmbase v0.88
  \ResetMacros:
    added, 3340, 3350, 3354
gmbase v0.92
  \@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, 2208
  \@parindent:
    added, 3772
gmbase v0.93
  \@gobbleeight:
    added, 911, 939

\pdef:
added, 885
\pprovide:
added, 926
gmbase v0.94
\@gobbleeight:
\if removed from parameters' string, 1239
\@parindent:
\includegraphics works well in XeTeX so I remove the complicated version with \XeTeXpicfile, 3681
\@xau:
added, 806
\addto@forlist:
added. All \@ifundefineds used by me changed to this, 2396
made robust to unbalanced \ifs and \fis the same way as LaTeX's \@ifundefined (after a heavy debug :-), 2396
\addtomacro:
order of arguments reversed, 2320
\RestoringDo:
the XeTeX version enriched with \iffontchar due to lack of bullets with the default settings reported by Morten Høgholm and Edd Barrett, 3506
gmbase v0.96
\RestoringDo:
moved here from my private document class, 3511
gmbase v0.97
\@gobbleeight:
added, 1203
gmbase v0.98
\@XAtoks:
moved here from gmdoc and rewritten, including split to \IfAmong because the latter is needed in \DeclareCommand, 1708
\@gobbleeight:
renamed from \@secondofmany, 1203
\@ifnif:
added test for \egroup, 2792
extracted from \@ifnextac, 2787
renamed from \@ifstar since something redefines \@ifstar, 2766
\@parindent:
added, 3649
moved here from my personal macro package since I needed it in the documentation of gmutils, 3643
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, 3842
\g@relaxen:
split from \IfAmong, 1337
gmbase v0.99
\@parindent:
moved to a separate macro from \@ifenvir and made symmetric to both arguments, 3842
gmbase v0.991
\@xau:
made 1-parameter, 806
\@xifncat:
inner macro fixed to handle active chars more properly (more as \if would do it), 2667
gmbase v0.992
\sups␣:
added, 2482
gmbase v0.993
\@parindent:
made \protected after an error at work, 3772
redefined deeply and made expandable thanks to \strcmp. Also renamed from \gmu@ifedetokens(the \gmu prefix added), 3842
since \gmu@ifedetokens turned to be expandable, we make this macro \protected, 3789
General:
package cut out from gmutils due to need of using \DeclareCommand with minimal chance of interference of other stuff, 4164
\RestoreMacros:
added, 3290
\StoreMacro:
fixed asymmetry with the unstarred version: till today the unstarred version for an undefined CS undefine the storage CS and the starred version made it \relax. Now both undefine, 3157
gmcommand v0.90
\XeTeXthree:
adjusted to the redefinition of \verb in xlxtra 2008/07/29, 8016
gmcommand v0.91
General:
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, 8672

gmcommand v0.94

General:

removed \unex@namedef and \unex@nameuse, probably never really used since they were incomplete: \edef@other undefined, 8672

The code from ancient xparse (1999) of TeXLive 2007 rewritten here, 8672

gmcommand v0.98

\ArgumentCatcher@PLoop:

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, 7137

\DeclareCommand:

added the S/T spec parsing, the s spec parsing rewritten to be a shorthand for S{*}, unused code removed, 5516

\enoughpage:

\par removed since to let it be used for \pagebreak inside a paragraph, 8107

made ε-TeX-ish, 8107

made ifthenelse-like with 'then' and 'else' optional default ⟨*nothing*⟩ and \newpage resp., 8107

\IfValueF:

added the \@bsphack—\@esphack option, 7278

added the Q{⟨*tokens*⟩} argument type (a word over the alphabet ⟨*tokens*⟩), 7278

gmcommand v0.993

\@UseStored:

added, 8351

General:

the pseudo-argument type \afterassignment renamed to =, 8672

\DeclareEnvironment:

we make the \end… command \DeclareCommand ed for the symmetry, for simplifying definition of \envirlet in particular, 7910

