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}
      [2008/08/06 v0.91 some rather TeXnical macros, some of them
           tricky (GM)]
```

Contents

* This file has version number v0.91 dated 2008/08/06.

Intro

The gmutils.sty package provides some macros that are analogous to the standard LaTeX ones but extend their functionality, such as `\@ifnextcat`, `\addtomacro` or `\begin(*)`. The others are just conveniences I like to use in all my TeX works, such as `\afterfi`, `\pk` or `\cs`.

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

For details just read the code part.

Installation

Unpack the gmutils-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 four files and a TDS-compliant archive.

gmutils.sty
README
gmutilsDoc.tex
gmutilsDoc.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 .sty file by running LaTeX on the gmutilsDoc.tex file twice (`xelatex gmutils.sty` in the directory you wish the documentation to be in, you don't have copy the .sty file there, TeX will find it), then MakeIndex on the 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.

```
4 \ifx\XeTeXversion\relax
5 \let\XeTeXversion\@undefined%
```

If someone earlier used the `\@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 1129

```
\fi
\ifdefined\XeTeXversion
\XeTeXinputencoding␣utf-8␣% we use Unicode dashes later in this file.
\fi% and if we are not in XeTeX, we skip them thanks to XeTeX-test.
```

A couple of abbreviations

\@xa
\@nx

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

The \newgif declaration's effect is used even in the LaTeX 2ε 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

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

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

\@gif

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

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

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

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

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

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

\grefstepcounter

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

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

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

Another shorthand. It may decrease a number of \expandafters 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
```

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

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

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

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

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

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

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

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 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\xiispace
    \xiispace
  \else
    \string#1%
  \fi}
```

\@ifnextcat, \@ifnextac

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

\@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).

The next command provides the real \if test for the next token. *It* should be called \@ifnextchar but that name is assigned for the future \ifx text, as we know. Therefore we call it \@ifnextif.

\@ifnextif
```
\long\def\@ifnextif#1#2#3{%
  \def\reserved@d{#1}%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
  \futurelet\@let@token\@ifnif}
```

\@ifnif
```
\def\@ifnif{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifnif
  \else
    \if\reserved@d\@nx\@let@token
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
```

```
{\def\:{\let\@sptoken=␣}␣\:␣%␣this␣makes␣|\@sptoken|␣a␣space␣
     token.
\def\:{\@xifnif}␣\@xa\gdef\:␣{\futurelet\@let@token\@ifnif}}
```

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

`\@ifnextac`
```
\long\def\@ifnextac#1#2{%
  \@ifnextcat\bgroup{#2}{\gm@ifnac{#1}{#2}}}
```

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

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

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

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

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

First use of this macro is for an active - that expands to --- if followed by a space. Another to make dot checking whether is followed by ~ without gobbling the space if it occurs instead.

`\afterfi` and Pals

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

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

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

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

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

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

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

And two more of that family:

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

\afteriffifi
```
\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.

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

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

\afterfififi
```
\long\def\afterfififi#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
```
\@ifdefinable\@begnamedgroup{\relax}
```

\@begnamedgroup
```
\def\@begnamedgroup#1{%
  \@ignorefalse% not to ignore blanks after group
  \begingroup\@endpefalse
```

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

For back compatibility with my earlier works

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

And for the ending

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

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

\old@begin
```
\let\old@begin\begin
```

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

Improvement of \end

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

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

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

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

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

From relsize

As file relsize.sty, v3.1 dated July 4, 2003 states, LaTeX 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
```
\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 `\catcode`s

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.

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

The next command, `\foone`, is intended as two-argument for shortening of the `\bgroup...\firstofone{\egroup...}` hack.

\foone
```
\long\def\foone#1{\bgroup#1\egroupfirstofone}
\long\def\egroupfirstofone#1{\egroup#1}
```

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

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

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

\gobbletwo
```
\let\gobbletwo\@gobbletwo
```

Some ‘other’ 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{\@makeother\_}%
\xiiunder {\def\xiiunder{_}}
\ifdefined\XeTeXversion
\xiiunder   \def\xiiunder{\char"005F }%
  \let\_\xiiunder
\fi
\foone{\catcode`\[=1 \@makeother\{
  \catcode`\]=2 \@makeother\}}%
[%
\xiilbrace   \def\xiilbrace[{]%
\xiirbrace   \def\xiirbrace[}]%
]% of \firstofone
```

Note that LaTeX’s `\@charlb` and `\@charrb` are of catcode 11 (‘letter’), cf. The LaTeX 2_ε 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   \newcommand*\smartunder{%
    \catcode`\_=\active
    \def_{\ifmmode\subs\else\_\fi}}}% We define it as \_ not just as \xiiunder
          because some font encodings don't have _ at the \char`\_ position.
\foone{\catcode`\!=0
  \@makeother\\}
\xiibackslash {!newcommand*!xiibackslash{\}}
\bslash \let\bslash=\xiibackslash
\foone{\@makeother\%}
\xiipercent {\def\xiipercent{%}}
\foone{\@makeother\&}%
\xiiand {\def\xiiand{&}}
\foone{\@makeother\ }%
\xiispace {\def\xiispace{ }}
```

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 recognized 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
  \else
```

```
    \let\visiblespace\xiispace
  \fi}
```

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."

`\meta`
```
\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.

```
    #1\/%
  }\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.

`\meta@font@select`
```
\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{\kernosp\discretionary{#1}{#2}{#3}%
    \penalty10000\hskiposp\relax}
```

`\discret`
```
\def\discret#1{\kernosp\discretionary{#1}{#1}{#1}\penalty10000%
    \hskiposp\relax}
```

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

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

```
\def\:{\ifmmode\afterfi{\mskip\medmuskip}\else\afterfi{\discret{%
    }}\fi}
```

\vs
```
\newcommand*{\vs}{\discre{\visiblespace}{}{\visiblespace}}
```

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

\printspaces
\gm@pswords
```
\def\printspaces#1{{\let~=\vs␣\let\ =\vs␣\gm@pswords#1␣\@@nil}}
\def\gm@pswords#1␣#2\@@nil{%
  \ifx\relax#1\relax\else#1\fi
  \ifx\relax#2\relax\else\vs\penalty\hyphenpenalty\gm@pswords#2\@@nil%
    \fi}% note that in the recursive call of \gm@pswords the argument string is
        not extended with a guardian space: it has been already by \printspaces.
```

\sfname
```
\DeclareRobustCommand*\sfname[1]{\textsf{\printspaces{#1}}}
```

\gmu@discretionaryslash
```
\def\gmu@discretionaryslash{\discre{/}{\hbox{}}{/}}% the second pseudo-
    argument nonempty to get \hyphenpenalty not \exhyphenpenalty.
```

\file
\gmu@printslashes
```
\DeclareRobustCommand*\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 below macro I use to format the packages' names.

\pk
```
\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
\-
```
\DeclareRobustCommand*{\cs}[2][\bslash]{{%
      \def\-{\discretionary{{\rmfamily-}}{}{}}%
      \def\{{\char`\{}\def\}{\char`\}}\ttfamily␣#1#2}}
```

\env
```
\DeclareRobustCommand*{\env}[1]{\cs[]{#1}}
```

And for the special sequences like ^^A:

\hathat
```
\foone{\@makeother\^}
  {\DeclareRobustCommand*\hathat[1]{\cs[^^]{#1}}}
```

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

\possfil
```
\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
```
\def\cmd#1{\cs{\@xa\cmd@to@cs\string#1}}
```

```
\cmd@to@cs 235 \def\cmd@to@cs#1#2{\char\number`#2\relax}
```

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

```
\marg 236 \def\marg#1{{\ttfamily\char`\{}\meta{#1}{\ttfamily\char`\}}}
```

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

```
\oarg 237 \def\oarg{\@ifnextchar[\@oargsq\@oarg}
\@oarg 238 \def\@oarg#1{{\ttfamily[}\meta{#1}{\ttfamily]}}
\@oargsq 239 \def\@oargsq[#1]{\@oarg{#1}}
```

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

```
\parg 240 \def\parg{\@ifnextchar(\@pargp\@parg}
\@parg 241 \def\@parg#1{{\ttfamily(}\meta{#1}{\ttfamily)}}
\@pargp 242 \def\@pargp(#1){\@parg{#1}}
```

But we can have all three in one command.

```
243 \AtBeginDocument{%
\arg 244   \let\math@arg\arg
\arg 245   \def\arg{\ifmmode\math@arg\else\afterfi{%
246       \@ifnextchar[%
247       \@oargsq{\@ifnextchar(%
248         \@pargp\marg}}\fi}%
249 }
```

Storing and Restoring the Meanings of CSs

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

```
\ifgmu@SMglobal 250 \newif\ifgmu@SMglobal
\SMglobal 251 \def\SMglobal{\gmu@SMglobaltrue}
```

The subsequent commands are defined in such a way that you can 'prefix' them with `\SMglobal` to get global (re)storing.

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

```
\StoreMacro 252 \def\StoreMacro{%
253   \bgroup\makeatletter\@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 316.

