Grzegorz Murzynowski

The gmutils Package*

Written by Grzegorz Murzynowski,
natror at o2 dot pl

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmutils}
      [2007/04/24 v0.78 some rather TeXnical macros, some of them
            tricky (GM)]
```

Contents

Intro

The gmutils.sty package provides some macros that are analogous to the standard LaTeX ones but extend their functionality, such as \@ifnextcat, \addtomacro or \begin(*).

* This file has version number v0.78 dated 2007/04/24.

The others are just conveniences I like to use in all my TeX works, such as `\afterfi`, `\pk` or `\cs`.

I wouldn't say they are only for the package writers but I assume some nonzero (LA)TEX-awareness of the user.

For details just read the code part.

Installation

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 four files.

gmutils.sty
README
gmutilsDoc.tex
gmutilsDoc.pdf

Compiling of the Documentation

The last of the above files (the .pdf, i.e., *this file*) is a documentation compiled from the .sty file by running LATEX on the gmutilsDoc.tex file twice, then MakeIndex on the gmutils.idx file, and then LATEX on gmutilsDoc.tex once more.

MakeIndex shell command:

```
makeindex␣-r␣gmutilsDoc
```

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, gmutils.sty, gmiflink.sty and also some standard packages: hyperref.sty, color.sty, geometry.sty, multicol.sty, lmodern.sty, fontenc.sty that should be installed on your computer by default.

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

A couple of abbreviations

\@xa 4 `\let\@xa\expandafter`
\@nx 5 `\let\@nx\noexpand`

The `\newgif` declaration's effect is used even in the LATEX 2_ε source by redefining some particular user defined ifs (UD-ifs henceforth) step by step. The goal is to make the 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 6 `\def\newgif#1{%`

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

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

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

After `\newgif\iffoo` you may type `{\foogtrue}` and the `\iffoo` switch becomes globally equal `\iftrue`. Simili modo `\foogfalse`. Note the letter *g* added to underline globalness of the assignment.

If for any reason, no matter how queer ;-) may it be, you need *both* global and local switchers of your `\if...`, declare it both with `\newif` and `\newgif`.

Note that it's just a shorthand. `\global\if`⟨*switch*⟩`true/false` *does* work as expected.

There's a trouble with `\refstepcounter`: defining `\@currentlabel` is local. So let's `\def` a `\global` version of `\refstepcounter`.

Warning. I use it because of very special reasons in gmdoc and in general it is probably not a good idea to make `\refstepcounter` global since it is contrary to the original LaTeX approach.

\grefstepcounter
```
\newcommand*\grefstepcounter[1]{%
  {\let\protected@edef=\protected@xdef\refstepcounter{#1}}}
```

Naïve first try `\globaldefs=\tw@` raised an error `unknown␣command␣\reserved@e`. The matter was to globalize `\protected@edef` of `\@currentlabel`.

Thanks to using the true `\refstepcounter` inside, it observes the change made to `\refstepcounter` by hyperref.

Another shorthand. It may decrease a number of `\expandafter`s e.g.

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

LaTeX provides a very useful `\g@addto@macro` macro that adds its second argument to the current definition of its first argument (works iff the first argument is a no argument macro). But I needed it some times in a document, where `@` is not a letter. So:

\gaddtomacro
```
\let\gaddtomacro=\g@addto@macro
```

The redefining of the first argument of the above macro(s) is `\global`. What if we want it local? Here we are:

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

And for use in the very document,

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

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

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

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

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

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

\g@relaxen
\grelaxen
```
\newcommand*\g@relaxen{\global\@relaxen}
\@ifdefinable\grelaxen{\let\grelaxen\g@relaxen}
```

For the heavy debugs I was doing while preparing gmdoc, as a last resort I used `\showlists`. But this command alone was usually too little: usually it needed setting `\showboxdepth` and `\showboxbreadth` to some positive values. So,

\gmshowlists
```
\def\gmshowlists{\showboxdepth=1000␣\showboxbreadth=1000␣\showlists}
```

\nameshow
```
\newcommand*\nameshow[1]{\@xa\show\csname#1\endcsname}
```

Standard `\string` command returns a string of 'other' chars except for the space, for which it returns $_{\sqcup}10$. In gmdoc I needed the spaces in macros' and environments' names to be always $_{12}$, so I define

\xiistring
```
\def\xiistring#1{%
  \if\@nx#1\twelvespace
    \twelvespace
  \else
    \string#1%
  \fi}
```

\@ifnextcat, \@ifnextac

As you guess, we `\def \@ifnextcat` à la `\@ifnextchar`, see LaTeX 2_ε source dated 2003/12/01, file d, lines 253–271. The difference is in the kind of test used: while `\@ifnextchar` does `\ifx`, `\@ifnextcat` does `\ifcat` which means it looks not at the meaning of a token(s) but at their `\catcode`(s). As you (should) remember from The TeXbook, the former test doesn't expand macros while the latter does. But in `\@ifnextcat` the peeked token is protected against expanding by `\noexpand`. Note that the first parameter is not protected and therefore it shall be expanded if it's a macro. Because an assignment is involved, you can't test whether the next token is an active char.

\@ifnextcat
```
\long\def\@ifnextcat#1#2#3{%
  \def\reserved@d{#1}%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
  \futurelet\@let@token\@ifncat}
\def\@ifncat{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifncat
  \else
    \ifcat\reserved@d\@nx\@let@token
```

```
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
{\def\:{\let\@sptoken=␣}␣\:␣% this makes \@sptoken a space token.
\def\:{\@xifncat}␣\@xa\gdef\:␣{\futurelet\@let@token\@ifncat}}
```

Note the trick to get a macro with no parameter and requiring a space after it. We do it inside a group not to spoil the general meaning of `\:` (which we extend later).

But how to peek at the next token to check whether it's an active char? First, we look with `\@ifnextcat` whether there stands a group opener. We do that to avoid taking a whole `{...}` as the argument of the next macro, that doesn't use `\futurelet` but takes the next token as an argument, tests it and puts back intact.

\@ifnextac

