

Grzegorz Murzynowski

The gmutils Package*

Written by Grzegorz Murzynowski,
natror at o2 dot pl

©2005, 2006, 2007 by Grzegorz Murzynowski.

This program is subject to the L^AT_EX Project Public License.

See <http://www.ctan.org/tex-archive/help/Catalogue/licenses.lpp1.html> for the details of that license.

LPPL status: "author-maintained".

Many thanks to my T_EX Guru Marcin Woliński for his T_EXnical support.

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{gmutils}
3 [2007/04/28_v0.80_some_rather_TeXnical_macros,_some_of_them_
   tricky_(GM)]
```

Contents

Intro	1	Storing and Restoring the	
Installation	2	Meanings of CSs	12
Contents of the gmutils.zip Archive	2	Not only preamble!	13
Compiling of the Documentation	2	Third Person Pronouns	14
A couple of abbreviations	2	To Save Precious Count Registers	14
\@ifnextcat, \@ifnextac	4	Improvements to mwcls Sectioning	
\afterfi and Pals	5	Commands	15
Almost an Environment or		Compatibilising Standard and	
Redefinition of \begin	6	mwcls Sectionings	17
Improvement of \end	7	enumerate* and itemize*	19
From relsize	7	The Logos	20
\firstofone and the Queer \catcodes	8	Expanding turning stuff all into	
Metasymbols	9	'other'	21
Macros for Printing Macros and		Varia	22
Filenames	10	Change History	25
		Index	25

Intro

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

* This file has version number v0.80 dated 2007/04/28.

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 (L)A_TE_X-awareness of the user.

For details just read the code part.

Installation

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

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

Contents of the gmutils.zip Archive

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

```
gmutils.sty
README
gmutilsDoc.tex
gmutilsDoc.pdf
```

Compiling of the Documentation

The last of the above files (the `.pdf`, i.e., *this file*) is a documentation compiled from the `.sty` file by running L_AT_EX on the `gmutilsDoc.tex` file twice, then MakeIndex on the `gmutils.idx` file, and then L_AT_EX 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 `gm-docc.cls`), `gmverb.sty`, `gmutils.sty`, `gmiflink.sty` and also some standard packages: `hyper-ref.sty`, `color.sty`, `geometry.sty`, `multicol.sty`, `lmodern.sty`, `fontenc.sty` that should be installed on your computer by default.

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

A couple of abbreviations

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

The `\newgif` declaration's effect is used even in the L_AT_EX 2_ε source by redefining some particular user defined ifs (UD-ifs henceforth) step by step. The goal is to make the UD-if's assignment global. I needed it at least twice during `gmdoc` writing so I make it a macro. It's an almost verbatim copy of L_AT_EX's `\newif` modulo the letter *g* and the `\global` prefix. (File `d:ltdefns.dtx` Date: 2004/02/20 Version v1.3g, lines 139–150)

```
\newgif 6 \def\newgif#1{%
```

```

7   {\escapechar\m@ne
8     \global\let#1\iffalse
9     \@gif#1\iftrue
10    \@gif#1\iffalse
11   }}

```

‘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.

```

12 \def\@gif#1#2{%
13   \@xa\gdef\csname\@xa\@gobbletwo\string#1%
14   g% the letter g for ‘\global’.
15   \@xa\@gobbletwo\string#2\endcsname
16   {\global\let#1#2}}

```

After `\newgif\ifffoo` you may type `{\foogtrue}` and the `\ifffoo` 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 17 \newcommand*\grefstepcounter[1]{%
18   {\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 19 \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 20 \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 21 \long\def\addto@macro#1#2{%
22   \toks@\@xa{#1#2}%
23   \edef#1{\the\toks@}%
24   }% (\toks@ is a scratch register, namely \toks0.)

```

And for use in the very document,

```

\addtomacro 25 \let\addtomacro=\addto@macro

```

```

\@emptyify 26 \newcommand*\@emptyify[1]{\let#1=\@empty}
\emptyify 27 \@ifdefinable\emptyify{\let\emptyify\@emptyify}

```

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

```

\g@emptyify 28 \newcommand*\g@emptyify{\global\@emptyify}
\gemptyify 29 \@ifdefinable\gemptyify{\let\gemptyify\g@emptyify}

```

```

\@relaxen 30 \newcommand*\@relaxen[1]{\let#1=\relax}
\relaxen 31 \@ifdefinable\relaxen{\let\relaxen\@relaxen}

```

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

```

\g@relaxen 32 \newcommand*\g@relaxen{\global\@relaxen}
\grelaxen 33 \@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 34 \def\gmshowlists{\showboxdepth=1000\showboxbreadth=1000\showlists}
\nameshow 35 \newcommand*\nameshow[1]{\@xa\show\cscname#1\endcscname}

```

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 36 \def\xiistring#1{%
37   \if\@nx#1\twelvespace
38     \twelvespace
39   \else
40     \string#1%
41   \fi}

```

`\@ifnextcat`, `\@ifnextac`

As you guess, we `\def \@ifnextcat` à la `\@ifnextchar`, see L^AT_EX 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 T_EXbook, 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 42 \long\def\@ifnextcat#1#2#3{%
43   \def\reserved@d{#1}%
44   \def\reserved@a{#2}%
45   \def\reserved@b{#3}%
46   \futurelet\@let@token\@ifncat}
47 \def\@ifncat{%
48   \ifx\@let@token\@sptoken
49     \let\reserved@c\@xifncat
50   \else
51     \ifcat\reserved@d\@nx\@let@token

```

```

52     \let\reserved@c\reserved@a
53     \else
54     \let\reserved@c\reserved@b
55     \fi
56 \fi
57 \reserved@c}

58 {\def\:{\let\sptoken=_}\:_% this makes \sptoken a space token.
59 \def\:{\@xifncat}\_@xa\gdef\:{\futurelet\@let@token\@ifncat}}

```

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

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

```

\@ifnextcat 60 \long\def\@ifnextac#1#2{%
61   \@ifnextcat\bgroup{#2}{\gm@ifnac{#1}{#2}}}

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

```

Yes, it won't work for an active char \let to {, 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 =.

```

64 \let\@tempa\*%
65 \def\*{%
66   \let\*\@tempa
67   \let\gm@letspace=_}%
68 \*_%

\@ifnextspace 69 \def\@ifnextspace#1#2{%
70   \let\@reserveda\*%
71   \def\*{%
72     \let\*\@reserveda
73     \ifx\@let@token\gm@letspace\afterfi{#1}%
74     \else\afterfi{#2}%
75     \fi}%
76   \futurelet\@let@token\*}