```
\egStore@Macro 254 \long\def\egStore@Macro#1{\egroup\Store@Macro{#1}}
\egStore@MacroSt 255 \long\def\egStore@MacroSt#1{\egroup\Store@MacroSt{#1}}
\Store@Macro 256 \long\def\Store@Macro#1{%
257   \escapechar92
258   \ifgmu@SMglobal\afterfi\global\fi
259   \@xa\let\csname␣/gmu/store\string#1\endcsname#1%
260   \global\gmu@SMglobalfalse}
\Store@MacroSt 261 \long\def\Store@MacroSt#1{%
262   \edef\gmu@smtempa{%
263     \ifgmu@SMglobal\global\fi
```

```
    \@nx\let\@xa\@nx\csname/gmu/store\bslash#1\endcsname% we add backslash because to ensure compatibility between \(Re)StoreMacro and \(Re)StoreMacro*, that is. to allow writing e.g. \StoreMacro\kitten and then \RestoreMacro*{kitten} to restore the meaning of \kitten.
    \@xa\@nx\csname#1\endcsname}
  \gmu@smtempa
  \global\gmu@SMglobalfalse}% we wish the globality to be just once.
```

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

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

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

\StoreMacros
```
\long\def\StoreMacros{\bgroup\makeatletter\Store@Macros}
```
\Store@Macros
```
\long\def\Store@Macros#1{\egroup
  \gmu@setsetSMglobal
  \let\gml@StoreCS\Store@Macro
  \gml@storemacros#1.}
```
\gmu@setsetSMglobal
```
\def\gmu@setsetSMglobal{%
  \ifgmu@SMglobal
    \let\gmu@setSMglobal\gmu@SMglobaltrue
  \else
    \let\gmu@setSMglobal\gmu@SMglobalfalse
  \fi}
```

And the inner iterating macro:

\gml@storemacros
```
\long\def\gml@storemacros#1{%
```
\gmu@reserveda
```
  \def\gmu@reserveda{\@nx#1}% My TeX Guru's trick to deal with \fi and such, i.e., to hide #1 from TeX when it is processing a test's branch without expanding.
  \if\gmu@reserveda.% a dot finishes storing.
    \global\gmu@SMglobalfalse
  \else
    \if\gmu@reserveda,% 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{\gmu@setSMglobal\gml@storemacros}%
    \fi
  \fi}
```

And for the restoring

\RestoreMacro
```
\def\RestoreMacro{%
  \bgroup\makeatletter\@ifstar\egRestore@MacroSt\egRestore@Macro}
```
\egRestore@Macro
```
\long\def\egRestore@Macro#1{\egroup\Restore@Macro{#1}}
```
\egRestore@MacroSt
```
\long\def\egRestore@MacroSt#1{\egroup\Restore@MacroSt{#1}}
```
\Restore@Macro
```
\long\def\Restore@Macro#1{%
  \escapechar92
```

```
    \ifgmu@SMglobal\afterfi\global\fi
    \@xa\let\@xa#1\csname␣/gmu/store\string#1\endcsname
    \global\gmu@SMglobalfalse}
\Restore@MacroSt
\long\def\Restore@MacroSt#1{%
    \edef\gmu@smtempa{%
      \ifgmu@SMglobal\global\fi
      \@nx\let\@xa\@nx\csname#1\endcsname
      \@xa\@nx\csname/gmu/store\bslash#1\endcsname}% cf. the commentary
          in line 264.
    \gmu@smtempa
    \global\gmu@SMglobalfalse}
\RestoreMacros
\long\def\RestoreMacros{\bgroup\makeatletter\Restore@Macros}
\Restore@Macros
\long\def\Restore@Macros#1{\egroup
    \gmu@setsetSMglobal
    \let\gml@StoreCS\Restore@Macro% we direct the core CS towards restoring
        and call the same iterating macro as in line 272.
    \gml@storemacros#1.}
```

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

And to restore *and* use immediately:

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

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

```
\storedcsname
\def\storedcsname#1{%
  \csname␣/gmu/store\bslash#1\endcsname}
```

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

```
\StoreEnvironment
\def\StoreEnvironment#1{%
  \StoreMacro*{#1}\StoreMacro*{end#1}}
\RestoreEnvironment
\def\RestoreEnvironment#1{%
  \RestoreMacro*{#1}\RestoreMacro*{end#1}}
```

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

```
\StoringAndRelaxingDo
\long\def\StoringAndRelaxingDo{%
    \gmu@SMdo@setscope
    \long\def\do##1{%
      \gmu@SMdo@scope
      \@xa\let\csname␣/gmu/store\string##1\endcsname##1%
      \gmu@SMdo@scope\let##1\relax}}
\gmu@SMdo@setscope
\def\gmu@SMdo@setscope{%
    \ifgmu@SMglobal\let\gmu@SMdo@scope\global
    \else\let\gmu@SMdo@scope\relax
    \fi
    \global\gmu@SMglobalfalse}
```

And here is the counter-definition for restore.

\RestoringDo
```
\long\def\RestoringDo{%
  \gmu@SMdo@setscope
  \long\def\do##1{%
    \gmu@SMdo@scope
    \@xa\let\@xa##1\csname␣/gmu/store\string##1\endcsname}}
```

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

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

\n@melet
```
\def\n@melet#1#2{%
  \edef\gmu@nl@reserveda{%
    \let\@xa\@nx\csname#1\endcsname
    \@xa\@nx\csname#2\endcsname}%
  \gmu@nl@reserveda}
```

The `\global` prefix doesn't work with `\n@melet` so we define the alternative.

\gn@melet
```
\def\gn@melet#1#2{%
  \edef\gmu@nl@reserveda{%
    \global\let\@xa\@nx\csname#1\endcsname
    \@xa\@nx\csname#2\endcsname}%
  \gmu@nl@reserveda}
```

Not only preamble!

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

\not@onlypreamble
```
\newcommand\not@onlypreamble[1]{{%
  \def\do##1{\ifx#1##1\else\@nx\do\@nx##1\fi}%
  \xdef\@preamblecmds{\@preamblecmds}}}
\not@onlypreamble\@preamblecmds
\not@onlypreamble\@ifpackageloaded
\not@onlypreamble\@ifclassloaded
\not@onlypreamble\@ifl@aded
\not@onlypreamble\@pkgextension
```

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

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

\@nx \@nx

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 is \nocite that checks \ifx\@onlypreamble\document. So hoping this is a rare problem, we circumvent in with

\nocite
```
\def\nocite#1{%
  \@bsphack{\setboxo=\hbox{\cite{#1}}}\@esphack}
```

Third Person Pronouns

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

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

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

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

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

\heshe
```
\newcommand*\heshe{\gm@atppron{he}{she}}
```

\hisher
```
\newcommand*\hisher{\gm@atppron{his}{her}}
```

\himher
```
\newcommand*\himher{\gm@atppron{him}{her}}
```

\hishers
```
\newcommand*\hishers{\gm@atppron{his}{hers}}
```

\HeShe
```
\newcommand*\HeShe{\gm@atppron{He}{She}}
```

\HisHer
```
\newcommand*\HisHer{\gm@atppron{His}{Her}}
```

\HimHer
```
\newcommand*\HimHer{\gm@atppron{Him}{Her}}
```

\HisHers
```
\newcommand*\HisHers{\gm@atppron{His}{Hers}}
```

To Save Precious Count Registers

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

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

\stepnummacro
```
\newcommand*\stepnummacro[1]{%
  \@tempcnta=#1\relax
  \advance\@tempcnta␣by1\relax
  \xdef#1{\the\@tempcnta}}% Because of some mysterious reasons explicit \counto
      interferred with page numbering when used in \gmd@evpaddonce in gm-
      doc.
```

\addtonummacro
```
\newcommand*\addtonummacro[2]{%
  \counto=#1\relax
  \advance\countoby#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 \labels,
2. e.g. in a table of contents the hyperlink <rozdzia\l\ Kwiaty polskie> linked not to the chapter's heading but to the last-before-it change of \ref.

```
\AtBeginDocument{% because we don't know when exactly hyperref is loaded and
      maybe after this package.
  \@ifpackageloaded{hyperref}{\newcounter{NoNumSecs}%
    \setcounter{NoNumSecs}{617}% to make \refing 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}}% end of else
}% 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}
```

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

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

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

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

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

[2] A. Berg, *Wozzeck*.

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

The kernel of MW's sectioning commands:

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

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

```
      \@ifundefined{if@mainmatter}{}{%
           \gm@dontnumbersectionsoutofmainmatter}
  \fi
%   \ifHeadingNumbered
%     \refstepcounter{#1}%
%     \protected@edef\HeadingNumber{\csname
     the#1\endcsname\relax}%
%   \else
%     \let\HeadingNumber\@empty
%   \fi
```

\HeadingRHeadText
\HeadingTOCText
\HeadingText
\mw@HeadingType

```
  \def\HeadingRHeadText{#2}%
  \def\HeadingTOCText{#3}%
  \def\HeadingText{#4}%
  \def\mw@HeadingType{#1}%
  \if\mw@HeadingBreakBefore
    \if@specialpage\else\thispagestyle{closing}\fi
    \@ifundefined{if@openright}{}{\gm@clearpagesduetoopenright}%
    \if\mw@HeadingBreakAfter
      \thispagestyle{blank}\else
      \thispagestyle{opening}\fi
       \global\@topnum\z@
  \fi% of \if\mw@HeadingBreakBefore
```

placement of \refstep suggested by me (GM)

```
  \ifHeadingNumbered
    \refstepcounter{#1}%
    \protected@edef\HeadingNumber{\csname␣the#1\endcsname\relax}%
  \else
    \let\HeadingNumber\@empty
    \gm@hyperrefstepcounter
  \fi% of \ifHeadingNumbered
  \if\mw@HeadingRunIn
    \mw@runinheading
  \else
    \if\mw@HeadingWholeWidth
```

```
      \if@twocolumn
        \if\mw@HeadingBreakAfter
        \onecolumn
        \mw@normalheading
        \pagebreak\relax
              \if@twoside
                \null
                \thispagestyle{blank}%
                \newpage
              \fi% of \if@twoside
        \twocolumn
        \else
          \@topnewpage[\mw@normalheading]%
        \fi% of \if\mw@HeadingBreakAfter
      \else
        \mw@normalheading
        \if\mw@HeadingBreakAfter\pagebreak\relax\fi
      \fi% of \if@twocolumn
    \else
      \mw@normalheading
      \if\mw@HeadingBreakAfter\pagebreak\relax\fi
    \fi% of \if\mw@HeadingWholeWidth
  \fi% of \if\mw@HeadingRunIn
  }
```

An improvement of MW's `\SetSectionFormatting`

A version of MW's `\SetSectionFormatting` that lets to leave some settings unchanged by leaving the respective argument empty ({} or []).

Notice: If we adjust this command for new version of MWCLS, we should name it `\SetSectionFormatting` and add issuing errors if the inner macros are undefined.

