Grzegorz Murzynowski

The gmutils Package*

Written by Grzegorz Murzynowski,
natror at o2 dot pl

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmutils}
      [2010/03/04 v0.991 some rather TeXnical macros, some
           of them tricky (GM)]
```

Contents

* This file has version number v0.991 dated 2010/03/04.

Intro

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

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

For details just read the code part.

Installation

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

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

Contents of the gmutils.zip archive

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

gmutils.sty
README
gmutils.pdf
gmutils.tds.zip

Compiling of the documentation

The last of the above files (the .pdf, i.e., *this file*) is a documentation compiled from the .sty file by running LaTeX on the gmutils.sty file twice (`xelatex␣gmutils.sty` in the

directory you wish the documentation to be in, you don't have copy the .sty file there, TEX will find it), then MakeIndex on the gmutils.idx file, and then LATEX on gmutils.sty 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.

```
\ifx\XeTeXversion\relax
  \let\XeTeXversion\@undefined% If someone earlier used
      % \@ifundefined{XeTeXversion} to test whether the engine is XeTeX,
      then \XeTeXversion is defined in the sense of ε-TeX tests. In that case
      we \let it to something really undefined. Well, we might keep sticking
      to \@ifundefined, but it's a macro and it eats its arguments, freezing their
      catcodes, which is not what we want in line 6155.
\fi

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

Options

\ifgmu@quiet
\newif\ifgmu@quiet

quiet
\DeclareOption{quiet}{\gmu@quiettrue}

\ProcessOptions

A couple of abbreviations

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

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

Note that there's only one `\expandafter`, after `\unexpanded`. It's because `\unexpanded` expands expandable tokens and gobbles `\relax`es while looking for an opening brace or `\bgroup`. Note also that it *requires* a text in braces.

Note also that (since v0.991) this is a 1-parameter macro so doesn't expand subsequent tokens until meets a ⟨*balanced text*⟩ but just takes first single token or ⟨*text*⟩.

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

And this one is defined, I know, but it's not `\long` with the standard definition and I want to be able to `\gobble` a `\par` sometimes.

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

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

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

\gmu@gobdef \long\def\gmu@gobdef#1#{%
              \def\gmu@tempa{}% it's a junk macro assignment to absorb possible prefixes.
              \@gobble}

\pprovide   \def\pprovide{\protected\provide}
```

Note that both `\provide` and `\pprovide` may be prefixed with `\global`, `\outer`, `\long` and `\protected` because the prefixes stick to `\def` because all before it is expandable. If the condition(s) is false (`#1` is defined) then the prefixes are absorbed by a junk assignment.

Note moreover that unlike LaTeX's `\providecommand`, our `\(p)provide` allow any parameters string just like `\def` (because they just *expand* to `\def`).

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

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

```
\gmu@if     \long\def\gmu@if#1#2{%
              \csname␣if#1\endcsname#2\@xa\@firstoftwo\else\@xa%
                   \@secondoftwo\fi
            }
```

`\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 `\cat¦codes` locally for a definition *outside* the changed `\catcodes`' group. Just see the below usage of this macro 'with TeX's eyes', as my TeX Guru taught me.

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

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

\@firstofthree  \long\def\@firstofthree#1#2#3{#1}
\@secondofthree \long\def\@secondofthree#1#2#3{#2}
\@thirdofthree  \long\def\@thirdofthree#1#2#3{#3}
\@secondoffive  \long\def\@secondoffive#1#2#3#4#5{#2}
```

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

```
\@firstofmany  319 \long\def\@firstofmany#1#2\@@nil{#1}
\@allbutfirstof  322 \long\def\@allbutfirstof#1#2\@@nil{#2}
```

\afterfi and pals

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

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

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

```
\longafterfi  353 \def\longafterfi{%
   \afterfi  354   \long\def\afterfi##1##2\fi{\fi##1}}
  355 \longafterfi
```

And two more of that family:

```
\afterfifi  357 \long\def\afterfifi#1#2\fi#3\fi{\fi\fi#1}
\afteriffifi  359 \long\def\afteriffifi#1#2\fi#3\fi{\fi#1}
```

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

```
\afterififfifi  363 \long\def\afterififfifi#1#2\fi#3\fi#4\fi{\fi#1}
\afteriffifi  364 \long\def\afteriffifi#1#2\fi#3\fi#4\fi{\fi\fi#1}
\afterfifi  365 \long\def\afterfifi#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}
```

\foone

The next command, \foone, is intended as two-argument for shortening of the \begingroup...% \firstofone{\endgroup...} hack.

```
\foone  372 \long\def\foone#1{\begingroup#1\relax\egfirstofone}
  374 \long\def\egfirstofone#1{\endgroup#1}
\fooatletter  376 \def\fooatletter{\foone\makeatletter}
```

\@ifempty, \IfAmong, \IfIntersect \@ifinmeaning

After a short deliberation I make the \IfAmong…\among and \IfIntersect macros' names apeless since they are intended for the macro and class writers, at the same level of abstraction as \DeclareCommand.

```
\@ifempty  386 \long\pdef\@ifempty#1{%
          % #1  the token(s) we test, it may be just {},
          % #2  the stuff executed if #1 is empty (is {}),
```

```
                % #3 the stuff executed if #1 consists of at least one token.
\gmu@reserveda  \def\gmu@reserveda{#1}%
                \ifx\gmu@reserveda\@empty\@xa\@firstoftwo
                \else\@xa\@secondoftwo
                \fi}
\@ifnonempty    \long\def\@ifnonempty#1#2#3{\@ifempty{#1}{#3}{#2}}
\IfAmong        \long\pdef\IfAmong#1\among#2{%
                    % #1 the token(s) whose presence we check,
                    % #2 the list of tokens in which we search #1,
                    % #3 the 'if found' stuff,
                    % #4 the 'if not found' stuff.
\gmu@among@     \long\def\gmu@among@##1#1##2\gmu@among@{%
                  \@ifempty{##2}\@secondoftwo\@firstoftwo}%
                \gmu@among@#2#1\gmu@among@}
\IfIntersect    \long\def\IfIntersect#1#2{% this (not expandable) macro checks whether
                    the list of tokens #1 and #2 have nonempty intersection and if so executes
                    % \@firstoftwo or else \@secondoftwo, so it's a 4-argument com-
                    mand:
                    % #1 first list to match,
                    % #2 second list to match,
                    % #3 if match (nonempty intersection),
                    % #4 if not match (empty intersection).
                \let\IfIntersect@next\@secondoftwo
                \gmu@foreach#1\gmu@foreach{%
                  \@xa\IfAmong\gmu@forarg\among{#2}
                  {\let\IfIntersect@next\@firstoftwo}{}}%
                \IfIntersect@next}
```

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

```
\gmu@foreach    \long\def\gmu@foreach#1\gmu@foreach#2{%
 \gmu@forer       \long\def\gmu@forer##1{%
                    \ifx\gmu@foreach##1\else
\gmu@forarg         \long\def\gmu@forarg{##1}%
                    #2%
                    \@xa\gmu@forer
                    \fi}%
                  \gmu@forer#1\gmu@foreach}
\@ifinmeaning   \long\pdef\@ifinmeaning#1\of#2{%
                    % #1 the token(s) whose presence we check,
                    % #2 the macro in whose meaning we search #1 (the first token of this
                        argument is expanded one level with \expandafter),
                    % #3 the 'if found' stuff,
                    % #4 the 'if not found' stuff.
\gmu@reserveda  \def\gmu@reserveda{\IfAmong#1\among}%
                \@xa\gmu@reserveda\@xa{#2}}
```

Global Boolean switches

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

UD-if's assignment global. I needed it at least twice during gmdoc writing so I make it a macro. It's an almost verbatim copy of LaTeX's `\newif` modulo the letter *g* and the `\global` prefix. (File d: ltdefns.dtx Date: 2004/02/20 Version v1.3g, lines 139–150)

`\newgif`

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

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

`\@gif`

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

```
\pdef\newif#1{% We not only make \newif \protected but also make it to
        define \protected assignments so that premature expansion doesn't
        affect \if…\fi nesting.
  \count@\escapechar␣\escapechar\m@ne
  \let#1\iffalse
  \@if#1\iftrue
  \@if#1\iffalse
  \escapechar\count@}
```

`\@if`

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

`\hidden@iffalse`
`\hidden@iftrue`

```
\pdef\hidden@iffalse{\iffalse}
\pdef\hidden@iftrue{\iftrue}
```

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

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

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

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

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

`\grefstepcounter`

```
\pdef\grefstepcounter#1{%
  {\let\protected@edef=\protected@xdef\refstepcounter{#1}}}
```

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

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

2008/08/10 I spent all the night debugging `\penalty 10000` that was added after a hypertarget in vertical mode. I didn't dare to touch hyperref's guts, so I worked it around with ensuring every `\grefstepcounter` to be in hmode:

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

By the way I read some lines from *The TEXbook* and was reminded that `\unskip` strips any last skip, whether horizontal or vertical. And I use `\unskip` mostly to replace a blank space with some fixed skip. Therefore define

\hunskip
```
\pdef\hunskip{\ifhmode\unskip\fi}
```

Note the two macros defined above are `\protected`. I think it's a good idea to make `\protected` all the macros that contain assignments. There is one more thing with `\ifhmode`: it can be different at the point of `\edef` and at the point of execution.

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

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

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

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

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

\addto@macro
```
\long\def\addto@macro#1#2{%
  \edef#1{\@xa\unexpanded\@xa{#1}\unexpanded{#2}}%  \edef and \unexpanded to double the hashes.
}
```

And for use in the very document,

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

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

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

Note that `\prependtomacro` can be prefixed.

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

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

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

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

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

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

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

`\gm@ifundefined`—a test that doesn't create any hash entry unlike `\@ifundefined`

I define it under another name not redefine `\@ifundefined` because I can imagine an odd case when something works thanks to `\@ifundefined`'s 'relaxation' effect.

`\gm@ifundefined`

```
\long\def\gm@ifundefined#1{% not \protected because expandable.
  \ifcsname#1\endcsname% defined…
    \@xa\ifx\csname␣#1\endcsname\relax% but only as \relax—then 2nd argument.
      \afterfifi\@firstoftwo%
    \else% defined and not \relax—then 3rd argument.
      \afterfifi\@secondoftwo%
    \fi
  \else% not defined—then 3rd.
    \afterfi\@firstoftwo%
  \fi}
```

`\gm@testdefined`

```
\long\def\gm@testdefined#1\iftrue{% This is a macro that expands to \iftrue or \iffalse depending on whether its argument is defined in the LaTeX sense. Its syntax requires an \iftrue to force balancing \ifs with \fis.
  \csname
  \ifdefined#1%
    \ifx#1\relax
       iffalse%
    \else␣iftrue%
    \fi
  \else␣iffalse%
  \fi\endcsname
}
```

`\gm@testundefined`

```
\long\def\gm@testundefined#1\iftrue{% we expand the last macro two levels. We repeat the parameter string to force the proper syntax.
  \@xa\@xa\@xa\unless\gm@testdefined#1\iftrue}
```

Some 'other' and active stuff

Here I define a couple of macros expanding to special chars made 'other'. It's important the CS are expandable and therefore they can occur e.g. inside `\csname`…`\endcsname` unlike e.g. CS'es `\chardef`ed.

```
\foone{\catcode`\_=8␣}
```

`\subs`

```
{\let\subs=_}
```

```
\foone{\catcode`\^=7␣}
```

`\sups`

```
{\let\sups=^}
```

```
\foone{\@makeother\_}
```

`\xiiunder`

```
{\def\xiiunder{_}}
```

```
\let\all@unders\xiiunder
\foone{\catcode`\_=8␣}
{\addtomacro\all@unders{_}}
\foone{\catcode`\_=11␣}
{\addtomacro\all@unders{_}}
\foone{\catcode`\_=13␣}
```

```
{\addtomacro\all@unders{_}}
```
Now `\all@unders` bears underscores of categories 8, 11, 12 and 13.

`\all@stars`
```
\def\all@stars{*}
\foone{\catcode`\*=11␣}
{\addtomacro\all@stars{*}}
\foone{\catcode`\*=13␣}
{\addtomacro\all@stars{*}}
```
And `\all@stars` bears stars of categories 11, 12 and 13.

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

`\backquote`
```
\foone{\@makeother\`}%
{\def\backquote{`}}
```

`\xiilbrace` `\xiirbrace`
```
\foone{\catcode`\[=1␣\@makeother\{
  \catcode`\]=2␣\@makeother\}}%
[%
  \def\xiilbrace[{]%
  \def\xiirbrace[}]%
]% of \firstofone
```

Note that LaTeX's `\@charlb` and `\@charrb` are of catcode 11 ('letter'), cf. The LaTeX 2_ε Source file k, lines 129–130.

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

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

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

`\bslash`
```
\let\bslash=\xiibackslash
```

`\xiipercent`
```
\foone{\@makeother\%}
{\def\xiipercent{%}}
```

`\xiiand`
```
\foone{\@makeother\&}%
{\def\xiiand{&}}
```

`\xiispace`
```
\foone{\@makeother\␣}%
{\def\xiispace{␣}}
```

`\xiihash`
```
\foone{\@makeother\#}%
{\def\xiihash{#}}
```

We introduce `\visiblespace` from Will Robertson's xltxtra if available. It's not sufficient `\@ifpackageloaded{xltxtra}` since `\xxt@visiblespace` is defined only unless `no-verb` option is set. 2008/08/06 I recognised the difference between `\xiispace` which has to be plain 'other' char (used in `\xiistring`) and something visible to be printed in any font.

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

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

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

`\@ifnextcat`, `\@ifnextac`, catcode-independent `\gm@ifstar`, `\@ifnextsingle`, `\@ifnextgroup`

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 expandable. Because an assignment is involved, you can't test whether the next token is an active char. But you can test if the next token is {$_1$ or }$_2$: `\@ifnextcat\bgroup`…, `\@ifnextcat\egroup`….

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

```
\@ifncat  \def\@ifncat{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifncat
  \else
    \ifcat\reserved@d\@nx\@let@token
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
```

```
{\def\:{\let\@sptoken=␣}␣\global\:␣%
```
this makes `\@sptoken` a space token.

```
\def\:{\@xifncat}␣\@xa\gdef\:␣{\futurelet\@let@token%
    \@ifncat}}
```

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

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

Having `\@ifnextcat` defined, let's apply it immediately. Similar thing does probably the xspace package.

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

\@ifnextif
```
\long\pdef\@ifnextif#1#2#3{%
```

(the future token in `\noexpand`ed, unlike `#1`)

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

\@ifnif
```
\def\@ifnif{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifnif
  \else
    \if\reserved@d\@nx\@let@token
      \let\reserved@c\reserved@a
    \else␣% #1 of \@ifnextif is not \if-equivalent the future token. But this
          may be because the future token is active so it would be \if-equivalent
          if not passed through \futurelet. Let's manage this case.
      \begingroup
      \edef\gmu@tempa{%
        \lccode`\@nx~=`\reserved@d
      }\gmu@tempa
      \lowercase{\endgroup
        \ifx~}\@let@token
        \let\reserved@c\reserved@a
      \else
        \let\reserved@c\reserved@b
      \fi
    \fi
  \fi
  \reserved@c}
```

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

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

\@ifnextac
```
\long\pdef\@ifnextac#1#2{%
  \@ifnextsingle
  {\gm@ifnac{#1}{#2}}%
  {#2}}
```

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

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

Having defined such tools, let's redefine `\gm@ifstar` to make it work with whatever catcode $\star$ may be. make a version of `\gm@ifstar` that would work with $\star_{11}$.

`\gmu@lowstar`

```
\pdef\gmu@lowstar{%
  \iffontchar\font"22C6␣\char"22C6␣\else\gmu@lowstarfake\fi}%
```

we could define it to be expandable (with `^^^^22c6`) but the only goal of it would be allowing this low star in `\csname`…`\endcsname`. But it would be misleading (not everyone can easily distinguish `*` from $\star$) so IMHO it's better to raise an error should such an active occur in a `\csname`. This macro is intended to be `\let` to an active star only in verbatims. For usual text there's

Commands around making star low (via `\active`ation) are defined in line 7048 because they use `\DeclareCommand`.

`\gmu@tempa`

```
\def\gmu@tempa{*}
\foone{\catcode`\*=\active}
```

`\gmu@tempb`

```
{\def\gmu@tempb{*}%
```

it's defined in line **??** to make $\star$ defined (when it was undefined, `\newcommand`'s `\gm@ifstar` test turned true, the next undefined token was gobbled and raised an error).

```
  \let*=\gmu@lowstar}

\edef\gmu@tempa{%
  \long\pdef\@nx\gm@ifstar##1##2{% ]
    \@nx\@ifnextif\gmu@tempa%
    {\@nx\@firstoftwo{##1}}%
    {%
      \@nx\@ifnextchar\@xa\@nx\gmu@tempb
      {\@nx\@firstoftwo{##1}}{##2}}%
  }% of \gm@ifstar.
}\gmu@tempa
```

A test whether we can pick a single token. We have to check whether we are not next to { and whether we are not next to }.

`\@ifnextsingle`

```
\long\pdef\@ifnextsingle#1#2{%
```

This macro checks whether the next token is able to be picked or is it a braced list of tokens or is it a group closer so there's no token to be picked.

```
  \@ifnextcat\bgroup{#2}{%
    \@ifnextcat\egroup{#2}%
    {#1}}}
```

`\@ifnextgroup`

```
\long\pdef\@ifnextgroup#1#2{%
```

Note this macro turns true both before a group opener and before a group closer.

```
  \@ifnextsingle{#2}{#1}}
```

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

`\ignoreactiveM`

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

\foone\obeylines{%
```

we know it's a single token since we use this macro only in `\@`ifnextgroup's 'else'.

```
\gmu@checkM  \long\pdef\gmu@checkM#1{%
               \ifx
               #1\@xa\ignoreactiveM%
               \else\@xa#1\fi}}
```

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

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

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

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

Now a test if the next token is an active line end. I use it in `gmdoc` and later in this package for active long dashes.

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

Storing and restoring the meanings of CSes

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

```
\ifgmu@SMglobal  \newif\ifgmu@SMglobal
```

```
\SMglobal  \pdef\SMglobal{\gmu@SMglobaltrue}
```

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

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

```
\StoreMacro  \pdef\StoreMacro{%
               \begingroup\makeatletter\gm@ifstar\egStore@MacroSt%
                    \egStore@Macro}
```

The unstarred version takes a CS and the starred version a text, which is intended for special control sequences. For storing environments there is a special command in line 1177.

```
\egStore@Macro  \long\def\egStore@Macro#1{\endgroup\Store@Macro{#1}}
```

```
\egStore@MacroSt
\long\def\egStore@MacroSt#1{\endgroup\Store@MacroSt{#1}}
\Store@Macro
\long\def\Store@Macro#1{%
  \escapechar92
  \ifgmu@SMglobal\afterfi\global\fi
  \@xa\let\csname␣/gmu/store\string#1\endcsname#1%
  \global\gmu@SMglobalfalse}
\Store@MacroSt
\long\def\Store@MacroSt#1{%
  \edef\gmu@smtempa{%
    \ifgmu@SMglobal\global\fi
    \@nx\let\@xa\@nx\csname/gmu/store\bslash#1\endcsname%
        we add backslash because to ensure compatibility
        between \(Re)StoreMacro and \(Re)StoreMacro*, that
        is. to allow writing e.g. \StoreMacro\kitten and
        then \RestoreMacro*{kitten} to restore the meaning of \kitten.
    \@xa\@nx\csname#1\endcsname}
  \gmu@smtempa
  \global\gmu@SMglobalfalse}% we wish the globality to be just once.
```

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

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

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

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

And the inner iterating macro:

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

```
      \afterfi{\gmu@setSMglobal\gml@storemacros}%
    \fi
  \fi}
```

And for the restoring

\RestoreMacro
```
\pdef\RestoreMacro{%
  \begingroup\makeatletter\gm@ifstar\egRestore@MacroSt%
      \egRestore@Macro}
```

\egRestore@Macro
\egRestore@MacroSt
```
\long\def\egRestore@Macro#1{\endgroup\Restore@Macro{#1}}
\long\def\egRestore@MacroSt#1{\endgroup\Restore@MacroSt{#1}}
```

\Restore@Macro
```
\long\def\Restore@Macro#1{%
  \escapechar92
  \gmu@ifstored#1{%
    \ifgmu@SMglobal\afterfi\global\fi
    \@xa\let\@xa#1\csname␣/gmu/store\string#1\endcsname
    \global\gmu@SMglobalfalse}%
  {\unless\ifgmu@quiet
      \PackageWarning{gmutils}{\@nx#1␣is␣not␣stored,␣I␣do␣
          nothing␣with
        it}%
    \fi
  }%
}
```

\gmu@ifstored
```
\long\def\gmu@ifstored#1#2#3{%
  \gm@ifundefined{/gmu/store%
    \if\relax\@nx#1\else\bslash\fi
    \string#1}{#3}{#2}%
}
```

\gmu@storeifnotyet
```
\long\pdef\gmu@storeifnotyet#1{%
  \if\relax\@nx#1% we check if it's a CS
  \gmu@ifstored{#1}{}{\StoreMacro#1}%
  \fi}
```

\Restore@MacroSt
```
\long\def\Restore@MacroSt#1{%
  \gm@ifundefined{/gmu/store\bslash#1}%
  {\unless\ifgmu@quiet
    \PackageWarning{gmutils}{\bslash#1␣is␣not␣stored.␣I~do␣
        nothing}%
    \fi}%
  {\edef\gmu@smtempa{%
      \ifgmu@SMglobal\global\fi
      \@nx\let\@xa\@nx\csname#1\endcsname
      \@xa\@nx\csname/gmu/store\bslash#1\endcsname}% cf. the commentary in line 1049.
    \gmu@smtempa}%
  \global\gmu@SMglobalfalse}
```

\RestoreMacros
```
\long\pdef\RestoreMacros{\begingroup\makeatletter%
    \Restore@Macros}
```

\Restore@Macros
```
\long\def\Restore@Macros#1{\endgroup
  \gmu@setsetSMglobal
```

```
1159    \let\gml@StoreCS\Restore@Macro%
```
we direct the core CS towards restoring and call the same iterating macro as in line 1077.
```
1162    \gml@storemacros#1.}
```

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

And to restore *and* use immediately:

`\StoredMacro`
`\Stored@Macro`
```
1168 \def\StoredMacro{\begingroup\makeatletter\Stored@Macro}
1169 \long\def\Stored@Macro#1{\endgroup\Restore@Macro#1#1}
```

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

`\storedcsname`
```
1172 \def\storedcsname#1{%
1173    \csname␣/gmu/store\bslash#1\endcsname}
```

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

`\StoreEnvironment`
```
1177 \pdef\StoreEnvironment#1{%
1179    \StoreMacro*{#1}\StoreMacro*{end#1}}
```

`\RestoreEnvironment`
```
1181 \pdef\RestoreEnvironment#1{%
1183    \RestoreMacro*{#1}\RestoreMacro*{end#1}}
```

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

`\StoringAndRelaxingDo`
```
1198 \def\StoringAndRelaxingDo{%
1199    \gmu@SMdo@setscope
1200    \long\def\do##1{%
1201       \gmu@SMdo@scope
1202       \@xa\let\csname␣/gmu/store\string##1\endcsname##1%
1203       \gmu@SMdo@scope\let##1\relax}}
```

`\gmu@SMdo@setscope`
```
1205 \def\gmu@SMdo@setscope{%
1206    \ifgmu@SMglobal\let\gmu@SMdo@scope\global
1207    \else\let\gmu@SMdo@scope\relax
1208    \fi
1209    \global\gmu@SMglobalfalse}
```

And here is the counter-definition for restore.

`\RestoringDo`
```
1218 \long\def\RestoringDo{%
1219    \gmu@SMdo@setscope
1220    \long\def\do##1{%
1221       \gmu@SMdo@scope
1222       \@xa\let\@xa##1\csname␣/gmu/store\string##1\endcsname}}
```

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

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

`\n@melet`
```
1230 \long\def\n@melet#1#2{%
1231    \edef\gmu@nl@reserveda{%
```

```
      \let\@xa\@nx\csname#1\endcsname
      \@xa\@nx\csname#2\endcsname}%
    \gmu@nl@reserveda}
```

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

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

`\envirlet`
```
\long\pdef\envirlet#1#2{%
  \n@melet{#1}{#2}%
  \n@melet{end#1}{end#2}%
}
```

`\DeclareCommand` and `\DeclareEnvironment`

The code of this section is based on the xparse package version 0.17 dated 1999/09/10, the version available in TeX Live 2007-13, in Ubuntu packages at least. Originally considered a stub 'im Erwartung' (Schönberg) for the LaTeX3 bundle, it evolved to quite a nice tool I think.

I rewrote the ancient xparse's code to use the usual catcodes (only with `@` a letter) and not to use the ldcsetup package (which caused an error of undefined CS `\KV@def`). Then I added the `\protected` prefix to the first macro. And I added my concept of `S`/`T` arg. spec. And of `b` and `B` spec. And the `Q` argument type. And made `\DeclareEnvironment` makes the arguments passed also to the end macro.

After a short deliberation I rename the command to `\DeclareCommand` which is much shorter than original `\DeclareDocumentCommand` and more adequate at least in my case: I don't only use this powerful declaration for 'document' commands.

In the args specification you can use:

`m` for mandatory argument (braced or not),

`O{`⟨*default*⟩`}` for optional argument (in square brackets) with default value ⟨*default*⟩ (which is passed as `#`⟨*n*⟩ if the argument is missing),

`o` equivalent of `O{\NoValue}`: optional argument (in square brackets) lack of which makes the `\NoValue` macro passed as `#`⟨*n*⟩. You can check it with `\If[No]Value(T|F|TF){\#`⟨*n*⟩`}`,

`s` for optional star (if star is present, it becomes respective `#`⟨*n*⟩, or else `\NoValue` is at the `#`⟨*n*⟩ (unlike in xparse!).

`S{`⟨*list*⟩`}{`⟨*default*⟩`}` for a Single token, checks whether on the respective position there's an unbraced token one of ⟨*list*⟩ and returns it on `#`⟨*n*⟩ if present or `{`⟨*default*⟩`}` if absent. The `{`⟨*default*⟩`}` is a braced optional. If absent, `\NoValue` is the default.

`T` as 'Token', just an alias for `S`.

Like in xparse, `s` is a shorthand for `S{*}`, but *unlike* in xparse, in the parsed arguments you get `*` or `\NoValue` as `#`⟨*n*⟩. You can now declare