```

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

\afterfi and Pals

It happens from time to time that you have some sequence of macros in an \if... and you would like to expand \fi before expanding them (e.g., when the macros should take some tokens next to \fi... as their arguments. If you know how many macros are there, you may type a couple of \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 T_EX 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 $\wedge M_{12}$. To avoid pollution write

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

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

And two more of that family:

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

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

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

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

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

```
82 \long\def\afterfiffiffifi#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 L^AT_EX's `\begin`:

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

```
\@begnamedgroup 83 \@ifdefinable\@begnamedgroup{\relax}
84 \def\@begnamedgroup#1{%
85   \@ignorefalse% not to ignore blanks after group
86   \begingroup\@endpefalse
87   \def\@currenvir{#1}%
88   \edef\@currenvline{\on@line}%
89   \csname_#1\endcsname}% if the argument is a command's name (an environment's
    e.g.), this command will now be executed. (If the corresponding control
    sequence hasn't been known to TEX, this line will act as \relax.)
```

For back compatibility with my earlier works

```
\bnamegroup 90 \let\bnamegroup\@begnamedgroup
And for the ending
\enamegroup 91 \def\enamegroup#1{\end{#1}}
And we make it the starred version of \begin.
\old@begin 92 \let\old@begin\begin
\begin 93 \def\begin{\@ifstar{\@begnamedgroup}{\old@begin}}
\begin*
```

Improvement of \end

It's very clever and useful that `\end` checks whether its argument is `ifx`-equivalent `@currentenv`. 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'.

```
94 \def\@checkend#1{%
95   \edef\reserved@a{\@xa\string\csname#1\endcsname}%
96   \edef\exii@currentenv{\@xa\string\csname\@currentenv\endcsname}%
97   \ifx\reserved@a\exii@currentenv\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, L^AT_EX_{2 ϵ} version of these macros was written by Donald Arseneau asnd@triumf.ca and Matt Swift swift@bu.edu after the L^AT_EX 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
\smaller      to \LARGE size. Negative n selects smaller fonts. \smaller == \relsize{-1};
\larger       \larger == \relsize{1}. \smallerr(my addition) == \relsize{-2}; \largerr
\smallerr     guess yourself.
\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 98 \DeclareRobustCommand*\relsize[1]{%
99   \ifmmode\@nomath\relsize\else
100   \begingroup
101   \@tempcnta\@ assign number representing current font size
102   \ifx\@currsz\normalsize\@four\else\@four\@funny order is to have most ...
```

```

103      \ifx\@currsiz3\else\small\else\large\else\huge\else\Huge\else
104      \ifx\@currsiz\footnotesize\else
105      \ifx\@currsiz\large\else
106      \ifx\@currsiz\Large\else
107      \ifx\@currsiz\LARGE\else
108      \ifx\@currsiz\scriptsize\else
109      \ifx\@currsiz\tiny\else
110      \ifx\@currsiz\huge\else
111      \ifx\@currsiz\Huge\else
112      4\rs@unknown@warning\% unknown state: \normalsize as start-
        ing point
113      \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
        Change the number by the given increment:
114      \advance\@tempcnta#1\relax
        watch out for size underflow:
115      \ifnum\@tempcnta<\z@\rs@size@warning{small}{\string\tiny}%
        \@tempcnta\z@\fi
116      \xa\endgroup
117      \ifcase\@tempcnta% set new size based on altered number
118      \tiny\or\scriptsize\or\footnotesize\or\small\or\%
        \normalsize\or
119      \large\or\Large\or\LARGE\or\huge\or\Huge\else
120      \rs@size@warning{large}{\string\Huge}\Huge
121      \fi\fi}% end of \relsize.
\rs@size@warning 122 \providecommand*\rs@size@warning[2]{\PackageWarning{gmutils}
        (relsize)}{%
123      Size requested is too #1. \MessageBreak Using #2 instead}}
\rs@unknown@warning 124 \providecommand*\rs@unknown@warning{\PackageWarning{gmutils}
        (relsize)}{\Currentfontsize
125      is unknown! (Why?!?) \MessageBreak Assuming \string\normalsize}}
        And a handful of shorthands:
        \larger 126 \DeclareRobustCommand*\larger[1][\@ne]{\relsize{+ #1}}
        \smaller 127 \DeclareRobustCommand*\smaller[1][\@ne]{\relsize{- #1}}
        \textlarger 128 \DeclareRobustCommand*\textlarger[2][\@ne]{\relsize{+ #1} #2}}
        \textsmaller 129 \DeclareRobustCommand*\textsmaller[2][\@ne]{\relsize{- #1} #2}}
        \largerr 130 \DeclareRobustCommand*\largerr{\relsize{+ 2}}
        \smallerr 131 \DeclareRobustCommand*\smallerr{\relsize{- 2}}

```

\firstofone and the Queer \catcodes

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

```
\firstofone 132 \long\def\firstofone#1{#1}
```

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


```

\gobble 133 \long\def\gobble#1{}
\gobbletwo 134 \let\gobbletwo\@gobbletwo
135 \bgroup\catcode'\_ =8\_%
136 \firstofone{\egroup
\subs 137 \let\subs=_}
138 \bgroup\@makeother\_%
139 \firstofone{\egroup
\twelveunder 140 \def\twelveunder{_{}}
Now, let's define such a smart _ (underscore) which will be usual _8 in the math mode
and _12 ('other') outside math.
141 \bgroup\catcode'\_ =\active
142 \firstofone{\egroup
\smartunder 143 \newcommand*\smartunder{%
144 \catcode'\_ =\active
145 \def_{\ifmmode\subs\else\_ \fi}}}% We define it as \_ not just as \twelveunder
because some font encodings don't have _ at the \char'\_ position.
146 \begingroup\catcode'\!=0
147 \@makeother\!
148 !firstofone{!endgroup%
\twelvebackslash 149 !newcommand*!\twelvebackslash{\}}
\bslash 150 \@ifundefined{bslash}{\let\bslash=\twelvebackslash}{\}
151 \begingroup \@makeother\%
152 \firstofone{\endgroup
\twelvepercent 153 \def\twelvepercent{%%}}
154 \begingroup\_ \@makeother\&%
155 \firstofone{\endgroup%
\twelveand 156 \def\twelveand{&}}
157 \begingroup\@makeother\_ \_ %
158 \firstofone{\endgroup%
\twelvespace 159 \def\twelvespace{\_ }}

```

Metasymbols

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

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

```

160 \ifx\l@nohyphenation\undefined
161 \newlanguage\l@nohyphenation
162 \fi
\meta 163 \DeclareRobustCommand*\meta[1]{%

```

“Since the old implementation of `\meta` could be used in math we better ensure that this is possible with the new one as well. So we use `\ensuremath` around `\langle`

and `\rangle`. However this is not enough: if `\meta@font@select` below expands to `\itshape` it will fail if used in math mode. For this reason we hide the whole thing inside an `\nfss@text` box in that case.”