#1 (optional) 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}%
    \@ifempty{#1}{}{\mw@processflags#1,\relax}% If #1 is omitted, the flags
          are left unchanged. If #1 is given, even as [], the flags are first cleared and
          then processed again.
  \fi
  \@ifundefined{#2}{\@namedef{#2}{\mw@section{#2}}}{}%
  \mw@secdef{#2}{@preskip}␣{#3}{2␣oblig.}%
  \mw@secdef{#2}{@head}␣␣␣␣{#4}{3␣oblig.}%
  \mw@secdef{#2}{@postskip}{#5}{4␣oblig.}%
  \ifx\empty#1\relax
    \mw@secundef{#2@flags}{1␣(optional)}%
  \else\mw@setflags{#2}%
```

\SetSectionFormatting
\mw@HeadingRunIn
\mw@HeadingBreakBefore
\mw@HeadingBreakAfter
\mw@HeadingWholeWidth

```
  \fi}
\mw@secdef \def\mw@secdef#1#2#3#4{% #1 the heading name,
             % #2 the command distinctor,
             % #3 the meaning,
             % #4 the number of argument to error message.
  \@ifempty{#3}
    {\mw@secundef{#1#2}{#4}}
    {\@namedef{#1#2}{#3}}}
\mw@secundef \def\mw@secundef#1#2{%
  \@ifundefined{#1}{%
    \ClassError{mwcls/gm}{%
      command␣\bslash#1␣␣undefined␣\MessageBreak
      after␣\bslash␣SetSectionFormatting!!!\MessageBreak}{%
      Provide␣the␣#2␣argument␣of␣\bslash␣
           SetSectionFormatting.}}{}}
```

First argument is a sectioning command (wo. \) and second the stuff to be added at the beginning of the heading declarations.

```
\addtoheading \def\addtoheading#1#2{%
    \n@melet{gmu@reserveda}{#1@head}%
    \toks\z@=\@xa{\gmu@reserveda}%
    \toks\tw@={#2}%
    \edef\gmu@reserveda{\the\toks\tw@\the\toks\z@}%
    \n@melet{#1@head}{gmu@reserveda}%
}
```

Negative `\addvspace`

When two sectioning commands appear one after another (we may assume that this occurs only when a lower section appears immediately after higher), we prefer to put the *smaller* vertical space not the larger, that is, the preskip of the lower sectioning not the postskip of the higher.

For that purpose we modify the very inner macros of MWCLS to introduce a check whether the previous vertical space equals the postskip of the section one level higher.

```
\@ifnotmw{}{% We proceed only in MWCLS
```

The information that we are just after a heading will be stored in the `\gmu@prevsec` macro: any heading will define it as the section name and `\everypar` (any normal text) will clear it.

```
\@afterheading \def\@afterheading{%
  \@nobreaktrue
  \xdef\gmu@prevsec{\mw@HeadingType}% added now
  \everypar{%
    \grelaxen\gmu@prevsec% added now. All the rest is original LaTeX.
    \if@nobreak
    \@nobreakfalse
    \clubpenalty␣\@M
    \if@afterindent␣\else
    {\setbox\z@\lastbox}%
    \fi
    \else
    \clubpenalty␣\@clubpenalty
```

```
    \everypar{}%
    \fi}}
```

If we are (with the current heading) just after another heading (one level lower I suppose), then we add the less of the higher header's post-skip and the lower header pre-skip or, if defined, the two-header-skip. (We put the macro defined below just before `\addvspace` in MWCLS inner macros.)

\gmu@checkaftersec
```
\def\gmu@checkaftersec{%
  \@ifundefined{gmu@prevsec}{}{%
    \ifgmu@postsec% an additional switch that is true by default but may be turned into an \ifdim in special cases, see line 534.
    {\@xa\mw@getflags\@xa{\gmu@prevsec}%
      \glet\gmu@reserveda\mw@HeadingBreakAfter}%
```
\gmu@reserveda
```
    \if\mw@HeadingBreakBefore\def\gmu@reserveda{11}\fi% if the current heading inserts page break before itself, all the play with vskips is irrelevant.
    \if\gmu@reserveda\else
    \penalty10000\relax
    \skip\z@=\csname\gmu@prevsec␣@postskip\endcsname\relax
    \skip\tw@=\csname\mw@HeadingType␣@preskip\endcsname\relax
    \@ifundefined{\mw@HeadingType␣@twoheadskip}{
      \ifdim\skip\z@>\skip\tw@
      \vskip-\skip\z@% we strip off the post-skip of previous header if it's bigger than current pre-skip
      \else
      \vskip-\skip\tw@% we strip off the current pre-skip otherwise
      \fi}{% But if the two-header-skip is defined, we put it
      \penalty10000
      \vskip-\skip\z@
      \penalty10000
      \vskip-\skip\tw@
      \penalty10000
      \vskip\csname\mw@HeadingType␣@twoheadskip\endcsname
      \relax}%
    \penalty10000
    \hrule␣height\z@\relax% to hide the last (un)skip before subsequent \addvspaces.
    \penalty10000
    \fi
    \fi
  }% of \@ifundefined{gmu@prevsec} 'else'
}% of \def\gmu@checkaftersec
```

\ParanoidPostsec
```
\def\ParanoidPostsec{% this version of \ifgmu@postsec is intended for the special case of sections may contain no normal text, as while gmdocing.
```
\ifgmu@postsec
```
  \def\ifgmu@postsec{% note this macro expands to an open \if.
    \skip\z@=\csname\gmu@prevsec␣@postskip\endcsname\relax
    \ifdim\lastskip=\skip\z@\relax% we play with the vskips only if the last skip is the previous heading's postskip (a counter-example I met while gmdocing).
  }}
\let\ifgmu@postsec\iftrue
```

\gmu@getaddvs
```
\def\gmu@getaddvs#1\addvspace#2\gmu@getaddvs{%
  \toks\z@={#1}
```

```
  \toks\tw@={#2}}
```

And the modification of the inner macros at last:

\gmu@setheading
```
\def\gmu@setheading#1{%
  \@xa\gmu@getaddvs#1\gmu@getaddvs
  \edef#1{%
    \the\toks\z@\@nx\gmu@checkaftersec
    \@nx\addvspace\the\toks\tw@}}
\gmu@setheading\mw@normalheading
\gmu@setheading\mw@runinheading
```

\SetTwoheadSkip
```
\def\SetTwoheadSkip#1#2{\@namedef{#1@twoheadskip}{#2}}
```

```
}% of \@ifnotmw
```

My heading setup for mwcls

The setup of heading skips was tested in 'real' typesetting, for money that is. The skips are designed for 11/13 pt leading and together with my version of mw11.clo option file for mwcls make the headings (except paragraph and subparagraph) consist of an integer number of lines. The name of the declaration comes from my employer, "Wiedza Powszechna" Editions.

```
\@ifnotmw{}{% We define this declaration only when in mwcls.
```

\WPheadings
```
\def\WPheadings{%
  \SetSectionFormatting[breakbefore,wholewidth]
      {part}{\z@\@plus1fill}{}{\z@\@plus3fill}%
  \@ifundefined{chapter}{}{%
    \SetSectionFormatting[breakbefore,wholewidth]
      {chapter}
      {66\p@}% {67\p@} for Adventor/Schola 0,95.
      {\FormatHangHeading{\LARGE}}
      {27\p@\@pluso,2\p@\@minus1\p@}%
  }%
  \SetTwoheadSkip{section}{27\p@\@pluso,5\p@}%
  \SetSectionFormatting{section}
      {24\p@\@pluso,5\p@\@minus5\p@}%
      {\FormatHangHeading␣{\Large}}
      {10\p@\@pluso,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@\@pluso,5\p@\@minus1\p@}%
  \SetSectionFormatting{subsection}
      {19\p@\@pluso,4\p@\@minus6\p@}
      {\FormatHangHeading␣{\large}}% 12/14 pt
      {6\p@\@pluso,3\p@}% after-skip 6 pt due to p.12, not to squeeze the before-skip too much.
  \SetTwoheadSkip{subsubsection}{10\p@\@plus1,75\p@\@minus1\p@}%
  \SetSectionFormatting{subsubsection}
      {10\p@\@pluso,2\p@\@minus1\p@}
      {\FormatHangHeading␣{\normalsize}}
      {3\p@\@pluso,1\p@}% those little skips should be smaller than you calculate out of a geometric progression, because the interline skip enlarges them.
  \SetSectionFormatting[runin]{paragraph}
      {7\p@\@pluso,15\p@\@minus1\p@}
```

```
      {\FormatRunInHeading{\normalsize}}
      {2\p@}%
  \SetSectionFormatting[runin]{subparagraph}
      {4\p@\@plus1\p@\@minuso,5\p@}
      {\FormatRunInHeading{\normalsize}}
      {\z@}%
}% of \WPheadings
}% of \@ifnotmw
```

Compatibilising Standard and mwcls Sectionings

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

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

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

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

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

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

```
\@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those written with two optionals.
  \def\gm@secini{gm@la}%
  \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 argument.
        \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 sectioning command was written starless, we pass the 'true' sectioning command #3 as the optional argument. Otherwise the sectioning command was written with star so the 'true' s.c. takes no optional.
```

```
    \fi
    \@xa\@xa\csname\gm@secini#1\endcsname
    \gm@secstar{#4}}%
}{% we are in mwcls and want to reverse MW's optionals order i.e., if there's just one optional, it should go both to toc and to running head.
\gm@secini  \def\gm@secini{gm@mw}%
  \let\gm@secmarkh\@gobble% in mwcls there's no need to make tricks for special version to running headings.
\gm@secxx   \def\gm@secxx#1#2[#3]#4{%
    \@xa\@xa\csname\gm@secini#1\endcsname
    \gm@secstar[#2][#3]{#4}}%
}
\gm@sec  \def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
\gm@secx  \def\gm@secx#1[#2]{%
  \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}% if there's only one optional, we double it not the mandatory argument.
\gm@straightensec  \def\gm@straightensec#1{% the parameter is for the command's name.
  \@ifundefined{#1}{}{% we don't change the ontological status of the command because someone may test it.
    \n@melet{\gm@secini#1}{#1}%
    \@namedef{#1}{%
\gm@secstar       \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
\gm@secstar         \def\gm@secstar{}\gm@sec{#1}}}%
}%
\let\do\gm@straightensec
\do{part}\do{chapter}\do{section}\do{subsection}\do{subsubsection}
\@ifnotmw{}{\do{paragraph}}% this 'straightening' of \paragraph with the standard article caused the 'TeX capacity exceeded' error. Anyway, who on Earth wants paragraph titles in toc or running head?
```

`enumerate*` and `itemize*`

We wish the starred version of `enumerate` to be just numbered paragraphs. But hyperref redefines `\item` so we should do it a smart way, to set the LaTeX's `list` parameters that is.

(Marcin Woliński in mwcls defines those environments slightly different: his item labels are indented, mine are not; his subsequent paragraphs of an item are not indented, mine are.)

