Grzegorz 'Natror' Murzynowski

The gmutils Package*

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmutils}
     [2006/11/29␣v0.74␣some␣rather␣TeXnical␣macros,␣some␣of␣them␣
          tricky␣(GM)]
```

Contents

Intro 1
Installation 2
Contents of the gmutils.zip Archive . . 2
Compiling of the Documentation . . . 2
\newgif and Other Globals 2
\@ifnextcat 3
\afterfi and Pals 4
Almost an Environment or Redefinition of \begin 4
Improvement of \end 5
From relsize 5
\firstofone and the Queer \catcodes 7
Metasymbols 8
Macros for Printing Macros and Filenames 8
Storing and Restoring the Meanings of CSs 10
Third Person Pronouns 11
To Save Precious Count Registers . 12
Improvements to mwcls Sectioning Commands 12
Compatibilising Standard and mwcls Sectionings 15
Varia 16
Index 19

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.

* This file has version number v0.74 dated 2006/11/29.

Installation

Just put the gmutils.sty somewhere in the texmf/tex/latex branch. Creating a texmf/tex/latex/gm directory may be advisable if you consider using other packages written by me.

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

Contents of the gmutils.zip Archive

The distribution of the gmutils package consists of the following four files.

gmutils.sty
README
gmutilsDoc.tex
gmutilsDoc.pdf

Compiling of the Documentation

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

MakeIndex shell command:

```
makeindex␣-r␣gmutilsDoc
```

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

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

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

`\newgif` and Other Globals

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.

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

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

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

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

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

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

\grefstepcounter

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

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

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

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

\glet

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

`\@ifnextcat`

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

\@ifnextcat

```
\long\def\@ifnextcat#1#2#3{%
  \let\reserved@d=#1%
  \def\reserved@a{#2}%
  \def\reserved@b{#3}%
  \futurelet\@let@token\@ifncat}
```

```
\def\@ifncat{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifncat
  \else
    \ifcat\reserved@d\noexpand\@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}␣\expandafter\gdef\:␣{\futurelet\@let@token\@ifncat}}
```

Note the trick to achieve 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).

`\afterfi` and Pals

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

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

\afterfi

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

One more of that family:

\afterelsefifi

```
\long\def\afterelsefifi#1\else#2\fi#3\fi{\fi\fi#1}
```

. . . and some other:

\afterelsefi
\afterfifi
\afterelseiffifi

```
\long\def\afterelsefi#1\else#2\fi{\fi#1}
\long\def\afterfifi#1\fi#2\fi{\fi\fi#1}
\long\def\afterelseiffifi#1\else#2\if#3\fi#4\fi{\fi#1}
```

Note, if you fancy this smart trick, that some '`else`' cases are covered by proper non-`else` `\after...` macros, e.g., `\afterfielsefi`'s task would be fulfilled by `\afterfifi` and `\afterelsefifi` covers also the '`\afterelsefielsefi`' case.

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

For back compatibility with my earlier works

\bnamegroup

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

And for the ending

\enamegroup

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

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

\old@begin
\begin
\begin*

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

Improvement of `\end`

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

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

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

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

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

From relsize

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

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

\relsize You declare the font size with `\relsize{`⟨*n*⟩`}` where ⟨*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.

\smaller \larger \smallerr \largerr

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

\relsize
```
\DeclareRobustCommand*\relsize[1]{%
  \ifmmode␣\@nomath\relsize\else
    \begingroup
     \@tempcnta␣% assign number representing current font size
       \ifx\@currsize\normalsize␣4\else␣␣␣% funny order is to have most ...
        \ifx\@currsize\small␣3\else␣␣␣␣␣␣␣% ...likely sizes checked first
         \ifx\@currsize\footnotesize␣2\else
          \ifx\@currsize\large␣5\else
           \ifx\@currsize\Large␣6\else
            \ifx\@currsize\LARGE␣7\else
             \ifx\@currsize\scriptsize␣1\else
              \ifx\@currsize\tiny␣0\else
               \ifx\@currsize\huge␣8\else
                \ifx\@currsize\Huge␣9\else
                 4\rs@unknown@warning␣% unknown state: \normalsize as starting point
     \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
```

Change the number by the given increment:

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

watch out for size underflow:

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

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

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

And a handful of shorthands:

\larger \smaller
```
\DeclareRobustCommand*\larger[1][\@ne]{\relsize{+#1}}
\DeclareRobustCommand*\smaller[1][\@ne]{\relsize{-#1}}
```

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

`\firstofone` and the Queer `\catcodes`

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

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

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

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

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

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

Now, let's define such a smart `_` (underscore) which will be usual $\_{}_8$ in the math mode and $\_{}_{12}$ ('other') outside math.

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

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

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

                  \begingroup \@makeother\%
                  \firstofone{\endgroup
\twelvepercent      \def\twelvepercent{%}}

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

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

Metasymbols

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

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