```
164 \ensuremath\langle
165 \ifmmode\@xa\nfss@text\fi
166 {%
167 \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.

```
168 \edef\meta@hyphen@restore{%
169 \hyphenchar\the\font\the\hyphenchar\font}%
170 \hyphenchar\font\m@ne
171 \language\l@nohyphenation
172 #1\/%
173 \meta@hyphen@restore
174 }\ensuremath\rangle
175 }
```

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.

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

The below `\meta`’s drag¹ is a version of The `TEXbook`’s one.

```
\<...> 177 \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 178 \def\discre#1#2#3{\kern0sp\discretionary{#1}{#2}{#3}\penalty10000^
\hskip0sp\relax}
\discret 179 \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.

```
180 \def\:{\ifmmode\afterfi{\mskip\medmuskip}\else\afterfi{\discret{}}\fi}
\vs 181 \newcommand*{\vs}{\discre{\textvisiblespace}{\textvisiblespace}}
```

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

```
\printspaces 182 \def\printspaces#1{\let~=\vs\let\=\vs\gm@pswords#1\@@nil}}
183 \def\gm@pswords#1\#2\@@nil{%
184 \if\relax#1\relax\else#1\fi
```

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

```

185 \if\relax#2\relax\else\vs\penalty\hyphenpenalty\gm@pswords#2\@@nil\fi}%
      note that in the recursive call of \gm@pswords the argument string is not
      extended with a guardian space: it has been already by \printspaces.
\sfname 186 \DeclareRobustCommand*\sfname[1]{\textsf{\printspaces{#1}}}
\file 187 \let\file\sfname% it allows the spaces in the filenames (and prints them as ).
      The below macro I use to format the packages' names.
\pk 188 \DeclareRobustCommand*\pk[1]{\textsf{\textup{#1}}}
      Some (if not all) of the below macros are copied from doc and/or ltxdoc.
      A macro for printing control sequences in arguments of a macro. Robust to avoid
      writing an explicit \ into a file. It calls \ttfamily not \tt to be usable in headings
      which are boldface sometimes.
\cs 189 \DeclareRobustCommand*\cs[2][\bslash]{%
190     \def\{-\discretionary{\rmfamily-}{\}{\}}%
191     \def\{\char'\}\def\{\char'\}\ttfamily_\#1#2}}
\env 192 \DeclareRobustCommand*\env[1]{\cs[] {#1}}
      And one for encouraging linebreaks e.g., before long verbatim words.
\possfil 193 \newcommand*\possfil{\hfil\penalty1000\hfilneg}
      The five macros below are taken from the ltxdoc.dtx.
      “\cmd{\foo} Prints \foo verbatim. It may be used inside moving arguments. \cs{^
      foo} also prints \foo, for those who prefer that syntax. (This second form may even be
      used when \foo is \outer).”
\cmd 194 \def\cmd#1{\cs{\@xa\cmd@to@cs\string#1}}
195 \def\cmd@to@cs#1#2{\char\number'#2\relax}
      \marg{text} prints {\text}, ‘mandatory argument’.
\marg 196 \def\marg#1{\ttfamily\char'\meta{#1}\ttfamily\char'\}}
      \oarg{text} prints [\text], ‘optional argument’. Also \oarg[text] does that.
\oarg 197 \def\oarg{\@ifnextchar[\@oargsq\@oarg}
198 \def\@oarg#1{\ttfamily[]\meta{#1}\ttfamily}}
199 \def\@oargsq[#1]{\@oarg{#1}}
      \parg{te,xt} prints (\text), ‘picture mode argument’.
\parg 200 \def\parg{\@ifnextchar(\@pargp\@parg}
201 \def\@parg#1{\ttfamily()\meta{#1}\ttfamily}}
202 \def\@pargp(#1){\@parg{#1}}
      But we can have all three in one command.
203 \AtBeginDocument{%
\arg 204 \let\math@arg\arg
205 \def\arg{\ifmmode\math@arg\else\afterfi{%
206     \@ifnextchar[%
207     \@oargsq{\@ifnextchar(%
208     \@pargp\marg}}\fi}%
209 }

```

Storing and Restoring the Meanings of CSs

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

```
\StoreMacro 210 \def\StoreMacro{%
211   \bgroup\makeatletter\@ifstar\egStore@MacroSt\egStore@Macro}
212 \long\def\egStore@Macro#1{\egroup\Store@Macro{#1}}
213 \long\def\egStore@MacroSt#1{\egroup\Store@MacroSt{#1}}
214 \long\def\Store@Macro#1{%
215   \@xa\let\csname_/gml/store/string#1\endcsname#1}
216 \long\def\Store@MacroSt#1{%
217   \edef\gmu@smtempa{%
218     \@nx\let\@xa\@nx\csname/gml/store#1\endcsname\@xa\@nx\csname#1^
219     \endcsname}
219   \gmu@smtempa}
```

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

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

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

```
\StoreMacros 220 \long\def\StoreMacros{\bgroup\makeatletter\Store@Macros}
221 \long\def\Store@Macros#1{\egroup
222   \let\gml@StoreCS\Store@Macro
223   \gml@storemacros#1.}
```

And the inner iterating macro:

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

And for the restoring

```
\RestoreMacro 235 \def\RestoreMacro{%
236   \bgroup\makeatletter\@ifstar\egRestore@MacroSt\egRestore@Macro}
237 \long\def\egRestore@Macro#1{\egroup\Restore@Macro{#1}}
238 \long\def\egRestore@MacroSt#1{\egroup\Restore@MacroSt{#1}}
239 \long\def\Restore@Macro#1{%
```

```

240      \@xa\let\@xa#1\csname_/gml/store/string#1\endcsname}
241 \long\def\Restore@MacroSt#1{%
242   \edef\gmu@smtempa{%
243     \@nx\let\@xa\@nx\csname#1\endcsname\@xa\@nx\csname/gml/store#1^
        \endcsname}
244   \gmu@smtempa}
\RestoreMacros 245 \long\def\RestoreMacros{\bgroup\makeatletter\Restore@Macros}
246 \long\def\Restore@Macros#1{\egroup
247   \let\gml@storeCS\Restore@Macro% we direct the core CS towards restoring and
        call the same iterating macro as in line 223.
248   \gml@storemacros#1.}

```

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

And to restore *and* use immediately:

```

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

```

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

```

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

```

And here is the counter-definition for restore.