```
enumerate*  \@namedef{enumerate*}{%
  \ifnum\@enumdepth>\thr@@
    \@toodeep
  \else
    \advance\@enumdepth\@ne
    \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
    \@xa\list\csname␣label\@enumctr\endcsname{%
      \partopsep\topsep␣\topsep\z@␣\leftmargin\z@
      \itemindent\@parindent␣% %\advance\itemindent\labelsep
      \labelwidth\@parindent
      \advance\labelwidth-\labelsep
```

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

itemize*

```
\@namedef{itemize*}{%
  \ifnum\@itemdepth>\thr@@
    \@toodeep
  \else
    \advance\@itemdepth\@ne
    \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
    \@xa\list\csname\@itemitem\endcsname{%
      \partopsep\topsep␣\topsep\z@␣\leftmargin\z@
      \itemindent\@parindent
      \labelwidth\@parindent
      \advance\labelwidth-\labelsep
      \listparindent\@parindent
      \def\makelabel##1{##1\hfil␣}}%
  \fi}
\@namedef{enditemize*}{\endlist}
```

The Logos

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

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

\DeclareLogo

```
\newcommand*\DeclareLogo[3][\relax]{%
```

#1 is for non-LaTeX spelling and will be used in the PD1 encoding (to make pdf bookmarks);
#2 is the command, its name will be the PD1 spelling by default,
#3 is the definition for all the font encodings except PD1.

\gmu@reserveda

```
  \ifx\relax#1\def\gmu@reserveda{\@xa\@gobble\string#2}%
  \else
```

\gmu@reserveda

```
    \def\gmu@reserveda{#1}%
  \fi
  \edef\gmu@reserveda{%
```

\@nx

```
    \@nx\DeclareTextCommand\@nx#2{PD1}{\gmu@reserveda}}
  \gmu@reserveda
  \DeclareTextCommandDefault#2{#3}%
```

\DeclareRobustCommand*

```
  \DeclareRobustCommand*#2{#3}}% added for XeTeX
```

\DeclareLogo

```
\DeclareLogo\LaTeX{%
  {%
    L%
    \setbox\z@\hbox{\check@mathfonts
      \fontsize\sf@size\z@
      \math@fontsfalse\selectfont
      A}%
    \kern-.57\wd\z@
```

```
    \sbox\tw@␣T%
    \vbox␣to\ht\tw@{\copy\z@␣\vss}%
    \kern-.2\wd\z@}% originally −, 15 em for T.
  {%
    \ifdim\fontdimen1\font=\z@
    \else
      \count\z@=\fontdimen5\font
      \multiply\count\z@␣by␣64\relax
      \divide\count\z@␣by\p@
      \count\tw@=\fontdimen1\font
      \multiply\count\tw@␣by\count\z@
      \divide\count\tw@␣by␣64\relax
      \divide\count\tw@␣by\tw@
      \kern-\the\count\tw@␣sp\relax
    \fi}%
  \TeX}
\LaTeXe \DeclareLogo\LaTeXe{\mbox{\m@th␣\if
    b\expandafter\@car\f@series\@nil\boldmath\fi
    \LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}
\StoreMacro\LaTeX
\StoreMacro*{LaTeX␣}
```

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

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

"Here are a few definitions which can usefully be employed when documenting package files: now we can readily refer to $\mathcal{AMS}$-TEX, BIBTEX and SLITEX, as well as the usual TEX and LATEX. There's even a PLAIN TEX and a WEB."

```
\@ifundefined{AmSTeX}
\AmSTeX   {\def\AmSTeX{\leavevmode\hbox{$\mathcal␣A\kern-.2em%
        \lower.376ex%
          \hbox{$\mathcal␣M$}\kern-.2em\mathcal␣S$-\TeX}}}{}
\BibTeX \DeclareLogo\BibTeX{{\rmfamily␣B\kern-.05em%
```

```
      \textsc{i{\kern-.025em}b}\kern-.08em% the kern is wrapped in braces
           for my \fakescaps' sake.
      \TeX}}
\SliTeX \DeclareLogo\SliTeX{{\rmfamily␣S\kern-.06emL\kern-.18em%
       \raise.32ex\hbox
          {\scshape␣i}\kern␣-.03em\TeX}}
\PlainTeX \DeclareLogo\PlainTeX{\textsc{Plain}\kern2pt\TeX}
    \Web \DeclareLogo\Web{\textsc{Web}}
```

There's also the (LA)TEX logo got with the `\LaTeXpar` macro provided by gmutils. And here *The TEXbook*'s logo:

```
\TeXbook \DeclareLogo[The␣TeX␣book]\TeXbook{\textsl{The␣\TeX␣book}}
         \let\TB\TeXbook% TUG Boat uses this.
    \eTeX \DeclareLogo[e-TeX]\eTeX{%
      \ensuremath{\varepsilon}-\kern-.125em\TeX}% definition sent by Karl Berry
           from TUG Boat itself.
\pdfeTeX \DeclareLogo[pdfe-TeX]\pdfeTeX{pdf\eTeX}
 \pdfTeX \DeclareLogo\pdfTeX{pdf\TeX}
         \@ifundefined{XeTeX}{%
  \XeTeX   \DeclareLogo\XeTeX{X\kern-.125em\relax
             \@ifundefined{reflectbox}{%
               \lower.5ex\hbox{E}\kern-.1667em\relax}{%
               \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
             \TeX}}{}
         \@ifundefined{XeLaTeX}{%
\XeLaTeX   \DeclareLogo\XeLaTeX{X\kern-.125em\relax
             \@ifundefined{reflectbox}{%
               \lower.5ex\hbox{E}\kern-.1667em\relax}{%
               \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
             \LaTeX}}
```

As you see, if TEX doesn't recognize `\reflectbox` (graphics isn't loaded), the first `E` will not be reversed. This version of the command is intended for non-XETEX usage. With XETEX, you can load the xltxtra package (e.g. with the gmutils `\XeTeXthree` declaration) and then the reversed `E` you get as the Unicode Latin Letter Reversed E.

Expanding turning stuff all into 'other'

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

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

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

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

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

```
\edef\gmu@reserveda{%
```

```
\@nx      744   \def\@nx\gm@gobmacro##1\@xa\@gobble\string\macro:->{}}
\gm@gobmacro 745 \gmu@reserveda
```

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

```
\unex@namedef 746 \long\def\unex@namedef#1#2{%
              747   \edef@other\gmu@reserveda{#1}%
              748   \@xa\long\@xa\def\csname\gmu@reserveda\endcsname{#2}}
\unex@nameuse 749 \long\def\unex@nameuse#1{%
              750   \edef@other\gmu@reserveda{#1}%
              751   \csname\gmu@reserveda\endcsname}
```

Brave New World of XƎTEX

```
\@ifXeTeX 752 \newcommand\@ifXeTeX[2]{%
          753   \ifdefined\XeTeXversion
          754   \unless\ifx\XeTeXversion\relax\afterfifi{#1}\else\afterfifi{%
                    #2}\fi
          755   \else\afterfi{#2}\fi}
\XeTeXthree 756 \def\XeTeXthree{%
          757   \@ifXeTeX{%
          758     \@ifpackageloaded{gmverb}{\StoreMacro\verb}{}%
          759     \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.
          760     \@ifpackageloaded{gmverb}{\RestoreMacro\verb}{}%
          761     \AtBeginDocument{%
          762       \RestoreMacro\LaTeX\RestoreMacro*{LaTeX␣}}% my version of the LaTeX logo has been stored just after defining, in line 693.
          763 }{}}
```

The `\udigits` declaration causes the digits to be typeset uppercase. I provide it since by default I prefer the lowercase (nautical) digits.

```
          764 \AtBeginDocument{%
          765   \@ifpackageloaded{fontspec}{%
\udigits  766     \DeclareRobustCommand*\udigits{%
          767       \addfontfeature{Numbers=Uppercase}}%
          768   }{%
          769     \emptify\udigits}}
```

Fractions

```
\Xedekfracc 770 \def\Xedekfracc{\@ifstar\gmu@xedekfraccstar\gmu@xedekfraccplain}
```

(plain) The starless version turns the font feature `frac` on. (*) But nor Minion GM 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 `\@namedefs` below) and the `numr` and `dnom` features with the fractional slash otherwise (via `\gmu@dekfracc`). (**) But Latin Modern Sans Serif Quotation doesn't support the numerator and denominator positions so we provide the double star version for it, which takes the char from font if it exist and typesets with lowers and kerns otherwise.

```
\gmu@xedekfraccstar 771 \def\gmu@xedekfraccstar{%
\gmu@xefraccdef 772   \def\gmu@xefraccdef##1##2{%
773       \iffontchar\font␣##2
774         \@namedef{gmu@xefracc##1}{\char##2␣}%
775       \else
776         \n@melet{gmu@xefracc##1}{relax}%
777       \fi}%
\gmu@dekfracc 778     \def\gmu@dekfracc##1/##2{%
779       {\addfontfeature{VerticalPosition=Numerator}##1}%
              \gmu@numeratorkern
780       \char"2044␣\gmu@denominatorkern
781       {\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.

```
782     \gmu@xefraccdef{1/4}{"BC}%
783     \gmu@xefraccdef{1/2}{"BD}%
784     \gmu@xefraccdef{3/4}{"BE}%
785     \gmu@xefraccdef{1/3}{"2153}%
786     \gmu@xefraccdef{2/3}{"2154}%
787     \gmu@xefraccdef{1/8}{"215B}%
788     \gmu@xefraccdef{3/8}{"215C}%
789     \gmu@xefraccdef{5/8}{"215D}%
790     \gmu@xefraccdef{7/8}{"215E}%
\dekfracc 791     \def\dekfracc##1/##2{%
\gm@duppa 792       \def\gm@duppa{##1/##2}%
793       \@ifundefined{gmu@xefracc\all@other\gm@duppa}{%
794         \gmu@dekfracc{##1}/{##2}}{%
795         \csname␣gmu@xefracc\all@other\gm@duppa\endcsname}}%
796     \@ifstar{\let\gmu@dekfracc\gmu@dekfraccsimple}{}%
797   }
\gmu@xedekfraccplain 798 \def\gmu@xedekfraccplain{% 'else' of the main \@ifstar
\dekfracc 799     \def\dekfracc##1/##2{{%
800         \addfontfeature{Fractions=On}%
801         ##1/##2}}%
802     }
\gmu@numeratorkern 803 \def\gmu@numeratorkern{\kern-.05em\relax}
804 \let\gmu@denominatorkern\gmu@numeratorkern
```

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 ⅞. To achieve that, a handful of macros is defined that expand to the Unicodes of built-in fractions and `\dekfracc` command is defined to use them.