```
\DeclareCommand\mimbla{S{+-}m}{%
 \leavevmode
 \ifx#1+\raise\fi
 \ifx#1-\lower\fi
 \ifx#1\NoValue\@tempdima=\fi
 7pt  \hbox{#2}%
 }
```

and after `\mimbla+{raised}␣\mimbla-{lowered}␣\mimbla␣{untouched}` get:

raised
lowered untouched

This example is rather silly but shows that you spend only one `#` for two symbols while in xparse you would spend two. What if you wanted 10 options for the optional symbol? Here it's no problem. And with any number k of tokens on the list the next `#` is always $n+1$ not $n+k+1$.

`Q{`⟨*list*⟩`}` as 'se**Q**uence', a sequence of symbols from ⟨*list*⟩; for the fundamental usage see line 2244. More clearly, an argument specified with `Q{`⟨*tokens*⟩`}` is a word over the alphabet ⟨*tokens*⟩∪␣ where ␣ is ignored and shall not be passed in the respective `#`⟨*n*⟩.

`c` for an optional argument in parentheses. Historically it comes from a 'coordinate' argument and may serve as such: if you declare `\DCcoordinate`, then its catcher will be redefined to check for presence of a comma and pass the argument as `{%` {⟨*before comma*⟩} {⟨*after comma*⟩} `}` if comma present. `\NoValue` is passed if absent. By default `\DCnocoordinate` is executed that defines the `c` type argument as just an optional in parentheses.

`\DCcoordinate`

`\DCnocoordinate`

`C{`⟨*default*⟩`}` as `c`, with default value.

`b` for for an *optional* argument in curly braces. That's strange for anyone acquainted with LaTeX and contrary to its basic convention, but practised by Til Tantau in the beamer class. If missing, `\NoValue` is passed as `#`⟨*n*⟩ and therefore all the tests `If[No]Value(T|F|TF)` apply.

`B{`⟨*default*⟩`}` for a braced *optional* argument with the default value ⟨*default*⟩.

`K{`⟨*#-string*⟩`}[{`⟨*replacement*⟩`}]` for a *mandatory* **K**nuthian delimited or undelimited macro parameters. This concept comes from Stephen Hicks' suggestion at BachoTeX 2009 of implementing arguments delimited with `#{`. So, in the mandatory first argument to `K` you give an arbitrary parameters string as described in Chapter 20 of *The TeXbook*. If you don't provide the replacement, only `#1` of those parameters will be passed to the core macro of your command as `#`⟨*n*⟩. In both arguments you use single `#` char.

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

`A` (for 'angles') a shorthand for `G{<>}` (accepts the second braced as the default value).

`\afterassignment{`⟨*register*⟩`}` a pseudo-argument that in fact *interrupts* parsing arguments to let an assignment to ⟨*register*⟩. The default ⟨*register*⟩ is `\@tempcnta`. For example,

```
\DeclareCommand\llf{\afterassignment}
 {{\lastlinefit\@tempcnta\par}}
```

defines `\llf` to be a command that expects a value for a numerical assignment, assigns it to (`\@tempcnta` and then to) the `\lastlinefit` special register and executes `\par` with that setting inside a group.

Actually, ⟨*register*⟩ may be empty `{}` and then each time our command is used the left side of the assignment has to be given explicitly and may even be different each time.

Doesn't add anything to the arguments' list.

This pseudo-argument type should be considered experimental.

Note that you could limit acceptable values of a mandatory or an optional-in-brackets argument to a list inside definition of the command, using `\IfAmong…\among…` defined in line 399.

The `S/T`, `s` and `Q` arguments are always 'long', they allow `\par` as their value that

is. They have to be unbraced to be parsed so there's no danger of "runaway argument".

\long!lL By default, all the `c,C,o,O,m,b` and `B` are 'short', they don't allow `\par` in them that is. Note however that `\par` is allowed in the default values. If you wish to allow `\par` for all the arguments, you can say `\DeclareCommand\mycommand!`… — the optional `!` makes all the arguments 'long'. Instead of `!` you can use `L` or `l` for 'long' or just `\long` itself.

In the arguments specification string you can write `>[{]`⟨*prefix*⟩`[}]` to make subsequent argument 'long' or ignored:

Pp!lL \long \par iI Any of `Pp!lL\long\par` to make a particular argument 'long', allowing `\par` in it that is, and/or any of `iI` to make the argument ignored (just gobbled).

(Note that also the `c` and `C` arguments may be made 'long'. That's because I use them not as coordinates but as just another kind of optional argument.

The concept of ignored arguments came to my head when I was declaring a command with three braced optionals and put optional stars only to distinguish the braced optionals.

For example, after

```
\DeclareCommand\GLBTQKi{%
    G{&&}{\#1 default}
    >LB{\#2 default}
    >iT{*\ht}
    Q{0123456789}{0}
    K{#1\par#2\par}{{\text{#1}\over\text{#2}}}
```

and you get `\GLBTQKi` 4-argument with:

`&#1& G{&&}` optional short in a pair of `&` with `\NoValue` as the default,
`{#2} >{L}B` optional long in curly braces,
⟨*ign.*⟩`>{i}T{*\ht}` star or `\ht` control sequence,
`#3 Q{0123456789}{0}` optional sequence of decimal digits with default 0,
`#4 K{#1\par#2\par}{{\text{#1}\over\text{#2}}}` mandatory sequence of two arguments both delimited with `\par`, that will be passed the inner macro as `{\text{#1}\over\text{#2}}`.

\IfLong The arguments may be tested inside the command with `\IfLong{#`⟨*n*⟩`}{`⟨*what if long*⟩`}{`⟨*what if short*⟩`}`. The test looks for `\par` at any level of nesting (`{\par}`—`\par` in braces will not hide) since it `\detokenize`s its argument.

\global \outer If you wish to define your command `\global`ly, you can specify `\DeclareCommand%\mycommand\global`. If you wish to forbid usage of your command in arguments of macros, add the `\outer` prefix. As with original TEX's `\def` and like, the prefixes are allowed in any order and in any number only here they come between the command's name and the arguments specification. You can also add `\long`, as we mentioned above, and, for the symmetry, also `\protected`, although the latter is *always* added since the command is not expandable.

Handling of white spaces with optionals seems to me too complicated compared to the estimated weight of the problem and I haven't faced it so far so I don't provide anything. But!—but there are some commands that should be invisible in the typeset text, such as indexing commands and font declarations. For those there is a working LATEX mechanism of `\@bsphack`—`\@esphack` and to use it I provide yet another 'prefix':

iIwW \sphack if you type `W` or `w` (for 'white') or `I` or `i` (for 'invisible') or, if you prefer a prefix-like,

`\sphack`[1] between the CS to be defined and arguments spec, `\@bsphack` and `\@esphack` will be added in proper places.

The original inner macros of the ancient xparse had names like `\@dc@o` etc. According to my TEX Guru's advice I changed them to `\ArgumentCatcher@`⟨*letter(s)*⟩ to make the error messages less confusing. Well, I don't know if they are but `\ArgumentCatcher@PO` looks better than `\@dc@PO` doesn't it?

`\IfDCMessages` `\DCMessagesfalse` `\DCMessagestrue`

Talking of messages, there's a Boolean switch `\IfDCMessages`. Its default setting is `\DCMessagesfalse` but if you set `\DCMessagestrue`, every `\command` created with my `\DeclareCommand` will issue a message "Parsing arguments for `\command`" at the beginning of its execution.

`.qQ`

If you are positive that no such message will ever be useful, you can suppress the very placing of it in the command's definition with an optional argument to `\Declarecommand` with all other 'prefixes', `.` or `q` or `Q` for 'quiet'.

To sum up, `\DeclareCommand` takes the following arguments:

- `#1` `>{P}m` the command to be defined (can be even `\par` if you really wish),
- `#2` `Q{\long\global\outer\protected!lL.qQiIwW\sphack}` optional ε-TEX's prefix(es) and/or symbols for making all the arguments long (`!`, `l` or `L`) and/or to suppress placing of the diagnostic message in the definition (`.`, `q`, `Q`) and/or for placing `\@sphack`—`\@esphack` (`i`, `I`, `w`, `W`, `\sphack`),
- `#3` `>{P}m` the arguments specification (can contain `\par` as you see),
- `#4` `>{P}m` the definition body. You refer to the arguments with `#`⟨*n*⟩ and can test their presence and absence with `\If[No]Value(T|F|TF){#`⟨*n*⟩`}` and if the argument was specified with the `>P` prefix (allows `\par` in itself), you can test it with `\IfLong{#`⟨*n*⟩`}{`⟨*if long*⟩`}{`⟨*if short*⟩`}`. You are also provided the `\IfAmong...%\among` and `\IfIntersect` tests defined earlier in this package to process the arguments, especially of the `S`/`T` and `Q` type.

`\DeclareEnvironment`

There is also the `\DeclareEnvironment` command to define environments with sophisticated optionals. It takes the arguments analogous to those of `\DeclareCommand`.

The `iIwW\sphack` specifier however acts different: it doesn't add `\@bsphack` nor `\@esphack` but only `\@ignoretrue` to the end macro so the spaces following `\end{%myenvir}` will be ignored. (I tried the space hack but it's problematic ("bad space factor" error) if an environment begins in the vertical mode and ends in horizontal.

So the arguments to `\DeclareEnvironment` are:

- `#1` `>{P}m` the environment's name; you may wonder why it allows `\par`; it's to allow environments like `\string\par`—I met such an environment once.
- `#2` `Q{\long\outer\global\protected!lL.qQiIwW\sphack}` to prefix the command (note that `\outer` prefix will actually not work since the command is called with `\csname`), make all the arguments 'long' (`\long`, `!`, `l` or `L`), not to place the message issuer in the command (`.`, `q`, `Q`) or to make the environment ignores spaces following its end (`iIwW\sphack`).
- `#3` `>{P}m` the arguments specification (both for the begin and for the end macros)
- `#4` `>{P}m` the begin definition,
- `#5` `>{P}m` the end definition; it can use the same parameters (`#`⟨*n*⟩'s) as the begin definition. Note however that there is only one specification of the arguments and both begin and end have to have 9 parameters in total at most.

`\@temptokenb`

```
\unless\ifdefined\@temptokenb
\newtoks\@temptokenb
```

[1] I don't define the `\sphack` control sequence and don't assume it's defined. I use it only as a marker and my use of it doesn't create an entry in the hash table.

```
1584 \fi
```

`\if@dc@alllong@` `\if@dc@quiet@`

```
1586 \newif\if@dc@alllong@␣% for an option of all arguments long.
1587 \newif\if@dc@quiet@␣% for suppressing the message of using of declared command.
```

`\IfDCMessages`

```
1590 \newif\IfDCMessages␣% a global switch to suppress the message about parsing.
```

`\@dc@long@yes`

```
1594 \def\@dc@long@yes{P}
```

`\@dc@arguments`

```
1596 \newtoks\@dc@arguments
```

`\@dc@catchernum` `\@dc@argnum`

```
1598 \newcount\@dc@catchernum
1599 \newcount\@dc@argnum
```

`\if@dc@ignore`

```
1601 \newif\if@dc@ignore
```

```
1605 \long\pdef\DeclareCommand#1#2#3{%
       % #1 command to be defined,
       % #2 arguments specification,
       % #3 definition body.
```

`\@dc@catchernum`

```
1616   \@dc@catchernum\z@
1617   \@dc@argnum\z@
1618   \toks@{}% in this register we will store the created sequence of argument catchers.
1622   \@temptokena\toks@␣% in this register we will store the sequence of #1#2….
1623   \@temptokenb\toks@␣% in this register we will store the sequences of m's.
1627   \emptify\@dc@long@
1628   \@dc@ignorefalse
1629   \@dc#2X% X is the sentinel of parsing.
1630   \protected\edef#1{%
1631     \@nx\@dc@␣␣␣␣{\the\toks@}%
1632     \@xa\@nx\csname\string#1\endcsname
1633     \@nx␣#1%
1634   }%
1635   \edef\gmu@reserveda{%
1636     \long\def\@xa\@nx\csname\string#1\endcsname
1637     \the\@temptokena{%
1639       \unexpanded{#3}}}%
1640   \gmu@reserveda}% of the 'draft' \DeclareCommand.
```

`\@dc@`

```
1643 \long\def\@dc@
1644   #1% the argument catchers in a pair of braces (their arguments may contain \par so the macro is long).
1646   #2% \\thecommand
1647   #3% \thecommand
1648   {%
1649   \ifx\protect\@typeset@protect
1650     \@xa\@firstofone
1651   \else
1652     \protect#3\@xa\@gobble
1653   \fi
1654   {\@dc@arguments{#2}#1\the\@dc@arguments}}
```

`\@dc`

```
1657 \def\@dc#1{% The main parsing macro.
1658   \ifx#1X% if we meet the sentinel we stop. Note there may be some m's in
         % \@temptokenb and we might not to care about them since they denote
```

mandatory arguments and as such they will be grabbed by `% \\⟨declared command⟩`. However, if any of them contains `\par` (while it shouldn't), then it wouldn't be noticed since `\\/<declared␣command>` is always long. Therefore:

```
    \@dc@add@ms
  \else
    \ifx#1>%
      \let\next@dc\grab@prefix
    \else␣% not X neither >
      \if@dc@alllong@\let\@dc@long@\@dc@long@yes\fi
      \ifx#1\afterassignment\@dc@ignoretrue\fi␣% if we parse the
          assigned pseudo-argument, we don't want to add anything to the ar-
          guments' toks register so we apply general mechanism of ignoring an
          argument.
      \ifx#1\gobblespace\@dc@ignoretrue\fi␣% again, as above for ig-
          noring spaces.
      \ifnum␣\ifx#1m0\else1\fi␣\ifx\@dc@long@\@empty␣0\else␣
          2\fi
          \if@dc@ignore␣1\else0\fi␣=␣0␣% we check whether #1 is
              % m and not prefixed.
        \addto@hook\@temptokenb␣m% as you see, \@temptokenb serves
            as a storage of m's.
      \else␣% (not m) or prefixed
        \@dc@add@ms
        \if@dc@ignore\addtotoks\toks@{\@dc@ignorearg}\fi
        \@xa\addtotoks\@xa\toks@\@xa{%
          \csname␣ArgumentCatcher@\@dc@long@\detokenize{#1}\endcsname}%
              argument catcher's name is \ArgumentCatcher@>[P>]/<arg.␣
              % type>.
        \@temptokenb{}%
      \fi% of \if short m or not
      \advance\@dc@catchernum\@ne
      \unless\if@dc@ignore
        \advance\@dc@argnum\@ne
        \@temptokena\@xa{%
          \the\@xa\@temptokena\@xa##\the\@dc@argnum}%
      \else
        \@dc@ignorefalse
      \fi
      \IfAmong#1\among{ABCO}%
      {\let\next@dc\grab@default}%
      {\IfAmong#1\among{GQST}%
        {\let\next@dc\grab@defaults}% G, Q, S and T have second argu-
            ment, braced optional, which is the default value (\NoValue by
            default).
        {\gmu@if␣␣x␣{K#1}%
          {\let\next@dc\@dc@define@K}%
          {\gmu@if␣x␣{\afterassignment#1}%
            {\let\next@dc\dc@grab@afterass}%
            {\let\next@dc\@dc}%
          }%
        }%
      }%
```

```
        \unless\ifx\next@dc\@dc@define@K
          \emptify\@dc@long@
        \fi
    \fi␣% of \if#1>.
      \@xa\next@dc
    \fi
}%
```

\@dc@add@ms

```
\def\@dc@add@ms{%
    \@xa\@ifempty\@xa{\the\@temptokenb}%
    {}%
    {\toks@\@xa{%
        \the\@xa\toks@
        \csname␣ArgumentCatcher@\the\@temptokenb\endcsname}}}
```

\grab@default

```
\long\def\grab@default#1{%  the default value or the list of tokens for S may
                            contain \par (why not?), that's why we make this macro \long.
    \toks@\@xa{\the\toks@{#1}}%
    \let\next@dc\@dc
    \@dc
}
```

\grab@defaults

```
\long\def\grab@defaults#1{%  default (when default is absent)
    \toks@\@xa{\the\toks@{#1}}%
    \@ifnextchar\bgroup
    {\grab@default}%
    {\grab@default{\NoValue}}%
}
```

\dc@grab@afterass

```
\def\dc@grab@afterass{%
    \@ifnextchar\bgroup
    {\grab@default}%
    {\grab@default{\@tempcnta}}%
}
```

\@dc@addargum

```
\long\def\@dc@addargum{%
    \addtotoks\@dc@arguments}

\StoreMacro\@dc@addargum
```

\@dc@ignorearg
\@dc@addargum

```
\pdef\@dc@ignorearg{%
    \long\def\@dc@addargum##1{%
      \RestoreMacro\@dc@addargum}}
```

Parsing of Knuthian parameters string is easier to implement with full-feature \DeclareCommand so we postpone it till line 2363

\grab@prefix
\@dc@long@

```
\def\grab@prefix#1{%
    \IfIntersect{#1}{\long!lL\par␣Pp}%
    {\def\@dc@long@{P}}{}%
    \IfIntersect{#1}{Ii}%
    {\@dc@ignoretrue}{}%
    \@dc
}
```

\ArgumentCatcher@o

```
\long\def\ArgumentCatcher@o#1\@dc@arguments{%  parsing of an optional
                                               with no default
    \@ifnextchar[%
```

```
\@dc@parse@next 1773    {\def\@dc@parse@next{#1}\ArgumentCatcher@o@}% the last macro has
                            to be short so we can't give it all the arguments' catchers since some de-
                            fault may contain \par. Therefore we hide all the argument catchers in
                            % \@dc@parse@next.
                1778    {\@dc@addargum\NoValue#1\@dc@arguments}}

\ArgumentCatcher@o@ 1780 \def\ArgumentCatcher@o@[#1]{%
                1781    \@dc@addargum{{#1}}\@dc@parse@next\@dc@arguments}

\ArgumentCatcher@Po 1783 \long\def\ArgumentCatcher@Po#1\@dc@arguments{% parsing of an optional
                            with no default.
                1784    \@ifnextchar[%
                1785    {\ArgumentCatcher@Po@{#1}}%
                1786    {\@dc@addargum\NoValue#1\@dc@arguments}}

\ArgumentCatcher@Po@ 1788 \long\def\ArgumentCatcher@Po@#1[#2]{%
                1789    \@dc@addargum{{#2}}#1\@dc@arguments}

\ArgumentCatcher@O 1791 \long\def\ArgumentCatcher@O#1#2\@dc@arguments{% parsing of optional
                            with default value.
                            % #1 the default value,
                            % #2 the tail of args parser.
                1797    \@ifnextchar[%
\@dc@parse@next 1798    {\def\@dc@parse@next{#2}\ArgumentCatcher@o@}% note that even when
                            an optional is short, its default may contain \par.
                1800    {\@dc@addargum{{#1}}#2\@dc@arguments}}

\ArgumentCatcher@PO 1803 \long\def\ArgumentCatcher@PO#1#2\@dc@arguments{% parsing of optional
                            with default value.
                            % #1 the default value,
                            % #2 the tail of args parser.
                1809    \@ifnextchar[%
                1810    {\ArgumentCatcher@Po@#2}% note that even when an optional is short, its de-
                            fault may contain \par.
                1812    {\@dc@addargum{{#1}}#2\@dc@arguments}}

\ArgumentCatcher@b 1815 \long\def\ArgumentCatcher@b#1\@dc@arguments{% parsing of an optional
                            braced short argument.
                1816    \@ifnextcat\bgroup
\@dc@parse@next 1817    {\def\@dc@parse@next{#1}\ArgumentCatcher@m@}% the last macro has
                            to be short so we can't give it all the arguments' parsers since some de-
                            fault may contain \par. Therefore we hide all the argument catchers in
                            % \@dc@parse@next.
                1821    {\@dc@addargum\NoValue#1\@dc@arguments}}

\ArgumentCatcher@Pb 1824 \long\def\ArgumentCatcher@Pb#1\@dc@arguments{% parsing of a braced
                            and long optional with \NoValue default.
                1826    \@ifnextcat\bgroup
                1827    {\ArgumentCatcher@Pm#1\@dc@arguments}%
                1828    {\@dc@addargum\NoValue#1\@dc@arguments}%
                1829 }

\ArgumentCatcher@B 1831 \long\def\ArgumentCatcher@B#1#2\@dc@arguments{% parsing of a braced
                        optional with default value.
                        % #1 the default value,
                        % #2 the tail of args catchers.
```

```
1837    \@ifnextcat\bgroup
\@dc@parse@next 1838    {\def\@dc@parse@next{#2}\ArgumentCatcher@m@}% note that even when
                        an optional is short, its default may contain \par.
1840    {\@dc@addargum{{#1}}#2\@dc@arguments}}

\ArgumentCatcher@PB 1842 \long\def\ArgumentCatcher@PB#1#2\@dc@arguments{% parsing of a braced
                        and long optional with default value.

                    #1 the default value,
                    #2 the tail of args catchers.
1849    \@ifnextcat\bgroup
1850    {\ArgumentCatcher@Pm#2\@dc@arguments}%
1851    {\@dc@addargum{{#1}}#2\@dc@arguments}}

\ArgumentCatcher@S@ 1854 \long\def\ArgumentCatcher@S@% parsing of a Single continued:
1855 #1% the list to search #4 on,
1856 #2% the default value,
1857 #3% the tail of args parsers,
1858 #4% the token we search in #1 (it's an unbraced single token as we checked in line
                        1873).
1860 {%
1861    \IfAmong#4\among{#1}%
1862    {\@dc@addargum{{#4}}#3\@dc@arguments}%
1863    {\@dc@addargum{{#2}}#3\@dc@arguments#4}%
1864 }

\ArgumentCatcher@S 1866 \long\def\ArgumentCatcher@S␣% parsing of a Single token. It's always long
                        since we look for a single unbraced token so there is no danger of "runaway
                        argument".
1869 #1% the list we search optional token on,
1870 #2% the default value,
1871 #3% the tail of args parser.
1872 \@dc@arguments{%
1873    \@ifnextsingle
1875    {\ArgumentCatcher@S@{#1}{#2}{#3}}%
1876    {\@dc@addargum{{#2}}#3\@dc@arguments}% if we parse an opening or
                        closing brace, then we are sure it's not any expected Single.
1879 }

1882 \let\ArgumentCatcher@PS\ArgumentCatcher@S␣% as above: we always act
                        'long' with single tokens.
1883 \let\ArgumentCatcher@T\ArgumentCatcher@S␣% we allow T (for Token) as
                        an alias of S.

1887 \edef\ArgumentCatcher@s{%
1888    \@nx\ArgumentCatcher@S{\@xa\@nx\all@stars}{\@nx\NoValue}}% the
                        s arg spec is a shorthand for S{*}. Except the star may be of any category
                        from {11, 12, 13}.
1891 \let\ArgumentCatcher@Ps\ArgumentCatcher@s

\ArgumentCatcher@Q 1893 \long\def\ArgumentCatcher@Q% parsing of a seQuence of tokens from the list.
                        It's always long since we look for unbraced tokens so there is no danger of
                        "runaway argument".
1896 #1% the list we search optional tokens on,
1897 #2% the default value,
1898 #3% the tail of args parser.
```

```
\@dc@arguments␣% delimiter
{%
  \def\@dc@parse@next{#3}%
  \emptify\@dc@seQuence
  \ArgumentCatcher@Q@{#1}{#2}}
\let\ArgumentCatcher@PQ\ArgumentCatcher@Q
\long\def\ArgumentCatcher@Q@#1#2{% list, default
  \@ifnextsingle
  {\ArgumentCatcher@Q@@{#1}{#2}}%
  {\@dc@seQuence@finish{#2}}%
}
\long\def\ArgumentCatcher@Q@@#1#2#3{% list, default, token to be checked
```

we only launch this macro when not before opening curly brace. `#1` is the list of acceptable values, `#2` the default value, `#3` is the token to be sought on the list.

```
  \IfAmong␣#3\among{#1}%
  {\addtomacro\@dc@seQuence#3%
    \ArgumentCatcher@Q@{#1}{#2}}%
  {\@dc@seQuence@finish{#2}#3}%
}
\long\def\@dc@seQuence@finish#1{%
  \ifx\@dc@seQuence\@empty\def\@dc@seQuence{#1}\fi
  \@xa\@dc@addargum\@xa{\@xa{\@dc@seQuence}}%
  \@dc@parse@next\@dc@arguments␣}
\long\def\ArgumentCatcher@c#1\@dc@arguments{%
  \@ifnextchar(%
  {\def\@dc@parse@next{#1}\ArgumentCatcher@c@}%
  {%
    \@dc@addargum{{\NoValue}}#1\@dc@arguments}%
}
\def\DCcoordinate{%
  \def\ArgumentCatcher@c@(##1){%
    \IfAmong,\among{##1}%
    {\@xa\@dc@addargum\@dc@commasep##1\@dc@commasep}%
    {\@dc@addargum{{##1}}}\@dc@parse@next\@dc@arguments}%
  \def\@dc@commasep##1,##2\@dc@commasep{{{{##1}{##2}}}}%
  \long\def\ArgumentCatcher@Pc@(##1){%
  \IfAmong,\among{##1}%
  {\@xa\@dc@addargum\@dc@commasep##1\@dc@commasep}%
  {\@dc@addargum{{##1}}}\@dc@parse@next\@dc@arguments}%
}
\def\DCnocoordinate{%
  \def\ArgumentCatcher@c@(##1){%
    \@dc@addargum{{##1}}\@dc@parse@next\@dc@arguments}%
  \long\def\ArgumentCatcher@Pc@(##1){%
    \@dc@addargum{{##1}}\@dc@parse@next\@dc@arguments}%
}
\DCnocoordinate
```

```
\ArgumentCatcher@C
\@dc@parse@next
\long\def\ArgumentCatcher@C#1#2\@dc@arguments{%
  \@ifnextchar(%
  {\def\@dc@parse@next{#2}\ArgumentCatcher@c@}%
  {\@dc@addargum{{#1}}#2\@dc@arguments}}

\ArgumentCatcher@Pc
\@dc@parse@next
\long\def\ArgumentCatcher@Pc#1\@dc@arguments{%
  \@ifnextchar(%
  {\def\@dc@parse@next{#1}\ArgumentCatcher@Pc@}%
  {\@dc@addargum{{\NoValue}}#1\@dc@arguments}%
}

\ArgumentCatcher@PC
\@dc@parse@next
\long\def\ArgumentCatcher@PC#1#2\@dc@arguments{%
  \@ifnextchar(%
  {\def\@dc@parse@next{#2}\ArgumentCatcher@Pc@}%
  {\@dc@addargum{{#1}}#2\@dc@arguments}}

\ArgumentCatcher@Pm
\long\def\ArgumentCatcher@Pm#1\@dc@arguments#2{%
  \@dc@addargum{{#2}}#1\@dc@arguments}

\gmu@hashes
\def\gmu@hashes#1#2{%  this is a fully expandable loop analogous to that of The
                       ε-TEX Manual p. 9.
  \ifnum#1<#2%
  ####\number#1
  \expandafter\gmu@hashes
  \expandafter{\number\numexpr#1+1\expandafter}%
  \expandafter{\number#2\expandafter}%
  \fi}% of \gmu@hashes.