```

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

```

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

```

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

```

Not only preamble!

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

```

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

```

```

265 \not@onlypreamble\@preamblecmds
266 \not@onlypreamble\@ifpackageloaded
267 \not@onlypreamble\@ifclassloaded
268 \not@onlypreamble\@ifl@aded
269 \not@onlypreamble\@pkgextension

```

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

```

270 \def\gm@notprerr{\can be used only in preamble(\on@line)}
271 \AtBeginDocument{%
272   \def\do#1{\@nx\do\@nx#1}%
273   \edef\@preamblecmds{%
274     \def\@nx\do##1{%
275       \def##1{!\@nx\string##1\@nx\gm@notprerr}}%
276   \@preamblecmds}}

```

Third Person Pronouns

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

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

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

```

277 \newcounter{gm@PronounGender}
\gm@atppron 278 \newcommand*\gm@atppron[2]{%
279   \stepcounter{gm@PronounGender}% remember \stepcounter is global.
280   \ifodd\arabic{gm@PronounGender}#1\else#2\fi}

\heshe 281 \newcommand*\heshe{\gm@atppron{he}{she}}
\hisher 282 \newcommand*\hisher{\gm@atppron{his}{her}}
\himher 283 \newcommand*\himher{\gm@atppron{him}{her}}
\hishers 284 \newcommand*\hishers{\gm@atppron{his}{hers}}

\HeShe 285 \newcommand*\HeShe{\gm@atppron{He}{She}}
\HisHer 286 \newcommand*\HisHer{\gm@atppron{His}{Her}}
\HimHer 287 \newcommand*\HimHer{\gm@atppron{Him}{Her}}
\HisHers 288 \newcommand*\HisHers{\gm@atppron{His}{Hers}}

```

To Save Precious Count Registers

It's a contribution to T_EX's ecology ;-). You can use as many CSs as you wish and you may use only 256 count registers (although in eT_EX there are 2¹⁶ count registers, which makes the following a bit obsolete).

```

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

```

```

290 \newcommand*\stepnummacro[1]{%
291   \@tempcnta=#1\relax
292   \advance\@tempcnta_\by1\relax
293   \xdef#1{\the\@tempcnta}}% Because of some mysterious reasons explicit \count0
      interferred with page numbering when used in \gmd@evpaddonce in gmdoc.

294 \newcommand*\addtonummacro[2]{%
295   \count0=#1\relax
296   \advance\count0by#2\relax
297   \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. pdf \TeX 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`.

```

298 \AtBeginDocument{% because we don't know when exactly hyperref is loaded and
      maybe after this package.
299   \@ifpackageloaded{hyperref}{\newcounter{NoNumSecs}%
300     \setcounter{NoNumSecs}{617}% to make \refing to an unnumbered section
      visible (and funny?).
301     \def\gm@hyperrefstepcounter{\refstepcounter{NoNumSecs}}%
302     \DeclareRobustCommand*\gm@targetheading[1]{%
303       \hypertarget{#1}{#1}}}% end of then
304   {\def\gm@hyperrefstepcounter{%
305     \def\gm@targetheading#1{#1}}}% end of else
306 }% of \AtBeginDocument

```

Auxiliary macros for the kernel sectioning macro:

```

307 \def\gm@dontnumbersectionsoutofmainmatter{%
308   \if@mainmatter\else_\HeadingNumberedfalse_\fi}
309 \def\gm@clearpagesduetoopenright{%
310   \if@openright\cleardoublepage\else_\clearpage\fi}

```

² A. Berg, *Wozzeck*.

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

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

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

```
\@ifnotmw 311 \long\def\@ifnotmw#1#2{\@ifundefined{mw@sectionxx}{#1}{#2}}
312 \let\gmu@def\def
313 \@ifnotmw{%
314   \StoreMacro\gmu@def\def\gmu@def#14#2{\RestoreMacro\gmu@def}}{}
```

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

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

The kernel of MW's sectioning commands:

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

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

```
320     \@ifundefined{if@mainmatter}{~}{^
321       \gm@dontnumbersectionsoutofmainmatter}
322   \fi
323   % \ifHeadingNumbered
324   % \refstepcounter{#1}%
325   % \protected@edef\HeadingNumber{\csname the#1\endcsname\relax}%
326   % \else
327   % \let\HeadingNumber\@empty
328   % \fi
329   \def\HeadingRHeadText{#2}%
330   \def\HeadingTOCText{#3}%
331   \def\HeadingText{#4}%
332   \def\mw@HeadingType{#1}%
333   \if\mw@HeadingBreakBefore
334     \if@specialpage\else\thispagestyle{closing}\fi
335     \@ifundefined{if@openright}{~}{\gm@clearpagesduetoopenright}%
336     \if\mw@HeadingBreakAfter
337       \thispagestyle{blank}\else
338       \thispagestyle{opening}\fi
```



```

332     \global\@topnum\z@
333     \fi% of \if\mw@HeadingBreakBefore
placement of \refstep suggested by me (GM)
334     \ifHeadingNumbered
335         \refstepcounter{#1}%
336         \protected@edef\HeadingNumber{\csname\the#1\endcsname\relax}%
337     \else
338         \let\HeadingNumber\@empty
339         \gm@hyperrefstepcounter
340     \fi% of \ifHeadingNumbered
341     \if\mw@HeadingRunIn
342         \mw@runinheading
343     \else
344         \if\mw@HeadingWholeWidth
345             \if@twocolumn
346                 \if\mw@HeadingBreakAfter
347                     \onecolumn
348                     \mw@normalheading
349                     \pagebreak\relax
350                     \if@twoside
351                         \null
352                         \thispagestyle{blank}%
353                         \newpage
354                     \fi% of \if@twoside
355                 \twocolumn
356             \else
357                 \@topnewpage[\mw@normalheading]%
358             \fi% of \if\mw@HeadingBreakAfter
359         \else
360             \mw@normalheading
361             \if\mw@HeadingBreakAfter\pagebreak\relax\fi
362         \fi% of \if@twocolumn
363     \else
364         \mw@normalheading
365         \if\mw@HeadingBreakAfter\pagebreak\relax\fi
366     \fi% of \if\mw@HeadingWholeWidth
367 \fi% of \if\mw@HeadingRunIn
368 }