```
\long\def\@ifnextac#1#2{%
  \@ifnextcat\bgroup{#2}{\gm@ifnac{#1}{#2}}}
\long\def\gm@ifnac#1#2#3{%
  \ifcat\@nx~\@nx#3\afterfi{#1#3}\else\afterfi{#2#3}\fi}
```

Yes, it won't work for an active char `\let` to `{`$_1$, but it *will* work for an active char `\let` to a char of catcode $\neq 1$. (Is there anybody on Earth who'd make an active char working as `\bgroup`?)

Now, define a test that checks whether the next token is a genuine space, ␣$_{10}$ that is. First define a CS let such a space. The assignment needs a little trick (The TEXbook appendix D) since `\let`'s syntax includes one optional space after `=`.

```
\let\@tempa\*%
\def\*{%
  \let\*\@tempa
  \let\gm@letspace=␣}%
\*␣%
```

\@ifnextspace

```
\def\@ifnextspace#1#2{%
  \let\@reserveda\*%
  \def\*{%
    \let\*\@reserveda
    \ifx\@let@token\gm@letspace\afterfi{#1}%
    \else\afterfi{#2}%
    \fi}%
  \futurelet\@let@token\*}
```

First use of this macro is for an active `-` that expands to `---` if followed by a space.

`\afterfi` and Pals

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

There's the Knuthian trick with `\next`. And here another, revealed to me by my TEX Guru.

I think the situations when the Knuthian (the former) trick is not available are rather seldom, but they are imaginable at least: the `\next` trick involves an assignment so it won't work e.g. in `\edef`. But in general it's only a matter of taste which one to use.

One warning: those macros peel the braces off, i.e.,

```
\if..\afterfi{\@makeother\^^M}\fi
```

causes a leakage of `^^M`$_{12}$. To avoid pollution write

```
\if..\afterfi{\bgroup\@makeother\^^M\egroup}\fi .
```

\afterfi 77 `\long\def\afterfi#1#2\fi{\fi#1}`

And two more of that family:

\afterfifi 78 `\long\def\afterfifi#1#2\fi#3\fi{\fi\fi#1}`

\afteriffifi 79 `\long\def\afteriffifi#1#2\if#3\fi#4\fi{\fi#1}`

Notice the refined elegance of those macros, that cover both 'then' and 'else' cases thanks to `#2` that is discarded.

80 `\long\def\afterififfifi#1#2\fi#3\fi#4\fi{\fi#1}`

81 `\long\def\afteriffifi#1#2\fi#3\fi#4\fi{\fi\fi#1}`

82 `\long\def\afterfifi#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}`

Almost an Environment or Redefinition of `\begin`

We'll extend the functionality of `\begin`: the non-starred instances shall act as usual and we'll add the starred version. The difference of the latter will be that it won't check whether the 'environment' has been defined so any name will be allowed.

This is intended to structure the source with named groups that don't have to be especially defined and probably don't take any particular action except the scoping.

(If the `\begin*`'s argument is a (defined) environment's name, `\begin*` will act just like `\begin`.)

Original LATEX's `\begin`:

```
\def\begin#1{%
  \@ifundefined{#1}%
    {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
    {\def\reserved@a{\def\@currenvir{#1}%
        \edef\@currenvline{\on@line}%
        \csname #1\endcsname}}%
    \@ignorefalse
    \begingroup\@endpefalse\reserved@a}
```

\@begnamedgroup 83 `\@ifdefinable\@begnamedgroup{\relax}`

84 `\def\@begnamedgroup#1{%`

85 `\@ignorefalse%` not to ignore blanks after group

86 `\begingroup\@endpefalse`

87 `\def\@currenvir{#1}%`

88 `\edef\@currenvline{\on@line}%`

89 `\csname␣#1\endcsname}%` if the argument is a command's name (an environment's e.g.), this command will now be executed. (If the corresponding control sequence hasn't been known to TEX, this line will act as `\relax`.)

For back compatibility with my earlier works

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

And for the ending

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

And we make it the starred version of `\begin`.

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

Improvement of `\end`

It's very clever and useful that `\end` checks whether its argument is `ifx`-equivalent `@currenvir`. However, it works not quite as I would expect: Since the idea of environment is to open a group and launch the cs named in the `\begin`'s argument. That last thing is done with `\csname...\endcsname` so the char catcodes are equivalent. Thus should be also in the `\end`'s test and therefore we ensure the compared texts are both expanded and made all 'other'.

```
\def\@checkend#1{%
  \edef\reserved@a{\@xa\string\csname#1\endcsname}%
  \edef\exii@currenvir{\@xa\string\csname\@currenvir\endcsname}%
  \ifx\reserved@a\exii@currenvir\else\@badend{#1}\fi}
```

Thanks to it you may write `\begin{macrocode*}` with $*_{12}$ and end it with `\end{macrocode*}` with $*_{11}$ (that was the problem that led me to this solution). The error messages looked really funny:

```
!␣LaTeX␣Error:␣\begin{macrocode*}␣on␣input␣line␣1844␣ended␣by␣\end{macrocode*}.
```

Of course, you might write also `\end{macrocode\star}` where `\star` is defined as 'other' star or letter star.

From relsize

As file relsize.sty, v3.1 dated July 4, 2003 states, LaTeX 2_ε version of these macros was written by Donald Arseneau asnd@triumf.ca and Matt Swift swift@bu.edu after the LaTeX 2.09 smaller.sty style file written by Bernie Cosell cosell@WILMA.BBN.COM.

I take only the basic, non-math mode commands with the assumption that there are the predefined font sizes.

\relsize
\smaller
\larger
\smallerr
\largerr

You declare the font size with `\relsize{`$\langle n\rangle$`}` where $\langle n\rangle$ gives the number of steps ("mag-step" = factor of 1.2) to change the size by. E.g., $n = 3$ changes from `\normalsize` to `\LARGE` size. Negative n selects smaller fonts. `\smaller` == `\relsize{-1}`; `\larger` == `\relsize{1}`. `\smallerr`(my addition) == `\relsize{-2}`; `\largerr` guess yourself.

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

\relsize
```
\DeclareRobustCommand*\relsize[1]{%
  \ifmmode␣\@nomath\relsize\else
    \begingroup
     \@tempcnta␣% assign number representing current font size
       \ifx\@currsize\normalsize␣4\else␣␣␣% funny order is to have most ...
```

```
          \ifx\@currsize\small 3\else␣␣␣␣␣␣␣% ...likely sizes checked first
           \ifx\@currsize\footnotesize 2\else
            \ifx\@currsize\large 5\else
             \ifx\@currsize\Large 6\else
              \ifx\@currsize\LARGE 7\else
               \ifx\@currsize\scriptsize 1\else
                \ifx\@currsize\tiny 0\else
                 \ifx\@currsize\huge 8\else
                  \ifx\@currsize\Huge 9\else
                  4\rs@unknown@warning␣% unknown state: \normalsize as starting point
      \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
```

Change the number by the given increment:

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

watch out for size underflow:

```
       \ifnum\@tempcnta<\z@␣\rs@size@warning{small}{\string\tiny}%
            \@tempcnta\z@␣\fi
       \@xa\endgroup
       \ifcase\@tempcnta␣␣% set new size based on altered number
          \tiny␣\or␣\scriptsize␣\or␣\footnotesize␣\or␣\small␣\or␣%
               \normalsize␣\or
          \large␣\or␣\Large␣\or␣\LARGE␣\or␣\huge␣\or␣\Huge␣\else
           \rs@size@warning{large}{\string\Huge}\Huge
\fi\fi}% end of \relsize.
```

\rs@size@warning
```
\providecommand*\rs@size@warning[2]{\PackageWarning{gmutils␣(relsize)}{%
 Size␣requested␣is␣too␣#1.\MessageBreak␣Using␣#2␣instead}}
```

\rs@unknown@warning
```
\providecommand*\rs@unknown@warning{\PackageWarning{gmutils␣(relsize)}{Current␣font␣size
 is␣unknown!␣(Why?!?)\MessageBreak␣Assuming␣\string\normalsize}}
```

And a handful of shorthands:

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

`\firstofone` and the Queer `\catcodes`

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

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

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

```
\gobble 133 \long\def\gobble#1{}
\gobbletwo 134 \let\gobbletwo\@gobbletwo

135 \bgroup\catcode`\_=8␣%
136 \firstofone{\egroup
\subs 137   \let\subs=_}

138 \bgroup\@makeother\_%
139 \firstofone{\egroup
\twelveunder 140   \def\twelveunder{_}}
```

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

```
141 \bgroup\catcode`\_=\active
142 \firstofone{\egroup
\smartunder 143   \newcommand*\smartunder{%
144     \catcode`\_=\active
145     \def_{\ifmmode\subs\else\_\fi}}}% We define it as \_ not just as \twelveunder
                because some font encodings don't have _ at the \char`\_ position.

146 \begingroup\catcode`\!=0
147 \@makeother\\
148 !firstofone{!endgroup%
\twelvebackslash 149   !newcommand*!twelvebackslash{\}}

\bslash 150 \@ifundefined{bslash}{\let\bslash=\twelvebackslash}{}

151 \begingroup \@makeother\%
152 \firstofone{\endgroup
\twelvepercent 153   \def\twelvepercent{%}}

154 \begingroup␣\@makeother\&%
155 \firstofone{\endgroup%
\twelveand 156   \def\twelveand{&}}

157 \begingroup\@makeother\␣%
158 \firstofone{\endgroup%
\twelvespace 159 \def\twelvespace{␣}}
```

Metasymbols

I fancy also another Knuthian trick for typesetting ⟨*metasymbols*⟩ in The TEXbook. So I repeat it here. The inner `\meta` macro is copied verbatim from `doc`'s v2.1b documentation dated 2004/02/09 because it's so beautifully crafted I couldn't resist. I only don't make it `\long`.

"The new implementation fixes this problem by defining `\meta` in a radically different way: we prevent hypenation by defining a `\language` which has no patterns associated with it and use this to typeset the words within the angle brackets."

```
160 \ifx\l@nohyphenation\undefined
161   \newlanguage\l@nohyphenation
162 \fi
\meta 163 \DeclareRobustCommand*\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."

```
  \ensuremath\langle
  \ifmmode␣\@xa␣\nfss@text␣\fi
  {%
    \meta@font@select
```

Need to keep track of what we changed just in case the user changes font inside the argument so we store the font explicitly.

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

But I define \meta@font@select as the brutal and explicit \it instead of the original \itshape to make it usable e.g. in the gmdoc's \cs macro's argument.

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

The below \meta's drag[1] is a version of The TEXbook's one.

\<...>
```
\def\<#1>{\meta{#1}}
```

Macros for Printing Macros and Filenames

First let's define three auxiliary macros analogous to \dywiz from polski.sty: a shorthands for \discretionary that'll stick to the word not spoiling its hyphenability and that'll won't allow a linebreak just before nor just after themselves. The \discretionary TEX primitive has three arguments: #1 'before break', #2 'after break', #3 'without break', remember?