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

\@tempcnta=1
\@whilenum\@tempcnta<9\do{%
  \edef\gmu@ms{\romannumeral\numexpr\@tempcnta*1000␣}%
  \edef\gmu@tempa{%
    \long\def\@xa\@nx\csname
       ArgumentCatcher@\gmu@ms
    \endcsname
    ##1\@nx\@dc@arguments{\def\@nx\@dc@parse@next{##1}%
      \@xa\@nx\csname
        ArgumentCatcher@\gmu@ms␣@%
      \endcsname}%
    \def\@xa\@nx\csname
    ArgumentCatcher@\gmu@ms␣@%
    \endcsname
    \gmu@hashes1{\numexpr\@tempcnta+1}%
    {\@nx\@dc@addargum{%
        \gmu@hashesbraced1{\numexpr\@tempcnta+1}}%
      \@nx\@dc@parse@next\@nx\@dc@arguments
    }% of \ArgumentCatcher@⟨m's⟩@.
  }% of \edef.
```

```
  \gmu@tempa
  \advance\@tempcnta1\relax
}% of the loop.
```

The loop above defines 8 pairs of macros as we see thanks to the code below:

```
\if␣1␣1
\@tempcnta\@ne
\@whilenum\@tempcnta<9\do{%
  \edef\gmu@ms{\romannumeral\numexpr\@tempcnta*1000}%
  \typeout{\bslash␣␣ArgumentCatcher@\gmu@ms:%
    ^^J\@xa\meaning\csname␣ArgumentCatcher@\gmu@ms%
        \endcsname}%
  \typeout{\bslash␣␣ArgumentCatcher@\gmu@ms␣@:%
    ^^J\@xa\meaning\csname␣ArgumentCatcher@\gmu@ms␣@%
        \endcsname}%
  \advance\@tempcnta\@ne
}
\fi
```

The code above produces on the terminal:

```
\ArgumentCatcher@m:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@m@
\ArgumentCatcher@m@:
macro:#1->\addto@hook \@dc@arguments {{#1}}\@dc@parse@next
    \@dc@arguments
\ArgumentCatcher@mm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mm@
\ArgumentCatcher@mm@:
macro:#1#2->\addto@hook \@dc@arguments
    {{#1}{#2}}\@dc@parse@next \@dc@arguments
\ArgumentCatcher@mmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmm@
\ArgumentCatcher@mmm@:
macro:#1#2#3->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}}\@dc@parse@next \@dc@arguments
\ArgumentCatcher@mmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmm@
\ArgumentCatcher@mmmm@:
macro:#1#2#3#4->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}}\@dc@parse@next \@dc@arguments
\ArgumentCatcher@mmmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmm@
\ArgumentCatcher@mmmmm@:
macro:#1#2#3#4#5->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}}\@dc@parse@next \to
ks@
\ArgumentCatcher@mmmmmm:
```

```
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmmm@
\ArgumentCatcher@mmmmmm@:
macro:#1#2#3#4#5#6->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}{#6}}\@dc@parse@ne
xt \@dc@arguments
\ArgumentCatcher@mmmmmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmmmm@
\ArgumentCatcher@mmmmmmm@:
macro:#1#2#3#4#5#6#7->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}{#6}{#7}}\@dc@pa
rse@next \@dc@arguments
\ArgumentCatcher@mmmmmmmm:
macro:#1\@dc@arguments ->\def \@dc@parse@next
    {#1}\ArgumentCatcher@mmmmmmmm@
\ArgumentCatcher@mmmmmmmm@:
macro:#1#2#3#4#5#6#7#8->\addto@hook \@dc@arguments
    {{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}}\@
dc@parse@next \@dc@arguments
```

\ArgumentCatcher@mmmmmmmmm

```
\def\ArgumentCatcher@mmmmmmmmm\the\@dc@arguments
#1#2#3#4#5#6#7#8#9{% yes, it has to be delimited (well, actually: started) by % \the\@dc@arguments not only \@dc@arguments. And it doesn't need to be long and launch inner short macro because there's nothing more to catch.
  \@dc@addargum{%
    {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}\the\@dc@arguments}
```

\ArgumentCatcher@Pm

```
\long\def\ArgumentCatcher@Pm#1\@dc@arguments#2{% catcher of a long mandatory.
  \@dc@addargum{{#2}}#1\@dc@arguments}
```

\ArgumentCatcher@K

```
\long\def\ArgumentCatcher@K␣% a Knuthian delimited or undelimited arguments parsed and passed to the command's body in a digest (Knuthian replacement)
#1% the particular Knuthian macro
#2% the tail of args parser.
\@dc@arguments␣% tail delimiter
```

\@dc@parse@next

```
{\def\@dc@parse@next{#2}#1}

\let\ArgumentCatcher@PK\ArgumentCatcher@K
```

\gmu@twostring

```
\long\def\gmu@twostring#1#2{\string#1\string#2}
```

\ArgumentCatcher@G

```
\long\def\ArgumentCatcher@G␣% general catcher
#1% left and right delimiter
#2% default value
#3% the tail of args parser.
\@dc@arguments␣% tail delimiter
{%
  \edef\@dc@Gname{ArgumentCatcher@G\gmu@twostring#1}%
  \unless\ifcsname␣\@dc@Gname␣\endcsname
    \@xa\def\csname␣\@dc@Gname\@xa\endcsname
    \@dc@Gparams#1{\@dc@addargum{{##2}}##1\@dc@arguments}%
  \fi
  \@xa\@ifnextchar\@firstoftwo#1%
```

```
\dc@parse@next
    {\def\dc@parse@next{#3}% we hide possible \pars.
      \csname\@dc@Gname\endcsname\dc@parse@next}%
    {\@dc@addargum{{#2}}#3\@dc@arguments}%
}

\ArgumentCatcher@PG
\long\def\ArgumentCatcher@PG␣% general catcher
#1% left and right delimiter
#2% default value
#3% the tail of args parser.
\@dc@arguments␣% tail delimiter
{%
  \edef\@dc@Gname{ArgumentCatcher@PG\gmu@twostring#1}%
  \unless\ifcsname␣\@dc@Gname␣\endcsname
    \long\@xa\def\csname␣\@dc@Gname\@xa\endcsname
    \@dc@Gparams#1{\@dc@addargum{{##2}}##1\@dc@arguments}%
  \fi
  \@xa\@ifnextchar\@firstoftwo#1%
  {\csname\@dc@Gname\endcsname{#3}}%
  {\@dc@addargum{{#2}}#3\@dc@arguments}%
}

\@dc@Gparams
\long\def\@dc@Gparams#1#2{##1#1##2#2}% expands to parameters string with
    % #1 delimited with first argument and #2 delimited with second. In fact it's
    intended to pass #1 further (it will always be braced) and to catch an argument
    put in a pair of tokens given \@dc@Gparams as the arguments.

\ArgumentCatcher@A
\long\def\ArgumentCatcher@A{\ArgumentCatcher@G{<>}}
\ArgumentCatcher@PA
\long\def\ArgumentCatcher@PA{\ArgumentCatcher@PG{<>}}

\ArgumentCatcher@a
\long\def\ArgumentCatcher@a{\ArgumentCatcher@G{<>}{\NoValue}}
\ArgumentCatcher@Pa
\long\def\ArgumentCatcher@Pa{\ArgumentCatcher@PG{<>}{%
    \NoValue}}

\NoValue
\def\NoValue{-NoValue-}

\IfNoValueTF
\long\def\IfNoValueTF#1{%
  \@xa\ifx\@firstofmany#1\@empty\@@nil\NoValue␣\afterfi%
      \@firstoftwo
  \else\afterfi\@secondoftwo
  \fi}

\IfNoValueT
\long\def\IfNoValueT#1#2{\IfNoValueTF{#1}{#2}\@empty}

\IfNoValueF
\long\def\IfNoValueF#1#2{\IfNoValueTF{#1}\@empty{#2}}

\IfValueTF
\long\def\IfValueTF#1#2#3{\IfNoValueTF{#1}{#3}{#2}}

\PutIfValue
\long\def\PutIfValue#1{\IfValueT{#1}{#1}}

\IfValueT
\let\IfValueT\IfNoValueF

\IfValueF
\let\IfValueF\IfNoValueT

\IfLong
\long\pdef\IfLong#1{%
    % #1 the argument to check for \par,
    % #2 what if \par found,
    % #3 what if \par not found.
```

```
\edef\gmu@reserveda{\detokenize{#1}}%
\edef\gmu@reserveda{%
  \IfAmong\string\par\@nx\among{\gmu@reserveda}}%
\gmu@reserveda}
```

`\afterassignment` pseudo-argument

```
\long\@namedef{ArgumentCatcher@\detokenize{%
    \afterassignment}}%
#1% register used in the assignment
#2% the tail of args parser.
\@dc@arguments{%
```

`\@dc@parse@next`

```
  \def\@dc@parse@next{%
    \@dc@addargum\NoValue␣␣% to make ignoring mechanism work
    #2\@dc@arguments}%
  \afterassignment\@dc@parse@next
  #1}% optional space and = is left at user's discretion. In fact, #1 may be empty
      and then each time our command is used the left side of the assignment has
      to be given explicitly and may even be different each time.

\n@melet{ArgumentCatcher@P\detokenize{\afterassignment}}
{ArgumentCatcher@\detokenize{\afterassignment}}
```

`\gobblespace` pseudo-argument: useful between optional and Knuthian-type arguments and after all parsing. Equivalent `>iT{\somedummymacro}`. At first I wanted mark it `\ignorespaces` but the gobbling doesn't expand macros: it doesnt't use `\ignorespaces` but `\@ifnextchar`'s space trick.

```
\long\@namedef{ArgumentCatcher@\detokenize{\gobblespace}}%
{\ArgumentCatcher@S{\dc@gobblespace@dummy}{\NoValue}}

\n@melet{ArgumentCatcher@P\detokenize{\gobblespace}}
{ArgumentCatcher@\detokenize{\gobblespace}}
```

`\dc@gobblespace@dummy`

```
\def\dc@gobblespace@dummy{\dc@gobblespace@dummy}

\let\gobblespace\dc@gobblespace@dummy
```

end of catchers' definitions

`\DeclareCommand`

```
\DeclareCommand\DeclareCommand␣␣␣% This is the final version by now. Note
    we introduce the 'prefixes' at the #2 position only at this point so we have yet
    to prefix every argument specifier separately. Note also we only here 'teach'
    \DeclareCommand to pass the name of its main argument in the \dc@bsname
    macro.
{>Pm␣␣␣␣␣␣␣␣␣␣␣␣␣␣% (1) command to be defined (may be \par if you really
    wish),
  Q{\global\protected\outer\long!lL.qQiIwW\sphack\envhack}% (2)
    an optional seQuence of ε-TEX's prefixes and/or ! or l or L for 'all long'
    and/or . or q or Q for 'quiet' (message about calling the command is
    suppressed) and/or i or I as 'invisible' or W or w as 'white' or \sphack
    to make the command 'invisible' in the leading with the \@bsphack...
    % \@esphack[2],
  >Pm␣% (3) arguments specification (may contain \par),
```

2 To deal with an 'invisible' environment, there is one more acceptable value of this argument: `\envhack`, which suppresses placing `\@bsphack`—`\@esphack` and places `\@ignoretrue` instead in the end macro.

```
>Pm␣%
```
(4) the definition body; may contain nearly anything including `\par` and tests for presence/absence of arguments `% \If[No]Value(T|F|TF)` and for their longness `\IfLong`; you refer to the arguments with `#`⟨*n*⟩.

```
}{%
  \edef\dc@bsname{\string#1}%
```
we'll use it in K-type arg. catcher and to allow `#1 == \par` (`\PackageWarning` is short and `\typeout` too!).

```
  \gm@testdefined#1\iftrue
    \IfDCMessages
      \PackageWarning{gmutils␣(gmparse)}{Redefining␣command␣
          \dc@bsname\space
        with␣\string\DeclareCommand}%
    \fi
  \else
    \IfDCMessages
```
`\dc@bsname`
```
      \typeout{^^J\string\DeclareCommand\dc@bsname\space␣
          (new)^^J}%
    \fi
  \fi
```

First we parse the declaration options to know whether all the arguments are to be long and whether we should suppress the message "Parsing arguments for…".

```
  \@dc@alllong@false
  \@dc@quiet@false
  \IfIntersect{#2}{\long!lL}{%
    \@dc@alllong@true}{}%
  \IfIntersect{#2}{.qQ}{\@dc@quiet@true}{}%
  \IfIntersect{#2}{iIwW\sphack}%
  {% if 'invisibility' is specified:
    \def\@dc@bsphack@{\@nx\@bsphack}%
    \def\@dc@esphack@{\@nx\@esphack}%
  }%
  {\emptify\@dc@bsphack@\emptify\@dc@esphack@}%
```
if 'invisibility' is not specified.

```
  \IfAmong\envhack\among{#2}%
  {% but if we define an environment:
    \emptify\@dc@bsphack@\emptify\@dc@esphack@}%
  {% and if we don't define an environment:
    \emptify\@dc@env@argstoend}%
  \relaxen\@dc@global@␣\relaxen\@dc@outer@
  \IfAmong\outer\among{#2}{\let\@dc@outer@\outer}{}%
  \IfAmong\global\among{#2}{\let\@dc@global@\global}{}%
```

We initialise the scratch count and toks registers before parsing of the arguments specifiers.

```
  \@dc@catchernum\z@
  \@dc@argnum\z@
  \toks@{}%
  \@temptokena\toks@
  \@temptokenb\toks@
```

We parse the arguments specifiers:

```
  \@dc#3X%
```
`X` is the sentinel of parsing.

Now we define the basic macro:

```
    \@dc@outer@\@dc@global@␣% possibly the prefixes,
    \protected\edef#1{% always \protected ;-)
      \@dc@bsphack@␣% perhaps \@bsphack
      \unless\if@dc@quiet@
        \unexpanded{%
          \IfDCMessages
          \typeout}{Parsing␣arguments␣for␣\dc@bsname.}%
          \@nx\fi
      \fi
      \@nx\@dc@{\the\toks@}% in the braces appear the argument catchers.
      \@xa\@nx\csname\dc@bsname\endcsname
      \ifx\@dc@outer@\outer
        \@xa\@nx\csname␣\@xa\gobble\string#1␣\endcsname␣% note the
              blank space after #1! If the command is to be \outer, we can't put
              it itself, so we put another, whose name is the same plus a space.
              Anyway, since #1 becomes \protected, this case will never be ex-
              ecuted.
        \else␣\@nx␣#1%
        \fi
    }%
    \edef\gmu@reserveda{%
      \@dc@global@␣% perhaps \global
      \long\def\@xa\@nx\csname\string#1\endcsname% for
            a command \command, define \\command
      \the\@temptokena% it's #1#2#3...
      {% the definition body:
        \@dc@env@argstoend␣% if we define an environment, this macro will
              provide passing the arguments to the end macro (see line 2409).
        \unexpanded{#4}%
        \@dc@esphack@}%
    }% of \edef.
    \gmu@reserveda
}% of \DeclareCommand.
```

Now we have a convenient tool to implement parsing of a Knuthian argument(s).

Knuthian argument's default replacement

\@dc@K@defaultrepl

```
\def\@dc@K@defaultrepl{##1}
```

2009/11/22 inner pair of braces removed (as leading to additional group causing errors).

\@dc@define@K

```
\DeclareCommand\@dc@define@K\long{mb}{% first we add the particular CS
      to the toks register:
    \edef\gmu@tempa{%
      \unexpanded{\toks@\@xa}%
      {\@nx\the\toks@{%
            \@xa\@nx\csname\dc@bsname␣@K\the\@dc@catchernum%
                  \endcsname}}%
    }\gmu@tempa
```

and define this macro:

```
    \edef\gmu@tempa{%
      \def\@xa\@nx\csname\dc@bsname␣@K\the\@dc@catchernum%
            \endcsname
```

```
      \unexpanded{#1}{%
      \@nx\@dc@addargum{{%
           \IfValueTF{#2}{\unexpanded{#2}}{\@xau{%
                \@dc@K@defaultrepl}}%
        }}% we add the Knuthian replacement to the toks register as #n.
      \@nx\@dc@parse@next\@dc@arguments}% and continue parsing.
  }% of \edef. Now we execute this temporary macro, which means we \def\⟨command⟩@K⟨n
  \@dc@global@␣% set properly before \@dc#3X.
  \if␣P\@dc@long@\relax\long\fi
  \gmu@tempa
```

Now we clean up and continue construction of arguments parser.

```
  \emptify\@dc@long@
  \@dc
}
```

\DeclareEnvironment

```
\DeclareCommand\DeclareEnvironment\long
{m% Pm the environment's name; you may wonder why it allows \par. Once I met
      an environment with a name containing
      % \string\par.
    Q{\outer\global\protected\long!lL.qQiIwW\sphack\envhack}% a se-
        quence of ε-TEX's prefixes and/or specifiers to make all the arguments
        'long' ( \long, !, l or L) and/or not to place the message issuer in the
        command ( ., q, Q) and/or to make the environment ignore spaces fol-
        lowing its end (i, I, w, W or \sphack),
  m␣␣␣% the arguments specification (both for the begin and the end definitions),
 m␣% the begin definition,
 m␣% the end definition; it can contain any #⟨n⟩—the arguments of the environ-
      ment are passed to it, too.
}{%
```

\@dc@env@argstoend

```
  \def\@dc@env@argstoend{%
    \edef
    \@xa\@nx\csname␣\bslash#1(arguments)\endcsname
    {\unexpanded{%
         \@xa\@xa\@xa\unexpanded\@xa\@xa\@xa{\@xa\@gobble%
              \the\@dc@arguments}%
      }% of outer \unexpanded,
    }% of \edef's body.
```

Note that \@dc@env@argstoend will be put in an \edef and the arguments bearer it sets will be defined as custom for this particular environment (its name contains the name of the environment) and inside the environment's group and will bear a list of tokens {⟨*arg.1*⟩} {⟨*arg.2*⟩} … not the CS \@dc@arguments so it's robust to nested environments and even to \DeclareEnvironment's occurrences inside the environment.

```
  }% of \@dc@env@argstoend.

  \def\gmu@reserveda{\envhack}% we add the information we define an en-
      vironment.
  \IfValueT{#2}%
  {\addtomacro\gmu@reserveda{#2}}% we pass all the 'prefixes' to \Decl¦
      % areCommand

  \@xa\DeclareCommand\csname#1\@xa\endcsname
  \gmu@reserveda{#3}{#4}%
```

Now the begin definition is done. Let's get down to the end definition.

```
    \@dc@global@␣% note this CS was for sure redefined by the last \Declare|
        % Command (and by nothing else) and that's exactly what we want.
    \@nameedef{end#1}% It's \edef inside.
    {\@nx\@xa\@xa\@nx\csname\bslash␣end#1\endcsname
      \@xa\@nx\csname␣\bslash#1(arguments)\endcsname}%
```

We `\edef` the `\end`⟨*env. name*⟩ macro to be

`\expandafter\\end`⟨*env. name*⟩`\\`⟨*env. name*⟩`(arguments)`

(it's `\expandafter` and two control sequences, the latter with `(arguments)` in its name).

```
    \IfIntersect{#2}{iIwW\sphack}%
    {\@dc@global@\@xa\addtomacro
      \csname␣end#1\endcsname{\@ignoretrue}}{}%
    \edef\gmu@reserveda{%
      \@dc@global@\long\def␣% the inner end macro is always \long and
          perhaps defined \globally.
      \@xa\@nx\csname\bslash␣end#1\endcsname
      \the\@temptokena% it's #1#2#3...—the inner end macro takes the same
          number of parameters as the begin macro.
      {% the definition body
        \unexpanded{#5}}%
    }% of \edef.
    \gmu@reserveda
}
```

Ampulex Compressa-like modifications of macros

Ampulex Compressa is a wasp that performs brain surgery on its victim cockroach to lead it to its lair and keep alive for its larva. Well, all we do here with the internal LaTeX macros resembles Ampulex's actions but here is a tool for a replacement of part of macro's definition.

The `\ampulexdef` command takes its `#2` which has to be a macro and replaces part of its definition delimited with `#5` and `#6` with the replacement `#7`. The redefinition may be prefixed with `#1`. `#2` may have parameters and for such a macro you have to set the parameters string and arguments string (the stuff to be taken by the one-step expansion of the macro) as the optional `[#3]` and `[#4]`. . If `\ampulexdef` doesn't find the start and end tokens in the meaning of the macro, it does nothing to it. You have to write `####` instead of `#` or you can use `\ampulexhash` as well. For an example use see line 3366.

`\ampulexdef`

```
\DeclareCommand\ampulexdef\long
{Q{\outer\long\global\protected}␣% (1) (optional) prefix(es); allowed is
    any sequence of them in any order, just like for the original TeX's \def.
  S{\def\edef\gdef\xdef\pdef}␣% (2) (optional) kind of definition; if not
    specified, \def will be used.
  m␣% (3) macro to be redefined,
  O{}␣% (4) \def's parameters string; empty by default,
  O{}␣% (5) definition body's parameters to be taken in a one-step expansion of
    the redefined macro; empty by default,
  m␣% (6) start token(s),
  m␣% (7) end token(s)
  m␣% (8) the replacement
```

```
} {%   For the example of usage see 3366.
  \def\gmu@tempa{#6}%
  \def\gmu@tempb{#7}%
  \def\gmu@tempc{#8}% we wrap the start, end and replacement tokens in
      macros to avoid unbalanced \ifs.
  \edef\gmu@tempd{%
    \long\def\@nx\gmu@tempd
    ####1\detoken@xa\gmu@tempa
    ####2\detoken@xa\gmu@tempb
    ####3\@nx\gmu@tempd{%
      \unexpanded{%
        \@ifempty{##3}{\hidden@iffalse}{\hidden@iftrue}}}}%
  \gmu@tempd% it defines \gmu@tempc to produce an open \if depending on
      whether the start and end token(s) are found in the meaning of #3.
  \edef\gmu@tempe{%
    \@nx\gmu@tempd\all@other#3% note that #3 may have parameters so we
        have to look at its meaning not at its one-level expansion, which could
        produce an 'extra \}' error.
    \detoken@xa\gmu@tempa
    \detoken@xa\gmu@tempb\@nx\gmu@tempd
  }%
  \gmu@tempe% we apply the checker and it produces an open \if.
  \edef\gmu@tempd{%
    \long\def\@nx\gmu@tempd
    ####1\@xa\unexpanded\@xa{\gmu@tempa}%
    ####2\@xa\unexpanded\@xa{\gmu@tempb}%
    ####3\@nx\ampulexdef{% we define a temporary macro with the param-
        eters delimited with the 'start' and 'end' parameters of \ampulexdef.
      \@nx\unexpanded{####1}%
      \@nx\@xa\@nx\unexpanded
      \@nx\@xa{\@nx\gmu@tempc}% we replace the part of the redefined macro's
          meaning with the replacement text.
      \@nx\unexpanded{####3}}}%
  \gmu@tempd
  \edef\gmu@tempf{#5}%
  \edef\gmu@tempe{%
    \IfValueT{#1}{#1}\IfValueTF{#2}{#2}{\def}%
    \@nx#3#4{%
      \@xa\@xa\@xa\gmu@tempd\@xa#3\gmu@tempf\ampulexdef}}%
  \gmu@tempe
  \fi}
```

\ampulexhash
```
\def\ampulexhash{####}% for your convenience (not to count the hashes).
```

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

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

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

\nameshowthe
```
\newcommand\nameshowthe[1]{\@xa\showthe\csname#1\endcsname}
```

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

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

`\xiistring`
```
\long\def\xiistring#1{%
  \if\@nx#1\xiispace
    \xiispace
  \else
    \afterfi{\string#1}% to make the same error as bare \string would
         cause in case of empty #1.
  \fi}
```

The next macro is applied to a `\detokenized` nonempty string to convert the spaces into 'other'.

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

Environments redefined

Almost an environment or redefinition of `\begin`

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

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

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

Original LaTeX's `\begin`:

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

`\@begnamedgroup`
```
\long\def\@begnamedgroup#1{%
  \edef\@prevgrouplevel{\the\currentgrouplevel}% added 2009/03/24
       to handle special pseudo-environments that don't increase \current|
       grouplevel(such as document). Note it's \edefed outside the environ-
       ment's group.
  \@ignorefalse% not to ignore blanks after group
  \begingroup\@endpefalse
```

```
\edef\@prevenvir{\@currenvir}%
```
Note we `\edef` it inside the group (for obvious reason), unlike the 'previous' grouplevel.
```
\edef\@currenvir{#1}%
```
We could do recatcoding through `\string` or `% \detokenize` but all the name 'other' and 10 could affect a thousand packages so we don't do that and we'll recatcode in a testing macro, see line 2748.
```
\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`.)

Let us make it the starred version of `\begin`.

\begin* \begin
```
\def\begin{\gm@ifstar{\@begnamedgroup}{%
        \@begnamedgroup@ifcs}}
```

\@begnamedgroup@ifcs
```
\def\@begnamedgroup@ifcs#1{%
  \ifcsname#1\endcsname\afterfi{\@begnamedgroup{#1}}%
  \else\afterfi{\@latex@error{Environment␣#1␣undefined}%
        \@eha}%
  \fi}%
```

`\@ifenvir` and improvement of `\end`

It's very clever and useful that `\end` checks whether its argument is `\ifx`-equivalent `\@currenvir`. However, in standard LATEX it works not quite as I would expect: Since the idea of environment is to open a group and launch the CS named in the `\begin`'s argument. That last thing is done with `\csname…\endcsname` so the catcodes of chars are irrelevant (until they are `\active`, 1, 2 etc.). Thus should be also in the `\end`'s test and therefore we ensure the compared texts are both expanded and made all 'other'.

First a (not expandable) macro that checks whether current environment is as given in `#1`. Why is this macro `\long`?—you may ask. It's `\long` to allow environments such as `\string\par`.

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

\@ifedetokens
```
\long\pdef\@ifedetokens#1#2{%
```
```
% #1 first list of tokens to be expanded and detokenized
% #2 second list
% #3 if agree
% #4 else
```
```
  \edef\gmu@tempa{#1}%
  \edef\gmu@reserveda{\@xa\detokenize\@xa{\gmu@tempa}}%
```
with our brave new `\begin`, `\@currenvir` is fully expanded, remember?
```
  \edef\gmu@tempa{#2}%
  \edef\gmu@reservedb{\@xa\detokenize\@xa{\gmu@tempa}}%
  \ifx\gmu@reserveda\gmu@reservedb\@xa\@firstoftwo
```

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

```
    \else\@xa\@secondoftwo
    \fi}
\@ifjobname \def\@ifjobname#1{\@ifedetokens{\jobname}{#1}}
\@ifprevenvir \long\def\@ifprevenvir#1{%
                % #1 enquired environment name which will be confronted with \@pre|
                   venvir
                % #2 what if true (if the names are equivalent[4])
                % #3 what if false
\@ifedetokens{\@prevenvir}{#1}}
```

Note that `\@ifjobname` and `\@ifenvir` are expandable and in an `\edef` they expand to

`\@ifedetokens{`⟨*current jobname*⟩`}{`⟨*arg.1*⟩`}`

and

`\@ifedetokens{`⟨*current envir*⟩`}{`⟨*arg.1*⟩`}`

resp. which may be useful for some TEXvert.

```
\@checkend \def\@checkend#1{\@ifenvir{#1}{}{\@badend{#1}}}
```

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

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

You might also write also `\end{macrocode\star}` where `\star` is defined as 'other' star or letter star.