```

(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.)

```

369 \@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those
      written with two optionals.
370   \def\gm@secini{\gm@la}%
\gm@secxx 371   \def\gm@secxx#1#2[#3]#4{%
372     \ifx\gm@secstar\@empty
373       \n@melet{\gm@true@#1mark}{#1mark}% a little trick to allow a special version
      of the heading just to the running head.
374       \@namedef{#1mark}##1{% we redefine \<sec>mark to gobble its argument and
      to launch the stored true marking command on the appropriate argu-
      ment.
375       \csname_\gm@true@#1mark\endcsname{#2}%
376       \n@melet{#1mark}{\gm@true@#1mark}% after we've done what we wanted
      we restore original \#1mark.
377     }%
378     \def\gm@secstar{[#3]}% if \gm@secstar is empty, which means the section-
      ing command was written starless, we pass the 'true' sectioning com-
      mand #3 as the optional argument. Otherwise the sectioning command
      was written with star so the 'true' s.c. takes no optional.
379     \fi
380     \@xa\@xa\csname\gm@secini#1\endcsname
381     \gm@secstar{#4}%
382 }{% 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.
383   \def\gm@secini{\gm@mw}%
384   \let\gm@secmarkh\@gobble% in mwcls there's no need to make tricks for special
      version to running headings.
\gm@secxx 385   \def\gm@secxx#1#2[#3]#4{%
386     \@xa\@xa\csname\gm@secini#1\endcsname
387     \gm@secstar[#2][#3]{#4}%
388   }
389   \def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
390   \def\gm@secx#1[#2]{%
391     \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}% if there's only
      one optional, we double it not the mandatory argument.

```

```

392 \def\gm@straightensec#1{% the parameter is for the command's name.
393   \@ifundefined{#1}{}{% we don't change the ontological status of the command
        because someone may test it.
394   \n@melet{\gm@secini#1}{#1}%
395   \@namedef{#1}{%
396     \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
397       \def\gm@secstar{}\gm@sec{#1}}}%
398 }%

399 \let\do\gm@straightensec
400 \do{part}\do{chapter}\do{section}\do{subsection}\do{subsubsection}
401 \@ifnotmw{}{\do{paragraph}}% this 'straightening' of \paragraph with the stan-
        dard article caused the 'TeX capacity exceeded' error. Anyway, who on Earth
        wants paragraph titles in toc or running head?

```

enumerate* and itemize*

We wish the starred version of `enumerate` to be just numbered paragraphs. But `hyperref` redefines `\item` so we should do it a smart way, to set the L^AT_EX'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* 402 \@namedef{enumerate*}{%
403   \ifnum\@enumdepth>\thr@@
404     \@toodeep
405   \else
406     \advance\@enumdepth\@ne
407     \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
408     \@xa\list\csname\label\@enumctr\endcsname{%
409       \partopsep\topsep\topsep\z@\leftmargin\z@
410       \itemindent\@parindent\%
411       \labelwidth\@parindent
412       \advance\labelwidth-\labelsep
413       \listparindent\@parindent
414       \usecounter\@enumctr
415       \def\makelabel##1{##1\hfil}}%
416   \fi}
417 \@namedef{endenumerate*}{\endlist}

itemize* 418 \@namedef{itemize*}{%
419   \ifnum\@itemdepth>\thr@@
420     \@toodeep
421   \else
422     \advance\@itemdepth\@ne
423     \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
424     \@xa\list\csname\@itemitem\endcsname{%
425       \partopsep\topsep\topsep\z@\leftmargin\z@
426       \itemindent\@parindent
427       \labelwidth\@parindent
428       \advance\labelwidth-\labelsep

```

```

429     \listparindent\@parindent
430     \def\makelabel##1{##1\hfil\}}%
431 \fi}
432 \@namedef{enditemize*}{\endlist}

```

The Logos

We'll modify The L^AT_EX logo now to make it fit better to various fonts.

```

433 \let\oldLaTeX\LaTeX
434 \let\oldLaTeXe\LaTeXe
435 \def\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
436 \newcommand*{\DeclareLogo[3][\relax]}{

```

#1 is for non-L^AT_EX 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.

```

437 \ifx\relax#1\def\@tempa{\@xa\@gobble\string#2}%
438 \else
439     \def\@tempa{#1}%
440 \fi
441 \edef\@tempa{
442     \@nx\DeclareTextCommand\@nx#2{PD1}{\@tempa}}
443 \@tempa
444 \DeclareTextCommandDefault#2{#3}}
445 \DeclareLogo\LaTeX{
446     {%
447         L%
448         \setbox\z@\hbox{\check@mathfonts
449             \fontsize\sf@size\z@
450             \math@fontsfalse\selectfont
451             A}%
452         \kern-.57\wd\z@
453         \sbox\tw@_T%
454         \vbox_\to\ht\tw@{\copy\z@_\vss}%
455         \kern-.2\wd\z@}% originally —, 15 em for T.
456     {%
457         \ifdim\fontdimen1\font=\z@
458         \else
459             \count\z@=\fontdimen5\font
460             \multiply\count\z@_by_64\relax
461             \divide\count\z@_by_p@
462             \count\tw@=\fontdimen1\font
463             \multiply\count\tw@_by\count\z@
464             \divide\count\tw@_by_64\relax
465             \divide\count\tw@_by\tw@
466             \kern-\the\count\tw@_sp\relax
467         \fi}%
468     \TeX}

```

```

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

```

‘(L)A_TE_X’ in my opinion better describes what I work with/in than just ‘L_AT_EX’.

```

\LaTeXpar 472 \DeclareLogo[(La)TeX]{\LaTeXpar}{%
473   {%
474     \setbox\z@\hbox{({}% )
475     \copy\z@
476     \kern-.2\wd\z@L%
477     \setbox\z@\hbox{\check@mathfonts
478       \fontsize\sf@size\z@
479       \math@fontsfalse\selectfont
480       A}%
481     \kern-.57\wd\z@
482     \sbox\tw@L%
483     \vbox\to\ht\tw@{\box\z@
484       \vss}%
485   }%
486   \kern-.07em% originally -, 15 em for T.
487   {% (
488     \sbox\z@)%
489     \kern-.2\wd\z@\copy\z@
490     \kern-.2\wd\z@}\TeX
491 }

```

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 couldn’t be used within `\csname...\endcsname`. My T_EXGuru 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 turns its #2 all into ‘other’ chars and assigns them to #1 which has to be a CS or an active char.

```

\def@other 492 \long\def\def@other#1#2{%
493   \long\def\gm@def@other@tempa{#2}%
494   \all@other#1{#2}}

```

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 495 \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.

```

\gm@gobmacro 496 \edef\@tempa{%
497   \def\@nx\gm@gobmacro##1\@xa@gobble\string\macro:->{}}
498 \@tempa

```

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

```

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

\unex@nameuse 502 \long\def\unex@nameuse#1{%
503   \edef@other\gmu@tempa{#1}%
504   \csname\gmu@tempa\endcsname}

```

Varia

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

```

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

```

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

(This definition is repeated in `gmverb`.)