\discre
```
\def\discre#1#2#3{\kern0sp\discretionary{#1}{#2}{#3}\penalty10000^
      \hskip0sp\relax}
```
\discret
```
\def\discret#1{\kern0sp\discretionary{#1}{#1}{#1}\penalty10000^
      \hskip0sp\relax}
```

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

```
\def\:{\ifmmode\afterfi{\mskip\medmuskip}\else\afterfi{\discret{}}\fi}
```
\vs
```
\newcommand*{\vs}{\discre{\textvisiblespace}{}{\textvisiblespace}}
```

Then we define a macro that makes the spaces visible even if used in an argument (i.e., in a situation where re\catcodeing has no effect).

\printspaces
```
\def\printspaces#1{{\let~=\vs␣\let\␣=\vs␣\gm@pswords#1␣\@@nil}}
\def\gm@pswords#1␣#2\@@nil{%
  \if\relax#1\relax\else#1\fi
```

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

```
185   \if\relax#2\relax\else\vs\penalty\hyphenpenalty\gm@pswords#2\@@nil\fi}%
```

note that in the recursive call of `\gm@pswords` the argument string is not extended with a guardian space: it has been already by `\printspaces`.

```
\sfname 186 \DeclareRobustCommand*\sfname[1]{\textsf{\printspaces{#1}}}
  \file 187 \let\file\sfname% it allows the spaces in the filenames (and prints them as ␣).
```

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

```
   \pk 188 \DeclareRobustCommand*{\pk}[1]{\textsf{\textup{#1}}}
```

Some (if not all) of the below macros are copied from doc and/or ltxdoc.

A macro for printing control sequences in arguments of a macro. Robust to avoid writing an explicit `\` into a file. It calls `\ttfamily` not `\tt` to be usable in headings which are boldface sometimes.

```
   \cs 189 \DeclareRobustCommand*{\cs}[2][\bslash]{{%
190       \def\-{\discretionary{{\rmfamily-}}{}{}}%
191       \def\{{\char`\{}\def\}{\char`\}}\ttfamily␣#1#2}}
  \env 192 \DeclareRobustCommand*{\env}[1]{\cs[]{#1}}
```

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

```
\possfil 193 \newcommand*\possfil{\hfil\penalty1000\hfilneg}
```

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

"`\cmd{\foo}` Prints `\foo` verbatim. It may be used inside moving arguments. `\cs{^ foo}` also prints `\foo`, for those who prefer that syntax. (This second form may even be used when `\foo` is `\outer`)."

```
   \cmd 194 \def\cmd#1{\cs{\@xa\cmd@to@cs\string#1}}
195 \def\cmd@to@cs#1#2{\char\number`#2\relax}
```

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

```
  \marg 196 \def\marg#1{{\ttfamily\char`\{}\meta{#1}{\ttfamily\char`\}}}
```

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

```
  \oarg 197 \def\oarg{\@ifnextchar[\@oargsq\@oarg}
198 \def\@oarg#1{{\ttfamily[}\meta{#1}{\ttfamily]}}
199 \def\@oargsq[#1]{\@oarg{#1}}
```

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

```
  \parg 200 \def\parg{\@ifnextchar(\@pargp\@parg}
201 \def\@parg#1{{\ttfamily(}\meta{#1}{\ttfamily)}}
202 \def\@pargp(#1){\@parg{#1}}
```

But we can have all three in one command.

```
203 \AtBeginDocument{%
  \arg 204   \let\math@arg\arg
205   \def\arg{\ifmmode\math@arg\else\afterfi{%
206       \@ifnextchar[%
207       \@oargsq{\@ifnextchar(%
208         \@pargp\marg}}\fi}%
209 }
```

Storing and Restoring the Meanings of CSs

A command to store the current meaning of a CS in another macro to temporarily redefine the CS and be able to set its original meanig back (when grouping is not recommended):

\StoreMacro
```
\def\StoreMacro{%
  \bgroup\makeatletter\@ifstar\egStore@MacroSt\egStore@Macro}
\long\def\egStore@Macro#1{\egroup\Store@Macro{#1}}
\long\def\egStore@MacroSt#1{\egroup\Store@MacroSt{#1}}
\long\def\Store@Macro#1{%
  \@xa\let\csname␣/gml/store\string#1\endcsname#1}
\long\def\Store@MacroSt#1{%
  \edef\gmu@smtempa{%
    \@nx\let\@xa\@nx\csname/gml/store#1\endcsname\@xa\@nx\csname#1^
          \endcsname}
  \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 CSs and stores each of them. The CS may be separated with commas but they don't have to.

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

And the inner iterating macro:

```
\long\def\gml@storemacros#1{%
  \def\@tempa{\@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\@tempa.% a dot finishes storing.
  \else
    \if\@tempa,% The list this macro is put before may contain commas and that's
          O.K., we just continue the work.
      \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 storing
            macro as above and to the restoring one as below.
      \afterfifi\gml@storemacros
    \fi
  \fi}
```

And for the restoring

\RestoreMacro
```
\def\RestoreMacro{%
  \bgroup\makeatletter\@ifstar\egRestore@MacroSt\egRestore@Macro}
\long\def\egRestore@Macro#1{\egroup\Restore@Macro{#1}}
\long\def\egRestore@MacroSt#1{\egroup\Restore@MacroSt{#1}}
\long\def\Restore@Macro#1{%
```

```
240     \@xa\let\@xa#1\csname␣/gml/store\string#1\endcsname}
241 \long\def\Restore@MacroSt#1{%
242   \edef\gmu@smtempa{%
243     \@nx\let\@xa\@nx\csname#1\endcsname\@xa\@nx\csname/gml/store#1^
            \endcsname}
244   \gmu@smtempa}
```

\RestoreMacros