Storing and restoring the catcodes of specials

```
\gmu@storespecials \DeclareCommand\gmu@storespecials{o}{% we provide a possibility of adding
                     stuff. For usage see line ??.
                   \def\do##1{\catcode`\@nx##1=\the\catcode`##1\relax}%
\gmu@restorespecials \edef\gmu@restorespecials{%
                     \dospecials\do\^^M}\IfValueT{#1}{#1}}
\gmu@septify \pdef\gmu@septify{% restoring the standard catcodes of specials. The name is
                     the opposite of 'sanitize' :-). It restores also the original catcode of ^^M.
                   \def\do{\relax\catcode`}%
                   \do\␣10\do\\0\do\{1\do\}2\do\$3\do\&4%
                   \do\#6\do\^7\do\_8\do\%14\do\~13\do\^^M5\relax
                   %%␣\let\do\@makeother
                   %%␣\do0\do1\do2\do3\do4\do5\do6\do7\do8\do9\relax
}
```

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

From relsize

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

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

\relsize \smaller \larger \smallerr \largerr

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

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

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

Change the number by the given increment:

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

watch out for size underflow:

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

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

\rs@unknown@warning
```
\providecommand*\rs@unknown@warning{\PackageWarning{gmutils␣
```

```
        (relsize)}{Current␣font␣size
2863  is␣unknown!␣(Why?!?)\MessageBreak␣Assuming␣\string%
        \normalsize}}
```

And a handful of shorthands:

```
\larger     2867 \DeclareRobustCommand*\larger[1][\@ne]{\relsize{+#1}}
\smaller    2868 \DeclareRobustCommand*\smaller[1][\@ne]{\relsize{-#1}}
\textlarger 2869 \DeclareRobustCommand*\textlarger[2][\@ne]{{\relsize{+#1}#2}}
\textsmaller 2870 \DeclareRobustCommand*\textsmaller[2][\@ne]{{\relsize{-#1}#2}}
\largerr    2871 \pdef\largerr{\relsize{+2}}
\smallerr   2872 \pdef\smallerr{\relsize{-2}}
```

Meta-symbols

I fancy also another Knuthian trick for typesetting ⟨*meta-symbols*⟩ 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 hyphenation by defining a `\language` which has no patterns associated with it and use this to typeset the words within the angle brackets."

```
\meta       2894 \pdef\meta#1{%
```

"Since the old implementation of `\meta` could be used in math we better ensure that this is possible with the new one as well. So we use `\ensuremath` around `\langle` and `\rangle`. However this is not enough: if `\meta@font@select` below expands to `\itshape` it will fail if used in math mode. For this reason we hide the whole thing inside an `\nfss@text` box in that case."

```
2902   {\meta@fontsetting\ensuremath\langle}%
2903   \ifmmode\@xa\nfss@text\fi
2904   {% this has to be a begin-group because \nfss@text becomes \hbox in math
         mode.
2906    \gmu@activespaceblank
2907     \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.

```
2915     #1\/%
2917   }%
2918   {\meta@fontsetting\ensuremath\rangle}%
2919 }% of \meta.
```

```
\gmu@activespaceblank 2921 \pdef\gmu@activespaceblank{%
  \gmu@activespace    2922   \@xa\def\gmu@activespace{\space\ignorespaces}% note the subtle perversity of this definition: if we meet more than one subsequent active spaces, then the first of them will typeset \space and its \ignorespaces will gobble \space of the second and will stop at \ignorespaces and this \ignorespaces will gobble the next \space and so on.
2929 }
```

```
\meta@fontsetting 2931 \def\meta@fontsetting{\color{red!50!black}}
\meta@font@select 2933 \def\meta@font@select{\meta@fontsetting\it}
```

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

The below `\meta`'s drag[5] is a version of *The TEXbook*'s one.

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

`\metachar` `\metacharfont`
```
\pdef\metachar#1{\begingroup\metacharfont␣#1\endgroup}
\def\metacharfont{\meta@fontsetting\rm}
```

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 line break just before nor just after themselves. The `\discretionary` TEX primitive has three arguments: `#1` 'before break', `#2` 'after break', `#3` 'without break', remember?

`\discre`
```
\pdef\discre#1#2#3{\leavevmode\kernosp%
  \discretionary{#1}{#2}{#3}\penalty10000\hskiposp\relax}
```

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

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

```
\def\:{%
  \ifmmode\afterfi{\mskip\medmuskip}%
  \else\afterfi{\discre{\null}{}{}}%  \null to get \hyphenpenalty not \exhyphenpenalty.
  \fi}
```

```
\let\gmu@discretionaryhyphen\-␣%  for the cases when we don't redefine \ but use \
```

`\bihyphen`
```
\DeclareCommand\bihyphen{O{*}}{%  a redefinition of \- that makes it optional-argument to allow further hyphenation
```

`mu@discretionaryhyphen`
```
  \DeclareCommand\gmu@discretionaryhyphen{T{#1}ca}{%
    \IfValueT{##2}{%
      \@ifempty{##2}{}{%
```

`\gmu@bihyphen@char`
```
        \def\gmu@bihyphen@char{##2}}%
    }%
    \IfValueT{##3}{%
      \@ifempty{##3}{}{%
```

`\gmu@bihyphen@corr`
```
        \def\gmu@bihyphen@corr{##3}}%
    }%
    \IfValueTF{##1}\discre\discretionary
    {% before break
      \IfValueTF{##2}{%
        \@ifempty{##2}{\gmu@bihyphen@char}{##2}%
      }{%
        \ifnum\hyphenchar\font>\z@
          \char\hyphenchar\font
        \fi}}% end of before break
```

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

```
      {% after break
        \IfValueT{##3}{%
          \@ifempty{##3}{\gmu@bihyphen@corr}{##3}%
        }%
      }%
      {% without break
      }% almost as in The TeXbook: unlike The TeXbook, we allow hyphenchars ≥ 255
          as we are XeTeX.
    }% of \DeclareCommand\-
    \gmu@storeifnotyet\-% original \- is a TeX's primitive, therefore we should
        store it.
    \let\-\gmu@discretionaryhyphen
    \let\@dischyph\gmu@discretionaryhyphen␣% to override framed.sty
}% of \bihyphen

\relaxen\gmu@bihyphen@corr
```

\vs
```
\pdef\vs{\discre{\visiblespace}{}{\visiblespace}}
```

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

\printspaces
```
\def\printspaces#1{{\let~=\vs␣\let\␣=\vs␣\gm@pswords#1␣%
    \@@nil}}
```

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

\sfname
```
\pdef\sfname#1{\textsf{\printspaces{#1}}}
```

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

\file
```
\pdef\file#1{\gmu@printslashes#1/\gmu@printslashes}
```

\gmu@printslashes
```
\def\gmu@printslashes#1/#2\gmu@printslashes{%
  \sfname{#1}%
  \ifx\gmu@printslashes#2\gmu@printslashes
  \else
  \textsf{\gmu@discretionaryslash}%
  \afterfi{\gmu@printslashes#2\gmu@printslashes}\fi}
```

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

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

\pk
```
\pdef\pk#1{\textsf{#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
```
\DeclareCommand\cs{O{\type@bslash\penalty\@M\hskip\z@skip}}{%
    % [#1] O{\bslash} the control sequence's prefix, by default it's \ allowing
      hyphenation of subsequent word,
```

```
%  #2  m  the control sequence or anything to be typeset in typewriter font.
\begingroup
\ifdefined\verbatim@specials\verbatim@specials\fi
\edef\-{\discretionary{%
    \ifdefined\gmv@hyphen\gmv@hyphen
    \else\unexpanded{{\normalfont-}}%
    \fi}{}{}}%
\def\{{\type@lbrace\yeshy}\def\}{\char`\}}%
\narrativett
\edef\narrativett@storedhyphenchar{\the\hyphenchar\font}%
\hyphenchar\font=\ifdefined\gmv@hyphenchar\gmv@hyphenchar
\else␣"A6␣\fi
\cs@inner{#1}}
```

\cs@inner
```
\pdef\cs@inner#1#2{%
  #1#2%
  \hyphenchar\font=\narrativett@storedhyphenchar\relax
  \endgroup}
```

\narrativett
```
\def\narrativett{\ttfamily}%
```
such name because I introduce it to distinguish the narrative verbatims from the code in gmdoc.

\env
```
\long\pdef\env{\cs[]}
```

And for the special sequences like ^^A:

\hathat
```
\foone{\@makeother\^}
  {\pdef\hathat{\cs[^^]}}
```

\hash
```
\AtBeginDocument{%
  \@ifpackageloaded{gmdoc}{\def\hash{\cs[\#]}}{}}
```

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

\possfil
```
\def\possfil{\hfil\penalty1000\hfilneg}
```

\possvfil
```
\def\possvfil{\vfil
  \penalty\numexpr
  \gmu@minnum{\clubpenalty+\widowpenalty}{9999}%
  \relax␣% eaten by \gmu@maxnum
  \relax␣% eaten by \numexpr
  \vfilneg}
```

\gmu@extremnum
```
\long\def\gmu@extremnum#1#2#3{%
      % #1  inequality sign
      % #2  left side of comparison
      % #3  right side of comparison
  \@xa\ifx\@firstofmany#3\@@nil\relax␣% this complicated test is to allow arguments beginning with some \if⟨...⟩ as in \possvfil e.g.
    \@xa\@firstoftwo
  \else\@xa\@secondoftwo
  \fi
  {#2}%
  {%
    \ifnum\numexpr#2\relax#1\numexpr#3\relax
      \@xa\@firstoftwo
    \else\@xa\@secondoftwo
    \fi
```

```
      {\gmu@extremnum#1{#2}}%
      {\gmu@extremnum#1{#3}}%
  }%
}
```

\gmu@maxnum
\gmu@minnum

```
\def\gmu@maxnum{\gmu@extremnum>}
\def\gmu@minnum{\gmu@extremnum<}
```

Typesetting arguments and commands

\arg

We define a conditional and iterating command `\arg` that in math mode does what it used to do was in math and outside math it typesets mandatory, optional and picture (parenthesed) and angled arguments and optional stars. You can write

```
\arg[gefilte]*<fisch>(mit){baigele}
```

to get

[⟨*gefilte*⟩][*]{⟨*fisz*⟩}(⟨*mit*⟩){⟨*bajgele*⟩}

or even

```
\verb+\MoltoAdagio/arg*{Dankgesang}<an>[die␣Gottheit]+
```

(where / is the escape char in verbatims) to get

\MoltoAdagio[*]{⟨*Dankgesang*⟩}<⟨*an*⟩>[⟨*die Gottheit*⟩]

(in der lydischen Tonart).

For more complicated arguments configurations consider using gmdoc's environment `enumargs`.

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

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

\cmd

```
3182 \long\def\cmd#1{\@xa\cs\@xa{\@xa\cmd@to@cs\string#1}}%
```

it has to be un `\protected`! It has so many `\expandafter`s to allow `\cmd\par` and still keep the `\cs` command 'short'.

\cmd@to@cs

```
3186 \def\cmd@to@cs#1#2{\char\number`#2\relax}
```

It can be short since it never gets actual control sequence as an argument only a string of 'other' tokens (and maybe spaces).

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

\marg

```
3192 \pdef\marg#1{{\narrativett\type@lbrace}\meta{#1}{%
       \narrativett\char`\}}}
```

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

\oarg

```
3198 \pdef\oarg{\@ifnextchar[\@oargsq\@oarg}
3200 \pdef\@oarg#1{{\narrativett[}\meta{#1}{\narrativett]}}
3201 \pdef\@oargsq[#1]{\@oarg{#1}}
```

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

\parg

```
3205 \pdef\parg{\@ifnextchar(\@pargp\@parg}
3207 \def\@parg#1{{\narrativett(}\meta{#1}{\narrativett)}}
```

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

\pdef\aarg{\@ifnextchar<\@aarga\@aarg}
\def\@aarg#1{{\narrativett<}\meta{#1}{\narrativett>}}
\def\@aarga<#1>{\@aarg{#1}}

\def\@verbaarga#1#2>{\@aarg{#2}\arg@dc}

\foone{\catcode`>\active}{%
  \def\@verbaargact#1#2>{\@aarg{#2}\arg@dc}%
}

\foone{\@makeother\{\@makeother\}%
  \catcode`[=\@ne\catcode`]=\tw@}
[%
  \def\@verbmargm#1#2}[% for an argument in curly braces in a verbatim,
      where the braces are not groupers and not necessary 'other'. We'll know
      by \@ifnextif that the future token is an opening brace. Note this macro
      has 2nd parameter delimited with 'other' closing brace (so may not act
      correctly when braces are nested (then hide them with special verbatim
      groupers)).
    \marg[#2]%
    \arg@dc
  ]%
]% of \foone
```

\arg@dc

```
\DeclareCommand\arg@dc!{%
  s␣% (1)
  o␣% (2)
  c␣% (3)
  b␣% (4)
  a␣% (5)
  S{\arg}␣% (6) just gobbled (for backwards compatibility)
}{% This command iterates while it has arguments and typesets them in brackets,
    parentheses or curly braces. Note it gobbles subsequent \args and just iter-
    ates.
  \def\next{0}%
  \IfValueT{#1}%
  {\metachar[\scanverb{*}\metachar]\def\next{1}}%
  \IfValueT{#2}{\@oarg{#2}\def\next{1}}%
  \IfValueT{#3}{\@parg{#3}\def\next{1}}%
  \IfValueT{#4}{\marg{#4}\def\next{1}}%
  \IfValueT{#5}{\aarg{#5}\def\next{1}}%
  \@ifnextchar\egroup{\endgroup}{%
    \if1\next\@xa\arg@dc
    \else␣% it's crucial that we look for verbatim braces after we checked there
        were no #4, otherwise there would be an error.
      \def\next{%
        \@ifnextif\xiilbrace{\@verbmargm}%
        {% not active or other lbrace
          \@ifnextif<{% then we look for angles
            \ifnum\catcode`>=\active
              \@xa\@verbaargact
            \else\@xa\@verbaarga
            \fi}%
```

```
3267                {% and if not angles neither verbatim braces, then
3268                  \endgroup␣␣% if we have no more arguments to typeset, we
                          close the group opened in line 3286
3270                }%
3271              }%
3272            }%
3273            \@xa␣\next
3274          \fi
3275        }% of not egroup
3276 }% of \arg@dc
```

Now define the front-end macro of the `\arg` command:

```
3280 \foone{\obeylines}{%
3281   \AtBeginDocument{%
3282     \let\math@arg\arg␣%
\arg 3283     \pdef\arg{\ifmmode\math@arg␣%
3285       \else\afterfi{%
3286         \begingroup␣%
3288         \ifdefined\@ifQueerEOL\@ifQueerEOL{%
3289           \def^^M{\unskip\space}% in the 'queer' EOLs scope we keep
                  line end active in case we have \arg {⟨arg.⟩} ending a line: the
                  next char peeper touches line end or, if the line end was 5, gob-
                  bles the space it turns into so the comment layer would 'leak'
                  to the code layer.
3295         }{}\fi␣%
3296         \arg@dc}%
3297       \fi}% of \arg,
3298   }% of \AtBeginDocument,
3299 }% of \foone.
```

Now you can write

```
\arg{mand.␣arg}␣[opt.␣arg]␣(pict.␣arg)
```

to get {⟨*mand. arg*⟩} [⟨*opt. arg*⟩] (⟨*pict. arg*⟩). (Yes, with only one `\arg`!)
And `$\arg(1+i)␣=␣\pi/4$` for $\arg(1+i) = \pi/4$.

```
\cat 3311 \DeclareCommand\cat{Q{'"0123456789ABCDEF}}{%
3312   ${}_{\the\numexpr#1}\m@th$\@ifnextcat␣a\space{}}
```

Not only preamble!

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

```
\not@onlypreamble 3328 \newcommand\not@onlypreamble[1]{{%
3329   \def\do##1{\ifx#1##1\else\@nx\do\@nx##1\fi}%
3330   \xdef\@preamblecmds{\@preamblecmds}}}

3332 \not@onlypreamble\@preamblecmds
3333 \not@onlypreamble\@ifpackageloaded
3334 \not@onlypreamble\@ifclassloaded
3335 \not@onlypreamble\@ifl@aded
3336 \not@onlypreamble\@pkgextension
```

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

\gm@notprerr
```
\def\gm@notprerr{␣can␣be␣used␣only␣in␣preamble␣(\on@line)}

\AtBeginDocument{%
  \def\do#1{\@nx\do\@nx#1}%
  \edef\@preamblecmds{%
    \def\@nx\do##1{%
      \def##1{\@nx\PackageError{gmutils/LaTeX}%
        {\@nx\string##1␣\@nx\gm@notprerr}\@nx\@eha}}%
    \@preamblecmds}}
```

A subtle error raises: the LaTeX standard `\@onlypreamble` and what `\document` does with `\@preamblecmds` makes any two of 'only preamble' CS's `\ifx`-identical inside `document`. And my change makes any two CS's `\ifx`-different. The first it causes a problem with is standard LaTeX's `\nocite` that checks `\ifx\@onlypreamble\document`. So hoping this is a rare problem, we circumvent it. 2008/08/29 a bug is reported by Edd Barrett that with natbib an 'extra }' error occurs so we wrap the fix in a conditional.

\gmu@nocite@ampulex
```
\def\gmu@nocite@ampulex{%  we wrap the stuff in a macro to hide an open \if.
    And not to make the begin-input hook too large. the first optional argument
    is the parameters string and the second the argument for one-level expansion
    of \nocite. Both hash strings are doubled to pass the first \def.
  \ampulexdef\nocite[####1][{{####1}}]% note the double brace around
      % #3.
  \ifx
  {\@onlypreamble\document}%
  \iftrue}

\AtBeginDocument{\gmu@nocite@ampulex}%
```

Third person pronouns

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

Previous versions of this documentation were consequently alternating 'he' and 'she' and provided specific macros for that purpose. Now I'm not that queer and gender so I take what is normally used, 'they' that is.

(The issue of human sexes and genders (certainly much more numerous than 2) is complex and delicate and a TeX macro package is probably not the best place to discuss it.)

\heshe \hisher \himher \hishers
```
\def\heshe{they}
\def\hisher{their}
\def\himher{them}
\def\hishers{theirs}
```

\HeShe \HisHer \HimHer \HisHers
```
\def\HeShe{They}
\def\HisHer{Their}
\def\HimHer{Them}
\def\HisHers{Theirs}
```

That is, 'Expe-ri-mente'[6] mit MW sectioning & `\refstepcounter` to improve mwcls's cooperation with hyperref. They shouldn't make any harm if another class (non-mwcls) is loaded.

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

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

```
3454 \AtBeginDocument{%
```
because we don't know when exactly hyperref is loaded and maybe after this package.
```
                NoNumSecs  3456   \@ifpackageloaded{hyperref}{\newcounter{NoNumSecs}%
3457     \setcounter{NoNumSecs}{617}%
```
to make `\ref`ing to an unnumbered section visible (and funny?).
```
gm@hyperrefstepcounter  3459     \def\gm@hyperrefstepcounter{\refstepcounter{NoNumSecs}}%
    \gm@targetheading   3460     \pdef\gm@targetheading#1{%
3461       \hypertarget{#1}{#1}}%
```
end of then
```
gm@hyperrefstepcounter  3462   {\def\gm@hyperrefstepcounter{}%
    \gm@targetheading   3463     \def\gm@targetheading#1{#1}}%
```
end of else
```
3464 }% of \AtBeginDocument
```

Auxiliary macros for the kernel sectioning macro:

```
ectionsoutofmainmatter  3467 \def\gm@dontnumbersectionsoutofmainmatter{%
3468   \if@mainmatter\else␣\HeadingNumberedfalse␣\fi}
earpagesduetoopenright  3469 \def\gm@clearpagesduetoopenright{%
3470   \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  3483 \long\def\@ifnotmw#1#2{\gm@ifundefined{mw@sectionxx}{#1}{#2}}
```

The kernel of MW's sectioning commands:

```
3508 \@ifnotmw{}{%
    \mw@sectionxx  3509 \def\mw@sectionxx#1#2[#3]#4{%
3510   \edef\mw@HeadingLevel{\csname␣#1@level\endcsname
3511         \space}%
```
space delimits level number!
```
3512   \ifHeadingNumbered
3513     \ifnum␣\mw@HeadingLevel>\c@secnumdepth␣%
           \HeadingNumberedfalse␣\fi
```
line below is in `\gm@ifundefined` to make it work in classes other than mwbk
```
3516     \gm@ifundefined{if@mainmatter}{}{%
           \gm@dontnumbersectionsoutofmainmatter}
3517   \fi

%    \ifHeadingNumbered
```

6 A. Berg, *Wozzeck*.

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

\HeadingRHeadText
\HeadingTOCText
\HeadingText
\mw@HeadingType

```
  \def\HeadingRHeadText{#2}%
  \def\HeadingTOCText{#3}%
  \def\HeadingText{#4}%
  \def\mw@HeadingType{#1}%
  \if\mw@HeadingBreakBefore
    \if@specialpage\else\thispagestyle{closing}\fi
    \gm@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
  \fi% of \ifHeadingNumbered

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

```
      \if\mw@HeadingBreakAfter\pagebreak\relax\fi
    \fi% of \if\mw@HeadingWholeWidth
  \fi% of \if\mw@HeadingRunIn
  }
```

An improvement of MW's \SetSectionFormatting

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

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

`[#1]` the flags, e.g. `breakbefore`, `breakafter`;
`#2` the sectioning name, e.g. `chapter`, `part`;
`#3` preskip;
`#4` heading type;
`#5` postskip

```
\relaxen\SetSectionFormatting
\newcommand*\SetSectionFormatting[5][\empty]{%
  \ifx\empty#1\relax\else% empty (not \empty!) #1 also launches \else.
    \def\mw@HeadingRunIn{10}\def\mw@HeadingBreakBefore{10}%
    \def\mw@HeadingBreakAfter{10}\def\mw@HeadingWholeWidth{%
        10}%
    \@ifempty{#1}{}{\mw@processflags#1,\relax}% If #1 is omitted, the
        flags are left unchanged. If #1 is given, even as [], the flags are first
        cleared and then processed again.
  \fi
  \gm@ifundefined{#2}{\@namedef{#2}{\mw@section{#2}}}{}%
  \mw@secdef{#2}{@preskip}␣{#3}{2␣oblig.}%
  \mw@secdef{#2}{@head}␣␣␣␣{#4}{3␣oblig.}%
  \mw@secdef{#2}{@postskip}{#5}{4␣oblig.}%
  \ifx\empty#1\relax
    \mw@secundef{#2@flags}{1␣(optional)}%
  \else\mw@setflags{#2}%
  \fi}
```

`\mw@secdef`

```
\def\mw@secdef#1#2#3#4{%
        % #1 the heading name,
        % #2 the command distincter,
        % #3 the meaning,
        % #4 the number of argument to error message.
  \@ifempty{#3}
    {\mw@secundef{#1#2}{#4}}
    {\@namedef{#1#2}{#3}}}
```

`\mw@secundef`

```
\def\mw@secundef#1#2{%
  \gm@ifundefined{#1}{%
    \ClassError{mwcls/gm}{%
      command␣\bslash#1␣␣undefined␣\MessageBreak
      after␣\bslash␣SetSectionFormatting!!!\MessageBreak}{%
      Provide␣the␣#2␣argument␣of␣\bslash␣
          SetSectionFormatting.}}{}}
```

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

```
\addtoheading 3638 \def\addtoheading#1#2{%
3639       \n@melet{gmu@reserveda}{#1@head}%
3640       \edef\gmu@reserveda{\unexpanded{#2}\@xa\unexpanded{%
               \gmu@reserveda}}%
3641       \n@melet{#1@head}{gmu@reserveda}%
3643 }
3645 }% of \@ifnotmw's else.
```

Negative \addvspace

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

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

```
3657 \@ifnotmw{}{% We proceed only in MWCLS.
```

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

```
\@afterheading 3662 \def\@afterheading{%
3663   \@nobreaktrue
3664   \xdef\gmu@prevsec{\mw@HeadingType}% added now
3665   \everypar{%
3666     \grelaxen\gmu@prevsec% added now. All the rest is original LaTeX.
3667     \if@nobreak
3668     \@nobreakfalse
3669     \clubpenalty␣\@M
3670     \if@afterindent␣\else
3671     {\setbox\z@\lastbox}%
3672     \fi
3673     \else
3674     \clubpenalty␣\@clubpenalty
3675     \everypar{}%
3676     \fi}}
```

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

```
\gmu@checkaftersec 3683 \def\gmu@checkaftersec{%
3684   \gm@ifundefined{gmu@prevsec}{}{%
3685     \ifgmu@postsec% an additional switch that is true by default but may be
               turned into an \ifdim in special cases, see line 3721.
3688     {\@xa\mw@getflags\@xa{\gmu@prevsec}%
3689       \glet\gmu@reserveda\mw@HeadingBreakAfter}%
3690     \if\mw@HeadingBreakBefore\def\gmu@reserveda{11}\fi% if the cur-
               rent heading inserts page break before itself, all the play with vskips is
               irrelevant.
3693     \if\gmu@reserveda\else
3694     \penalty10000\relax
```

```
      \skip\z@=\csname\gmu@prevsec␣@postskip\endcsname\relax
      \skip\tw@=\csname\mw@HeadingType␣@preskip\endcsname\relax
      \gm@ifundefined{\mw@HeadingType␣@twoheadskip}{%
        \ifdim\skip\z@>\skip\tw@
        \vskip-\skip\z@% we strip off the post-skip of previous header if it's
            bigger than current pre-skip
        \else
        \vskip-\skip\tw@% we strip off the current pre-skip otherwise
        \fi}{% But if the two-header-skip is defined, we put it
        \penalty10000
        \vskip-\skip\z@
        \penalty10000
        \vskip-\skip\tw@
        \penalty10000
        \vskip\csname\mw@HeadingType␣@twoheadskip\endcsname
        \relax}%
      \penalty10000
      \hrule␣height\z@\relax% to hide the last (un)skip before
          subsequent \addvspaces.
      \penalty10000
      \fi
      \fi
    }% of \gm@ifundefined{gmu@prevsec} 'else'.
}% of \def\gmu@checkaftersec.
```

\ParanoidPostsec

```
\def\ParanoidPostsec{% this version of \ifgmu@postsec is intended for the
    special case of sections may contain no normal text, as while gmdocing.
```

\ifgmu@postsec

```
  \def\ifgmu@postsec{% note this macro expands to an open \if.
    \skip\z@=\csname\gmu@prevsec␣@postskip\endcsname\relax
    \ifdim\lastskip=\skip\z@\relax% we play with the vskips only if the
        last skip is the previous heading's postskip (a counter-example I met
        while gmdocing).
  }}
\let\ifgmu@postsec\iftrue
```

\gmu@getaddvs

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

And the modification of the inner macros at last:

\gmu@setheading

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

\SetTwoheadSkip

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

```
}% of \@ifnotmw's else.
```

My heading setup for mwcls

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

```
\@ifnotmw{}{% We define this declaration only when in mwcls.
\DeclareCommand\WPheadings{T{\chapter}}{%
  \SetSectionFormatting[breakbefore,wholewidth]
      {part}{\z@\@plus1fill}{}{\z@\@plus3fill}%
  \IfValueF{#1}{%
    \gm@ifundefined{chapter}{}{%
      \SetSectionFormatting[breakbefore,wholewidth]
      {chapter}
      {66\p@}% {67\p@} for Adventor/Schola 0,95.
      {\FormatHangHeading{\LARGE}}
      {27\p@\@pluso,2\p@\@minus1\p@}%
    }%
  }% of unless #1

  \SetTwoheadSkip{section}{27\p@\@pluso,5\p@}%
  \SetSectionFormatting{section}
      {24\p@\@pluso,5\p@\@minus5\p@}%
      {\FormatHangHeading␣{\Large}}
      {10\p@\@pluso,5\p@}% ed. Krajewska of "Wiedza Powszechna", as we
          understand her, wants the skip between a heading and text to be rigid.

  \SetTwoheadSkip{subsection}{11\p@\@pluso,5\p@\@minus1\p@}%
  \SetSectionFormatting{subsection}
      {19\p@\@pluso,4\p@\@minus6\p@}
      {\FormatHangHeading␣{\large}}% 12/14 pt
      {6\p@\@pluso,3\p@}% after-skip 6 pt due to p.12, not to squeeze the
          before-skip too much.

  \SetTwoheadSkip{subsubsection}{10\p@\@plus1,75\p@\@minus1%
      \p@}%
  \SetSectionFormatting{subsubsection}
      {10\p@\@pluso,2\p@\@minus1\p@}
      {\FormatHangHeading␣{\normalsize}}
      {3\p@\@pluso,1\p@}% those little skips should be smaller than you cal-
          culate out of a geometric progression, because the interline skip en-
          larges them.

  \SetSectionFormatting[runin]{paragraph}
      {7\p@\@pluso,15\p@\@minus1\p@}
      {\FormatRunInHeading{\normalsize}}
      {2\p@}%
  \SetSectionFormatting[runin]{subparagraph}
      {4\p@\@plus1\p@\@minuso,5\p@}
      {\FormatRunInHeading{\normalsize}}
      {\z@}%
}% of \WPheadings
}% of \@ifnotmw
```

\WPheadings

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.