```

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

```

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

```

509 \bgroup\catcode'\^M\active% the comment signs here are crucial.
510 \firstofone{\egroup%
\defobeylines 511 \def\defobeylines{\catcode'\^M=13\_ \def\^M{\par}}}%

```

Another thing I dislike in L^AT_EX yet is doing special things for `\dotskip`’s, ‘cause I like the Knuthian simplicity. So I sort of restore Knuthian meanings:

```

\dekssmallskip 512 \def\dekssmallskip{\vskip\smallskipamount}
\undeekssmallskip 513 \def\undeekssmallskip{\vskip-\smallskipamount}
\dekmedskip 514 \def\dekmedskip{\vskip\medskipamount}
\dekbigskip 515 \def\dekbigskip{\vskip\bigskipamount}
\hfillneg 516 \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 517 \long\def\@firstofmany#1#2\@@nil{#1}

```

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

```

\TODO 518 \newcommand*{\TODO}[1][ ]{\%
519   \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 L^AT_EX macro instead.

```

\twocoltoc 520 \newcommand*\twocoltoc{%

```

```

521 \RequirePackage{multicol}%
\starttoc 522 \def\starttoc##1{%
523 \begin{multicols}{2}\makeatletter\input{\jobname.##1}%
524 \if@filesw\@xa\newwrite\csname tf@##1\endcsname
525 \immediate\openout\csname tf@##1\endcsname\jobname.##1^
\relax
526 \fi
527 \@nobreakfalse\end{multicols}}
528 \@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 529 \newcommand\enoughpage[1]{%
530 \par
531 \dimen0=\pagegoal
532 \advance\dimen0 by-\pagetotal
533 \ifdim\dimen0<#1\relax\newpage\fi}

```

Two shorthands for debugging:

```

\tOnLine 534 \newcommand*\tOnLine{\typeout{\on@line}}
\OnAtLine 535 \let\OnAtLine\on@line

```

An equality sign properly spaced:

```

\equals 536 \newcommand*\equals{${}={}}$}

```

And for the L^AT_EX's pseudo-code statements:

```

\eequals 537 \newcommand*\eequals{${}=={}}$}

```

The job name without extension.

```

538 \def\gm@jobn#1.#2\@nil{#1}
\jobnamewoe 539 \def\jobnamewoe{\@xa\gm@jobn\jobname.\@nil}% We add the dot to be sure
there is one although I'm not sure whether you can TEX a file that has no
extension.

```

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 540 \def\freeze@actives{%
541 \count\z@\z@
542 \@whilenum\count\z@<\@ccclvi\do{%
543 \ifnum\catcode\count\z@=\active
544 \uccode'\~=\count\z@
545 \uppercase{\let~\relax}%
546 \fi
547 \advance\count\z@\@ne}}

```

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

```
\ShowFont 548 \newcommand*\ShowFont[1][6]{%
549   \begin{multicols}{#1}[The current font (the \f@encoding\encoding):]
550   \parindent\z@
551   \count\z@\m@ne
552   \@whilenum\count\z@<\ccclv\do{
553     \advance\count\z@\@ne
554     \the\count\z@:\char\count\z@\par}
555   \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 556 \newcommand*\liturgiques[1][red]{% Requires the color package.
557   \gmu@RP{color}%
558   \newcommand*\czerwo{\small\color{#1}}% environment
559   \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.
560   \def\*{\czer{${}$}}
561   \def\+{\czer{$\dag$}}
562   \newcommand*\nieczer[1]{\textcolor{black}{##1}}
```

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

```
563 \let\gmu@RP\RequirePackage
564 \AtBeginDocument{%
\gmu@RP 565   \renewcommand*\gmu@RP[2][ ]{%
566     \@ifpackageloaded{#2}{}{%
567       \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 568 \providecommand*\continuum{\gmu@RP{eufrak}\mathfrak{c}}
```

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

```
569 \providecommand*\acro[1]{\scshape\lowercase{#1}}
570 \newcommand*\IMO{\acro{IMO}}
571 \newcommand*\AKA{\acro{AKA}}
```

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