```
245 \long\def\RestoreMacros{\bgroup\makeatletter\Restore@Macros}
246 \long\def\Restore@Macros#1{\egroup
247   \let\gml@StoreCS\Restore@Macro% we direct the core CS towards restoring and
          call the same iterating macro as in line 223.
248   \gml@storemacros#1.}
```

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

And to restore *and* use immediately:

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

It happended (see the definition of `\@docinclude` in gmdoc.sty) that I needed to `\relax` a bunch of macros and restore them after some time. Because the macros were rather numerous and I wanted the code more readable, I wanted to `\do` them. After a proper defining of `\do` of course. So here is this proper definition of `\do`, provided as a macro (a declaration).

\StoringAndRelaxingDo

```
251 \long\def\StoringAndRelaxingDo{%
252   \def\do##1{\@xa\let\csname␣/gml/store\string##1\endcsname##1%
253     \let##1\relax}}
```

And here is the counter-definition for restore.

\RestoringDo

```
254 \long\def\RestoringDo{%
255   \def\do##1{%
256     \@xa\let\@xa##1\csname␣/gml/store\string##1\endcsname}}
```

And to store a cs as explicitly named cs, i.e. to `\let` one csname another (`\n@melet` not `\@namelet` becasuse the latter is defined in Till Tantau's beamer class another way):

```
257 \def\n@melet#1#2{%
258   \edef\@tempa{%
259     \let\@xa\@nx\csname#1\endcsname
260     \@xa\@nx\csname#2\endcsname}%
261   \@tempa}
```

Not only preamble!

Let's remove some commands from the list to erase at begin document! Primarily that list was intended to save memory not to forbid anything. Nowadays, when memory is cheap, the list of only-preamble commands should be rethought IMO.

```
262 \newcommand\not@onlypreamble[1]{%
263   \def\do##1{\ifx#1##1\else\@nx\do\@nx##1\fi}%
264   \edef\@preamblecmds{\@preamblecmds}}
```

```
\not@onlypreamble\@preamblecmds
\not@onlypreamble\@ifpackageloaded
\not@onlypreamble\@ifclassloaded
\not@onlypreamble\@ifl@aded
\not@onlypreamble\@pkgextension
```

And let's make the message of only preamble command's forbidden use informative a bit:

```
\def\gm@notprerr{␣can␣be␣used␣only␣in␣preamble␣(\on@line)}
\AtBeginDocument{%
  \def\do#1{\@nx\do\@nx#1}%
  \edef\@preamblecmds{%
    \def\@nx\do##1{%
      \def##1{!␣\@nx\string##1␣\@nx\gm@notprerr}}%
    \@preamblecmds}}
```

Third Person Pronouns

Is a reader of my documentations 'she' or 'he' and does it make a difference?

Not to favour any gender in the personal pronouns, define commands that'll print alternately masculine and feminine pronoun of third person. By 'any' I mean not only typically masculine and typically feminine but the entire amazingly rich variety of people's genders, *including* those who do not describe themselves as 'man' or 'woman'.

One may say two pronouns is far too little to cover this variety but I could point Ursula's K. LeGuin's *The Left Hand Of Darkness* as another acceptable answer. In that moody and moderate SF novel the androgynous persons are usually referred to as 'mister', 'sir' or 'he': the meaning of reference is extended. Such an extension also my automatic pronouns do suggest. It's *not* political correctness, it's just respect to people's diversity.

```
\newcounter{gm@PronounGender}
```

\gm@atppron
```
\newcommand*\gm@atppron[2]{%
  \stepcounter{gm@PronounGender}% remember \stepcounter is global.
  \ifodd\arabic{gm@PronounGender}#1\else#2\fi}
```

\heshe
```
\newcommand*\heshe{\gm@atppron{he}{she}}
```
\hisher
```
\newcommand*\hisher{\gm@atppron{his}{her}}
```
\himher
```
\newcommand*\himher{\gm@atppron{him}{her}}
```
\hishers
```
\newcommand*\hishers{\gm@atppron{his}{hers}}
```

\HeShe
```
\newcommand*\HeShe{\gm@atppron{He}{She}}
```
\HisHer
```
\newcommand*\HisHer{\gm@atppron{His}{Her}}
```
\HimHer
```
\newcommand*\HimHer{\gm@atppron{Him}{Her}}
```
\HisHers
```
\newcommand*\HisHers{\gm@atppron{His}{Hers}}
```

To Save Precious Count Registers

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

```
\newcommand*\nummacro[1]{\gdef#1{0}}
```

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

Need an explanation? The `\nummacro` declaration defines its argument (that should be a CS) as `{0}` which is analogous to `\newcount` declaration but doesn't use up any count register.

Then you may use this numeric macro as something between TEX's count CS and LATEX's counter. The macros `\stepnummacro` and `\addtonummacro` are analogous to LATEX's `\stepcounter` and `\addtocounter` respectively: `\stepnummacro` advances the number stored in its argument by 1 and `\addtonummacro` advances it by the second argument. As the LATEX's analogoi, they have the global effect (the effect of global warming ;-)).

So far I've used only `\nummacro` and `\stepnummacro`. Notify me if you use them and whether you need sth. more, `\multiplynummacro` e.g.

Improvements to mwcls Sectioning Commands

That is, 'Expe-ri-mente'[2] mit MW sectioning & `\refstepcounter` to improve mwcls's cooperation with hyperref. They shouldn't make any harm if another class (non-mwcls) is loaded.

We `\refstep` sectioning counters even if the sectionings are not numbered, because otherwise

1. pdfTEX cried of multiply defined `\label`s,
2. e.g. in a table of contents the hyperlink `<rozdzia\l\␣Kwiaty␣polskie>` linked not to the chapter's heading but to the last-before-it change of `\ref`.

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

Auxiliary macros for the kernel sectioning macro:

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

[2] A. Berg, *Wozzeck*.

To avoid `\def`ing of `\mw@sectionxx` if it's undefined, we redefine `\def` to gobble the definition and restore the original meaning of itself.

Why shouldn't we change the ontological status of `\mw@sectionxx` (not define if undefined)? Because some macros (in gmdocc e.g.) check it to learn whether they are in an mwcls or not.

But let's make a shorthand for this test since we'll use it three times in this package and maybe also somewhere else.

`\@ifnotmw`

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

I know it may be of bad taste (to write such a way *here*) but I feel so lonely and am in an alien state of mind after 3 hour sleep last night and, worst of all, listening to sir Edward Elgar's flamboyant Symphonies d'Art Nouveau.

A *decent* person would just wrap the following definition in `\@ifundefined`'s Else. But look, the definition is so long and I feel so lonely etc. So, I define `\def` (for some people there's nothing sacred) to be a macro with two parameters, first of which is delimited by digit 4 (the last token of `\mw@sectionxx`'s parameter string) and the latter is undelimited which means it'll be the body of the definition. Such defined `\def` does nothing else but restores its primitive meaning by the way sending its arguments to the Gobbled Tokens' Paradise. Luckily, `\RestoreMacro` contains `\let` not `\def`.

The kernel of MW's sectioning commands:

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

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

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