\gm@secini
```
  \def\gm@secini{gm@la}%
  \StoreMacros{\partmark\chaptermark\sectionmark%
      \subsectionmark\subsubsectionmark\paragraphmark}%
```
\gm@secxx
```
  \def\gm@secxx#1#2[#3]#4{%
    \ifx\gm@secstar\@empty
```
a little trick to allow a special version of the heading just to the running head.
```
      \@namedef{#1mark}##1{%
```
we redefine \⟨*sec*⟩`mark` to gobble its argument and to launch the stored true marking command on the appropriate argument.
```
        \storedcsname{#1mark}{#2}%
        \RestoreMacro*{#1mark}%
```
after we've done what we wanted we restore original `\#1mark`.
```
      }%
```
\gm@secstar
```
      \def\gm@secstar{[#3]}%
```
if `\gm@secstar` is empty, which means the sectioning command was written starless, we pass the 'true' sectioning command `#3` as the optional argument. Otherwise the sectioning command was written with star so the 'true' s.c. takes no optional.
```
    \fi
    \@xa\@xa\csname\gm@secini#1\endcsname
    \gm@secstar{#4}}%
```

```
}{%
```
we are in mwcls and want to reverse MW's optionals order i.e., if there's just one optional, it should go both to toc and to running head.

\gm@secini
```
  \def\gm@secini{gm@mw}%
  \let\gm@secmarkh\@gobble%
```
in mwcls there's no need to make tricks for special version to running headings.

```
\gm@secxx
3885    \def\gm@secxx#1#2[#3]#4{%
3886      \@xa\@xa\csname\gm@secini#1\endcsname
3887      \gm@secstar[#2][#3]{#4}}%
3888 }

\gm@sec
3890 \def\gm@sec#1{\@dblarg{\gm@secx{#1}}}
\gm@secx
3891 \def\gm@secx#1[#2]{%
3892   \@ifnextchar[{\gm@secxx{#1}{#2}}{\gm@secxx{#1}{#2}[#2]}}% if there's
          only one optional, we double it not the mandatory argument.

\gm@straightensec
3896 \def\gm@straightensec#1{% the parameter is for the command's name.
3897   \gm@ifundefined{#1}{}{% we don't change the ontological status of the
          command because someone may test it.
3899     \n@melet{\gm@secini#1}{#1}%
3900     \@namedef{#1}{%
\gm@secstar
3901       \gm@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
\gm@secstar
3902         \def\gm@secstar{}\gm@sec{#1}}}%
3903 }%

3905 \let\do\gm@straightensec
3906 \do{part}\do{chapter}\do{section}\do{subsection}\do{%
      subsubsection}
3907 \@ifnotmw{}{\do{paragraph}}% this 'straightening' of \paragraph with the
      standard article caused the 'TEX capacity exceeded' error. Anyway, who on
      Earth wants paragraph titles in toc or running head?
```

enumerate* and itemize*

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

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

```
enumerate*
3923 \@namedef{enumerate*}{%
3924   \ifnum\@enumdepth>\thr@@
3925     \@toodeep
3926   \else
3927     \advance\@enumdepth\@ne
3928     \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
3929     \@xa\list\csname␣label\@enumctr\endcsname{%
3930       \partopsep\topsep␣\topsep\z@␣\leftmargin\z@
3931       \itemindent\@parindent␣% %\advance\itemindent\labelsep
3932       \labelwidth\@parindent
3933       \advance\labelwidth-\labelsep
3934       \listparindent\@parindent
3935       \usecounter␣\@enumctr
3936       \def\makelabel##1{##1\hfil}}%
3937   \fi}
3938 \@namedef{endenumerate*}{\endlist}

itemize*
3941 \@namedef{itemize*}{%
3942   \ifnum\@itemdepth>\thr@@
3943     \@toodeep
```

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

The logos

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

```
\let\oldLaTeX\LaTeX
\let\oldLaTeXe\LaTeXe

\pdef\TeX{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX\@}
\StoreMacro\TeX
\AtBeginDocument{\RestoreMacro\TeX}
```

`\DeclareLogo`

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

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

`\gmu@reserveda`

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

`\gmu@reserveda`

```
    \def\gmu@reserveda{#1}%
  \fi
  \edef\gmu@reserveda{%
    \@nx\DeclareTextCommand\@nx#2{PD1}{\gmu@reserveda}}
  \gmu@reserveda
  \DeclareTextCommandDefault#2{#3}%
```

`\pdef`

```
  \pdef#2{#3}%
```

added for XeTeX.

```
}
```

`\DeclareLogo`

```
\DeclareLogo\LaTeX{%
  {%
    L%
    \setbox\z@\hbox{\check@mathfonts
      \fontsize\sf@size\z@
      \math@fontsfalse\selectfont
      A}%
    \kern-.57\wd\z@
    \sbox\tw@␣T%
    \vbox␣to\ht\tw@{\copy\z@␣\vss}%
    \kern-.2\wd\z@%
```

originally −.15em for T.

```
  }%
  {%
    \ifdim\fontdimen1\font=\z@
```

```
      \else
        \count\z@=\fontdimen5\font
        \multiply\count\z@␣by␣64\relax
        \divide\count\z@␣by\p@
        \count\tw@=\fontdimen1\font
        \multiply\count\tw@␣by\count\z@
        \divide\count\tw@␣by␣64\relax
        \divide\count\tw@␣by\tw@
        \kern-\the\count\tw@␣sp\relax
      \fi}%
    \TeX}
\LaTeXe \DeclareLogo\LaTeXe{\mbox{\m@th␣\if
        b\expandafter\@car\f@series\@nil\boldmath\fi
        \LaTeX\kern.15em2$_{\textstyle\varepsilon}$}}
    \StoreMacro\LaTeX
    \StoreMacro*{LaTeX␣}
```

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

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

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

```
\gm@ifundefined{AmSTeX}
\AmSTeX   {\def\AmSTeX{\leavevmode\hbox{$\mathcal␣A\kern-.2em%
            \lower.376ex%
              \hbox{$\mathcal␣M$}\kern-.2em\mathcal␣S$-\TeX}}}{}
\BibTeX \DeclareLogo\BibTeX{{\rmfamily␣B\kern-.05em%
        \textsc{i{\kern-.025em}b}\kern-.08em% the kern is wrapped in braces
              for my \fakescaps' sake.
        \TeX}}
\SliTeX \DeclareLogo\SliTeX{{\rmfamily␣S\kern-.06emL\kern-.18em%
```

```
        \raise.32ex\hbox
4066        {\scshape␣i}\kern␣-.03em\TeX}}
\PlainTeX 4068 \DeclareLogo\PlainTeX{\textsc{Plain}\kern2pt\TeX}
    \Web 4070 \DeclareLogo\Web{\textsc{Web}}
```

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

```
\TeXbook 4073 \DeclareLogo[The␣TeX␣book]\TeXbook{\textsl{The␣\TeX␣book}}
4074 \let\TB\TeXbook%
```

TUG Boat uses this.

```
   \eTeX 4076 \DeclareLogo[e-TeX]\eTeX{%
4077   \iffontchar\font"03B5{\itshape␣ε}\else
4078   \ensuremath{\varepsilon}\fi-\kern-.125em\TeX}%
```

definition sent by Karl Berry from *TUG Boat* itself.

```
4081 \StoreMacro\eTeX
\pdfeTeX 4083 \DeclareLogo[pdfe-TeX]\pdfeTeX{pdf\eTeX}
 \pdfTeX 4085 \DeclareLogo\pdfTeX{pdf\TeX}
\pdfLaTeX 4086 \DeclareLogo\pdfLaTeX{pdf\LaTeX}
4089 \gm@ifundefined{XeTeX}{%
 \XeTeX 4090   \DeclareLogo\XeTeX{X\kern-.125em\relax
4091     \gm@ifundefined{reflectbox}{%
4092       \lower.5ex\hbox{E}\kern-.1667em\relax}{%
4093       \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
4094     \TeX}}{}
4096 \gm@ifundefined{XeLaTeX}{%
\XeLaTeX 4097   \DeclareLogo\XeLaTeX{X\kern-.125em\relax
4098     \gm@ifundefined{reflectbox}{%
4099       \lower.5ex\hbox{E}\kern-.1667em\relax}{%
4100       \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
4101     \LaTeX}}
```

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

```
\LuaTeX 4109 \DeclareLogo[LuaTeX]\LuaTeX{\textsc{Lua}\TeX}
```

Expandable turning stuff all into 'other'

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

```
\detoken@xa 4128 \long\def\detoken@xa#1{\detokenize\@xa{#1}}
```

The next macro originates from the ancient era when I didn't know about ε-TEX's `\detokenize`. A try to redefine it to `\detokenize\@xa{#1}` resulted in error so (v0.991) I leave it and use as is.

Note however it acts different than `\detoken@xa` for a macro with parameters: while `\detoken@xa` produces and 'extra }' error, `\all@other` expands to the detokenised meaning.

```
\all@other 4138 \long\def\all@other#1{\@xa\gm@gobmacro\meaning#1}
```

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

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

Brave New World of XeTeX

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

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

\XeTeXthree
```
\DeclareCommand\XeTeXthree{o}{%
  \@ifXeTeX{%
    \IfValueT{#1}{\PassOptionsToPackage{#1}{fontspec}}%
    \@ifpackageloaded{gmverb}{\StoreMacro\verb}{}%
    \RequirePackage{xltxtra}% since v 0.4 (2008/07/29) this package re-
        defines \verb and verbatim*, and quite elegantly provides an option
        to suppress the redefinitions, but unfortunately that option excludes also
        a nice definition of \xxt@visiblespace which I fancy.
    \@ifpackageloaded{gmverb}{\RestoreMacro\verb}{}%
    \AtBeginDocument{%
      \RestoreMacro\LaTeX\RestoreMacro*{LaTeX␣}% my version of
          the LaTeX logo has been stored just after defining, in line 4023.
      \RestoreMacro\eTeX}%
```

\adddefaultfontfeatures
```
    \pdef\adddefaultfontfeatures##1{%
      \addtomacro\zf@default@options{#1,}}
  }% of \@ifXeTeX's first argument,
  {}% \@ifXeTeX's second argument,
}% of \XeTeXthree's body.
```

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

\udigits
```
\AtBeginDocument{%
  \@ifpackageloaded{fontspec}{%
    \pdef\udigits{%
      \addfontfeature{Numbers=Uppercase}}%
  }{%
    \emptify\udigits}}
```

Fractions

\Xedekfracc
```
\def\Xedekfracc{\gm@ifstar\gmu@xedekfraccstar%
      \gmu@xedekfraccplain}
```

(plain) The starless version turns the font feature frac on.

(⋆) But nor my modification of Minion Pro neither TeX Gyre Pagella doesn't feature the frac font feature properly so, with the starred version of the declaration we use the

characters from the font where available (see the `\@namedef`s below) and the `numr` and `dnom` features with the fractional slash otherwise (via `\gmu@dekfracc`).

(⋆⋆) But Latin Modern Sans Serif Quotation doesn't support the numerator and denominator positions so we provide the double star version for it, which takes the char from font if it exist and typesets with lowers and kerns otherwise.

```
\gmu@xedekfraccstar 4214 \def\gmu@xedekfraccstar{%
  \gmu@xefraccdef   4215   \def\gmu@xefraccdef##1##2{%
                    4216     \iffontchar\font␣##2
                    4217       \@namedef{gmu@xefracc##1}{\char##2␣}%
                    4218     \else
                    4219       \n@melet{gmu@xefracc##1}{relax}%
                    4220     \fi}%
    \gmu@dekfracc   4222   \def\gmu@dekfracc##1/##2{%
                    4223     {\addfontfeature{VerticalPosition=Numerator}##1}%
                                \gmu@numeratorkern
                    4224     \char"2044␣\gmu@denominatorkern
                    4225     {\addfontfeature{VerticalPosition=Denominator}##2}}%
```

We define the fractional macros. Since Adobe Minion Pro doesn't contain $\frac{n}{5}$ nor $\frac{n}{6}$, we don't provide them here.

```
                    4229   \gmu@xefraccdef{1/4}{"BC}%
                    4230   \gmu@xefraccdef{1/2}{"BD}%
                    4231   \gmu@xefraccdef{3/4}{"BE}%
                    4232   \gmu@xefraccdef{1/3}{"2153}%
                    4233   \gmu@xefraccdef{2/3}{"2154}%
                    4234   \gmu@xefraccdef{1/5}{"2155}%
                    4235   \gmu@xefraccdef{2/5}{"2156}%
                    4236   \gmu@xefraccdef{3/5}{"2157}%
                    4237   \gmu@xefraccdef{4/5}{"2158}%
                    4238   \gmu@xefraccdef{1/6}{"2159}%
                    4239   \gmu@xefraccdef{5/6}{"215A}%
                    4240   \gmu@xefraccdef{1/8}{"215B}%
                    4241   \gmu@xefraccdef{3/8}{"215C}%
                    4242   \gmu@xefraccdef{5/8}{"215D}%
                    4243   \gmu@xefraccdef{7/8}{"215E}%
  \dekfracc@args    4244   \pdef\dekfracc@args##1/##2{%
                    4245     \gm@ifundefined{gmu@xefracc\detokenize{##1/##2}}{%
                    4246       \gmu@dekfracc{##1}/{##2}}{%
                    4247       \csname␣gmu@xefracc\detokenize{##1/##2}\endcsname}%
                    4248     \if@gmu@mmhbox\egroup\fi
                    4249   }% of \dekfracc@args.
                    4250   \gm@ifstar{\let\gmu@dekfracc\gmu@dekfraccsimple}{}%
                    4251 }
\gmu@xedekfraccplain 4253 \def\gmu@xedekfraccplain{% 'else' of the main \gm@ifstar
  \dekfracc@args    4254   \pdef\dekfracc@args##1/##2{%
                    4255     \ifmmode\hbox\fi{%
                    4256       \addfontfeature{Fractions=On}%
                    4257       ##1/##2}%
                    4258     \if@gmu@mmhbox\egroup\fi
                    4259   }% of \dekfracc@args
                    4260 }
```

\if@gmu@mmhbox
```
\newif\if@gmu@mmhbox%
```
we'll use this switch for \dekfracc and also for \thous (hacky thousand separator).

\dekfracc
```
\pdef\dekfracc{%
  \ifmmode\hbox\bgroup\@gmu@mmhboxtrue\fi
  \dekfracc@args}
```

\gmu@numeratorkern
\gmu@denominatorkern
```
\def\gmu@numeratorkern{\kern-.055em\relax}
\def\gmu@denominatorkern{\kern-.05em\relax}
```

What have we just done? We defined two versions of the \Xefractions declaration. The starred version is intended to make use only of the built-in fractions such as ½ or 7/8. To achieve that, a handful of macros is defined that expand to the Unicodes of built-in fractions and \dekfracc command is defined to use them.

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

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

A simpler version of \dekfracc is provided in line 5722.

\resizegraphics

\resizegraphics
```
\pdef\resizegraphics#1#2#3{%  2009/11/17 works bad with a file whose name contains spaces so I return \XeTeXpicfile
  \resizebox{#1}{#2}{%
    \edef\gmu@tempa{\@nx\csname␣XeTeX\@nx\@ifendswithpdf{%
        \@xa\string\csname#3\endcsname}{pdf}{pic}file\@nx%
          \endcsname}%
    \gmu@tempa␣"#3"\relax}}

\edef\gmu@tempa{%
  \def\@nx\@ifendswithpdf##1{%
    \unexpanded{%
      \ifnum
      \if\relax\gmu@pdfdetector}##1%
    \detokenize{pdf}\unexpanded{\relax0\else1\fi}% we expand to 1 if #1 ends with lowercase 'pdf' of cat. 12
    \unexpanded{\if\relax\gmu@PDFdetector}##1%
    \detokenize{PDF}\unexpanded{\relax0\else1\fi}% we expand to 1 if #1 ends with uppercase 'PDF' of cat. 12
    >0
    \unexpanded{\@xa\@firstoftwo\else\@xa\@secondoftwo\fi}%
  }% of \@ifendswithpdf

  \def\@nx\gmu@pdfdetector##1\detokenize{pdf}{}%
  \def\@nx\gmu@PDFdetector##1\detokenize{PDF}{}%
}\gmu@tempa
```

\textsuperscript@@
\@textsuperscript
```
\pdef\textsuperscript@@#1{%
  \@textsuperscript{\selectfont#1}}%
\def\@textsuperscript#1{%
  {\m@th\ensuremath{^{\mbox{\fontsize\sf@size\z@#1}}}}}

\let\textsuperscript@@\textsuperscript
```

```
\GMtextsuperscript
\textsuperscript
\truetextsuperscript
```

```
\def\GMtextsuperscript{%
  \@ifXeTeX{%
    \DeclareCommand\textsuperscript{sm}{%
      \IfValueTF{##1}{\textsuperscript@@{##2}}%
      {%
        \begingroup
        \addfontfeature{VerticalPosition=Numerator}##2%
        \endgroup}}%
  }{\truetextsuperscript}}
\def\truetextsuperscript{%
  \let\textsuperscript\textsuperscript@@
}
```

Settings for mathematics in main font

```
\gmath
\garamath
```

I used these terrible macros while typesetting E. Szarzyński's *Letters* in 2008. The `\gmath` declaration introduces math-active digits and binary operators and redefines Greek letters and parentheses, the `\garamath` declaration redefines the quantifiers and is more Garamond Premier Pro-specific.

```
\gmath
```

So, when you set default fonts (in the preamble), put `\gmath` to set what possible from them to math. This sets the `normal` math version. If you want to use another set of fonts elsewhere in your document and have according math for them, set them 'for a while' in the preamble and in their scope declare `\gmath[`⟨*version*⟩`]`. Then put `\mathversion{normal}` right after `\begin{document}` and `\gmath[`⟨*version*⟩`]` where you want those other fonts.

`\gmath` takes second optional argument, in parentheses, which should be (sth. that expands to) a `\fontspec` font selecting command. A font selected by this command will be declared and used when some char in basic font is missing and only else the default math font will be left for such a char.

So,

`\gmath[`⟨*version*⟩`](`⟨*rescue font(spec-ification)*⟩`)`

Note that `\gmath` without first optional argument has always to come first because otherwise it overwrite settings of your math version.

```
\gmu@getfontstring
\gmu@fontstring@
\gmu@fontstring@@
\gmu@getfontscale
\gmu@tempa
\gmu@tempb
\gmu@tempa
```

```
\def\gmu@getfontstring{%
  \xdef\gmu@fontstring{%
    \gmu@fontstring@}}
\def\gmu@fontstring@{%
  \@xa\@xa\@xa\gmu@fontstring@@\@xa\meaning\the\font\@@nil}
\def\gmu@fontstring@@#1"#2"#3\@@nil{"#2"}
\def\gmu@getfontscale#1Scale#2=#3,{%
  \ifx\gmu@getfontscale#3\else
    \def\gmu@tempa{MatchLowercase}%
    \def\gmu@tempb{#3}%
    \ifx\gmu@tempa\gmu@tempb
      \gmu@calc@scale{5}%
      \@xa\@firstoftwo
    \else
      \def\gmu@tempa{MatchUppercase}%
      \ifx\gmu@tempa\gmu@tempb
```

```
          \gmu@calc@scale{8}%
          \afterfifi\@firstoftwo
          \else\afterfifi\@secondoftwo
        \fi
      \fi
      {\xdef\gmu@fontscalebr{[\gmu@fontscale] }}%
      {\xdef\gmu@fontscalebr{[#3] }}%
      \afterfi\gmu@getfontscale
    \fi
}
```

\gmu@getfontdata

```
\def\gmu@getfontdata#1{%
  \global\emptify\gmu@fontscalebr
  \begingroup
  #1%
  \@xa\@xa\@xa\gmu@getfontscale
  \csname zf@family@options\f@family\endcsname
  ,Scale=\gmu@getfontscale,%
  \gmu@getfontstring
  \xdef\gmu@theskewchar{\the\skewchar\font}%
  \endgroup}
```

\gmu@stripchar

```
\def\gmu@stripchar#1"{"}
```

\gmath@getfamnum

```
\DeclareCommand\gmath@getfamnum{C{\gmath@fam}}{%
  \edef\gmath@famnum{\@xa\gmu@stripchar\meaning#1}%
}
```