```
\ifx\l@nohyphenation\undefined
  \newlanguage\l@nohyphenation
\fi
```

\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␣\expandafter␣\nfss@text␣\fi
  {%
    \meta@font@select
```

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

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

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

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

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

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

Macros for Printing Macros and Filenames

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

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

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

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

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

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

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

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

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

it allows the spaces in the filenames (and prints them as ␣).

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

```
              \@ifundefined{pk}{%
\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           \@ifundefined{cs}{\DeclareRobustCommand*{\cs}[2][\bslash]{{%
                     \def\-{\discretionary{{\rmfamily-}}{}{}}%
                     \def\{{\char`\{}\def\}{\char`\}}\ttfamily␣#1#2}}}{}
\env          \@ifundefined{env}{\DeclareRobustCommand*{\env}[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{\expandafter\cmd@to@cs\string#1}}
              \def\cmd@to@cs#1#2{\char\number`#2\relax}
```

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

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

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

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

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

\parg

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

But we can have all three in one command.

\arg

```
\AtBeginDocument{%
  \let\math@arg\arg
  \def\arg{\ifmmode\math@arg\else\afterfi
    \@ifnextchar[\@oargsq{\@ifnextchar(\@pargp\marg}\fi}%
}
```

Storing and Restoring the Meanings of CSs

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

\StoreMacro

```
\def\StoreMacro{\bgroup\makeatletter\egStore@Macro}
\long\def\egStore@Macro#1{\egroup\Store@Macro{#1}}
\long\def\Store@Macro#1{%
  \expandafter\let\csname␣/gml/store\string#1\endcsname#1}
```

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

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

\StoreMacros

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

And the inner iterating macro:

```
\long\def\gml@storemacros#1{%
  \def\@tempa{\noexpand#1}% My TEX Guru's trick to deal with \fi and such, i.e.,
             to hide #1 from TEX when it is processing a test's branch without expanding.
  \if\@tempa.% a dot finishes storing.
  \else
    \if\@tempa,% The list this macro is put before may contain commas and that's
               O.K., we just continue the work.
      \afterelsefifi\gml@storemacros
    \else% what is else this shall be stored.
      \gml@StoreCS{#1}% we use a particular CS to may \let it both to the storing
                macro as above and to the restoring one as below.
      \afterfifi\gml@storemacros
    \fi
  \fi}
```

And for the restoring

\RestoreMacro

```
\def\RestoreMacro{\bgroup\makeatletter\egRestore@Macro}
\long\def\egRestore@Macro#1{\egroup\Restore@Macro{#1}}
\long\def\Restore@Macro#1{%
    \expandafter\let\expandafter#1\csname␣/gml/store\string#1%
          \endcsname}
```

\RestoreMacros

```
\long\def\RestoreMacros{\bgroup\makeatletter\Restore@Macros}
\long\def\Restore@Macros#1{\egroup
  \let\gml@StoreCS\Restore@Macro% we direct the core CS towards restoring and
          call the same iterating macro as in line 174.
  \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:

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

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

\StoringAndRelaxingDo

```
\long\def\StoringAndRelaxingDo{%
  \def\do##1{\expandafter\let\csname␣/gml/store\string##1%
          \endcsname##1%
    \let##1\relax}}
```

And here is the counter-definition for restore.

\RestoringDo

```
\long\def\RestoringDo{%
  \def\do##1{%
    \expandafter\let\expandafter##1\csname␣/gml/store\string##1%
          \endcsname}}
```

And to store a cs as explicitly named cs, i.e. to `\let` one csname another:

```
\def\@namelet#1#2{%
  \edef\@tempa{%
    \let\expandafter\noexpand\csname#1\endcsname
    \expandafter\noexpand\csname#2\endcsname}%
  \@tempa}
```

Third Person Pronouns

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

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

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

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

\gm@atppron