```
      \global\@topnum\z@
  \fi% of \if\mw@HeadingBreakBefore
```

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

```
  \ifHeadingNumbered
    \refstepcounter{#1}%
    \protected@edef\HeadingNumber{\csname␣the#1\endcsname\relax}%
  \else
    \let\HeadingNumber\@empty
    \gm@hyperrefstepcounter
  \fi% of \ifHeadingNumbered
  \if\mw@HeadingRunIn
    \mw@runinheading
  \else
    \if\mw@HeadingWholeWidth
      \if@twocolumn
        \if\mw@HeadingBreakAfter
        \onecolumn
        \mw@normalheading
        \pagebreak\relax
              \if@twoside
                \null
                \thispagestyle{blank}%
                \newpage
              \fi% of \if@twoside
        \twocolumn
        \else
          \@topnewpage[\mw@normalheading]%
        \fi% of \if\mw@HeadingBreakAfter
      \else
        \mw@normalheading
        \if\mw@HeadingBreakAfter\pagebreak\relax\fi
      \fi% of \if@twocolumn
    \else
      \mw@normalheading
      \if\mw@HeadingBreakAfter\pagebreak\relax\fi
    \fi% of \if\mw@HeadingWholeWidth
  \fi% of \if\mw@HeadingRunIn
  }
```

(End of Experimente with MW sectioning.)

Compatibilising Standard and mwcls Sectionings

If you use Marcin Woliński's document classes (mwcls), you might have met their little queerness: the sectioning commands take two optional arguments instead of standard one. It's reasonable since one may wish one text to be put into the running head, another to the toc and yet else to the page. But the order of optionalities causes an incompatibility with the standard classes: MW section's first optional argument goes to the running head not to toc and if you've got a source file written with the standard classes in mind

and use the first (and only) optional argument, the effect with mwcls would be different if not error.

Therefore I counter-assign the commands and arguments to reverse the order of optional arguments for sectioning commands when mwcls are in use and reverse, to make mwcls-like sectioning optionals usable in the standard classes.

With the following in force, you may both in the standard classes and in mwcls give a sectioning command one or two optional arguments (and mandatory the last, of course). If you give just one optional, it goes to the running head and to toc as in scls (which is unlike in mwcls). If you give two optionals, the first goes to the running head and the other to toc (like in mwcls and unlike in scls).

(In both cases the mandatory last argument goes only to the page.)

What more is unlike in scls, it's that even with them the starred versions of sectioning commands allow optionals (but they still send them to the Gobbled Tokens' Paradise).

(In mwcls, the only difference between starred and non-starred sec commands is (not) numbering the titles, both versions make a contents line and a mark and that's not changed with my redefinitions.)

```
\@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those
      written with two optionals.
  \def\gm@secini{gm@la}%
\gm@secxx  \def\gm@secxx#1#2[#3]#4{%
    \ifx\gm@secstar\@empty
      \n@melet{gm@true@#1mark}{#1mark}% 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 argu-
            ment.
        \csname␣gm@true@#1mark\endcsname{#2}%
        \n@melet{#1mark}{gm@true@#1mark}% after we've done what we wanted
              we restore original \#1mark.
      }%
      \def\gm@secstar{[#3]}% if \gm@secstar is empty, which means the section-
            ing command was written starless, we pass the 'true' sectioning com-
            mand #3 as the optional argument. Otherwise the sectioning command
            was written with star so the 'true' s.c. takes no optional.
    \fi
    \@xa\@xa\csname\gm@secini#1\endcsname
    \gm@secstar{#4}}%
}{% we are in mwcls and want to reverse MW's optionals order i.e., if there's just one
      optional, it should go both to toc and to running head.
  \def\gm@secini{gm@mw}%
  \let\gm@secmarkh\@gobble% in mwcls there's no need to make tricks for special
        version to running headings.
\gm@secxx  \def\gm@secxx#1#2[#3]#4{%
    \@xa\@xa\csname\gm@secini#1\endcsname
    \gm@secstar[#2][#3]{#4}}%
}
\def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
\def\gm@secx#1[#2]{%
  \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}% if there's only
        one optional, we double it not the mandatory argument.
```