\XeTeXmathcode⟨*char slot*⟩ [⟨=⟩] ⟨*type*⟩ ⟨*family*⟩ ⟨*char slot*⟩

\gmathFams

```
\DeclareCommand\gmathFams{o  %  the name of math version
  C{\NoValue} %  'rescue' font. Will be accessible via \symgmathRoman/<version>
        (math family) and \gmath@fontt/<version> (font).
}{%
  \IfValueT{#1}{%
    \DeclareMathVersion{#1}%  this sets the defaults so no need to define
        them explicitly
    }%  of if #1 given

  \gmu@getfontdata{\rmfamily\itshape}%
  \edef\gmu@tempa{%
    \IfValueTF{#1}{\@nx\SetSymbolFont{letters}{#1}}%
    {\@nx\DeclareSymbolFont{letters}}%
    {\encodingdefault}{gmathit\PutIfValue{#1}}{m}{it}%
    \IfValueT{#1}{%
      \@nx\DeclareSymbolFont{letters#1}%
      {\encodingdefault}{gmathit\PutIfValue{#1}}{m}{it}%
      }%
    \@nx\DeclareFontFamily{\encodingdefault}{gmathit%
        \PutIfValue{#1}}{%
      \skewchar\font\gmu@theskewchar\space}%
    \@nx\DeclareFontShape{\encodingdefault}{gmathit%
        \PutIfValue{#1}}{m}{it}{%
      <-> \gmu@fontscalebr \gmu@fontstring}{}%
    \IfValueT{#1}{%
```

```
      \@nx\SetMathAlphabet\@nx\mathit{#1}{%
          \encodingdefault}{gmathit#1}{m}{it}}%
  }\gmu@tempa
  \IfValueT{#2}{%
    \gmu@getfontdata{#2\gmu@ifstored{\upshape}{%
        \storedcsname{upshape}}{\upshape}}%
    \edef\gmu@tempa{%
      \@nx\DeclareSymbolFont{gmathRoman\PutIfValue{#1}}%
      {\encodingdefault}{gmathRm\PutIfValue{#1}}{m}{n}%
      \@nx\DeclareFontFamily{\encodingdefault}{gmathRm%
          \PutIfValue{#1}}{%
        \skewchar\font\gmu@theskewchar\space}%
      \@nx\DeclareFontShape{\encodingdefault}{gmathRm%
          \PutIfValue{#1}}{m}{n}{%
        <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
    }\gmu@tempa
    \@xa\font\csname␣gmath@fontt\PutIfValue{#1}\endcsname
    =\gmu@fontstring\relax
    \gmu@getfontdata{#2\gmu@ifstored{\itshape}{%
        \storedcsname{itshape}}{\itshape}}%
    \edef\gmu@tempa{%
      \@nx\DeclareSymbolFont{gmathItalic\PutIfValue{#1}}%
      {\encodingdefault}{gmathIt\PutIfValue{#1}}{m}{n}%
      \@nx\DeclareFontFamily{\encodingdefault}{gmathIt%
          \PutIfValue{#1}}{%
        \skewchar\font\gmu@theskewchar\space}%
      \@nx\DeclareFontShape{\encodingdefault}{gmathIt%
          \PutIfValue{#1}}{m}{n}{%
        <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
    }\gmu@tempa
  }% of if #2 given
  \gmu@getfontdata{\rmfamily\upshape}%
  \edef\gmu@tempa{%
    \IfValueTF{#1}{\@nx\SetSymbolFont{gmathroman}{#1}}%
    {\@nx\DeclareSymbolFont{gmathroman}}%
    {\encodingdefault}{gmathrm\PutIfValue{#1}}{m}{n}%
    \@nx\DeclareFontFamily{\encodingdefault}{gmathrm%
        \PutIfValue{#1}}{%
      \skewchar\font\gmu@theskewchar\space}%
    \@nx\DeclareFontShape{\encodingdefault}{gmathrm%
        \PutIfValue{#1}}{m}{n}{%
      <->␣\gmu@fontscalebr␣\gmu@fontstring}{}%
    \IfValueT{#1}{%
      \@nx\SetMathAlphabet\@nx\mathrm{#1}{%
          \encodingdefault}{gmathrm#1}{m}{n}}%
  }\gmu@tempa
    \@xa\font\csname␣gmath@font\PutIfValue{#1}\endcsname
    =\gmu@fontstring\relax
    \gmathfamshook
}
\emptify\gmathfamshook
```

```
\gmathbase 4514 \DeclareCommand\gmathbase{oC{\NoValue}}{%
           4515   \gmathFams[#1](#2)%
 \gmath@do 4516   \DeclareCommand\gmath@do{%
           4517     m␣% (1) the character or CS to be declared,
           4518     o␣% (2) the Unicode to be assigned,
           4519     m␣% (3) math type (CS like \mathord etc.)
           4520     C{\gmath@fam}␣% font family
           4521   }{%
           4523     \gmath@getfamnum(##4)%
           4524     \IfValueTF{##2}{%
           4525       \edef\gmu@tempa{%
           4526         =␣\mathchar@type##3\space
           4527         \gmath@famnum\space
           4529         "##2\relax}%
           4530       \if\relax\@nx##1%
           4531         \gmu@ifstored{##1}{}{\StoreMacro##1}%
           4532         \edef\gmu@tempa{%
           4533           \XeTeXmathchardef␣\@nx##1\gmu@tempa}%
           4534         \else
           4535         \edef\gmu@tempa{%
           4536           \XeTeXmathcode␣`##1␣\gmu@tempa}
           4537         \fi%
           4538       }%
           4539       {% no value of ##2
           4540         \edef\gmu@tempa{%
           4541           \XeTeXmathcode␣`##1␣=
           4542           \mathchar@type##3\space
           4543           \gmath@famnum\space
           4545           `##1\relax}%
           4546       }%
           4547       \gmu@tempa
           4550     }% of \gmath@do
\gmath@doif 4552     \DeclareCommand\gmath@doif{%
           4553       m␣␣␣␣␣␣␣␣␣% (1) the Unicode hex of char enquired,
           4554       m␣␣␣␣␣␣␣% (2) the char or CS to be declared,
           4555       m␣␣␣␣␣␣␣% (3) math type CS(\mathord etc.),
           4556       o␣␣␣␣␣␣␣% (4) second-choice Unicode (taken if first-choice is absent),
           4557       o␣␣␣␣␣␣␣% (5) third-choice Unicode (as above if second-choice is absent
                           from font).
           4558       B{\gmath@fam}␣% (6) never used in this package. Why?
           4559       C{\NoValue}
           4560     }{%
           4561       \gmu@storeifnotyet{##2}%
           4562       \@xa\let\@xa\gmath@ft\csname
           4563         gmath@font%
           4564         \ifx\gmath@fam##6\else␣t\fi
           4565         \gmath@version
           4566       \endcsname
           4567       \iffontchar\gmath@ft"##1␣\gmath@do##2[##1]##3(##6)%
           4568       \else
           4569         \IfValueTF{##4}{%
           4570           \iffontchar\gmath@ft"##4␣%
```

```
                \gmath@do##2[##4]##3(##6)%
            \else
              \IfValueTF{##5}{%
                \iffontchar\gmath@ft"##5␣%
                    \gmath@do##2[##5]##3(##6)%
                \else
                  \gmath@restore{##1}{##2}{##3}{##4}{##5}{%
                      ##7}%
                \fi}%
              {\gmath@restore{##1}{##2}{##3}{##4}{##5}{##7}}%
            \fi}%
          {\gmath@restore{##1}{##2}{##3}{##4}{##5}{##7}}%
        \fi
      }% of \gmath@doif In the command above we try to define math char or
          a CS in the family given as ##6. If there're no respective chars, we try
          the same with the family given (as a word) in ##7.
\gmath@restore
      \def\gmath@restore##1##2##3##4##5##6{%
        \IfValueT{##6}%
        {\ifcsname␣##6\endcsname
            \edef\gmu@tempa{%
              \unexpanded{\gmath@doif{##1}{##2}{%
                  ##3}[##4][##5]}%
              {\@xa\@nx\csname##6\endcsname}\relax
            }\gmu@tempa
            \@xa\@gobbletwo␣% if family ##6 is defined, we gobble the other
                branch
          \fi
        }%
        \firstofone
        {\if\relax\@nx##2%
            \RestoreMacro##2%
          \fi
        }%
      }%
\iffalse␣% doesn't work in a non-math font.
\gmath@delc
      \DeclareCommand\gmath@delc{mo}{%
          %  #1   the char or CS to be declared,
          % [#2]  the Unicode (if not the same as the char).
      \gmath@getfamnum
      \IfValueTF{##2}{%
        \edef\gmu@tempa{%
          =␣\gmath@famnum\space␣"##2\relax}%
        \edef\gmu@tempa{%
          \XeTeXdelcode␣`##1␣\gmu@tempa}
      }%
      {%
        \edef\gmu@tempa{%
          \XeTeXdelcode␣`##1␣=
          \gmath@famnum\space
          `##1\relax}%
      }%
      \gmu@tempa
    }% of \gmath@delc
```

```
\gmath@delcif
  \def\gmath@delcif##1##2{%
       %  #1  the Unicode enquired,
       %  #2  the char to be delcode-declared
    \iffontchar\gmath@font"##1 \gmath@delc##2[##1]\fi}
\fi% of iffalse

\gmath@delimif
  \def\gmath@delimif##1##2##3{%
       %  #1  the Unicode enquired,
       %  #2  the CS defined as \XeTeXdelimiter,
       %  #3  the math type CS (probably \mathopen or \mathclose).
    \iffontchar\gmath@font"##1
      \gmath@getfamnum
      \protected\edef##2{\@nx\ensuremath{%
           \XeTeXdelimiter \mathchar@type##3\space
           \gmath@famnum\space "##1\relax}}%
    \fi}% of \gmath@delimif.

\rmopname
  \pdef\rmopname##1##2##3{%
    \mathop {##1\kern \z@ \mathrm{##3}}\csname n##2limits@%
        \endcsname
  }%

\gmu@dogmathbase
  \DeclareCommand\gmu@dogmathbase{oC{\NoValue}}{%
    \RestoreMacro\mathchar@type%
    \IfValueT{##1}{\mathversion{##1}}%
    \edef\gmath@version{\PutIfValue{##1}}%
    \@xa\let\@xa\gmath@fam\csname symgmathroman%
    \endcsname
    \edef\gmath@famm{symgmathRoman\gmath@version}% as you see, this
        is not a font family (number) but a macro containing the name: of the
        secondary ('rescue') family.

    \typeout{@@@ gmutils.sty: taking some math chars from
        the font^^J \gmu@fontstring@}%
    \gmath@do+\mathbin
    \gmath@doif{2212}-\mathbin[2013](\gmath@famm)% minus sign if
        present or else en dash
    \gmath@do=\mathrel
    \gmath@do0\mathord
    \gmath@do1\mathord
    \gmath@do2\mathord
    \gmath@do3\mathord
    \gmath@do4\mathord
    \gmath@do5\mathord
    \gmath@do6\mathord
    \gmath@do7\mathord
    \gmath@do8\mathord
    \gmath@do9\mathord
    \gmath@doif{2264}\le\mathrel(\gmath@famm)%
    \let\leq\le
    \let\leeng\le
    \gmath@doif{2265}\ge\mathrel(\gmath@famm)%
    \let\geq\ge
    \let\geeng\ge
    \gmath@doif{2A7D}\xleq\mathrel(\gmath@famm)%
```

```
\gmath@doif{2A7E}\xgeq\mathrel(\gmath@famm)%
\@ifpackageloaded{polski}{%
  \ifdefined\xleq
  \gmu@storeifnotyet\leq
  \let\leq=\xleq
  \let\le=\leq
  \fi
  \ifdefined\xgeq
  \gmu@storeifnotyet\geq
  \let\geq=\xgeq
  \let\ge=\geq
  \fi}{}%
\gmath@do.\mathpunct
\gmath@do,\mathpunct
\gmath@do;\mathpunct
\gmath@do…\mathpunct
\gmath@do(\mathopen
\gmath@do)\mathclose
\gmath@do[\mathopen
\gmath@do]\mathclose
\gmath@doif{00D7}×\mathbin␣(\gmath@famm)%
\gmath@do:\mathrel
\gmath@doif{00B7}·\mathbin(\gmath@famm)%
\gmath@doif{22C6}⋆\mathbin(\gmath@famm)% low star
\gmath@doif{2300}\varnothing\mathord(\gmath@famm)%
\gmath@doif{221E}\infty\mathord(\gmath@famm)%
\gmath@doif{2248}\approx\mathrel(\gmath@famm)%
\gmath@doif{2260}\neq\mathrel(\gmath@famm)%
\let\ne\neq
\gmath@doif{00AC}\neg\mathbin(\gmath@famm)%
\gmath@doif{00AC}\nego\mathord(\gmath@famm)%
\gmath@do/\mathop
\gmath@do<\mathrel
\gmath@do>\mathrel
\gmath@doif{2329}\langle\mathopen(\gmath@famm)%
\gmath@doif{232A}\rangle\mathclose(\gmath@famm)%
\gmath@doif{2202}\partial\mathord(\gmath@famm)%
\gmath@doif{00B1}\pm\mathbin(\gmath@famm)%
\gmath@doif{007E}\sim\mathrel(\gmath@famm)%
\gmath@doif{2190}\leftarrow\mathrel(\gmath@famm)%
\gmath@doif{2192}\rightarrow\mathrel(\gmath@famm)%
\gmath@doif{2194}\leftrightarrow\mathrel(\gmath@famm)% if not
    present, \gmathfurther will take care of it if left and right arrows are
    present.
\gmath@doif{2191}\uparrow\mathrel(\gmath@famm)% it should be
    a delimiter (declared with \gmath@delimif) but in a non-math font
    the delimiters don't work (2008/11/19) and I don't think I'll ever need
    up- and down- arrows as delimiters.
\gmath@doif{2193}\downarrow\mathrel(\gmath@famm)%
\gmath@doif{2208}\in\mathrel[03F5][0454](\gmath@famm)%
```

As a fan of modal logics I allow redefinition of `\lozenge` and `\square` iff both are in the font. I don't accept the 'ballot box' U+2610.

```
\if\iffontchar\gmath@font"25CA␣0\else␣1\fi
   \iffontchar\gmath@font"25FB␣0\else\iffontchar%
        \gmath@font"25A1␣0\else␣2\fi\fi
  \gmath@do\lozenge[25CA]\mathord
  \gmath@doif{25FB}\square\mathord[25A1](\gmath@famm)% 'medium white square (modal operator)' of just 'white square'.
\fi
\gmath@doif{EB08}\bigcircle\mathbin(\gmath@famm)%
\gmath@doif{2227}\wedge\mathbin(\gmath@famm)%
\gmath@doif{2228}\vee\mathbin(\gmath@famm)%
\gmath@doif{0393}\Gamma\mathalpha(\gmath@famm)%
\gmath@doif{0394}\Delta\mathalpha(\gmath@famm)%
\gmath@doif{0398}\Theta\mathalpha(\gmath@famm)%
\gmath@doif{039B}\Lambda\mathalpha(\gmath@famm)%
\gmath@doif{039E}\Xi\mathalpha(\gmath@famm)%
\gmath@doif{03A3}\Sigma\mathalpha(\gmath@famm)%
\gmath@doif{03A5}\Upsilon\mathalpha(\gmath@famm)%
\gmath@doif{03A6}\Phi\mathalpha(\gmath@famm)%
\gmath@doif{03A8}\Psi\mathalpha(\gmath@famm)%
\gmath@doif{03A9}\Omega\mathalpha(\gmath@famm)%
\@xa\let\@xa\gmath@fam\csname␣symletters\gmath@version%
\endcsname
\edef\gmath@famm{symgmathItalic\gmath@version}%
\gmath@doif{03B1}\alpha\mathalpha(\gmath@famm)%
\gmath@doif{03B2}\beta\mathalpha(\gmath@famm)%
\gmath@doif{03B3}\gamma\mathalpha(\gmath@famm)%
\gmath@doif{03B4}\delta\mathalpha(\gmath@famm)%
\gmath@doif{03F5}\epsilon\mathalpha(\gmath@famm)%
\gmath@doif{03B5}\varepsilon\mathalpha(\gmath@famm)%
\gmath@doif{03B6}\zeta\mathalpha(\gmath@famm)%
\gmath@doif{03B7}\eta\mathalpha(\gmath@famm)%
\gmath@doif{03B8}\theta\mathalpha(\gmath@famm)%
\gmath@doif{03D1}\vartheta\mathalpha(\gmath@famm)%
\gmath@doif{03B9}\iota\mathalpha(\gmath@famm)%
\gmath@doif{03BA}\kappa\mathalpha(\gmath@famm)%
\gmath@doif{03BB}\lambda\mathalpha(\gmath@famm)%
\gmath@doif{03BC}\mu\mathalpha(\gmath@famm)%
\gmath@doif{03BD}\nu\mathalpha(\gmath@famm)%
\gmath@doif{03BE}\xi\mathalpha(\gmath@famm)%
\gmath@doif{03C0}\pi\mathalpha(\gmath@famm)%
\gmath@doif{03A0}\Pi\mathalpha(\gmath@famm)%
\gmath@doif{03C1}\rho\mathalpha(\gmath@famm)%
\gmath@doif{03C3}\sigma\mathalpha(\gmath@famm)%
\gmath@doif{03DA}\varsigma\mathalpha(\gmath@famm)% 03C2?
\gmath@doif{03C4}\tau\mathalpha(\gmath@famm)%
\gmath@doif{03C5}\upsilon\mathalpha(\gmath@famm)%
\gmath@doif{03D5}\phi\mathalpha(\gmath@famm)%
\gmath@doif{03C7}\chi\mathalpha(\gmath@famm)%
\gmath@doif{03C8}\psi\mathalpha(\gmath@famm)%
\gmath@doif{03C9}\omega\mathalpha(\gmath@famm)%
\if␣1␣1%
\iffontchar\gmath@font"221A
  \fontdimen61\gmath@font=1pt
```

```
        \edef\sqrtsign{%
          \XeTeXradical␣\@xa\gmu@stripchar\meaning%
               \symgmathroman\space␣"221A\relax}%
      \fi
      \fi% of if 1 1.
\max  \def\max{\rmopname␣\relax␣m{max}}%
\min  \def\min{\rmopname␣\relax␣m{min}}%
\lim  \def\lim{\rmopname␣\relax␣m{lim}}%
\sin  \def\sin{\rmopname␣\relax␣o{sin}}%
\cos  \def\cos{\rmopname␣\relax␣o{cos}}%
 \tg  \def\tg{\rmopname␣\relax␣o{tg}}%
\ctg  \def\ctg{\rmopname␣\relax␣o{ctg}}%
\tan  \def\tan{\rmopname␣\relax␣o{tan}}%
\ctan \def\ctan{\rmopname␣\relax␣o{ctan}}%
    }% of \gmu@dogmathbase
    \AtBeginDocument{\gmu@dogmathbase[#1](#2)%
      \let\gmathbase\gmu@dogmathbase
    }% of atbd
      \not@onlypreamble\gmathbase
}% of \gmathbase
```

The `\gmatbase` declaration defines a couple of gmath defining commands and then launches them for the default font at begin document and becomes only that launching.

```
\@onlypreamble\gmathbase
```

It's a bit tricky: if `\gmathbase` occurs first time in a document inside `document` then an error error is raised. But if `\gmathbase` occurs first time in the preamble, then it removes itself from the only-preamble list and redefines itself to be only the inner macro of the former itself.

```
\gmathfurther \pdef\gmathfurther{%
    \def\do##1##2##3{\gmu@storeifnotyet##1%
      \def##1{%
        \mathop{\mathchoice{\hbox{%
               \rm
               \edef\gma@tempa{\the\fontdimen8\font}%
               \larger[3]%
               \lower\dimexpr(\fontdimen8\font-\gma@tempa)/2␣%
                 \hbox{##2}}}{\hbox{%
                 \rm
                 \edef\gma@tempa{\the\fontdimen8\font}%
                 \larger[2]%
                 \lower\dimexpr(\fontdimen8\font-%
                      \gma@tempa)/2␣%
                 \hbox{##2}}}%
             {\mathrm{##2}}{\mathrm{##2}}}##3}}%
      \iffontchar\gmath@font"2211␣␣␣␣\do\sum{\char"2211}{}%
          \fi%
      \do\forall{\gma@quantifierhook␣\rotatebox[origin=c]{%
          180}{A}%
        \gmu@forallkerning
      }{\nolimits}%
      \def\gmu@forallkerning{\setbox0=\hbox{A}\setbox2=\hbox{%
          \scriptsize␣x}%
```

```
      \kern\dimexpr\ht2/3*2 -\wd0/2\relax}% to be able to redefine it
                                              when the big quantifier is Bauhaus-like.
    \do\exists{\rotatebox[origin=c]{180}{\gma@quantifierhook E}}\nolimits%
    \def\do##1##2##3{\gmu@storeifnotyet##1%
      \def##1{##3{%
          \mathchoice{\hbox{\rm##2}}{\hbox{\rm##2}}%
          {\hbox{\rm\scriptsize##2}}{\hbox{\rm\tiny##2}}}}}%
    \unless\iffontchar\gmath@font"2227
      \do\vee{\rotatebox[origin=c]{90}{<}}\mathbin%
    \fi
    \unless\iffontchar\gmath@font"2228
      \do\wedge{\rotatebox[origin=c]{-90}{<}}\mathbin
    \fi
    \unless\iffontchar\gmath@font"2194
      \if\iffontchar\gmath@font"2190 0\else1\fi
        \iffontchar\gmath@font"2192 0\else2\fi
        \do\leftrightarrow{\char"2190\kern-0,1em %
            \char"2192}\mathrel
    \fi\fi
    \def\do##1##2##3{\gmu@storeifnotyet##1%
      \def##1####1{##2{\hbox{%
            \rm
            \setbox0=\hbox{####1}%
            \edef\gma@tempa{\the\ht0}%
            \edef\gma@tempb{\the\dp0}%
            ##3%
            \setbox0=\hbox{####1}%
          \lower\dimexpr(\ht0 +\dp0)/2-\dp0 -((%
              \gma@tempa+\gma@tempb)/2-\gma@tempb) %
          \box0}}}%
  \do\bigl\mathopen\larger
  \do\bigr\mathclose\larger
  \do\Bigl\mathopen\largerr
  \do\Bigr\mathclose\largerr
  \do\biggl\mathopen{\larger[3]}%
  \do\biggr\mathclose{\larger[3]}%
  \do\Biggl\mathopen{\larger[4]}%
  \do\Biggr\mathclose{\larger[4]}%
  \addtotoks\everymath{%
    \def\do##1##2{\gmu@storeifnotyet##1%
      \def##1{\ifmmode##2{\mathchoice
          {\hbox{\rm\char`##1}}{\hbox{\rm\char`##1}}%
          {\hbox{\rm\scriptsize\char`##1}}{\hbox{\rm\tiny\char`##1}}}%
        \else\char`##1\fi}}%
    \do\{{\mathopen
    \do\}\mathclose
    \def\={\mathbin{=}}%
    \def\neqb{\mathbin{\neq}}%
    \let\neb\neqb
    \def\do##1{\gmu@storeifnotyet##1%
```

```
          \edef\gma@tempa{%
            \def\@xa\@nx\csname␣\@xa\gobble\string##1r%
                 \endcsname{%
              \@nx\mathrel{\@nx##1}}}%
          \gma@tempa}%
        \do\vee␣\do\wedge␣\do\neg
        \def\fakern{\mkern-3mu}%
        \thickmuskip=8mu␣plus␣4mu\relax
        \gma@gmathhook
    }% of \everymath.
    \everydisplay\everymath
    \ifdefined\Url
      \ampulexdef\Url{\let\do}\@makeother
      {\everymath{}\let\do\@makeother}% I don't know why but the url
           package's \url typesets the argument inside a math which caused
           digits not to be typewriter but Roman and lowercase.
  \fi% of \ifdefined\Url.
}% of \def\gmathfurther.

\StoreMacro\mathchar@type

\gmath \DeclareCommand\gmath{oC{\NoValue}}{%
  \gmathbase[#1](#2)%
  \gmathfurther
  \IfValueT{#1}{\csname␣gmathhook#1\endcsname}% this allows adding
       version-specific stuff (I first used this for Fell fonts rescued with Garamond
       Premier)
}

\gmathscripts \pdef\gmathscripts{%
  \addtotoks\everymath{\catcode`\^=7\relax␣\catcode`\_=8%
       \relax␣}%
  \everydisplay\everymath}

\gmathcats \pdef\gmathcats{%
  \addtotoks\everymath{\gmu@septify}%
  \everydisplay\everymath}

\emptify\gma@quantifierhook
\quantifierhook \def\quantifierhook#1{%
\gma@quantifierhook   \def\gma@quantifierhook{#1}}

\emptify\gma@gmathhook
\gmathhook \def\gmathhook#1{\addtomacro\gma@gmathhook{#1}}

\gma@dollar \def\gma@dollar$#1${{\gmath$#1$}}%
\gma@bare \def\gma@bare#1{\gma@dollar$#1$}%
\gma@checkbracket \def\gma@checkbracket{\@ifnextchar\[%
  \gma@bracket\gma@bare}
\gma@bracket \def\gma@bracket\[#1\]{{\gmath\[#1\]}\@ifnextchar\par{}{%
       \noindent}}
\gma \def\gma{\@ifnextchar$%
  \gma@dollar\gma@checkbracket}

\garamath \DeclareCommand\garamath{%
  O{\rm}% the font command
}{%
```

Before 2009/10/19 all the stuff was added to \everymath which didn't work.

```
4992    \quantifierhook{\addfontfeature{OpticalSize=800}}%
\gma@arrowdash 4994    \def\gma@arrowdash{{%
4995        \setbox0=\hbox{\char"2192}\copy0\kern-0,6\wd0
4996        \bgcolor\rule[-\dp0]{0,6\wd0}{\dimexpr1,07\ht0+\dp0}%
                \kern-0,6\wd0}}%
\gma@gmathhook 4998    \def\gma@gmathhook{%
4999      \def\do####1####2####3{\gmu@storeifnotyet####1%
5000        \def####1{####3{%
  \mathchoice 5001            \mathchoice{\hbox{#1####2}}{\hbox{#1####2}}%
5002            {\hbox{#1\scriptsize####2}}{\hbox{#1%
                    \tiny####2}}}}}%
5003      \do\mapsto{\rule[0,4ex]{0,1ex}{0,4ex}\kern-0,05em%
5004        \gma@arrowdash\kern-0,05em\char"2192}\mathrel
5005      \do\cup{\scshape␣u}\mathbin
5006      \do\varnothing{\setbox0=\hbox{\gma@quantifierhook%
              \addfontfeature{Scale=1.272727}0}%
5007        \setbox2=\hbox{\char"2044}%
5008        \copy0␣\kern-0,5\wd0␣\kern-0,5\wd2␣\lower0,125\wd0␣%
              \copy2
5009        \kern0,5\wd0\kern-0,5\wd2}{}% of \varnothing
5010      \do\leftarrow{\char"2190\kern-0,05em\gma@arrowdash}%
              \mathrel
5011      \do\shortleftarrow{\char"2190}\mathrel
5012      \do\rightarrow{\gma@arrowdash\kern-0,05em\char"2192}%
              \mathrel
5013      \do\shortrightarrow{\char"2192\relax}\mathrel
5014      \do\in{\gma@quantifierhook\char"0454}\mathbin
5015      \do\prec{\gma@quantifierhook
5016        \rotatebox[origin=c]{-90}{%
5017          \glyphname{u03A5.a}}}\mathrel␣% added 2009/9/11
5018    }% of \gma@gmathhook
5019 }% of \garamath.
```

Minion and Garamond Premier kerning and ligature fixes

»Ws« shall not make long »s« because long »s« looks ugly next to »W«.

```
\gmu@tempa 5028 \def\gmu@tempa{\kern-0,08em\penalty10000\hskiposp\relax
5029   s\penalty10000\hskiposp\relax}

5031 \protected\edef\Vs{V\gmu@tempa}

5033 \protected\edef\Ws{W\gmu@tempa}

\Wz 5035 \pdef\Wz{W\kern-0,05em\penalty10000\hskiposp\relax␣z}
```

A left-slanted font

Or rather a left Italic *and* left slanted font. In both cases we sample the skewness of the itshape font of the current family, we reverse it and apply to \itshape in \litshape and \textlit and to \sl in \lsl. Note a slight asymmetry: \litshape and \textlit take the current family while \lsl and \textlsl the basic Roman family and basic (serif) Italic font. Therefore we introduce the \lit declaration for symmetry, that declaration left-slants \it.

I introduced them first while typesetting E. Szarzyński's *Letters* to follow his (elaborate) hand-writing and now I copy them here when need left Italic for his *Albert Camus' The Plague* to avoid using bold font.

Of course it's rather esoteric so I wrap all that in a declaration.

```
\leftslanting@ 5057 \pdef\leftslanting@{%
    \litdimen  5058   \def\litdimen{\strip@pt\fontdimen1\font␣ex}%
\litcorrection 5059   \def\litcorrection{%
               5060     \ifhmode\null\nobreak\hskip\litdimen\relax\fi}%
     \litkern  5061   \def\litkern{% note it's to be used inside the left slanted font, unlike \lit¦
                          correction, intended to be used before switching to left slant/italic.
               5064     \leavevmode\null
               5065     \kern-\litdimen\relax}%
   \dilitkern  5066   \def\dilitkern{\kern\litdimen\litkern}%

    \litshape  5068   \pdef\litshape{%
               5070     \litcorrection
               5071     \itshape
               5072     \@tempdima=-2\fontdimen1\font
               5073     \advance\leftskip␣by\strip@pt\fontdimen1\font␣ex␣% to assure
                            at least the lowercase letters not to overshoot to the (left) margin. Note
                            this has any effect only if there is a \par in the scope.
               5077     \litcorrection
               5078     \edef\gmu@tempa{%
               5079       \@nx\addfontfeature{FakeSlant=\strip@pt\@tempdima}}%
                              when not \edefed, it caused an error, which is perfectly
                              understandable.
               5082     \gmu@tempa}%
    \textlit   5085   \pdef\textlit##1{%
               5086     {\litshape##1}}%
       \lit    5088   \pdef\lit{\rm\litshape}%
       \lsl    5091   \pdef\lsl{{%
               5092       \litcorrection
               5093       \it
               5096       \@tempdima=-\fontdimen1\font
               5097       \litcorrection
               5098       \xdef\gmu@tempa{%
               5099         \@nx\addfontfeature{RawFeature={slant=\strip@pt%
                                \@tempdima}}}}%
               5100     \rm␣␣% Note in this declaration we left-slant the basic Roman font not the
                            itshape of the current family.
               5102     \gmu@tempa}%
```

Now we can redefine `\em` and `\emph` to use left Italic for nested emphasis. In Polish typesetting there is bold in nested emphasis as I have heard but we don't like bold since it perturbs homogeneous greyness of a page. So we introduce a three-cycle instead of two-: Italic, left Italic, upright.

```
      \em  5110   \pdef\em{%
           5111     \ifdim\fontdimen1\font=\z@␣␣\itshape
           5112     \else
           5113       \ifdim\fontdimen1\font>\z@␣\litshape
           5114       \else␣\upshape
           5115       \fi
           5116     \fi}%
```

```
    \pdef\emph##1{%
      {\em##1}}%
}% of \leftslanting@.
```

\leftslanting
```
\pdef\leftslanting{\AtBeginDocument\leftslanting@}
\AtBeginDocument{\let\leftslanting\leftslanting@}
```

Fake Old-style Numbers

While preparing documentation of this package I faced an æsthetic problem of lack of old-style numbers in a font I fancy. The font is for the sans serif and the digits occur only in the date in title so it would be a pity not too use a nice font when only one or two numbers are needed.