The unstarred version makes use of the Fraction font feature and therefore is much simpler.

Note that in the first argument of `\@ifstar` we wrote 8 (eight) #s to get the correct definition and in the second argument 'only' 4. (The LaTeX 2ε Source claims that that is changed in the 'new implementation' of `\@ifstar` so maybe it's subject to change.)

A simpler version of `\dekfracc` is provided in line 950

`\resizegraphics`

```
805 \@ifXeTeX{%
```

```
\resizegraphics 806 \def\resizegraphics#1#2#3{{%
807     \setboxo=\hbox{\XeTeXpicfile␣#3␣}%
808     \ifx!#1\else
809       \dimeno=#1\relax
810       \count2=\wdo
811       \divide\count2␣by1000\relax
812       \counto=\dimeno\relax
813       \divide\counto\count2
814     \fi
815     \ifx!#2\else
816       \dimeno=#1\relax
817       \count6=\hto
818       \divide\count6␣by1000\relax
819       \count4=\dimeno\relax
820       \divide\count4\count6
821     \fi
822     \ifx!#1\counto=\count4\fi
823     \ifx!#2\count4=\counto\fi
824     \XeTeXpicfile␣#3␣xscaled␣\counto␣yscaled␣\count4
825   }}}{%
\resizegraphics 826   \def\resizegraphics#1#2#3{%
827     \resizebox{#1}{#2}{%
828       \includegraphics{#3}}}}%
```

The [options] in the `\XeTeXpicfile` command use the following keywords:
`width` ⟨*dimen*⟩
`height` ⟨*dimen*⟩
`scaled` ⟨*scalefactor*⟩
`xscaled` ⟨*scalefactor*⟩
`yscaled` ⟨*scalefactor*⟩
`rotated` ⟨*degrees*⟩

```
\GMtextsuperscript 829 \def\GMtextsuperscript{%
830   \@ifXeTeX{%
\textsuperscript 831     \def\textsuperscript##1{{%
832         \addfontfeature{VerticalPosition=Numerator}##1}}%
833   }{\truetextsuperscript}}
\truetextsuperscript 834 \def\truetextsuperscript{%
\textsuperscript 835   \DeclareRobustCommand*\textsuperscript[1]{%
836     \@textsuperscript{\selectfont##1}}%
\@textsuperscript 837   \def\@textsuperscript##1{%
838     {\m@th\ensuremath{^{\mbox{\fontsize\sf@size\z@##1}}}}}}
```

Varia

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

```
\gmu@tilde 839 \def\gmu@tilde{%
840   \leavevmode\lower.8ex\hbox{$\,\widetilde{\mbox{␣}}\,$}}
```

Originally there was just `\ ` instead of `\mbox{ }` but some commands of ours do redefine `\ `.

```
\* 841 \DeclareRobustCommand*\*{\gmu@tilde}
```

```
\AtBeginDocument{% to bypass redefinition of \~ as a text command with various encodings
\texttilde  \DeclareRobustCommand*\texttilde{%
              \@ifnextchar/{\gmu@tilde\kern-0,1667em\relax}\gmu@tilde}}
```

We prepare the proper kerning for "~/".

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

(This definition is repeated in gmverb.)

```
\foone{\catcode`\ \active}%
\gmobeyspaces {\def\gmobeyspaces{\catcode`\ \active\let␣\ }}
```

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

```
\foone{\catcode`\^^M\active}% the comment signs here are crucial.
\defobeylines {\def\defobeylines{\catcode`\^^M=13␣\def^^M{\par}}}
```

Another thing I dislike in LaTeX yet is doing special things for `\...skip`'s, 'cause I like the Knuthian simplicity. So I sort of restore Knuthian meanings:

```
\deksmallskip   \def\deksmallskip{\vskip\smallskipamount}
\undeksmallskip \def\undeksmallskip{\vskip-\smallskipamount}
\dekmedskip     \def\dekmedskip{\vskip\medskipamount}
\dekbigskip     \def\dekbigskip{\vskip\bigskipamount}
\hfillneg       \def\hfillneg{\hskip␣opt␣plus␣-1fill\relax}
```

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

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

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

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

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

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

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

```
\enoughpage 866 \newcommand\enoughpage[1]{%
            867   \par
            868   \dimeno=\pagegoal
            869   \advance\dimeno␣by-\pagetotal
            870   \ifdim\dimeno<#1\relax\newpage\fi}
```

Two shorthands for debugging:

```
\tOnLine    871 \newcommand*\tOnLine{\typeout{\on@line}}
\OnAtLine   872 \let\OnAtLine\on@line
```

An equality sign properly spaced:

```
\equals     873 \newcommand*\equals{${}={}$}
```

And for the LaTeX's pseudo-code statements:

```
\eequals    874 \newcommand*\eequals{${}=={}$}
```

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

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

```
\freeze@actives 875 \def\freeze@actives{%
            876   \count\z@\z@
            877   \@whilenum\count\z@<\@cclvi\do{%
            878     \ifnum\catcode\count\z@=\active
\~          879       \uccode`\~=\count\z@
            880       \uppercase{\let~\relax}%
            881     \fi
            882     \advance\count\z@\@ne}}
```

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

```
\ShowFont   883 \newcommand*\ShowFont[1][6]{%
            884   \begin{multicols}{#1}[The␣current␣font␣(the␣\f@encoding%
                    \ encoding):]
            885     \parindent\z@
            886     \count\z@\m@ne
            887     \@whilenum\count\z@<\@cclv\do{
            888       \advance\count\z@\@ne
            889       \ \the\count\z@:~\char\count\z@\par}
            890   \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 891 \newcommand*\liturgiques[1][red]{% Requires the color package.
            892   \gmu@RPif{color}{color}%
\czerwo     893   \newcommand*\czerwo{\small\color{#1}}% environment
\czer       894   \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.
\*          895   \def\*{\czer{$*$}}
```

```
\+      \def\+{\czer{$\dag$}}
\nieczer  \newcommand*\nieczer[1]{\textcolor{black}{##1}}}
```

After the next definition you can write `\gmu@RP[⟨options⟩]{⟨package⟩}{⟨csname⟩}` to get the package #2 loaded with options #1 if the csname #3 is undefined.

```
\gmu@RPif \newcommand*\gmu@RPif[3][]{%
            \ifx\relax#1\relax
\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@RPif` to issue a request before the error issued by undefined CS.

```
          \AtBeginDocument{%
\gmu@RPif   \renewcommand*\gmu@RPif[3][]{%
              \@ifundefined{#3}{%
                \@ifpackageloaded{#2}{}{%
                  \typeout{^^J!␣Package␣`#2'␣not␣loaded!!!␣(%
                       \on@line)^^J}}}{}}}
```

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

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

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

```
\iteracro \def\iteracro{%
    \acro   \DeclareRobustCommand*\acro[1]{\gmu@acrospaces##1␣%
                 \gmu@acrospaces}%
          }
          \iteracro
```

```
\gmu@acrospaces \def\gmu@acrospaces#1␣#2\gmu@acrospaces{%
                  \gmu@acroinner#1\gmu@acroinner
                  \ifx\relax#2\relax\else
                    \space
                    \afterfi{\gmu@acrospaces#2\gmu@acrospaces}% when #2 is nonempty, it
                          is ended with a space. Adding one more space in this line resulted in an
                          infinite loop.
                  \fi}
```

```
\gmu@acroinner \def\gmu@acroinner#1{%
                 \ifx\gmu@acroinner#1\relax\else
                   \ifcat␣a\@nx#1\relax%
                     \ifnum`#1=\uccode`#1%
                       {\acrocore{#1}}%
                     \else{#1}% tu było \smallerr
                     \fi
                   \else#1%
                   \fi
                   \afterfi\gmu@acroinner
                 \fi}
```

We extract the very thing done to the letters to a macro because we need to redefine it in fonts that don't have small caps.

```
\acrocore 930 \def\acrocore{\scshape\lowercase}
```

Since the fonts I am currently using do not support required font feature, I skip the following definition.

```
\IMO 931 \newcommand*\IMO{\acro{IMO}}
\AKA 932 \newcommand*\AKA{\acro{AKA}}
\usc 933 \DeclareRobustCommand*\usc[1]{{\addfontfeature{%
           Letters=UppercaseSmallCaps}#1}}
\uscacro 934 \def\uscacro{\let\acro\usc}
\qxenc 935 \newcommand*\qxenc{\fontencoding{QX}\selectfont}
```

The `\copyright` command is unavailable in T1 and U (unknown) encodings so provide

```
\qxcopyright 936 \newcommand*\qxcopyright{{\qxenc\copyright}}
\qxcopyrights 937 \newcommand*\qxcopyrights{%
938   \let\gmu@copyright\copyright
939   \def\copyright{{\qxenc\gmu@copyright}}}
\fixcopyright 940 \newcommand*\fixcopyright{%
941   \@ifXeTeX{\def\copyright{\char"00A9␣}}{\qxcopyrights}}
```

Probably the only use of it is loading gmdocc.cls 'as second class'. This command takes first argument optional, options of the class, and second mandatory, the class name. I use it in an article about gmdoc.

```
\secondclass 942 \def\secondclass{%
\ifSecondClass 943   \newif\ifSecondClass
944   \SecondClasstrue
945   \@fileswithoptions\@clsextension}% [outeroff,gmeometric]{gmdocc}
```

it's loading gmdocc.cls with all the bells and whistles except the error message.

Cf. *The TEXbook* exc. 11.6.
A line from LaTeX:

```
% \check@mathfonts\fontsize\sf@size\z@\math@fontsfalse\selectfont
```

didn't work as I would wish: in a `\footnotesize`'s scope it still was `\scriptsize`, so too large.