```
572 \def\secondclass{%
573   \newif\ifSecondClass
574   \SecondClasstrue
575   \@fileswithoptions\@clsextension% [outeroff,gmeometric]{gmdocc} it's load-
    ing gmdocc.cls with all the bells and whistles except the error message.
576 \endinput
```


Change History

v0.74

General:

Added macros to make sectioning commands of `mwcls` and standard classes compatible. Now my sectionings allow two optionals in both worlds and with `mwcls` if there's only one optional, it's the title to toc and running head not just to the latter, [24](#)

The catcodes of `\begin` and `\end` argument(s) don't have to agree strictly anymore: an environment is properly closed if the `\begin`'s and `\end`'s arguments result in the same `\csname`, [7](#)

v0.75

`\@ifnextac`:

added, [5](#)

`\@ifnextcat`:

`\let` for `#1` changed to `\def` to allow things like `\noexpand~`, [4](#)

v0.76

General:

A 'fixing' of `\dots` was rolled back since it came out they were O.K. and that was the QX encoding that prints them very tight, [24](#)

`\freeze@actives`:

added, [23](#)

v0.77

General:

`\afterfi` & pals made two-argument as the Marcin Woliński's analogoi are. At this occasion some redundant macros of that family are deleted, [24](#)

v0.78

General:

`\@namelet` renamed to `\n@melet` to solve a conflict with the `beamer` class. The package contents regrouped, [24](#)

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`, [13](#)

v0.80

General:

Checksum 1689, [1](#)

`\hfillneg`:

added, [22](#)

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.

<code>*</code> , 64 , 65 , 66 , 68 , 70 , 71 , 72 , 76 , 505 , 560	<code>\@currentvir</code> , 87 , 96	<code>\@ifnextspace</code> , 69
<code>\+</code> , 561	<code>\@currentvline</code> , 88	<code>\@ifnotmw</code> , 311 , 313 , 369 , 401
<code>\-</code> , 190	<code>\@currsize</code> , 102 , 103 , 104 , 105 , 106 , 107 , 108 , 109 , 110 , 111	<code>\@itemdepth</code> , 419 , 422 , 423
<code>\<...></code> , 177	<code>\@emptyify</code> , 26 , 27 , 28	<code>\@itemitem</code> , 423 , 424
<code>\@@nil</code> , 182 , 183 , 185 , 517 , 538 , 539	<code>\@enumctr</code> , 407 , 408 , 414	<code>\@nobreakfalse</code> , 527
<code>\@badend</code> , 97	<code>\@enumdepth</code> , 403 , 406 , 407	<code>\@nx</code> , 5 , 37 , 51 , 63 , 218 , 225 , 243 , 259 , 260 , 263 , 272 , 274 , 275 , 442 , 497
<code>\@begnamedgroup</code> , 83 , 84 , 90 , 93	<code>\@fileswithoptions</code> , 575	<code>\@oarg</code> , 197 , 198 , 199
<code>\@car</code> , 470	<code>\@firstofmany</code> , 517	<code>\@oargsq</code> , 197 , 199 , 207
<code>\@ccclv</code> , 552	<code>\@gif</code> , 9 , 10 , 12	<code>\@onlypreamble</code> , 528
<code>\@ccclvi</code> , 542	<code>\@ifl@aded</code> , 268	<code>\@parg</code> , 200 , 201 , 202
<code>\@checkend</code> , 94	<code>\@ifncat</code> , 46 , 47 , 59	<code>\@pargp</code> , 200 , 202 , 208
<code>\@clsextension</code> , 575	<code>\@ifnextac</code> , 60	
	<code>\@ifnextcat</code> , 42 , 61	

<code>\@parindent</code> , 410, 411, 413, 426, 427, 429	<code>\czer</code> , 559, 560, 561	<code>\gm@gobmacro</code> , 495, 497
<code>\@pkgextension</code> , 269	<code>\czerwo</code> , 558, 559	<code>\gm@hyperrefstepcounter</code> , 301, 304, 339
<code>\@preamblecmds</code> , 264, 265, 273, 276	<code>\dag</code> , 561	<code>\gm@ifnac</code> , 61, 62
<code>\@relaxen</code> , 30 , 31, 32	<code>\DeclareLogo</code> , 436, 445, 469, 472	<code>\gm@jobn</code> , 538, 539
<code>\@reserveda</code> , 70, 72	<code>\DeclareRobustCommand*</code> , 98, 126, 127, 128, 129, 130, 131, 163, 186, 188, 189, 192, 302, 505	<code>\gm@letspace</code> , 67, 73
<code>\@starttoc</code> , 522	<code>\DeclareTextCommand</code> , 442	<code>\gm@notprerr</code> , 270, 275
<code>\@toodeep</code> , 404, 420	<code>\DeclareTextCommandDefault</code> , 444	<code>\gm@PronounGender</code> , 277
<code>\@topnewpage</code> , 357	<code>\def@other</code> , 492	<code>\gm@pswords</code> , 182, 183 , 185
<code>\@whilenum</code> , 542, 552	<code>\defobeylines</code> , 511	<code>\gm@sec</code> , 389, 396, 397
<code>\@xa</code> , 4 , 13, 15, 22, 35, 59, 95, 96, 116, 165, 194, 215, 218, 240, 243, 252, 256, 259, 260, 380, 386, 408, 424, 437, 495, 497, 501, 524, 539	<code>\dekbigskip</code> , 515	<code>\gm@secini</code> , 370, 380, 383, 386, 394
<code>\@xifncat</code> , 49, 59	<code>\dekmedskip</code> , 514	<code>\gm@secmarkh</code> , 384
<code>\acro</code> , 569, 570, 571	<code>\deksmallskip</code> , 512	<code>\gm@secstar</code> , 372, 378, 381, 387, 396, 397
<code>\addto@macro</code> , 21 , 25	<code>\dimen</code> , 531, 532, 533	<code>\gm@secx</code> , 389, 390
<code>\addtomacro</code> , 25	<code>\discre</code> , 178 , 181	<code>\gm@secxx</code> , 371 , 385 , 391
<code>\addtonummacro</code> , 294	<code>\discret</code> , 179 , 180	<code>\gm@straightenssec</code> , 392, 399
<code>\afterfi</code> , 63, 73, 74, 77 , 180, 205	<code>\divide</code> , 461, 464, 465	<code>\gm@targetheading</code> , 302, 305
<code>\afterfifi</code> , 78 , 229, 232	<code>\edef@other</code> , 500, 503	<code>\gml@storeCS</code> , 222, 231, 247
<code>\afterfififi</code> , 82	<code>\eequals</code> , 537	<code>\gml@storemacros</code> , 223, 224, 229, 232, 248
<code>\afterriffifi</code> , 79	<code>\egRestore@Macro</code> , 236, 237	<code>\gmobeyspaces</code> , 507
<code>\afterriffififi</code> , 81	<code>\egRestore@MacroSt</code> , 236, 238	<code>\gmshowlists</code> , 34
<code>\afterrififfififi</code> , 80	<code>\egStore@Macro</code> , 211, 212	<code>\gmu@def</code> , 312, 314, 315
<code>\AKA</code> , 571	<code>\egStore@MacroSt</code> , 211, 213	<code>\gmu@RP</code> , 557, 563, 565 , 568
<code>\all@other</code> , 494, 495	<code>\emptyify</code> , 27 , 27	<code>\gmu@smtempa</code> , 217, 219, 242, 244
<code>\arg</code> , 204 , 205	<code>\enamgroup</code> , 91	<code>\gmu@tempa</code> , 500, 501, 503, 504
<code>\AtBeginDocument</code> , 203, 271, 298, 564	<code>\endlist</code> , 417, 432	<code>\gobble</code> , 133
<code>\begin</code> , 92 , 93	<code>\enoughpage</code> , 529	<code>\gobbletwo</code> , 134