\if@dc@Mmode@:

made for-eaching to make it behave as expected, i.e. that latter settings prævalent over the former, 6119

\if@dc@xaeacher@:

implemented the \SameAs feature, 5516

\IfValueF:

added to modify naming of the inner macro of a \DeclareCommand ed commands: preceding the names with a backslash is OK for the letter CS'es, but it may cause a collision for the active chars. Therefore we both precede the name *and* succede it with a bslash, 7266

This strikingly simple idea of storing the specifiers' sequence in a macro came to my head only after some three years of developing \DeclareCommand, 7223

KV@gmutils.gmd@gm#1:

Incompatibilities with earlier versions: because of making all the specifiers "case-insensitive", i.e. aliasing the uppercases as lowercase, the S spec. ceased to be a **S**ingle-token-catcher and became a **S**tar-token-catcher. Moreover, the parameters for *all* the one-parameter specifiers became optional (not only for the lowercase as earlier). Moreover, catcher names of CS specifiers are now created by \stringing the CS and stripping its backslash (earlier: by crude detokenising it)., 4912

moved here from gmutils.sty (a bug fix), 4932

gmenvir v0.74

begin*:

The catcodes of \begin and \end argument(s) don't have to agree strictly anymore: an environment is properly closed if the \begin's and \end's arguments result in the same \csname, 9016

gmenvir v0.92

begin*:

added, 9041

shortened thanks to \@ifenvir, 9076

gmenvir v0.993

begin*:

made use \@envirstack, 9041

to be precise, it's not a change but rather a staus quo action: \gmu@ifedetokens suddenly turned to be expandable and un\protected so we make *this* macro \protected, 9041

gmlogos v0.96

\LuaTeX:

added, 9782

gmlogos v0.97

\pdfLaTeX:

added, 9722

gmlogos v0.993

\LaTeXpar:

added \leavevmode (a bug fix), 9666

\XeTeXpar:
added, 9745
gmmeta v0.96
\gmu@pswords:
\textup removed to allow slanting the name in titles (that are usually typeset in Italic font), 9316
gmmeta v0.98
\arg@dc:
made iterating thanks to \DeclareCommand\arg@dc, 9554
\cs:
added \verbatim@specials, 9330
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, 9330
gmmeta v0.99n
\cat:
added, 9571, 9577, 9579
gmnotonlypream 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, 9804
gmnotonlypream v0.93
\nocite:
a bug fixed: with natbib an 'extra }' error. Now it fixes only the standard version of \nocite, 9852
gmRCS v0.74
General:
Added macros to make sectioning commands of mwcls and standard classes compatible. Now my sectionings allow two optionals in both worlds and with mwcls if there's only one optional, it's the title to toc and running head not just to the latter, 13311
gmRCS 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, 13311
gmRCS 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, 13311
gmRCS v0.78
General:
\@namelet renamed to \n@melet to solve a conflict with the beamer class. The package contents regrouped, 13311
gmRCS v0.85
General:
fixed behaviour of too clever headings with gmdoc by adding an \ifdim test, 13311
gmRCS v0.87
General:
the package goes ε-TEX even more, making use of \ifdefined and the code using UTF-8 chars is wrapped in a XETEX-condition, 13311
gmRCS v0.89
General:
removed obsolete adjustment of pgf for XETEX, 13311
gmRCS v0.91
General:
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, 13311
gmRCS v0.93
General:
A couple of \DeclareRobustCommand* changed to \pdef, 13311
The numerical macros commented out as obsolete and never really used, 13311
gmRCS v0.94
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, 13311
removed \unex@namedef and \unex@nameuse, probably never really used since they were incomplete: \edef@other undefined, 13311
The code from ancient xparse (1999) of TEXLive 2007 rewritten here, 13311
gmRCS v0.96
General:
put to CTAN on 2008/11/21, 13311
gmtypos v0.76
\freeze@actives:
added, 11582