\romorzero
```
\def\romorzero#1{%
  \ifnum#1=0␣zero\else\romannumeral#1␣\fi}
```

\fakeonum
```
\DeclareCommand\fakeonum{%
  o␣%  fake bold for the digit »2« (for which emboldening improves look),
  >Pm␣␣␣%  the text to fake old-style numbers in.
}{%  I tried to use this command as a declaration but active digits are very uncomfortable, e.g. you can't define macros with arguments.
  \gmu@if@onum{#2}{%
    \begingroup
    \edef\gmu@tempa{#2}%
    \makeatletter%
    \IfValueT{#1}{%
      \prependtomacro\fake@onum@ii{%
        \begingroup\addfontfeature{FakeBold=#1}}%
      \addtomacro\fake@onum@ii\endgroup
    }%
    \endlinechar\m@ne␣%  to suppress the line end added by \scantokens, especially in active ^^M's scopes.
    \gmu@dofakeonum
    \@xa\scantokens\@xa{\gmu@tempa}%
    \endgroup
  }% of \gmu@ifonum 'else'.
}% of \fakeonum.
```

\gmu@dofakeonum
```
\def\gmu@dofakeonum{%
  \def\do##1{%
    \catcode`##1\active
    \scantokens{%
      \@xa\let\@xa##1%
      \csname␣fake@onum@\@xa\romorzero\string##1\endcsname%
          \empty}}%
  \do0\do1\do2\do3\do4\do5\do6\do7\do8\do9%
}

\def\do#1#2{%
  \@namedef{fake@onum@\romorzero#1}{#2}}
```

\gmu@tempa
```
\def\gmu@tempa#1{%
  \do#1{\leavevmode
    \gmu@calculateslant{#1}%  uses \gmu@tempa and \gmu@tempb, therefore goes first. And defines \gmu@tempd.
```

```
      \gmu@measurewd{#1}% the width of char #1 is in \gmu@tempa without
           kerning and in \gmu@tempb with kerning.
      \edef\gmu@tempc{\the\fontcharht\font`#1}%
      \hbox to \gmu@tempb {%
        \hss\resizebox{\gmu@tempa}%
        {\dimexpr\fontdimen5\font+\gmu@tempc-\fontdimen8%
             \font}%
        {\gmu@tempd#1}\hss}}}
```

`\gmu@measurewd`

```
\def\gmu@measurewd#1{%
  \edef\gmu@tempa{\the\fontcharwd\font`#1}%
  \settowidth{\@tempdimb}{% to preserve kerning
    \char`#1\char`#1\char`#1\char`#1\char`#1\char`#1%
    \char`#1\char`#1\char`#1\char`#1\char`#1\char`#1%
    \char`#1\char`#1\char`#1\char`#1\char`#1\char`#1%
    \char`#1\char`#1\char`#1}%
  \edef\gmu@tempb{\the\dimexpr(\@tempdimb-\gmu@tempa)/20}%
}
```

```
\gmu@tempa0 % \fake@onum@zero
\gmu@tempa1 % \fake@onum@i
\gmu@tempa2 % \fake@onum@ii
```

`\gmu@tempa`

```
\def\gmu@tempa#1{%
  \do#1{\leavevmode
    \gmu@measurewd{#1}%
    \lower
    \dimexpr\fontdimen8\font-\fontdimen5\font\relax
    \hbox to \gmu@tempb {\hss#1\hss}}%
}
```

```
\gmu@tempa3 % \fake@onum@iii
\gmu@tempa4 % \fake@onum@iv
\gmu@tempa5 % \fake@onum@v
\gmu@tempa7 % \fake@onum@vii
\gmu@tempa9 % \fake@onum@ix
```

`\gmu@tempa`

```
\def\gmu@tempa#1{% to preserve pseudo-kerning in digits sequences.
  \do#1{\leavevmode
  \gmu@measurewd#1%
  \hbox to \gmu@tempb {\hss#1\hss}}}
```

```
\gmu@tempa6 % \fake@onum@vi
\gmu@tempa8 % \fake@onum@viii
```

`\gmu@if@onum`

```
\protected\def\gmu@if@onum{%
  \edef\gmu@tempa{\@xa\meaning\the\font}%
  \@xa\@ifinmeaning\detokenize{+onum}\of\gmu@tempa
}
```

Thus `\gmu@if@onum` becomes a two-argument command that executes its `#1` if there is `+onum` in current font specification or its `#2` if `+onum` is absent.

One could easily generalise `\gmu@if@onum` to `\@if@fontfeature`, i.e. to a test for an arbitrary font feature, probably with employing that very nice feature specification of fontspec, so that you could write `\IfFontFeature{Numbers=OldStyle}{}{fake old-style digits}`.

`\gmu@getslant`

```
\pdef\gmu@getslant{% we define \gmu@tempa to the (fake) slant of current
```

font.

```
    \edef\gmu@tempa{\@xa\meaning\the\font\detokenize{%
        slant=0,}}%
    \edef\gmu@tempb{%
      \def\@nx\gmu@tempb####1%
      \detokenize{slant=}%
      ####2,####3}%
    \gmu@tempb\@@nil{##2}%
    \edef\gmu@tempa{\@xa\gmu@tempb\gmu@tempa\@@nil␣pt}%
}
\def\gmu@calculateslant#1{%
    \gmu@getslant
    \edef\gmu@tempa{\the\numexpr\dimexpr\fontdimen1\font␣+␣%
        \gmu@tempa}%
    \edef\gmu@tempa{\the\numexpr␣\gmu@tempa␣*
      \numexpr\fontdimen5\font\relax/\numexpr\fontcharht%
          \font`#1
      \relax}%
    \edef\gmu@tempd{%
      \the\dimexpr␣\gmu@tempa␣sp␣␣-␣\fontdimen1\font}%
    \ifdim\gmu@tempd=\z@␣\emptify\gmu@tempd
    \else\edef\gmu@tempd{%
      \@nx\addfontfeature{FakeSlant=\strip@pt\dimexpr%
          \gmu@tempd}}%
    \fi}
\DeclareCommand\gmu@cepstnof{O{\gmu@tempa}%
    s%
    m%
    O{,␣Scale=MatchLowercase}%
}
{%
    \def\gmu@reservedc##1:##2:\@@nil{%
      \ifx:##2:\else␣RawFeature={\gmu@maybestripcomma##2,,%
          \@@nil}␣\fi}%
    \def\gmu@reservedd##1/##2/\@@nil{%
      \ifx&##2&\@xa\@firstoftwo\else\@xa\@secondoftwo\fi}%
    \def\gmu@reservede##1:##2:\@@nil{%
      \ifx&##2&\@xa\@firstoftwo\else\@xa\@secondoftwo\fi}%
    \edef\gmu@reserveda{%
```

now, reserved B parses the font name and features. It uses an auxiliary reserved C because after / may be or may not be features specification.

```
      \@xa\@xa\@xa\gmu@reservedd\@xa\meaning\the\font//\@@nil
```

```
      {% font name doesn't contain a slash
        \@xa\@xa\@xa\gmu@reservede\@xa\meaning\the\font::%
            \@@nil
        {% nor does it contain a colon
          \def\@nx\gmu@reservedb\detokenize{select␣font
            "}####1\detokenize{"}####2\@nx\@@nil{%
            \ifx\fontspec#3%
              \@nx\@nx\@nx\fontspec[\@gobble#4\@empty]{####1}% gobble
                  a comma
            \else
              #3Font={####1},␣#3Features={\@gobble␣#4%
                  \@empty}%
            \fi
          }%
        }%
        {% no slash but there is a colon
          \def\@nx\gmu@reservedb\detokenize{select␣font
            "}####1:####2\detokenize{"}####3\@nx\@@nil{%
            \ifx\fontspec#3%
              \@nx\@nx\@nx\fontspec[\@nx\gmu@reservedc####2::%
                  \@nx\@@nil#4]{####1}%
            \else
              #3Font={####1},␣#3Features={\@nx%
                  \gmu@reservedc####2::\@nx\@@nil#4}%
            \fi
          }%
        }%
      }% of 'no slash' case
      {% font name contains a slash
        \def\@nx\gmu@reservedb\detokenize{select␣font
          "}####1/####2\detokenize{"}####3\@nx\@@nil{%
          \ifx\fontspec#3%
            \@nx\@nx\@nx\fontspec[\@nx\gmu@reservedc####2::%
                \@nx\@@nil#4]{####1}%
          \else
            #3Font={####1},␣#3Features={\@nx%
                \gmu@reservedc####2::\@nx\@@nil#4}%
          \fi
        }% of \gmu@reservedb
      }% of 'slash present' case
    }\gmu@reserveda
    \xdef#1{\@xa\@xa\@xa\gmu@reservedb\@xa\meaning\the\font%
        \@@nil}%
}%

\gmu@maybestripcomma  \def\gmu@maybestripcomma#1,,#2\@@nil{#1}

    \gmu@setbasefont  \pdef\gmu@setbasefont{\@xa\let\@xa\gmu@basefont\the\font}

                      \let\setbasefont\gmu@setbasefont

     \gmu@calc@scale  \DeclareCommand\gmu@calc@scale{%
      O{1}% a factor,
      m% number of the fontdimen
      }
```

```
{\begingroup
```

We 'descale' the current font:

```
  \gmu@cepstnof\fontspec[,␣Scale=1]\gmu@tempa
  \@xa\let\@xa\gmu@currfont@descaled\the\font
  \gmu@basefont
  \gmu@cepstnof\fontspec[,␣Scale=1]\gmu@tempa
```

now also the base font is descaled.

```
  \xdef\gmu@fontscale{%
    \strip@pt
    \dimexpr␣␣1pt␣*
     \numexpr\dimexpr#1\fontdimen#2\font\relax\relax␣/
    \numexpr\fontdimen#2\gmu@currfont@descaled\relax
    \relax}%
  \endgroup}
```

Varia

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

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

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

```
\AtBeginDocument{%
```
to bypass redefinition of \~ as a text command with various encodings

\texttilde
```
  \pdef\texttilde{%
    \@ifnextchar/{\gmu@tilde\kern-0,1667em\relax}\gmu@tilde}}
```

We prepare the proper kerning for "~/".

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

(This definition is repeated in gmverb.)

```
\foone{\catcode`\␣\active}%
```

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

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

```
\foone{\catcode`\^^M\active}%
```
the comment signs here are crucial.

\defobeylines
```
{\def\defobeylines{\catcode`\^^M=13␣\def^^M{\par}}}
```

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

\deksmallskip
```
\def\deksmallskip{\vskip\smallskipamount}
```

\undeksmallskip
```
\def\undeksmallskip{\vskip-\smallskipamount}
```

\dekmedbigskip
```
\def\dekmedbigskip{\vskip\glueexpr␣\medskipamount+%
      \smallskipamount}
```

\dekmedskip
```
\def\dekmedskip{\vskip\medskipamount}
```

```
\dekbigskip 5430 \def\dekbigskip{\vskip\bigskipamount}
\hfillneg   5433 \def\hfillneg{\hskip␣opt␣plus␣-1fill\relax}
```

A mark for the **TO-DO!**s:

```
\TODO       5438 \newcommand*{\TODO}[1][]{{%
            5439     \sffamily\bfseries\huge␣TO-DO!\if\relax#1\relax\else%
                         \space\fi#1}}
```

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

```
\twocoltoc  5474 \newcommand*\twocoltoc{%
            5475   \RequirePackage{multicol}%
\@starttoc  5476   \def\@starttoc##1{%
            5477     \begin{multicols}{2}\makeatletter\@input␣{\jobname␣
                         .##1}%
            5478       \if@filesw␣\@xa␣\newwrite␣\csname␣tf@##1\endcsname
            5479         \immediate␣\openout␣\csname␣tf@##1\endcsname␣%
                             \jobname␣.##1\relax
            5480       \fi
            5481       \@nobreakfalse\end{multicols}}}

            5483 \@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 5488 \DeclareCommand\enoughpage{%
            5489   Q{+-0123456789}% (optional short version (number of \baselineskips))
            5490   B{2\baselineskip}% (2) optional (formerly mandatory) long version of re-
                       quired room on a page
            5492   >is
            5493   B{}␣% (3) what if the room is enough
            5494   >isB{\newpage}␣% (4) what if there's to little room on a page
            5495 }{%
            5502   \if
            5503   \ifdim\dimexpr\pagegoal-\pagetotal<\dimexpr
            5504   \IfValueTF{#1}{#1\baselineskip}{#2}\relax
            5505    1\else0\fi
            5506    \unless\ifdim\dimexpr\pagegoal-\pagetotal<\z@
            5507     1\else2\fi
```

We check both whether space left on page is larger than we check *and* whether it's nonnegative (the latter happens when we are already on the next page). Therefore this test will not work properly for large values of `#1` or values close to `\textwidth`.

```
            5518     \@xa\@firstoftwo
            5519   \else
            5520     \@xa\@secondoftwo
            5521   \fi
            5522   {#4}{#3}%
            5523 }
```

An equality sign properly spaced:

```
\equals     5532 \pdef\equals{\hunskip${}={}$\ignorespaces}
```

And for the LaTeX's pseudo-code statements:

\eequals
```
\pdef\eequals{\hunskip${}=={}$\ignorespaces}
```
\·
```
\pdef\·{\hunskip${}·{}$\ignorespaces}
```

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

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

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

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

\ShowFont
```
\newcommand*\ShowFont[1][6]{%
  \begin{multicols}{#1}[The␣current␣font␣(the␣\f@encoding\␣
      encoding):]
    \parindent\z@
    \count\z@\m@ne
    \@whilenum\count\z@<\@cclv\do{
      \advance\count\z@\@ne
      \␣\the\count\z@:~\char\count\z@\par}
  \end{multicols}}
```

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

\liturgiques
```
\newcommand*\liturgiques[1][red]{%  Requires the color package.
  \gmu@RPfor{xcolor}\color%
```
\czerwo
```
  \newcommand*\czerwo{\small\color{#1}}%  environment
```
\czer
```
  \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.
  \StoreMacro\*%
```
*
```
  \def\*{\czer{\storedcsname{*}}}%
```
\+
```
  \def\+{\czer{†}}%
```
\nieczer
```
  \newcommand*\nieczer[1]{\textcolor{black}{##1}}%
}
```

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

\gmu@RPfor
```
\newcommand*\gmu@RPfor[3][]{%
  \ifx\relax#1\relax\emptify\gmu@resa
```
\gmu@resa
```
  \else␣\def\gmu@resa{[#1]}%
```

```
    \fi
    \@xa\RequirePackage\gmu@resa{#2}}
```

Since inside `document` we cannot load a package, we'll redefine `\gmu@RPfor` to issue a request before the error issued by undefined CS.

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

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

```
\continuum  \pprovide\continuum{%
      \gmu@RPfor{eufrak}\mathfrak\ensuremath{\mathfrak{c}}}
```

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

```
\iteracro  \def\iteracro{%
    \acro    \pdef\acro##1{%
        \begingroup
        \acropresetting
        \gmu@acrospaces##1␣\gmu@acrospaces
        \endgroup
    }%
}

\emptify\acropresetting

\iteracro

\gmu@acrospaces  \def\gmu@acrospaces#1␣#2\gmu@acrospaces{%
      \gmu@acroinner#1\gmu@acroinner
      \ifx\relax#2\relax\else
        \space
        \afterfi{\gmu@acrospaces#2\gmu@acrospaces}%
```

when `#2` is nonempty, it is ended with a space. Adding one more space in this line resulted in an infinite loop, of course.

```
      \fi}

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

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

```
\acrocore 5665 \pdef\acrocore{\smaller␣% was: \scshape\lowercase
          5666 }
```

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

```
\IMHO 5671 \def\IMHO{\acro{IMHO}}
\AKA  5672 \def\AKA{\acro{AKA}}
\usc  5674 \pdef\usc#1{{\addfontfeature{Letters=UppercaseSmallCaps}#1}}
\uscacro 5676 \def\uscacro{\let\acro\usc}
```

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

```
\secondclass   5694 \def\secondclass{%
\ifSecondClass 5695   \newif\ifSecondClass
               5696   \SecondClasstrue
               5697   \@fileswithoptions\@clsextension}% [outeroff,gmeometric]{gm¦
                        docc} it's loading gmdocc.cls with all the bells and whistles except the error
                        message.
```

Cf. *The TEXbook* ex. 11.6.
A line from LATEX:

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

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

```
\gmu@dekfraccsimple 5709 \def\gmu@dekfraccsimple#1/#2{\leavevmode\kern.1em
                    5710   \raise.7ex\hbox{%
                    5711     \gmu@fracfontsetup#1}\gmu@numeratorkern
                    5712   \dekfraccslash\gmu@denominatorkern
                    5714   {%
                    5715     \gmu@fracfontsetup#2}%
                    5716   \if@gmu@mmhbox\egroup\fi}
\gmu@fracfontsetup 5718 \def\gmu@fracfontsetup{%
                   5719   \smaller[3]\addfontfeature{FakeBold=1}}
\dekfraccsimple 5722 \def\dekfraccsimple{%
                5723   \let\dekfracc@args\gmu@dekfraccsimple
                5724 }
\dekfraccslash  5725 \@ifXeTeX{\def\dekfraccslash{\char"2044␣}}
\dekfraccslash  5726 {\def\dekfraccslash{/}}␣% You can define it as the fraction
                          slash, \char"2044␣
                5728 \dekfraccsimple
```

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

```
\ikern 5736 \newcommand*\ikern{\,\penalty\@M\hskip\z@skip\relax}
```

And a macro to forbid hyphenation of the next word:

```
\nohy  5740 \pdef\nohy{\leavevmode\kernosp\relax}
\yeshy 5741 \pdef\yeshy{\leavevmode\penalty\@M\hskip\z@skip}
```

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

`\include` not only .tex's

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

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

`\gmu@getext`

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

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

`\@include`

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

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

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

`\whenonly` `\gmu@whonly`

```
\newcommand\whenonly[3]{%
  \def\gmu@whonly{#1,}%
```

```
\ifx\gmu@whonly\@partlist\afterfi{#2}\else\afterfi{#3}\fi}
```

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

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

Switching on and off parts of one file

The \include facility is very nice only it forces you to split your source in many files. Therefore I provide a tool analogous to \include and using the same \includeonly mechanism/list to switch on and off parts of the same source file.

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

\@filepart
```
\def\@filepart#1␣{%
  \clearpage
  \edef\gmu@filepartname{#1}% we'll use it later
  \if@filesw
    \immediate\write\@mainaux{\string\@input{#1.aux}}%
  \fi
  \@tempswatrue
  \if@partsw
    \@tempswafalse
    \@for\reserved@a:=\@partlist\do{%
      \ifx\reserved@a\gmu@filepartname\@tempswatrue\fi}%
  \fi
  \if@tempswa
    \let\@auxout\@partaux
    \if@filesw
      \immediate\openout\@partaux␣#1.aux
      \immediate\write\@partaux{\relax}%
    \fi
    \@xa\@firstoftwo
  \else
```

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

```
    \deadcycles\z@
    \@nameuse{cp@\gmu@filepartname}%
    \let\@auxout\@mainaux
    \@xa\@secondoftwo
  \fi
  {\iftrue}%
  {\let\endfilepart\fi
    \csname␣gm@skipped@#1\endcsname
    \def\next{\RestoreMacro\endfilepart
      \@ifnextchar\bgroup{\show\NextBgroup\@gobble}{}}%
    \@xa\next\iffalse}%
}
```

\endfilepart
```
\DeclareCommand\endfilepart{b}{% Note the argument is not used really.
```
Maybe later we'll use it for checking of proper matching. Or maybe not.

```
    \inclasthook
    \clearpage
    \@writeckpt{\gmu@filepartname}%
    \if@filesw
    \immediate\closeout\@partaux
    \fi
    \fi% this \fi closes \Iftrue put by line 5841.
    \let\@auxout\@mainaux
}

\StoreMacro\endfilepart

\nofileparts \def\nofileparts{%
    \let\filepart\@gobble
\endfilepart \DeclareCommand\endfilepart{b}{}%
}
```

Fix of including when fontspec is used

The fontspec package creates counters for font families. If a fontspec command is used in a part of a document and then such a part is skipped, an error occurs 'No counter zf@fam@... defined'. Now we fix that by ensuring all the counters are defined before they are set.

Note it's a draft version which doesn't support resetting of one counter within another.

```
\includecountfix \def\includecountfix{%
  \@wckptelt      \def\@wckptelt##1{%
                    \immediate\write\@partaux{%
                      \providecounter{##1}% to provide the font counters defined in parts of the document.
                      \string\setcounter{##1}{\the\@nameuse{c@##1}}}}%
                  }

\providecounter \pdef\providecounter#1{%
          #1      \unless\ifcsname␣c@#1\endcsname\newcounter{#1}\fi}
```

Faked small caps

```
\gmu@scapLetters \def\gmu@scapLetters#1{%
                   \ifx#1\relax\relax\else% two \relaxes to cover the case of empty #1.
                     \ifcat␣a\@nx#1\relax
                       \ifnum\the\lccode`#1=`#1\relax
                         {\fakescapscore\MakeUppercase{#1}}% not Plain \uppercase because that works bad with inputenc.
                       \else#1%
                       \fi
                     \else#1%
                     \fi%
                     \@xa\gmu@scapLetters
                   \fi}%
\gmu@scapSpaces \def\gmu@scapSpaces#1␣#2\@@nil{%
                  \ifx#1\relax\relax
                  \else\gmu@scapLetters#1\relax
                  \fi
```

```
    \ifx#2\relax\relax
    \else\afterfi{\␣\gmu@scapSpaces#2\@@nil}%
    \fi}
\gmu@scapss \def\gmu@scapss#1\@@nil{{\def~{{\nobreakspace}}%
\nobreakspace     \gmu@scapSpaces#1␣\@@nil}}% %␣\def\\{{\newline}}\relax adding
                      redefinition of \\ caused stack overflow. Note it disallows hyphenation
                      except at \-.
 \fakescaps \pdef\fakescaps#1{{\gmu@scapss#1\@@nil}}
            \let\fakescapscore\gmu@scalematchX
```

Experimente z akcentami patrz no3.tex.

```
  \tinycae \def\tinycae{{\tiny\AE}}% to use in \fakescaps[\tiny]{...}
           \RequirePackage{calc}
```

wg `\zf@calc@scale` pakietu fontspec.

```
           \@ifpackageloaded{fontspec}{%
\gmu@scalar  \def\gmu@scalar{1.0}%
  \zf@scale  \def\zf@scale{}%
\gmu@scalematchX  \def\gmu@scalematchX{%
             \begingroup
\gmu@scalar    \ifx\zf@scale\empty\def\gmu@scalar{1.0}%
               \else\let\gmu@scalar\zf@scale\fi
               \setlength\@tempdima{\fontdimen5\font}% 5—ex height
               \setlength\@tempdimb{\fontdimen8\font}% 8—XeTeX synthesised
                     uppercase height.
               \divide\@tempdimb␣by1000\relax
               \divide\@tempdima␣by\@tempdimb
               \setlength{\@tempdima}{\@tempdima*\real{\gmu@scalar}}%
               \gm@ifundefined{fakesc@extrascale}{}{%
                 \setlength{\@tempdima}{\@tempdima*\real{%
                      \fakesc@extrascale}}}%
               \@tempcnta=\@tempdima
               \divide\@tempcnta␣by␣1000\relax
               \@tempcntb=-1000\relax
               \multiply\@tempcntb␣by\@tempcnta
               \advance\@tempcntb␣by\@tempdima
               \xdef\gmu@scscale{\the\@tempcnta.%
                 \ifnum\@tempcntb<100␣0\fi
                 \ifnum\@tempcntb<10␣0\fi
                 \the\@tempcntb}%
             \endgroup
             \addfontfeature{Scale=\gmu@scscale}%
           }}{\let\gmu@scalematchX\smallerr}
\fakescextrascale \def\fakescextrascale#1{\def\fakesc@extrascale{#1}}
\fakesc@extrascale
```

See above/see below

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

```
\wyzejnizej \newcommand*\wyzejnizej[1]{%
            \edef\gmu@tempa{\gm@ifundefined{r@#1}{\arabic{page}}{%
```

```
        \@xa\@xa\@xa\@secondoftwo\csname␣r@#1\endcsname}}%
  \ifnum\gmu@tempa<\arabic{page}\relax␣wy\.zej\fi
  \ifnum\gmu@tempa>\arabic{page}\relax␣ni\.zej\fi
  \ifnum\gmu@tempa=\arabic{page}\relax␣\@xa\ignorespaces\fi
}
```

`luzniej` and `napapierki`—environments used in page breaking for money

The name of first of them comes from Polish typesetters' phrase "rozbijać [skład] na papierki"—'to broaden [leading] with paper scratches'.

\napapierkistretch `\def\napapierkistretch{0,3pt}%` It's quite much for 11/13pt leading.

\napapierkicore
```
\def\napapierkicore{\advance\baselineskip%
  by␣optplus\napapierkistretch\relax}
```

```
\DeclareEnvironment{napapierki}{s}{%
  \par\IfValueT{#1}{\global
  }%
  \napapierkicore}
{%
  \par
  \IfValueT{#1}{\global\baselineskip=1\baselineskip\relax
  }%
}% so that you can use \napapierki*›…\endnapapierki* in interlacing environments.
```

\gmu@luzniej `\newcount\gmu@luzniej`

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

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

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

The starred version sets `\looseness` in `\everypar`, which has its advantages and disadvantages.

luzniej*
```
\newenvironment*{luzniej*}[1][1]{%
  \multiply\tolerance␣by␣2\relax
  \everypar{\looseness=#1\relax}}{\par}
```

\nawj `\newcommand*\nawj{\kern0,1em\relax}%` a kern to be put between parentheses and letters with descendants such as *j* or *y* in certain fonts.

The original `\pauza` of polski has the skips rigid (one is even a kern). We make the skips flexible. Moreover, our `\pauza` begins with `\ifhmode` to be usable also at the beginning of a line where it marks a part of a dialogue.

\pauza@skipcore
```
\def\pauza@skipcore{\hskip0.2em␣plus0.1em\relax
  \pauzacore
  \@ifnextchar,{% 2009/11/22 added a special case of a comma following pauza
  }{\hskip.2em␣plus0.1em\relax\ignorespaces}}%
```

\ppauza@skipcore
```
\def\ppauza@skipcore{\unskip\penalty10000\hskip0.2em␣
  plus0.1em\relax
```

```
6051              \ppauza@dash\hskip.2em␣plus0.1em\ignorespaces}

6054 \AtBeginDocument{%
\pauza 6055   \pdef\pauza{%
6056     \ifhmode
6057       \unskip\penalty10000
6058       \hskip0.2em␣plus0.1em\relax
6059       \pauzacore\hskip.2em␣plus0.1em\relax\ignorespaces%
6060     \else
6061       \pauzadial
6062     \fi}%
```

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

```
\pauzadial 6067   \pdef\pauzadial{%
6068     \leavevmode\pauzacore\penalty10000\hskip0,5em%
             \ignorespaces}
```

And a version with no space at the left, to begin a `\noindent`ed paragraph explaining e.g. a quotation:

```
\lpauza 6072   \pdef\lpauza{%
6073     \leavevmode
6074     \pauzacore\hskip.2em␣plus0.1em\ignorespaces}%
```

We define `\ppauza` as an en dash surrounded with thin stretchable spaces and sticking to the upper line or bare but discretionary depending on the next token being space$_{10}$. Of course you'll never get such a space after a literal CS so an explicit `\ppauza` will always result with a bare discretionary en dash, but if we `\let-\ppauza`…

```
\ppauza 6083    \pdef\ppauza{%
6084     \ifvmode␣␣␣␣\PackageError{gmutils}{%
6085       command␣\bslash␣ppauza␣(en␣dash)␣not␣intended␣for␣
               vmode.}{%
6086       Use␣\bslash␣ppauza␣(en␣dash)␣only␣in␣number␣and␣
               numeral␣ranges.}%
6087     \else
6088       \unskip\discretionary
6089       {\ppauza@dash}{\ppauza@dash}{\ppauza@dash}%
6090     \fi}%
6091 }% of at begin document

6093 \ifdefined\XeTeXversion
6095 \AtBeginDocument{% to be independent of moment of loading of polski.
\-  6096   \pdef\-{%
6097     \ifhmode
6098       \unskip\penalty10000
6099       \afterfi{%
6100         \@ifnextspace{\pauza@skipcore}%
6101         {\@ifnextchar,{\pauza@skipcore}% a special case of comma added
                2009/11/22
6102           {\@ifnextMac{\pauza@skipcore}%
6103             {\pauzacore\penalty\hyphenpenalty\hskip%
                   \z@skip}}}%
6104       }% of \afterfi's argument
6105       \else
```

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

```
6109        \leavevmode\pauzacore\penalty10000\hskip0,5em%
                \ignorespaces
6110      \fi}%
```

The next command's name consists of letters and therefore it eats any spaces following it, so `\@ifnextspace` would always be false, therefore we don't use it.