<code>\begin*</code> , 93	<code>\ensuremath</code> , 164, 174	<code>\grefstepcounter</code> , 17
<code>\bigskipamount</code> , 515	<code>enumerate*</code> , 402	<code>\grelaxen</code> , 33 , 33
<code>\bnamegroup</code> , 90	<code>\env</code> , 192	<code>\HeadingNumber</code> , 336, 338
<code>\boldmath</code> , 470	<code>\equals</code> , 536	<code>\HeadingNumberedfalse</code> , 308, 319
<code>\box</code> , 483	<code>\exii@currentvir</code> , 96, 97	<code>\HeadingRHeadText</code> , 322
<code>\bslash</code> , 150 , 189	<code>\f@encoding</code> , 549	<code>\HeadingText</code> , 324
<code>\c@secnumdepth</code> , 319	<code>\f@series</code> , 470	<code>\HeadingTOCText</code> , 323
<code>\cleardoublepage</code> , 310	<code>\file</code> , 187	<code>\HeShe</code> , 285
<code>\cmd</code> , 194	<code>\freeze@actives</code> , 540	<code>\heshe</code> , 281
<code>\cmd@to@cs</code> , 194, 195	<code>\g@emptyify</code> , 28 , 29	<code>\hfillneg</code> , 516
<code>\color</code> , 558	<code>\g@relaxen</code> , 32 , 33	<code>\HimHer</code> , 287
<code>\continuum</code> , 568	<code>\gaddtomacro</code> , 20	<code>\himher</code> , 283
<code>\copy</code> , 454, 475, 489	<code>\gemptify</code> , 29 , 29	<code>\HisHer</code> , 286
<code>\count</code> , 295, 296, 297, 459, 460, 461, 462, 463, 464, 465, 466, 541, 542, 543, 544, 547, 551, 552, 553, 554	<code>\glet</code> , 19	<code>\hisher</code> , 282
<code>\cs</code> , 189 , 192, 194	<code>\gm@atppron</code> , 278 , 281, 282, 283, 284, 285, 286, 287, 288	<code>\HisHers</code> , 288
	<code>\gm@clearpagesduetoopenright</code> , 309, 328	<code>\hishers</code> , 284
	<code>\gm@def@other@tempa</code> , 493	<code>\hyphenpenalty</code> , 185
	<code>\gm@dontnumbersectionsoutofmainheading</code> , 307, 320	<code>\if@filesw</code> , 524
		<code>\if@mainmatter</code> , 308
		<code>\if@openright</code> , 310
		<code>\if@specialpage</code> , 327
		<code>\if@twoside</code> , 350
		<code>\if@numbered</code> , 318, 334
		<code>\ifodd</code> , 280

<code>\ifSecondClass</code> , 573	<code>\newgif</code> , 6	<code>\showboxbreadth</code> , 34
<code>\IMO</code> , 570	<code>\newlanguage</code> , 161	<code>\showboxdepth</code> , 34
<code>\itemindent</code> , 410, 426	<code>\newwrite</code> , 524	<code>\ShowFont</code> , 548
<code>itemize*</code> , 418	<code>\nfss@text</code> , 165	<code>\showlists</code> , 34
<code>\jobnamewoe</code> , 539	<code>\nieczer</code> , 562	<code>\smaller</code> , <i>p.</i> 7, 127
<code>\l@nohyphenation</code> , 160 , 161, 171	<code>\not@onlypreamble</code> , 262 , 265, 266, 267, 268, 269	<code>\smallerr</code> , <i>p.</i> 7, 131
<code>\labelsep</code> , 412, 428	<code>\nummacro</code> , 289	<code>\smallskipamount</code> , 512, 513
<code>\labelwidth</code> , 411, 412, 427, 428	<code>\oarg</code> , 197	<code>\smartunder</code> , 143
<code>\larger</code> , <i>p.</i> 7, 126	<code>\old@begin</code> , 92 , 93	<code>\stepnummacro</code> , 290
<code>\largerr</code> , <i>p.</i> 7, 130	<code>\oldLaTeX</code> , 433	<code>\Store@Macro</code> , 212, 214, 222
<code>\LaTeXe</code> , 434, 469	<code>\oldLaTeXe</code> , 434	<code>\Store@Macros</code> , 220, 221
<code>\LaTeXpar</code> , 472	<code>\OnAtLine</code> , 535	<code>\Store@MacroSt</code> , 213, 216
<code>\leftmargin</code> , 409, 425	<code>\PackageWarning</code> , 122, 124	<code>\Stored@Macro</code> , 249, 250
<code>\list</code> , 408, 424	<code>\pagebreak</code> , 349, 361, 365	<code>\StoredMacro</code> , 249
<code>\listparindent</code> , 413, 429	<code>\pagegoal</code> , 531	<code>\StoreMacro</code> , 210 , 314
<code>\liturgiques</code> , 556	<code>\pagetotal</code> , 532	<code>\StoreMacros</code> , 220
<code>\macro</code> , 497	<code>\parg</code> , 200	<code>\StoringAndRelaxingDo</code> , 251
<code>\marg</code> , 196 , 208	<code>\partopsep</code> , 409, 425	<code>\subs</code> , 137 , 145
<code>\math@arg</code> , 204, 205	<code>\pk</code> , 188	<code>\textcolor</code> , 562
<code>\mathfrak</code> , 568	<code>\possfil</code> , 193	<code>\textlarger</code> , 128
<code>\medmuskip</code> , 180	<code>\printspaces</code> , 182 , 186	<code>\textsmaller</code> , 129
<code>\meta</code> , 163 , 177, 196, 198, 201	<code>\relaxen</code> , 31 , 31	<code>\textstyle</code> , 471
<code>\meta@font@select</code> , 167, 176	<code>\resize</code> , <i>p.</i> 7, 98 , 99, 126, 127, 128, 129, 130, 131	<code>\textvisiblespace</code> , 181
<code>\meta@hyphen@restore</code> , 168 , 173	<code>\renewcommand*</code> , 565	<code>\thr@@</code> , 403, 419
<code>\mskip</code> , 180	<code>\RequirePackage</code> , 521, 563	<code>\TODO</code> , 518
<code>\multiply</code> , 460, 463	<code>\Restore@Macro</code> , 237, 239, 247, 250	<code>\tOnLine</code> , 534
<code>\mw@HeadingBreakAfter</code> , 329, 346, 361, 365	<code>\Restore@Macros</code> , 245, 246	<code>\twelveand</code> , 156
<code>\mw@HeadingBreakBefore</code> , 326	<code>\Restore@MacroSt</code> , 238, 241	<code>\twelvebackslash</code> , 149 , 150
<code>\mw@HeadingLevel</code> , 316, 319	<code>\RestoreMacro</code> , 235 , 314	<code>\twelvepercent</code> , 153
<code>\mw@HeadingRunIn</code> , 341	<code>\RestoreMacros</code> , 245	<code>\twelvespace</code> , 37, 38, 159
<code>\mw@HeadingType</code> , 325	<code>\RestoringDo</code> , 254	<code>\twelveunder</code> , 140
<code>\mw@HeadingWholeWidth</code> , 344	<code>\romannumeral</code> , 407, 423	<code>\twocoltoc</code> , 520 , 528
<code>\mw@normalheading</code> , 348, 357, 360, 364	<code>\rs@size@warning</code> , 115, 120, 122	<code>\undeksmallskip</code> , 513
<code>\mw@runinheading</code> , 342	<code>\rs@unknown@warning</code> , 112, 124	<code>\unex@namedef</code> , 499
<code>\mw@sectionxx</code> , 315	<code>\scshape</code> , 569	<code>\unex@nameuse</code> , 502
<code>\n@melet</code> , 257, 373, 376, 394	<code>\secondclass</code> , 572	<code>\usecounter</code> , 414
<code>\nameshow</code> , 35	<code>\SecondClasstrue</code> , 574	<code>\varepsilon</code> , 471
<code>\newcounter</code> , 277, 299	<code>\sfname</code> , 186 , 187	<code>\vs</code> , 181 , 182, 185
		<code>\wd</code> , 452, 455, 476, 481, 489, 490
		<code>\xiistring</code> , 36