```
\def\gm@straightensec#1{% the parameter is for the command's name.
  \@ifundefined{#1}{}{% we don't change the ontological status of the command
        because someone may test it.
    \n@melet{\gm@secini#1}{#1}%
    \@namedef{#1}{%
      \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
        \def\gm@secstar{}\gm@sec{#1}}}}%
}%
\let\do\gm@straightensec
\do{part}\do{chapter}\do{section}\do{subsection}\do{subsubsection}
\@ifnotmw{}{\do{paragraph}}% this 'straightening' of \paragraph with the stan-
      dard article caused the 'TEX capacity exceeded' error. Anyway, who on Earth
      wants paragraph titles in toc or running head?
```

`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␣%
      \labelwidth\@parindent
      \advance\labelwidth-\labelsep
      \listparindent\@parindent
      \usecounter␣\@enumctr
      \def\makelabel##1{##1\hfil}}%
  \fi}
\@namedef{endenumerate*}{\endlist}
```

itemize*
```
\@namedef{itemize*}{%
  \ifnum\@itemdepth>\thr@@
    \@toodeep
  \else
    \advance\@itemdepth\@ne
    \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
    \@xa\list\csname\@itemitem\endcsname{%
      \partopsep\topsep␣\topsep\z@␣\leftmargin\z@
      \itemindent\@parindent
      \labelwidth\@parindent
      \advance\labelwidth-\labelsep
```

```
      \listparindent\@parindent
      \def\makelabel##1{##1\hfil␣}}%
  \fi}
\@namedef{enditemize*}{\endlist}
```

The Logos

We'll modify The LaTeX logo now to make it fit better to various fonts.

```
\let\oldLaTeX\LaTeX
\let\oldLaTeXe\LaTeXe
\def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
\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.

```
  \ifx\relax#1\def\@tempa{\@xa\@gobble\string#2}%
  \else
    \def\@tempa{#1}%
  \fi
  \edef\@tempa{%
    \@nx\DeclareTextCommand\@nx#2{PD1}{\@tempa}}
  \@tempa
  \DeclareTextCommandDefault#2{#3}}
\DeclareLogo\LaTeX{%
  {%
    L%
    \setbox\z@\hbox{\check@mathfonts
      \fontsize\sf@size\z@
      \math@fontsfalse\selectfont
      A}%
    \kern-.57\wd\z@
    \sbox\tw@␣T%
    \vbox␣to\ht\tw@{\copy\z@␣\vss}%
    \kern-.2\wd\z@}% originally −, 15 em for T.
  {%
    \ifdim\fontdimen1\font=\z@
    \else
      \count\z@=\fontdimen5\font
      \multiply\count\z@␣by␣64\relax
      \divide\count\z@␣by\p@
      \count\tw@=\fontdimen1\font
      \multiply\count\tw@␣by\count\z@
      \divide\count\tw@␣by␣64\relax
      \divide\count\tw@␣by\tw@
      \kern-\the\count\tw@␣sp\relax
    \fi}%
  \TeX}
```

```
\DeclareLogo\LaTeXe{\mbox{\m@th␣\if
    b\expandafter\@car\f@series\@nil\boldmath\fi
    \LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}
```

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

\LaTeXpar
```
\DeclareLogo[(La)TeX]{\LaTeXpar}{%
  {%
    \setbox\z@\hbox{(}% )
    \copy\z@
    \kern-.2\wd\z@␣L%
    \setbox\z@\hbox{\check@mathfonts
      \fontsize\sf@size\z@
      \math@fontsfalse\selectfont
      A}%
    \kern-.57\wd\z@
    \sbox\tw@␣T%
    \vbox␣to\ht\tw@{\box\z@%
      \vss}%
  }%
  \kern-.07em% originally −, 15 em for T.
  {% (
    \sbox\z@)%
    \kern-.2\wd\z@\copy\z@
    \kern-.2\wd\z@}\TeX
}
```

Expanding turning stuff all into 'other'

While typesetting a unicode file contents with inputenc package I got a trouble with some Unicode sequences that expanded to unexpandable CSs: they could'nt be used within `\csname...\endcsname`. My TEXGuru advised to use `\meanig` to make all the name 'other'. So—here we are.

Don't use them in `\edef`s, they would expand not quite.

The next macro turns its `#2` all into 'other' chars and assigns them to `#1` which has to be a CS or an active char.

\def@other
```
\long\def\def@other#1#2{%
  \long\def\gm@def@other@tempa{#2}%
  \all@other#1{#2}}
```

The next macro is intended to be put in `\edef`s with a macro argument. The meaning of the macro will be made all 'other' and the words '(long) macro:-¿' gobbled.

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

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

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

In the next two macros' names, 'unex' stands both for not expanding the argument(s) and for disastrously partial unexpandability of the macros themselves.

```
\unex@namedef 499 \long\def\unex@namedef#1#2{%
              500   \edef@other\gmu@tempa{#1}%
              501   \@xa\long\@xa\def\csname\gmu@tempa\endcsname{#2}}
\unex@nameuse 502 \long\def\unex@nameuse#1{%
              503   \edef@other\gmu@tempa{#1}%
              504   \csname\gmu@tempa\endcsname}
```

Varia

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

```
\*  505 \DeclareRobustCommand*\*{\leavevmode\lower.8ex\hbox{$\,\widetilde{\␣}^
          \,$}}
```

The standard \obeyspaces declaration just changes the space's \catcode to $_{13}$ ('active'). Usually it is fairly enough because no one 'normal' redefines the active space. But we are *not* normal and we do *not* do usual things and therefore we want a declaration that not only will \activeate the space but also will (re)define it as the \␣ primitive. So define \gmobeyspaces that obeys this requirement.

(This definition is repeated in gmverb.)

```
              506 \begin{catcode}`\␣\active
\gmobeyspaces 507 \gdef\gmobeyspaces{\catcode`\␣\active\let␣\␣}
              508 \end{catcode}
```

While typesetting poetry, I was surprised that sth. didn't work. The reason was that original \obeylines does \let not \def, so I give the latter possibility.