```
\gmu@dekfraccsimple 946 \def\gmu@dekfraccsimple#1/#2{\leavevmode\kern.1em
947   \raise.5ex\hbox{\udigits\smaller[3]#1}\gmu@numeratorkern
948   \dekfraccslash\gmu@denominatorkern
949   {\udigits\smaller[3]#2}}%
\dekfraccsimple 950 \def\dekfraccsimple{%
951   \let\dekfracc\gmu@dekfraccsimple
952 }
\dekfraccslash 953 \@ifXeTeX{\def\dekfraccslash{\char"2044␣}}{%
\dekfraccslash 954   \def\dekfraccslash{/}}␣% You can define it as the fraction slash, \char"2044
955 \dekfraccsimple
```

A macro that acts like `\,` (thin and unbreakable space) except it allows hyphenation afterwards:

```
\ikern 956 \newcommand*\ikern{\,\penalty10000\hskip0sp\relax}
```

And a macro to forbid hyphenation of the next word:

```
\nohy   \newcommand*\nohy{\kernosp\relax}
\yeshy  \newcommand*\yeshy{\penalty10000\hskipsp\relax}
```

In both of the above definitionc 'osp' not \z@ to allow their writing to and reading from files where @ is 'other'.

\@ifempty

```
\@ifempty       \long\def\@ifempty#1#2#3{%
\gmu@reserveda    \def\gmu@reserveda{#1}%
                  \ifx\gmu@reserveda\@empty\afterfi{#2}%
                  \else\afterfi{#3}\fi
                }
```

\include not only .tex's

\include modified by me below lets you to include files of any extension provided that extension in the argument.

If you want to \include a non-.tex file and deal with it with \includeonly, give the latter command full file name, with the extension that is.

```
\gmu@getext     \def\gmu@getext#1.#2\@@nil{%
\gmu@filename     \def\gmu@filename{#1}%
\gmu@fileext      \def\gmu@fileext{#2}}

                \def\include#1{\relax
                  \ifnum\@auxout=\@partaux
                  \@latex@error{\string\include\space␣cannot␣be␣nested}\@eha
                  \else␣\@include#1␣\fi}

\@include       \def\@include#1␣{%
                  \gmu@getext#1.\@@nil
\gmu@fileext      \ifx\gmu@fileext\empty\def\gmu@fileext{tex}\fi
                  \clearpage
                  \if@filesw
                    \immediate\write\@mainaux{\string\@input{\gmu@filename.aux}}%
                  \fi
                  \@tempswatrue
                  \if@partsw
                    \@tempswafalse
                    \edef\reserved@b{#1}%
                    \@for\reserved@a:=\@partlist\do{%
                      \ifx\reserved@a\reserved@b\@tempswatrue\fi}%
                  \fi
                  \if@tempswa
                    \let\@auxout\@partaux
                    \if@filesw
                      \immediate\openout\@partaux␣\gmu@filename.aux
                      \immediate\write\@partaux{\relax}%
                    \fi
                    \@input@{\gmu@filename.\gmu@fileext}%
                    \inclasthook
                    \clearpage
                    \@writeckpt{\gmu@filename}%
                    \if@filesw
                      \immediate\closeout\@partaux
```

```
    \fi
  \else
```

If the file is not included, reset `\@include` `\deadcycles`, so that a long list of non-included files does not generate an 'Output loop' error.

```
    \deadcycles\z@
    \@nameuse{cp@\gmu@filename}%
  \fi
  \let\@auxout\@mainaux}
```

\whenonly
\gmu@whonly

```
\newcommand\whenonly[3]{%
  \def\gmu@whonly{#1,}%
  \ifx\gmu@whonly\@partlist\afterfi{#2}\else\afterfi{#3}\fi}
```

I assume one usually includes chapters or so so the last page style should be closing.

\inclasthook

```
\def\inclasthook{\thispagestyle{closing}}
```

Faked small caps

\gmu@scapLetters

```
\def\gmu@scapLetters#1{%
  \ifx#1\relax\relax\else% two \relaxes to cover the case of empty #1.
    \ifcat␣a#1\relax
      \ifnum\the\lccode`#1=`#1\relax
        {\fakescapscore\MakeUppercase{#1}}% not Plain \uppercase because
              that works bad with inputenc.
      \else#1%
      \fi
    \else#1%
    \fi%
    \@xa\gmu@scapLetters
  \fi}%
```

\gmu@scapSpaces

```
\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}
```

\gmu@scapss
\nobreakspace

```
\def\gmu@scapss#1\@@nil{{\def~{{\nobreakspace}}%
    \gmu@scapSpaces#1␣\@@nil}}% % \def\\{{\newline}}\relax adding re-
        definition of \\ caused stack overflow Note it disallows hyphenation ex-
        cept at \-.
```

\fakescaps

```
\DeclareRobustCommand\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.

\gmu@scalar
\zf@scale

```
\@ifXeTeX{%
  \def\gmu@scalar{1.0}%
  \def\zf@scale{}%
```

```
\gmu@scalematchX 1035  \def\gmu@scalematchX{%
                 1036    \begingroup
      \gmu@scalar 1037      \ifx\zf@scale\empty\def\gmu@scalar{1.0}%
                 1038      \else\let\gmu@scalar\zf@scale\fi
                 1039      \setlength\@tempdima{\fontdimen5\font}% 5—ex height
                 1040      \setlength\@tempdimb{\fontdimen8\font}% 8—XeTeX synthesized uppercase height.
                 1041      \divide\@tempdimb␣by1000\relax
                 1042      \divide\@tempdima␣by\@tempdimb
                 1043      \setlength{\@tempdima}{\@tempdima*\real{\gmu@scalar}}%
                 1044      \@ifundefined{fakesc@extrascale}{}{%
                 1045        \setlength{\@tempdima}{\@tempdima*\real{%
                               \fakesc@extrascale}}}%
                 1046      \@tempcnta=\@tempdima
                 1047      \divide\@tempcnta␣by␣1000\relax
                 1048      \@tempcntb=-1000\relax
                 1049      \multiply\@tempcntb␣by\@tempcnta
                 1050      \advance\@tempcntb␣by\@tempdima
                 1051      \xdef\gmu@scscale{\the\@tempcnta.%
                 1052        \ifnum\@tempcntb<100␣0\fi
                 1053        \ifnum\@tempcntb<10␣0\fi
                 1054        \the\@tempcntb}%
                 1055    \endgroup
                 1056    \addfontfeature{Scale=\gmu@scscale}%
                 1057  }}{\let\gmu@scalematchX\smallerr}
\fakescextrascale 1058 \def\fakescextrascale#1{\def\fakesc@extrascale{#1}}
\fakesc@extrascale
```

See above/see below

To generate a phrase as in the header depending of whether the respective label is before of after.

```
\wyzejnizej 1059 \newcommand*\wyzejnizej[1]{%
            1060   \edef\gmu@tempa{\@ifundefined{r@#1}{\arabic{page}}{%
            1061       \@xa\@xa\@xa\@secondoftwo\csname␣r@#1\endcsname}}%
            1062   \ifnum\gmu@tempa<\arabic{page}\relax␣wy\.zej\fi
            1063   \ifnum\gmu@tempa>\arabic{page}\relax␣ni\.zej\fi
            1064   \ifnum\gmu@tempa=\arabic{page}\relax␣\@xa\ignorespaces\fi
            1065 }
```

`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 1066 \def\napapierkistretch{0,3pt}% It's quite much for 11/13pt typesetting
\napapierkicore 1067 \def\napapierkicore{\advance\baselineskip%
            1068   by␣optplus\napapierkistretch\relax}
napapierki  1069 \newenvironment*{napapierki}{%
            1070   \par\global\napapierkicore}{%
            1071   \par\dimen\z@=\baselineskip
            1072   \global\baselineskip=\dimen\z@}% so that you can use \endnapapierki in interlacing environments
```

```
\gmu@luzniej 1073 \newcount\gmu@luzniej
\luzniejcore 1074 \newcommand*\luzniejcore[1][1]{%
1075   \advance\gmu@luzniej\@ne% We use this count to check whether we open the
         environment or just set \looseness inside it again.
1076   \ifnum\gmu@luzniej=\@ne␣␣\multiply\tolerance␣by␣2␣\fi
1077   \looseness=#1\relax}
```

After `\begin{luzniej}` we may put the optional argument of `\luzniejcore`

```
luzniej 1078 \newenvironment*{luzniej}{\par\luzniejcore}{\par}
```

The starred version does that `\everypar`, which has its advantages and disadvantages.

```
luzniej* 1079 \newenvironment*{luzniej*}[1][1]{%
1080   \multiply\tolerance␣by␣2\relax
1081   \everypar{\looseness=#1\relax}}{\par}
\nawj 1082 \newcommand*\nawj{\kerno,1em\relax}% to put between parentheses and let-
         ters with lower … such as j or y in certain fonts.
```

The original `\pauza` of polski has the skips rigid (one is even a kern). It begins with `\ifhmode` to be usable also at the beginning of a line as the mark of a dialogue.

```
1083 \ifdefined\XeTeXversion
1084 \AtBeginDocument{% to be independent of moment of loading of polski.
\-  1085   \DeclareRobustCommand*\-{%
1086     \ifhmode
1087       \unskip\penalty10000
1088       \afterfi{%
1089         \@ifnextspace{\hskipo.2em␣pluso.1em\relax
\pauzacore 1090           \pauzacore\hskip.2em␣pluso.1em\relax\ignorespaces}%
1091         {\pauzacore\penalty\hyphenpenalty\hskip\z@}}%
1092       \else
```

According to *Instrukcja technologiczna. Skład ręczny i maszynowy* the dialogue dash should be followed by a rigid hskip of ½ em.

```
1093       \leavevmode\pauzacore\penalty10000\hskipo,5em\ignorespaces
1094     \fi}%
```

The next command's name consists of letters and therefore it eats any spaces following it, so `\@ifnextspace` would always be false.

```
\pauza 1095   \DeclareRobustCommand*\pauza{%
1096     \ifhmode
1097       \unskip\penalty10000
1098       \hskipo.2em␣pluso.1em\relax
1099       \pauzacore\hskip.2em␣pluso.1em\relax\ignorespaces%
1100     \else
```

According to *Instrukcja technologiczna. Skład ręczny i maszynowy* the dialogue dash should be followed by a rigid hskip of ½ em.