```
\newcommand*\gm@atppron[2]{%
```

```
            \stepcounter{gm@PronounGender}% remember \stepcounter is global.
            \ifodd\arabic{gm@PronounGender}#1\else#2\fi}
\heshe    \newcommand*\heshe{\gm@atppron{he}{she}}
\hisher   \newcommand*\hisher{\gm@atppron{his}{her}}
\himher   \newcommand*\himher{\gm@atppron{him}{her}}
\hishers  \newcommand*\hishers{\gm@atppron{his}{hers}}
\HeShe    \newcommand*\HeShe{\gm@atppron{He}{She}}
\HisHer   \newcommand*\HisHer{\gm@atppron{His}{Her}}
\HimHer   \newcommand*\HimHer{\gm@atppron{Him}{Her}}
\HisHers  \newcommand*\HisHers{\gm@atppron{His}{Hers}}
```

To Save Precious Count Registers

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

```
\newcommand*\nummacro[1]{\gdef#1{0}}
\newcommand*\stepnummacro[1]{%
  \@tempcnta=#1\relax
  \advance\@tempcnta␣by1\relax
  \xdef#1{\the\@tempcnta}}% Because of some mysterious reasons explicit \count0
        interferred with page numbering when used in \gmd@evpaddonce in gmdoc.
\newcommand*\addtonummacro[2]{%
  \count0=#1\relax
  \advance\count0by#2\relax
  \xdef#1{\the\count\z@}}
```

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

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

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

Improvements to mwcls Sectioning Commands

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

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

1. pdfTEX cried of multiply defined `\label`s,

[2] A. Berg, *Wozzeck.*

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

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

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

`\@ifnotmw`

```
\long\def\@ifnotmw#1#2{\@ifundefined{mw@sectionxx}{#1}{#2}}
```

```
\@ifnotmw{%
  \StoreMacro\def␣\def\def#14#2{\RestoreMacro\def}}{}
```

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

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

The kernel of MW's sectioning commands:

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

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

```
      \@ifundefined{if@mainmatter}{}{%
            \gm@dontnumbersectionsoutofmainmatter}
```

```
\fi
```

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

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

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

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

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

(End of Experimente with MW sectioning.)

Compatibilising Standard and mwcls Sectionings

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

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

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

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

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

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

```
\@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those
        written with two optionals.
  \def\gm@secini{gm@la}%
  \def\gm@secxx#1#2[#3]#4{%
    \ifx\gm@secstar\@empty
      \@namelet{gm@true@#1mark}{#1mark}% a little trick to allow a special ver-
            sion of the heading just to the running head.
      \@namedef{#1mark}##1{% we redefine \⟨sec⟩mark to gobble its argument and
            to launch the stored true marking command on the appropriate argu-
            ment.
        \csname␣gm@true@#1mark\endcsname{#2}%
        \@namelet{#1mark}{gm@true@#1mark}% after we've done what we wanted
              we restore original \#1mark.
      }%
      \def\gm@secstar{[#3]}% if \gm@secstar is empty, which means the section-
            ing command was written starless, we pass the 'true' sectioning com-
```

\gm@secxx

mand #3 as the optional argument. Otherwise the sectioning command was written with star so the 'true' s.c. takes no optional.

```
    \fi
    \expandafter\expandafter\csname\gm@secini#1\endcsname
    \gm@secstar{#4}}%
}{% we are in mwcls and want to reverse MW's optionals order i.e., if there's just one optional, it should go both to toc and to running head.
  \def\gm@secini{gm@mw}%
  \let\gm@secmarkh\@gobble% in mwcls there's no need to make tricks for special version to running headings.
```

\gm@secxx

```
  \def\gm@secxx#1#2[#3]#4{%
    \expandafter\expandafter\csname\gm@secini#1\endcsname
    \gm@secstar[#2][#3]{#4}}%
}
\def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
\def\gm@secx#1[#2]{%
  \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}% if there's only one optional, we double it not the mandatory argument.
\def\gm@straightensec#1{% the parameter is for the command's name.
  \@ifundefined{#1}{}{% we don't change the ontological status of the command because someone may test it.
    \@namelet{\gm@secini#1}{#1}%
    \@namedef{#1}{%
      \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
        \def\gm@secstar{}\gm@sec{#1}}}}%
}%
\let\do\gm@straightensec
\do{part}\do{chapter}\do{section}\do{subsection}\do{subsubsection}
\@ifnotmw{}{\do{paragraph}}% this 'straightening' of \paragraph with the standard article caused the 'TeX capacity exceeded' error. Anyway, who on Earth wants paragraph titles in toc or running head?
```

Varia

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@\expandafter{#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
```

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

\LaTeXpar

```
\DeclareRobustCommand*{\LaTeXpar}{(L\kern-.36em%
        {\sbox\z@␣T%
         \vbox␣to\ht\z@{\hbox{\check@mathfonts
                              \fontsize\sf@size\z@
                              \math@fontsfalse\selectfont
                              A}%
                        \vss}%
        }%
        \kern-.07em% originally −, 15 em
        )\TeX}
```