```
              509 \bgroup\catcode`\^^M\active% the comment signs here are crucial.
              510 \firstofone{\egroup%
\defobeylines 511 \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   512 \def\deksmallskip{\vskip\smallskipamount}
\undeksmallskip 513 \def\undeksmallskip{\vskip-\smallskipamount}
  \dekmedskip   514 \def\dekmedskip{\vskip\medskipamount}
  \dekbigskip   515 \def\dekbigskip{\vskip\bigskipamount}
```

In some \if(cat?) test I needed to look only at the first token of a tokens' string (first letter of a word usually) and to drop the rest of it. So I define a macro that expands to the first token (or {⟨*text*⟩}) of its argument.

```
\@firstofmany 516 \long\def\@firstofmany#1#2\@@nil{#1}
```

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

```
      \TODO 517 \newcommand*{\TODO}[1][]{{%
              518     \sffamily\bfseries\huge␣TODO!\if\relax#1\relax\else\space\fi#1}}
```

I like twocolumn tables of contents. First I tried to provide them by writing \begin{^ multicols}{2} and \end{multicols} outto the .toc file but it worked wrong in some cases. So I redefine the internal LaTeX macro instead.

```
  \twocoltoc 519 \newcommand*\twocoltoc{%
             520   \RequirePackage{multicol}%
  \@starttoc 521   \def\@starttoc##1{%
```

```
    \begin{multicols}{2}\makeatletter\@input␣{\jobname␣.##1}%
      \if@filesw␣\@xa␣\newwrite␣\csname␣tf@##1\endcsname
        \immediate␣\openout␣\csname␣tf@##1\endcsname␣\jobname␣.##1^
              \relax
      \fi
      \@nobreakfalse\end{multicols}}}
\@onlypreamble\twocoltoc
```

The macro given below is taken from the multicol package (where its name is `\enough@room`). I put it in this package since I needed it in two totally different works.

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

Two shorthands for debugging:

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

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

An equality sign properly spaced:

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

And for the LaTeX's pseudo-code statements:

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

The job name without extension.

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

\jobnamewoe
```
\def\jobnamewoe{\@xa\gm@jobn\jobname.\@@nil}%
```
We add the dot to be sure there is one although I'm not sure whether you can TeX a file that has no extrension.

While typesetting a UTF-8 ls-R result I found a difficulty that follows: UTF-8 encoding is handled by the inputenc package. It's O.K. so far. The UTF-8 sequences are managed using active chars. That's O.K. so far. While writing such sequences to a file, the active chars expand. You feel the blues? When the result of expansion is read again, it sometimes is again an active char, but now it doesn't star a correct UTF-8 sequence.

Because of that I wanted to 'freeze' the active chars so that they would be `\write`n to a file unexpanded. A very brutal operation is done: we look at all 256 chars' catcodes and if we find an active one, we `\let` it `\relax`. As the macro does lots and lots of assignments, it shouldn't be used in `\edef`s.

\freeze@actives
```
\def\freeze@actives{%
  \count\z@\z@
  \@whilenum\count\z@<\@cclvi\do{%
    \ifnum\catcode\count\z@=\active
      \uccode‘\~=\count\z@
      \uppercase{\let~\relax}%
    \fi
    \advance\count\z@\@ne}}
```

A macro that typesets all 256 chars of given font. It makes use of `\@whilenum`.

\ShowFont
```
\newcommand*\ShowFont[1][6]{%
```

```
  \begin{multicols}{#1}[The␣current␣font␣(the␣\f@encoding\␣encoding):]
    \parindent\z@
    \count\z@\m@ne
    \@whilenum\count\z@<\@cclv\do{
      \advance\count\z@\@ne
      \␣\the\count\z@:~\char\count\z@\par}
  \end{multicols}}
```

A couple of macros for typesetting liturgic texts such as psalmody of Liturgia Horarum. I wrap them into a declaration since they'll be needed not every time.

\liturgiques
```
\newcommand*\liturgiques[1][red]{% Requires the color package.
  \gmu@RP{color}%
  \newcommand*\czerwo{\small\color{#1}}% environment
  \newcommand{\czer}[1]{\leavevmode{\czerwo##1}}% we leave vmode because
        if we don't, then verse's \everypar would be executed in a group and thus
        its effect lost.
  \def\*{\czer{$*$}}
  \def\+{\czer{$\dag$}}
  \newcommand*\nieczer[1]{\textcolor{black}{##1}}}
```

A command that issues a message urging to load a package if it has not been loaded.

```
\let\gmu@RP\RequirePackage
```

```
\AtBeginDocument{%
```

\gmu@RP
```
  \renewcommand*\gmu@RP[2][]{%
    \@ifpackageloaded{#2}{}{%
      \typeout{^^J!␣Package␣`#2'␣not␣loaded!!!␣(\on@line)^^J}}}}
```

It's very strange to me but it seems that `\c` is not defined in the basic math packages. It is missing at least in the *Symbols* book.

\continuum
```
\providecommand*\continuum{\gmu@RP{eufrak}\mathfrak{c}}
```

And this macro I saw in the ltugproc document class nad I liked it.

```
\providecommand*\acro[1]{{\scshape\lowercase{#1}}}
```

```
\newcommand*\IMO{\acro{IMO}}
\newcommand*\AKA{\acro{AKA}}
```

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

```
\endinput
```

Change History

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, 24

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, 7

v0.75

\@ifnextac:

added, 5

\@ifnextcat:

\let for #1 changed to \def to allow things like \noexpand~ , 4

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, 24

\freeze@actives:

added, 23

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, 24

v0.78

General:

\@namelet renamed to \n@melet to solve a conflict with the beamer class. The package contents regrouped, 24

CheckSum 1685, 1

Index

Numbers written in italic refer to the code lines where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used. The numbers preceded with 'p.' are page numbers. All the numbers are hyperlinks.