```
1101       \leavevmode\pauzacore\penalty10000\hskipo,5em\ignorespaces
1102     \fi}%
```

And a version with no space at the left, to begin a `\noindent` paragraph or a dialogue in quotation marks:

```
\lpauza 1103   \DeclareRobustCommand*\lpauza{%
1104     \pauzacore\hskip.2em␣pluso.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…

```
\-  \DeclareRobustCommand*\-{%
      \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{\unskip\penalty10000\hskipo.2em␣pluso.1em%
              \relax
            -\hskip.2em␣pluso.1em\ignorespaces}{\unskip%
                \discretionary{-}{-}{-}}}%
      \fi}%
\ppauza  \DeclareRobustCommand*\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{-}{-}{-}%
      \fi}%
\emdash  \def\emdash{\char`\—}
}% of at begin document
\longpauza \def\longpauza{\def\pauzacore{—}}
\pauzacore \longpauza
\shortpauza \def\shortpauza{%
\pauzacore   \def\pauzacore{-\kern.23em\relax\llap{-}}}
\fi% of if XeTeX.
```

If you have all the three dashes on your keyboard (as I do), you may want to use them for short instead of \pauza, \ppauza and \dywiz. The shortest dash is defined to be smart in math mode and result with −.

```
\ifdefined\XeTeXversion
\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␣\let—\—%
             \catcode`-\active␣\let-\-%
}}}
\else
\relaxen\adashes
\fi
```

The hyphen shouldn't be active IMO because it's used in TEX control such as \hskip-2pt. Therefore we provide the \ahyphen declaration reluctanly, 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 \def\gmu@dywiz{\ifmmode-\else
             \ifvmode-\else\afterfifi\dywiz\fi\fi}%
```

```
\foone{\catcode`-\active}{%
\ahyphen   \def\ahyphen{\let-\gmu@dywiz\catcode`\-\active}}
```

To get current time. Works in ε-TEXs, icluding XETEX.

```
\czas   \newcommand*\czas[1][.]{%
          \the\numexpr(\time-30)/60\relax#1%
          \@tempcnta=\numexpr\time-(\time-30)/60*60\relax
          \ifnum\@tempcnta<10␣0\fi\the\@tempcnta}
```

To push the stuff up to the header and have the after heading skip after the stuff

```
\przeniesvskip   \long\def\przeniesvskip#1{%
                   \edef\gmu@LastSkip{\the\lastskip}%
                   \vskip-\gmu@LastSkip\relax
                   \vspace*{osp}%
                   #1\vskip\gmu@LastSkip\relax}
\textbullet   \@ifXeTeX{\chardef\textbullet="2022␣}{\def\textbullet{$\bullet$}}
tytulowa   \newenvironment*{tytulowa}{\newpage}{\par\thispagestyle{empty}%
               \newpage}
```

Nazwisko na stronę redakcyjną

```
\nazwired   \def\nazwired{\quad\textsc}
```

Settings for mathematics in main font

I used this terrible macros while typesetting E. Szarzyński's *Letters* in 2008.

```
\gmath   \def\gmath{%
           \def\do##1{\edef##1{{\@nx\mathit{\@xa\@gobble\string##1}}}}%
           \do\A␣\do\a␣\do\B␣\do\b␣\do\c␣\do\C\do\d␣\do\D␣\do\e␣\do\E\do\f
           \do\F\do\g\do\G␣\do\i\do\I␣\do\j\do\J␣\do\k\do\K␣\do\l␣\do\L␣%
               \do\m
           \do\M␣\do\n␣\do\N␣\do\P␣\do\p␣\do\q␣\do\Q␣\do\R␣\do\r
           \let\sectionsign\S␣\do\S␣\do\s␣\do\T␣\do\t␣\do\u␣\do\U␣\do\v%
               \do\V
           \do\w␣\do\W␣\do\x␣\do\X␣\do\Y␣\do\y␣\do\z\do\Z
           \def\do##1{\edef##1{{\@nx\mathrm{\@xa\@gobble\string##1}}}}%
           \do\0\do\1\do\2\do\3\do\4\do\5\do\6\do\7\do\8\do\9%
           \relaxen\do
           \newcommand*\do[4][\mathit]{\def##2{##3{##1{\char"##4}}}}%
           \do\alpha{}{03B1}%
           \do[\mathrm]\Delta{}{0394}%
           \do\varepsilon{}{03B5}%
           \do\vartheta{}{03D1}%
           \do\nu{}{03BD}%
           \do\pi{}{03C0}%
           \do\phi{}{03D5}%
           \do[\mathrm]\Phi{}{0424}%
           \do\sigma{}{03C3}%
           \do\varsigma{}{03DA}%
           \do\psi{}{03C8}%
           \do\omega{}{03C9}%
           \do\infty{}{221E}%
           \do[\mathrm]\neg{\mathbin}{00AC}%
```

```
  \do[\mathrm]\neq{\mathrel}{2260}%
  \do\partial{}{2202}%
  \do[\mathrm]\pm{}{00B1}%
  \do[\mathrm]\pm{\mathbin}{00B1}%
  \do[\mathrm]\sim{\mathrel}{007E}%
  \def\do##1##2##3{\def##1{%
\mathop      \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}}%
  \do\sum{\char"2211}{}%
  \do\forall{\gma@quantifierhook␣\rotatebox[origin=c]{180}{A}%
    \setboxo=\hbox{A}\setbox2=\hbox{\scriptsize␣x}%
    \kern\dimexpr\ht2/3*2␣-\wdo/2\relax}{\nolimits}%
  \do\exists{\rotatebox[origin=c]{180}{\gma@quantifierhook␣E}}%
       \nolimits%
  \def\do##1##2##3{\def##1{##3{%
\mathchoice       \mathchoice{\hbox{\rm##2}}{\hbox{\rm##2}}%
             {\hbox{\rm\scriptsize##2}}{\hbox{\rm\tiny##2}}}}}%
    \do\vee{\rotatebox[origin=c]{90}{<}}\mathbin
    \do\wedge{\rotatebox[origin=c]{-90}{<}}\mathbin
    \do\leftarrow{\char"2190}\mathrel
    \do\rightarrow{\char"2192}\mathrel
    \do\leftrightarrow{\char"2190\kern-0,1em␣\char"2192}\mathrel
   \def\do##1##2##3{%
     \catcode`##1=12\relax
     \scantokens{\mathcode`##1="8000\relax
       \foone{\catcode`##1=\active}{\def##1}{##3{%
\mathchoice        \mathchoice{\hbox{\rm##2}}{\hbox{\rm##2}}%
             {\hbox{\rm\scriptsize##2}}{\hbox{\rm\tiny##2}}}}%
       \ignorespaces}}% to eat the lineend (scantokens acts as \read icluding
                        line end.
   \do..\mathpunct␣␣\do,,\mathpunct␣␣\do……\mathpunct
   \do((\mathopen
   \@ifundefined{resetMathstrut@}{}{% an error occured 'bad mathchar etc.'
        because amsmath.sty doesn't take account of a possibility of ( ) being math-
        active.
\resetMathstrut@     \def\resetMathstrut@{%
       \setbox\z@\hbox{%
%% \mathchardef\@tempa\mathcode`\(\relax%% \def\@tempb##1"##2##3{%
\the\textfont"##3\char"}%%% \expandafter\@tempb\meaning\@tempa \relax
         (}%
       \ht\Mathstrutbox@\ht\z@␣\dp\Mathstrutbox@\dp\z@
     }}%
```

```
    \do))\mathclose
    \do[[\mathopen\do]]\mathclose
    \do-{\char"2212}\mathbin␣␣\do++\mathbin␣␣\do==\mathrel␣␣\do××%
         \mathbin
    \do::\mathbin␣␣\do··\mathbin␣␣\do//\mathbin␣\do<<\mathrel
    \do>>\mathrel
    \def\do##1##2##3{\def##1####1{##2{\hbox{%
            \rm
            \setboxo=\hbox{####1}%
            \edef\gma@tempa{\the\hto}%
            \edef\gma@tempb{\the\dpo}%
            ##3%
            \setboxo=\hbox{####1}%
            \lower\dimexpr(\hto␣+␣\dpo)/2-\dpo␣-((\gma@tempa+%
                 \gma@tempb)/2-\gma@tempb)␣%
            \boxo}}}}%
  \do\bigl\mathopen\larger
  \do\bigr\mathclose\larger
  \do\Bigl\mathopen\largerr
  \do\Bigr\mathclose\largerr
  \do\biggl\mathopen{\larger[3]}%
  \do\biggr\mathclose{\larger[3]}%
  \do\Biggl\mathopen{\larger[4]}%
  \do\Biggr\mathclose{\larger[4]}%
  \def\do##1##2{\def##1{\ifmmode##2{\mathchoice
        {\hbox{\rm\char`##1}}{\hbox{\rm\char`##1}}%
        {\hbox{\rm\scriptsize\char`##1}}{\hbox{\rm\tiny%
             \char`##1}}}%
    \else\char`##1\fi}}%
\StoreMacros{\{\}}%
  \do\{\mathopen
  \do\}\mathclose
  \def\={\mathbin{=}}%
\neqb   \def\neqb{\mathbin{\neq}}%
  \def\do##1{\edef\gma@tempa{%
\@xa        \def\@xa\@nx\csname␣\@xa\gobble\string##1r\endcsname{%
          \@nx\mathrel{\@nx##1}}}%
    \gma@tempa}%
  \do\vee␣\do\wedge␣\do\neg
\fakern   \def\fakern{\mkern-3mu}%
  \thickmuskip=8mu␣plus␣4mu\relax
  \gma@gmathhook
}% of def gmath
\emptify\gma@quantifierhook
\quantifierhook \def\quantifierhook#1{%
\gma@quantifierhook   \def\gma@quantifierhook{#1}}
\emptify\gma@gmathhook
\gmathhook \def\gmathhook#1{\addtomacro\gma@gmathhook{#1}}
\gma@dollar \def\gma@dollar$#1${{\gmath$#1$}}%
\gma@bare \def\gma@bare#1{\gma@dollar$#1$}%
\gma@checkbracket \def\gma@checkbracket{\@ifnextchar\[%
```

```
  \gma@bracket\gma@bare}
\gma@bracket \def\gma@bracket\[#1\]{{\gmath\[#1\]}\@ifnextchar\par{}{%
       \noindent}}
\gma \def\gma{\@ifnextchar$%
  \gma@dollar\gma@checkbracket}
\garamath \def\garamath{%
  \quantifierhook{\addfontfeature{OpticalSize=800}}%
\gma@arrowdash   \def\gma@arrowdash{{%
      \setboxo=\hbox{\char"2192}\copyo\kern-0,6\wdo
      \bgcolor\rule[-\dpo]{0,6\wdo}{\dimexpr\hto+\dpo}\kern-0,6%
           \wdo}}%
\gma@gmathhook   \def\gma@gmathhook{%
    \def\do####1####2####3{\def####1{####3{%
\mathchoice          \mathchoice{\hbox{\rm####2}}{\hbox{\rm####2}}%
          {\hbox{\rm\scriptsize####2}}{\hbox{\rm\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{\setboxo=\hbox{\gma@quantifierhook%
         \addfontfeature{Scale=1.272727}o}%
      \setbox2=\hbox{\char"2044}%
      \copyo\kern-0,5\wdo\kern-0,5\wd2\lowero,125\wdo\copy2
      \kerno,5\wdo\kern-0,5\wd2}{}%
    \do\leftarrow{\char"2190\kern-0,05em\gma@arrowdash}\mathrel
    \do\rightarrow{\gma@arrowdash\kern-0,05em\char"2192}\mathrel
    \do\in{\gma@quantifierhook\char"0454}\mathbin
  }}
```

Typesetting dates in my memoirs

A date in the YYYY-MM-DD format we'll transform into DD mmmm YYYY format or we'll just typeset next two tokens/{...} if the arguments' string begins with --. The latter option is provided to preserve compatibility with already used macros and to avoid a starred version of \thedate and the same time to be able to turn \datef off in some cases (for SevSev04.tex).