\@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]{\expandafter\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\noexpand#1\twelvespace
    \twelvespace
  \else
    \string#1%
  \fi}
```

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

*

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

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

(This definition is repeated in gmverb.)

\gmobeyspaces

```
\begin{catcode}`\␣\active
\gdef\gmobeyspaces{\catcode`\␣\active\let␣\␣}
```

```
\end{catcode}
```

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

\defobeylines

```
\bgroup\catcode`\^^M\active% the comment signs here are crucial.
\firstofone{\egroup%
\newcommand*\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 \undeksmallskip \dekmedskip \dekbigskip

```
\def\deksmallskip{\vskip\smallskipamount}
\def\undeksmallskip{\vskip-\smallskipamount}
\def\dekmedskip{\vskip\medskipamount}
\def\dekbigskip{\vskip\bigskipamount}
```

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

\@firstofmany

```
\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 \@starttoc

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

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

\enoughpage

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

The `\dots` didn't come out well. My small investigation revealed a mysterious replacement of the original LaTeX definition of `\textellipsis` with

```
> \textellipsis=macro:
->\PD1-cmd \textellipsis \PD1\textellipsis .
```

So, let's ensure `\dots` are given the proper kerning:

```
\ltxtextellipsis  391 \DeclareTextCommandDefault\ltxtextellipsis{%
                  392   .\kern\fontdimen3\font
                  393   .\kern\fontdimen3\font
                  394   .\kern\fontdimen3\font}
           \dots  395 \DeclareRobustCommand*\dots{%
                  396   \ifmmode\mathellipsis\else\ltxtextellipsis\fi}
                  397 \let\ldots\dots
```

Two shorthands for debugging:

```
        \tOnLine  398 \newcommand*\tOnLine{\typeout{\on@line}}
       \OnAtLine  399 \let\OnAtLine\on@line
```

An equality sign properly spaced:

```
         \equals  400 \newcommand*\equals{${}={}$}
```

And for the LaTeX's pseudo-code statements:

```
        \eequals  401 \newcommand*\eequals{${}=={}$}
```

The job name without extension.

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

```
     \jobnamewoe  403 \def\jobnamewoe{\expandafter\gm@jobn\jobname.\@@nil}%
```
We add the dot to be sure there is one although I'm not sure whether you can TeX a file that has no extrension.