```
\polskadata \newcommand*\polskadata{%
    \datef \def\datef##1-##2-##3##4{%
      \if\relax##2\relax##3##4%
      \else
      \ifnum##3##4=0\relax
      \else
        \ifnum##3=0\relax
        \else##3%
        \fi##4%
      \fi
      \ifcase##2\relax\or\ stycznia\or\ lutego%
        \or\ marca\or\ kwietnia\or\ maja\or\ czerwca\or\ lipca\or%
             \ sierpnia%
        \or\ września\or\ października\or\ listopada\or\ grudnia%
             \else
        {}%
```

```
    \fi
    \if\relax##1\relax\else\ \fi␣##1%
  \fi}%
\def\datefsl##1/##2/##3##4{%
  \if\relax##2\relax##3##4%
  \else
    \ifnum##3##4=0\relax
    \else
      \ifnum##3=0\relax
      \else##3%
      \fi##4%
    \fi
    \ifcase##2\relax\or\ stycznia\or\ lutego%
      \or\ marca\or\ kwietnia\or\ maja\or\ czerwca\or\ lipca\or%
          \ sierpnia%
      \or\ września\or\ października\or\ listopada\or\ grudnia%
          \else
      {}%
    \fi
    \if\relax##1\relax\else\ \fi␣##1%
  \fi}%
}% of \polskadata
\polskadata
```

For documentation in English:

```
\newcommand*\englishdate{%
  \def\datef##1-##2-##3##4{%
    \if\relax##2\relax##3##4%
    \else
      \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
      \ifnum##3##4=0\relax
      \else
        \ %
        \ifnum##3=0\relax
        \else##3%
        \fi##4%
        \ifcase##3##4\relax\or␣st\or␣nd\or␣rd\else␣th\fi
      \fi
      \if\relax##1\relax\else,\ \fi␣##1%
    \fi
  }%
  \def\datefsl##1/##2/##3##4{%
    \if\relax##2\relax##3##4%
    \else
      \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
```

```
      \ifnum##3##4=0\relax
      \else
        \ %
        \ifnum##3=0\relax
        \else##3%
        \fi##4%
        \ifcase##3##4\relax\or␣st\or␣nd\or␣rd\else␣th\fi
      \fi
      \if\relax##1\relax\else,\ \fi␣##1%
    \fi
  }%
}
```

\ifgmu@dash
```
\newif\ifgmu@dash
```

\gmu@ifnodash
```
\def\gmu@ifnodash#1-#2\@@nil{%
  \def\@tempa{#2}%
  \ifx\@tempa\@empty}
```

\gmu@testdash
```
\def\gmu@testdash#1\ifgmu@dash{%
  \gmu@ifnodash#1-\@@nil
    \gmu@dashfalse
  \else
    \gmu@dashtrue
  \fi
  \ifgmu@dash}
```

A word of explanation to the above pair of macros. `\gmu@testdash` sets `\iftrue` the `\ifgmu@dash` switch if the argument contains an explicit `-`. To learn it, an auxiliary `\gmu@ifdash` macro is used that expands to an open (un`\fi`ed) `\ifx` that tests whether the dash put by us is the only one in the argument string. This is done by matching the parameter string that contains a dash: if the investigated sequence contains (another) dash, `#2` of `\gmu@ifdash` becomes the rest of it and the 'guardian' dash put by us so then it's nonempty. Then `#2` is took as the definiens of `\@tempa` so if it was empty, `\@tempa` becomesx equal `\@empty`, otherwise it isx not.

Why don't we use just `\gmu@ifdash`? Because we want to put this test into another `\if...`. A macro that doesn't *mean* `\if...` wouldn't match its `\else` nor its `\fi` while TEX would skip the falsified branch of the external `\if...` and that would result in the 'extra `\else`' or 'extra `\fi`' error.

Therefore we wrap the very test in a macro that according to its result sets an explicit Boolean switch and write this switch right after the testing macro. (Delimiting `\gmu@testdash`'es parameter with this switch is intended to bind the two which are not one because of TEXnical reasons only.

Warning: this pair of macros may result in 'extra `\else`/extra `\fi`' errors however, if `\gmu@testdash` was `\expandaftered`.

Dates for memoirs to be able to typeset them also as diaries.

\ifdate
```
\newif\ifdate
%\newcounter{dateinsection}[section]
```

\data
```
\newcommand*{\data}[1]{%
  \ifdate\gmu@testdash#1\ifgmu@dash\datef#1\else\datefsl#1\fi\fi}
```

\linedate
```
\newcommand*{\linedate}[1]{\par\ifdate\addvspace{\dateskip}%
  \date@line{\footnotesize\itshape␣\date@biway{#1}}%
  \nopagebreak\else%%\ifnum\arabic{dateinsection}>0\dekbigskip\fi
```

```
    \addvspace{\bigskipamount}%
    \fi}% end of \linedate.
1391 \let\dateskip\medskipamount
\date@biway 1392 \def\date@biway#1{%
1393   \gmu@testdash#1\ifgmu@dash\datef#1\else\datefsl#1\fi}
    \rdate 1394 \newcommand*\rdate[1]{\let\date@line\rightline␣\linedate{#1}}
    \ldate 1395 \newcommand*\ldate[1]{\let\date@line\leftline␣\linedate{#1}}
\runindate 1396 \newcommand*\runindate[1]{%
1397   \paragraph{\footnotesize\itshape␣\datef#1\@@nil}\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.

```
1398 \let\thedate\ldate
  \zwrobcy 1399 \DeclareRobustCommand*\zwrobcy[1]{\emph{#1}}␣% ostinato, allegro con moto,
         garden party etc., także kompliment
  \tytul 1400 \DeclareRobustCommand*\tytul[1]{\emph{#1}}
```

Maszynopis w świecie justowanym zrobi delikatną chorągiewkę.

```
maszynopis 1401 \newenvironment{maszynopis}[1][]{#1\ttfamily
1402   \hyphenchar\font=45\relax% to przypisanie jest globalne do fontu.
1403   \@tempskipa=\glueexpr\rightskip+\leftskip\relax
1404   \ifdim\gluestretch\@tempskipa=\z@
1405   \tolerance900
```

sprawdziło się przy tolerancji 900

```
1406   \advance\rightskip␣by\z@␣pluso,5em\relax\fi
1407   \fontdimen3\font=\z@% zabraniamy rozciągania odstępów, ale %  \fontdimen4%
         \font=\z@ dopuszczamy ich skurczenie
1408   \hyphenpenaltyo␣% żeby nie stresować TEXa: w maszynopisie ten wspaniały al-
         gorytm dzielenia akapitu powinien być wyłączony, a każdy wiersz łamany
         na ostatnim dopuszczalnym miejscu przełamania.
1409   \StoreMacro\pauzacore
\pauzacore 1410   \def\pauzacore{-\rlap{\kern-o,3em-}-}%
1411 }{\par}
\justified 1412 \newcommand*\justified{%
1413   \leftskip=1\leftskip% to preserve the natural length and discard stretch and
         shrink.
1414   \rightskip=1\rightskip
1415   \parfillskip=1\parfillskip
1416   \advance\parfillskip␣by␣osp␣plus␣1fil\relax
1417   \let\\\@normalcr}
```

For dati under poems.

```
\wherncore 1418 \newcommand\wherncore[1]{%
1419     \rightline{%
1420     \parbox{o,7666\textwidth}{
1421       \leftskiposp␣plus␣\textwidth
1422       \parfillskiposp\relax
1423       \let\\\linebreak
1424       \footnotesize␣#1}}}
```

```
\whern      \newcommand\whern[1]{%
              \vskip\whernskip
              \wherncore{#1}}
\whernskip  \newskip\whernskip
            \whernskip2\baselineskip␣minus␣2\baselineskip\relax
\whernup    \newcommand\whernup[1]{\par\wherncore{#1}}
```

Minion and Garamond Premier kerning and ligature fixes

„Ws" nie będzie robiło długiego „s", bo źle wygląda przy „W"

```
\Ws   \DeclareRobustCommand*\Ws{W\kern-0,08em\penalty10000\hskip0sp%
        \relax
        s\penalty10000\hskip0sp\relax}
\Wz   \DeclareRobustCommand*\Wz{W\kern-0,05em\penalty10000\hskip0sp%
        \relax␣z}
\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, 48
- `\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`, 8

v0.75
- `\@ifnextac`:
  - added, 6
- `\@ifnextcat`:
  - `\let` for `#1` changed to `\def` to allow things like `\noexpand~` , 5
- `\@ifnextif`:
  - `\let` for `#1` changed to `\def` to allow things like `\noexpand~` , 5

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, 48
- `\freeze@actives`:
  - added, 33

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, 48

v0.78
- General:
  - `\@namelet` renamed to `\n@melet` to solve a conflict with the beamer class. The package contents regrouped, 48

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`, 16

v0.80
- General:
  - CheckSum 1689, 1
- `\hfillneg`:
  - added, 32

v0.81
- `\dekfraccslash`:
  - moved here from pmlectionis.cls, 35
- `\ifSecondClass`:

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.