```
                  404 \endinput
```

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.

*, 363
\-, 149
\<...>, 135
\@@nil, 140, 141, 143, 374, 402, 403
\@badend, 55
\@begnamedgroup, 41, 42, 48, 51
\@checkend, 52
\@currenvir, 45, 54
\@currenvline, 46
\@currsize, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69
\@emptify, 347, 348, 349
\@firstofmany, 374
\@gif, 7, 8, 10
\@ifncat, 22, 23, 35
\@ifnextcat, 18
\@ifnotmw, 241, 242, 298, 330
\@namelet, 202, 302, 305, 323
\@nobreakfalse, 384
\@oarg, 156, 157, 158
\@oargsq, 156, 158, 165
\@onlypreamble, 385
\@parg, 159, 160, 161
\@pargp, 159, 161, 165
\@relaxen, 351, 352, 353
\@starttoc, 379
\@topnewpage, 286
\@xifncat, 25, 35

\addto@macro, 332, 336
\addtomacro, 336
\addtonummacro, 224
\afterelsefi, 38, 138
\afterelsefifi, 37, 180
\afterelseiffifi, 40
\afterfi, 36, 138, 164
\afterfifi, 39, 183
\arg, 163, 164
\AtBeginDocument, 162, 228

\begin, 50, 51
\begin*, 51
\bigskipamount, 373
\bnamegroup, 48
\bslash, 108, 148

\c@secnumdepth, 248
\cleardoublepage, 240

\cmd, 153
\cmd@to@cs, 153, 154
\count, 225, 226, 227
\cs, 148, 151, 153

\DeclareRobustCommand*, 56, 84, 85, 86, 87, 88, 89, 121, 144, 147, 148, 151, 232, 337, 363, 395
\DeclareTextCommandDefault, 391
\defobeylines, 369
\dekbigskip, 373
\dekmedskip, 372
\deksmallskip, 370
\dimen, 388, 389, 390
\discre, 136, 139
\discret, 137, 138
\dots, 395, 397

\eequals, 401
\egRestore@Macro, 186, 187
\egStore@Macro, 167, 168
\emptify, 348, 348
\enamegroup, 49
\enoughpage, 386
\ensuremath, 122, 132
\env, 151
\equals, 400
\exii@currenvir, 54, 55

\file, 145

\g@emptify, 349, 350
\g@relaxen, 353, 354
\gaddtomacro, 331
\gemptify, 350, 350
\glet, 17
\gm@atppron, 208, 211, 212, 213, 214, 215, 216, 217, 218
\gm@clearpagesduetoopenright, 239, 257
\gm@dontnumbersectionsoutofmainmatter, 237, 249
\gm@hyperrefstepcounter, 231, 234, 268
\gm@jobn, 402, 403
gm@PronounGender, 207
\gm@pswords, 140, 141, 143
\gm@sec, 318, 325, 326
\gm@secini, 299, 309, 312, 315, 323
\gm@secmarkh, 313
\gm@secstar, 301, 307, 310, 316, 325, 326
\gm@secx, 318, 319
\gm@secxx, 300, 314, 320
\gm@straightensec, 321, 328
\gm@targetheading, 232, 235
\gml@StoreCS, 173, 182, 192
\gml@storemacros, 174, 175, 180, 183, 193
\gmobeyspaces, 365
\gmshowlists, 355
\gobble, 91
\gobbletwo, 92
\grefstepcounter, 15
\grelaxen, 354, 354

\HeadingNumber, 265, 267
\HeadingNumberedfalse, 238, 248
\HeadingRHeadText, 251
\HeadingText, 253
\HeadingTOCText, 252
\HeShe, 215
\heshe, 211
\HimHer, 217
\himher, 213
\HisHer, 216
\hisher, 212
\HisHers, 218
\hishers, 214
\hyphenpenalty, 143

\if@filesw, 381
\if@mainmatter, 238
\if@openright, 240
\if@specialpage, 256
\if@twoside, 279
\ifHeadingNumbered, 247, 263
\ifodd, 210

\jobnamewoe, 403

\l@nohyphenation, 118, 119, 129
\larger, *p. 6*, 84
\largerr, *p. 6*, 88
\LaTeXpar, 337
\ldots, 397
\ltxtextellipsis, 391, 396

\marg, 155, 165
\math@arg, 163, 164
\mathellipsis, 396
\medmuskip, 138
\meta, 121, 135, 155, 157, 160
\meta@font@select, 125, 134
\meta@hyphen@restore, 126, 131
\mskip, 138
\mw@HeadingBreakAfter, 258, 275, 290, 294
\mw@HeadingBreakBefore, 255
\mw@HeadingLevel, 245, 248
\mw@HeadingRunIn, 270
\mw@HeadingType, 254
\mw@HeadingWholeWidth, 273
\mw@normalheading, 277, 286, 289, 293
\mw@runinheading, 271
\mw@sectionxx, 244

\nameshow, 356
\newcounter, 207, 229
\newgif, 4
\newlanguage, 119
\newwrite, 381
\nfss@text, 123
\nummacro, 219

\oarg, 156
\old@begin, 50, 51
\OnAtLine, 399

\PackageWarning, 80, 82
\pagebreak, 278, 290, 294
\pagegoal, 388
\pagetotal, 389
\parg, 159
\pk, 147
\possfil, 152
\printspaces, 140, 144

\relaxen, 352, 352
\relsize, *p. 6*, 56, 57, 84, 85, 86, 87, 88, 89
\RequirePackage, 378
\Restore@Macro, 187, 188, 192, 195
\Restore@Macros, 190, 191
\RestoreMacro, 186, 243
\RestoreMacros, 190
\RestoringDo, 199
\rs@size@warning, 73, 78, 80
\rs@unknown@warning, 70, 82

\sfname, 144, 145
\showboxbreadth, 355
\showboxdepth, 355
\showlists, 355
\smaller, *p. 6*, 85
\smallerr, *p. 6*, 89
\smallskipamount, 370, 371
\smartunder, 101
\stepnummacro, 220
\Store@Macro, 168, 169, 173
\Store@Macros, 171, 172
\Stored@Macro, 194, 195
\StoredMacro, 194
\StoreMacro, 167, 243

\StoreMacros, 171
\StoringAndRelaxingDo, 196
\subs, 95, 103

\textlarger, 86
\textsmaller, 87
\textvisiblespace, 139
\TODO, 375
\tOnLine, 398
\twelveand, 114
\twelvebackslash, 107, 108
\twelvepercent, 111
\twelvespace, 117, 358, 359
\twelveunder, 98
\twocoltoc, 377, 385

\undeksmallskip, 371

\vs, 139, 140, 143

\xiistring, 357