```
      \-  6114    \pdef\-{%
6115      \ifvmode␣␣␣␣\PackageError{gmutils}{%
6116        command␣\bslash␣ppauza␣(en␣dash)␣not␣intended␣for␣
              vmode.}{%
6117        Use␣\bslash␣ppauza␣(en␣dash)␣only␣in␣number␣and␣
              numeral␣ranges.}%
6118      \else
6119        \afterfi{%
6120          \@ifnextspace{\ppauza@skipcore}{%
6121            \@ifnextMac\ppauza@skipcore
6122            {\unskip\discretionary
6123              {\ppauza@dash}{\ppauza@dash}{\ppauza@dash}}}%
6124        }%
6125      \fi
6126    }%
  \emdash  6128    \def\emdash{\char`\-}
6129  }% of at begin document

\longpauza  6131  \def\longpauza{\def\pauzacore{-}}
\pauzacore  6132  \longpauza
\shortpauza  6133  \def\shortpauza{%
\pauzacore  6134    \def\pauzacore{\hbox{-\kern,23em\relax\llap{-}}}}%
\ppauza@dash  6135  \def\ppauza@dash{-}%

6138  \else␣% not XeTeX
\longpauza  6139  \def\longpauza{\def\pauzacore{---}}
\pauzacore  6140  \longpauza
\shortpauza  6141  \def\shortpauza{%
\pauzacore  6142    \def\pauzacore{--\kern,23em\relax\llap{--}}}%
\ppauza@dash  6143  \def\ppauza@dash{--}%

6146  \fi% of if XeTeX.
```

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

```
6154  \ifdefined\XeTeXversion
6155  \foone{\catcode`-\active␣\catcode`-\active␣\catcode`-\active}{%
          %
  \adashes  6156    \def\adashes{\AtBeginDocument\adashes}% because \pauza is defined
                      at begin document.
  \adashes  6158    \AtBeginDocument{\def\adashes{%
        \-  6159        \catcode`-\active␣\def-{\-}%
        \-  6160        \catcode`-\active␣\def-{\-}%
6162        \addtomacro\dospecials{\do\-\do\-}%
6163        \addtomacro\@sanitize{\@makeother\-\@makeother\-}%
6164        \addtomacro\gmu@septify{\do\-13\do\-13\relax}%
```

```
}}}
\else
\relaxen\adashes
\fi
```

The hyphen shouldn't be active IMHO because it's used in TEX control such as `\hskip-2pt`. Therefore we provide the `\ahyphen` declaration reluctantly, because sometimes we need it and always use it with caution. Note that my active hyphen in vertical and math modes expands to $-_{12}$.

`\gmu@dywiz`
```
\def\gmu@dywiz{\ifmmode-\else
  \ifvmode-\else\afterfifi\dywiz\fi\fi}%
```

```
\foone{\catcode`-\active}{% aktivnyj diefis aktywny dywiz active hyphen
```
`\ahyphen`
```
    \def\ahyphen{\let-\gmu@dywiz\catcode`\-\active}}
```

To get current time. Works in ε-TEXs, including XETEX. `\czas` typesets 8.17 and `\czas[:]` typesets 8:17.

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

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

`tytulowa`
```
\newenvironment*{tytulowa}{\newpage}{\par\thispagestyle{%
    empty}\newpage}
```

To typeset peoples' names on page 4 (the editorial page):

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

Typesetting dates in my memoirs

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

`\polskadata`
`\gmu@datefsl`
```
\pdef\polskadata{%
  \DeclareCommand\gmu@datefsl{%
    Q{0123456789\bgroup}>iT{/-}␣%  (1) year
    Q{0123456789\bgroup}>iT{/-}␣%  (2) month
    Q{0123456789\bgroup}␣%  (3) day
    T{,}␣K{##1\gmu@datefsl}␣%  (4, 5) additional stuff after comma
  }{%
      \IfValueF{##2}{\PutIfValue{##3}}%
      \IfValueT{##2}{%
```

```
          \@tempcnta=0##3\relax\the\@tempcnta
          \ifcase##2\relax\or␣stycznia\or␣lutego%
          \or␣marca\or␣kwietnia\or␣maja\or␣czerwca\or␣
              lipca\or␣sierpnia%
          \or␣września\or␣października\or␣listopada\or␣
              grudnia\else
          {}%
          \fi}%
          \IfValueT{##1}{\space␣##1}%
          \PutIfValue{##4}\IfValueT{##5}{␣##5}%
        }% of \gmu@datefsl.
  }% of \polskadata

\polskadata
```

For documentation in English:

\englishdate
\gmu@datefsl

```
\pdef\englishdate{%
  \DeclareCommand\gmu@datefsl{%
    Q{0123456789\bgroup}>iT{/-}␣% (1) year
    Q{0123456789\bgroup}>iT{/-}␣% (2) month
    Q{0123456789\bgroup}␣% (3) day
    T{,}␣␣K{##1\gmu@datefsl}␣% (4, 5) additional stuff after comma
  }{%
    \IfValueF{##2}{\PutIfValue{##3}}%
    \IfValueT{##2}{%
      \ifcase##2\relax\or␣January\or␣February%
        \or␣March\or␣April\or␣May\or␣June\or␣July\or␣
            August%
        \or␣September\or␣October\or␣November\or␣December%
            \else
        {}%
      \fi}%
    \space
    \@tempcnta=##3\relax\the\@tempcnta,
    \IfValueT{##1}{␣##1}%
    \PutIfValue{##4}\IfValueT{##5}{␣##5}%
  }% of \gmu@datefsl.
}%
```

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

\ifdate
\bidate

```
\newif\ifdate
\pdef\bidate#1{%
  \gmu@datefsl#1\gmu@datefsl
}
```

\linedate
\linedate@@
\linedate@

```
\pdef\linedate{\gm@ifstar\linedate@@\linedate@}
\pdef\linedate@@#1{\linedate@{--}{}#1}}
\pdef\linedate@#1{\par
  \linedate@hook{#1}%
  \ifdate\addvspace{\dateskipamount}%
  \possvfil% if we put it before \addvspace, the v-space is always added.
  \date@line{\footnotesize\itshape␣\bidate{#1}}%
  \nopagebreak
  \else% %\ifnum\arabic{dateinsection}>0\dekbigskip\fi
```

```
      \addvspace{\bigskipamount}\possvfil
    \fi}% end of \linedate.

\let\linedate@hook\@gobble

\let\dateskipamount\medskipamount

\rdate  \pdef\rdate{\let\date@line\rightline␣\linedate}

\date@left  \def\date@left#1{\par{%
      \raggedright#1%
      \leftskip\z@skip
      \@@par}}%

\ldate  \pdef\ldate{%
    \let\date@line\date@left
    \linedate}

\runindate  \newcommand*\runindate[1]{%
    \paragraph{\footnotesize\itshape␣\gmu@datef#1\gmu@datef}%
    \stepcounter{dateinsection}}
```

I'm not quite positive which side I want the date to be put to so let's `let` for now and we'll be able to change it in the very documents.

```
\let\thedate\ldate

\zwrobcy  \pdef\zwrobcy#1{\emph{#1}␣%
```

ostinato, allegro con moto, garden party etc., także kompliment

```
\tytul  \pdef\tytul#1{\emph{#1}}
```

Maszynopis w świecie justowanym zrobi delikatną chorągiewkę. (The `maszynopis` environment will make a delicate ragged right if called in a justified world.)

```
\maszynopis  \newenvironment{maszynopis}[1][]{#1\ttfamily
    \hyphenchar\font=45\relax% this assignment is global for the font.
    \@tempskipa=\glueexpr\rightskip+\leftskip\relax
    \ifdim\gluestretch\@tempskipa=\z@
    \tolerance900
```

it worked well with tolerance = 900.

```
    \advance\rightskip␣by\z@␣plus0,5em\relax\fi
    \fontdimen3\font=\z@% we forbid stretching spaces…
 %␣␣\fontdimen4\font=\z@ but allow shrinking them.
    \hyphenpenalty0␣% not to make TEX nervous: in a typewriting this marvel-
```

lous algorithm of hyphenation should be turned off and every line broken at the last allowable point.

```
    \StoreMacro\pauzacore
\pauzacore  \def\pauzacore{-\rlap{\kern-0,3em-}-}%
}{\par}

\justified  \pdef\justified{%
    \leftskip=1\leftskip% to preserve the natural length and discard stretch
```

and shrink.

```
    \rightskip=1\rightskip
    \parfillskip=1\parfillskip
    \advance\parfillskip␣by␣0sp␣plus␣1fil\relax
    \let\\\@normalcr}
```

To conform Polish recommendation for typesetting saying that a paragraph's last line leaving less than \parindent should be stretched to fill the text width:

\fullpar
```
\DeclareCommand\fullpar{%
  T{+-}%
  Q{+-0123456789}␣% optional looseness (most probably negative)
}{%
  \begingroup
  \IfValueT{#1}{\looseness=#1\IfValueTF{#2}{#2}{1}\relax
    \multiply\tolerance␣by␣\tw@
  }%
  \fullparcore
  \par
  \endgroup}
```

\fullparcore
```
\pdef\fullparcore{%
  \hunskip
  \parfillskip\z@skip}
```

To conform Polish recommendation for typesetting that says that the last line of a paragraph has to be 2\parindent long at least. The idea is to set \parfillskip naturally rigid and long as \textwidth-2\parindent, but that causes non-negligible shrinking of the inter-word spaces so we provide a declaration to catch the proper glue where the parindent is set (e.g. in footnotes parindent is 0 pt)

\twoparinit
```
\newcommand*\twoparinit{% the name stands for 'last paragraph line's length
    minimum two \parindent.
```

\twopar@defts
```
  \def\twopar@defts{%
    \hsize-\leftskip-\rightskip-\fontcharwd\font`…}%
```

\twopar@atleast
```
  \def\twopar@atleast{2\@parindent}%
```

\twopar
```
  \DeclareCommand\twopar{%
    T{+-}% (1) you can specify loosening the paragraph by one only by typing
          single + and tightening by one by typing single -.
    Q{+-0123456789}% (2)
    A{\twopar@atleast}% (3)
    >iT{\cipolagwa}}{%
    \begingroup
    \IfValueT{##1}{%
      \looseness=##1\IfValueTF{##2}{##2}{1}\relax
      \multiply\tolerance␣by2
    }%
    \twoparcore<##3>%
    \par
    \endgroup
  }% of \twopar.
```

```
  \ifdefined\XeTeXversion
```

\twoparcore
```
    \DeclareCommand\twoparcore{%
      A{\twopar@atleast}>iT{\cipolągwa}}{%
      \hunskip␣% it's O.K. it's in a group, it'll work anyway.
      \edef\gmu@tempa{\the\dimexpr\twopar@defts-##1\relax}%
      \parfillskip=\glueexpr\gmu@tempa␣minus␣\gmu@tempa
      \relax% to delimit \glueexpr.
      \relax% to delimit the assignment.
    }%
  \else␣% not XeTeX—doesn't use \fontcharwd.
```

```
\twoparcore
      \DeclareCommand\twoparcore{%
        A{\twopar@default}>iT{\cipolagwa}}{%
        \hunskip␣% it's O.K. it's in a group, it'll work anyway.
        {\setboxo=\hbox{\dots}%
          \xdef\gmu@tempa{\the\wdo}}%
        \edef\gmu@tempa{%
          \the\dimexpr\hsize-\leftskip-\rightskip
          -\gmu@tempa-2\@parindent\relax}%
        \parfillskip=\glueexpr\gmu@tempa␣minus␣\gmu@tempa
        \relax% to delimit \glueexpr.
        \relax% to delimit the assignment.
      }%
    \fi
    \AtBeginDocument{%
      \unless\ifdefined\@parindent
\@parindent
        \newskip\@parindent
        \@parindent=\parindent
      \fi
\restoreparindent
      \def\restoreparindent{\parindent\@parindent}%
    }% of \AtBeginDocument.
}% of \twoparinit.
```

For dati under poems

Or explanations under results of time.

```
\wherncore
\DeclareCommand\wherncore{om}{%
      % [#1] o ptional value of \hskip of (left) indent of the parbox. If absent,
        parbox is aligned right;
      % [#2] o text for the datum parbox.
    \IfValueTF{#1}{\leftline{%
        \whernfont
        \hskip#1\relax\parbox
        {\dimexpr\textwidth-\leftskip-\rightskip-#1}%
        {#2}% of \parbox,
      }% of \leftline,
    }% of ValueT{#1}.
    {% ValueF{#1}:
      \rightline
      {\whernfont
        \whern@parbox{#2}%
      }% of \rightline,
      \setprevdepth
    }% of ValueF{#1},
}% of \wherncore.
\whern@parbox
\DeclareCommand\whern@parbox{%
    S{\leftskip\rightskip}% horizontal alignment of resulting box (the side to
        be ragged)
    O{t}␣% vertical alignment of parbox
    >is␣% separator
    O{0,7666\textwidth}␣% (3) width of parbox
    m␣% (4) parbox contents
}{%
```

```
% #1 S the skip of the ragged side,
% #2 S the \parbox's contents.
  \parbox[#2]{#3}{%
    \IfValueTF{#1}{#1}{\leftskip}=0sp␣plus␣\textwidth
    \parfillskip0sp\relax
    \let\\\linebreak
    \disobeylines
    \whernfont␣#4\unskip\strut\endgraf
    \getprevdepth
  }% of \parbox,
}% of \whern@parbox.
\whern \def\whern{%
  \endgraf\nopagebreak
  \gm@ifstar{\wherncore}%
  {\vskip\whernskip\wherncore}}
\let\whernfont\footnotesize
\whernskip \newskip\whernskip
\whernskip2\baselineskip␣minus␣2\baselineskip\relax
\whernup \DeclareCommand\whernup{%
  o␣% a vskip before
  >is␣% separating star (ignored)
  o␣␣% (2) custom width of parbox
  >Pm}{\par
  \IfValueT{#1}{\vskip#1\relax}%
  \leftline{%
    \IfValueTF{#2}{\whern@parbox\rightskip[b][#2]}%
    {\whern@parbox\rightskip[b]}%
    {#3}%
  }%
  \setprevdepth
  \nopagebreak\relax
  \@ifenvir{quote}{\noindent\ignorespaces}{}}
```

Thousand separator

```
\thousep \pdef\thousep#1{% a macro that'll put the thousand separator between every
        two three-digit groups.
```

First we check whether we have at least five digits.

```
  \gmu@thou@fiver#1\relax\relax\relax\relax\relax% we
      put five \relaxes after the parameter to ensure the string will
      meet \gmu@thou@fiver's definition.
  \gmu@thou@fiver{#1}{% if more than five digits:
    \emptify\gmu@thou@put
    \relaxen\gmu@thou@o\relaxen\gmu@thou@i\relaxen%
        \gmu@thou@ii
    \@tempcnta\z@
    \gmu@thou@putter#1\gmu@thou@putter
    \gmu@thou@put
  }}
\gmu@thou@fiver \def\gmu@thou@fiver#1#2#3#4#5\gmu@thou@fiver#6#7{% this macro only
```

checks if the text delimited with itself consists of at least five tokens/braces

```
\ifx\relax#5\relax\@xa\@firstoftwo
\else\@xa\@secondoftwo
\fi{#6}{#7}}
```

\gmu@thou@putter

```
\def\gmu@thou@putter#1#2{%
```

we are sure to have at least five tokens before the sentinel `\gmu@thou@putter`.

```
  \advance\@tempcnta\@ne
  \@tempcntb\@tempcnta
  \divide\@tempcntb3\relax
  \@tempcnta=\numexpr\@tempcnta-\@tempcntb*3
  \edef\gmu@thou@put{\@xau{\gmu@thou@put}\unexpanded{#1}%
    \ifx\gmu@thou@putter#2\else
      \ifcase\@tempcnta
        \gmu@thou@o\or\gmu@thou@i\or\gmu@thou@ii%
```

all three CSes are yet `\relax` so we may put them in an `\edef` safely.

```
      \fi
    \fi}% of \edef
  \ifx\gmu@thou@putter#2% if we are at end of the digits…
    \edef\gmu@tempa{%
      \ifcase\@tempcnta
        \gmu@thou@o\or\gmu@thou@i\or\gmu@thou@ii
      \fi}%
    \@xa\let\gmu@tempa\gmu@thousep% … we set the proper CS…
  \else% or …
    \afterfi{% iterate.
      \gmu@thou@putter#2}% of \afterfi
  \fi% of if end of digits.
}% of \gmu@thou@putter.
```

\gmu@thousep

```
\def\gmu@thousep{\,}%
```

in Polish the recommended thousand separator is a thin space.

So you can type `\thousep{7123123123123}` to get 7 123 123 123 123. But what if you want to apply `\thousep` to a count register or a `\numexpr`? You should write one or two `\expandafter`s and `\the`. Let's do it only once for all:

\xathousep

```
\pdef\xathousep#1{\@xa\thousep\@xa{\the#1}}
```

Now write `\xathousep{\numexpr␣10*9*8*7*6*120}` to get 3 628 800.

\shortthousep \thous

```
\def\shortthousep{%
  \DeclareCommand\thous{Q{+-0123456789}}{%
```

we declare it as a command with Q-type argument to allow spaces between digits.

```
    \ifmmode\hbox\bgroup\@gmu@mmhboxtrue\fi
    \IfValueTF{##1}{% we are given a sequence of digits
      \@tempcnta=##1\relax
      \ifnum\@tempcnta<0␣$-$%
        \@tempcnta=-\@tempcnta
      \fi
      \xathousep\@tempcnta
      \if@gmu@mmhbox\egroup
      \else\@xa\spifletter
      \fi
    }%
```

```
      {% no bare digits given, then we assume the argument is braced.
        \thousep
      }%
    }% of \thous.
}% of \shortthousep.
```

And now write `\thous␣3628800` to get 3 628 800 even with a blank space (beware of the range of TEX's counts).

hyperref's \nolinkurl into \url*

\urladdstar
\url

```
\def\urladdstar{%
  \AtBeginDocument{%
    \@ifpackageloaded{hyperref}{%
      \StoreMacro\url
      \pdef\url{\gm@ifstar{\nolinkurl}{\storedcsname{url}}}%
    }{}}}
\@onlypreamble\urladdstar
```

Footnotes suggested by Andrzej Tomaszewski

\ATfootnotes

```
\DeclareCommand\ATfootnotes{s}{%
```

We make the footnote mark in the footnote `\scriptsize` not `\scriptscriptsize`.

```
  \IfValueT{#1}% the following setting is suitable for old style numbers in footnote marks, therefore I place it in the starred version of the command.
  {\prependtomacro\gmu@ATfootnotes{%
      \pdef\@makefnmark{%
        \mbox␣{\normalfont␣\textsuperscript␣{\smaller[3]%
            \@thefnmark␣}}}%
    }% of prepend,
  }% of \IfValueT.

  \gmu@ATfootnotes
  \gmu@AT@ampulex\maketitle% without hyperref
  \ifdefined\HyOrg@maketitle
    \afterfi{\gmu@AT@ampulex\HyOrg@maketitle}% with hyperref
  \fi
}
```

\gmu@AT@ampulex

```
\pdef\gmu@AT@ampulex#1{%
  \ampulexdef#1{\def\@makefnmark}%
  \if@twocolumn
  {\gmu@ATfootnotes\if@twocolumn}% Ampulex redefinition of \maketitle for the standard classes.
```

\@makefntext

```
  \ampulexdef#1{\long\def\@makefntext}%
  \if@twocolumn{\gmu@ATfootnotes\if@twocolumn}% Ampulex redefinition of \maketitle for mwcls.
}
```

\gmu@ATfootnotes

```
\pdef\gmu@ATfootnotes{%
```

And we make the footnote number not be in superscript but on the base line, according to Andrzej Tomaszewski's suggestion on BachoTEX 2008, and the same size as in the footnote mark.

\@makefntext

```
  \long\pdef\@makefntext##1{%
```

```
      \ifdefined\@parindent␣\parindent\@parindent
      \else␣\parindent␣1em\relax
      \fi
      \indent{\ATf@font\scriptsize%
        {\@thefnmark}}%
      \gmu@fnhook
      \enspace\ignorespaces##1}%
}

\let\ATf@font\normalfont

\emptify\gmu@fnhook
```

\rrthis
```
\pdef\rrthis{% 'rag right this': make only the current paragraph ragged right
    (e.g. if the paragraph consists of a long URL).
  \begingroup\rightskip=0sp␣plus␣\hsize␣\endgraf\endgroup}
```

\centerthis
```
\pdef\centerthis{% 2009/12/15
  \begingroup
  \rightskip=1\rightskip␣plus␣\hsize
  \leftskip=1\leftskip␣plus\hsize
  \parfillskip=\z@skip
  \endgraf\endgroup}
```

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

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

A fix to the url package

It happened that a URLs typeset with the `\url` command of the url package came out sort of spaced because kerning was off because of the math mode. So I provide a redefinition of the internal macros of the url package which in my version uses not math mode but `\scantokens` and not `\relpenalty` and `\binoppenalty` but `\hyphenpenalty` (as it is in the paragraph) and `\discretionary`. I tried putting explicit penalties after the symbols but that spoiled kerning.

The rules of line breaking are somewhat different, too: in the original url package line breaks are forbidden between any two symbols listed in `\UrlBigBreaks`. In my version line breaks are forbidden between any two *identical* 'URL Breaks' and 'URL Big Breaks'.

There are some more differences in formatting some chars, i.a. `~`, `%` and angle brackets which I don't treat specially and just take from font assuming the font provides ASCII chars and checking whether it provides the angle brackets.

```
\@ifXeTeX{%
```

\UrlFix
```
  \pdef\UrlFix{\AtBeginDocument{%
      \@ifpackageloaded{url}{\gm@UrlFix}{}}%
    \relaxen\UrlFix}%
  \AtBeginDocument{%
```

\UrlFix
```
    \pdef\UrlFix{%
      \@ifpackageloaded{url}{\gm@UrlFix}{}%
      \relaxen\UrlFix}}%
}
{%
```

\UrlFix
```
  \pdef\UrlFix{\PackageWarning{gmutils}{!!!␣The␣\string%
```

```
      \UrlFix\space
      declaration␣works␣only␣with␣XeTeX}}%
}

\@ifXeTeX{}{%
  \edef\gmu@restoreUpUpUp{\catcode`\@nx\^^^=\the\catcode`%
      \^^^}%
  \AtEndOfPackage\gmu@restoreUpUpUp
  \catcode`\^^^=9␣}
```

`\gm@UrlFix`

```
\def\gm@UrlFix{%
```

default style assignments

`\UrlBreaks`

```
  \def\UrlBreaks{\do\.\do\@\do\\\do\/\do\!\do\_\do\|\do\;%
      \do\]%
    \do\)\do\,\do\?\do\'\do\"\do\+\do\=\do\#\do\%\do\~\do%
        \_\do\|%
    \do\{\do\}\do\$}%
```

`\UrlBigBreaks`

```
  \def\UrlBigBreaks{\do\:}%
```

`\UrlNoBreaks`

```
  \def\UrlNoBreaks{\do\(\do\[\do\{}%
```

`\UrlSpecials`

```
  \def\UrlSpecials{%
    \do\␣{\hbox{\visiblespace}}\do\^^M{\hbox{%
        \visiblespace}}}%
```

`\Url@Format`

```
  \def\Url@Format##1{%
    \UrlFont
    \ifdefined\verbatim@specials
      \catcode`\>\active
      \verbatim@specials
      \verbatim@mathhack
    \fi␣% setting of the escape char, begin and end group and optionally math
        shift, defined in gmverb.
    \gm@UrlSetup
    \UrlLeft
    \edef\gmu@theendlinechar{\the\endlinechar}%
    \endlinechar\m@ne
    \kern\z@% to forbid hyphenating the first word if the URL begins with
        a word
    \hyphenchar\font=\UrlHyphenchar\relax
    \let\-\gmu@discretionaryhyphen
    \scantokens{##1}%
    \endlinechar\gmu@theendlinechar\relax
    \UrlRight
  }% of \Url@Format.
```

```
  \edef\UrlHyphenchar{%
    \ifdefined\gmv@hyphenchar\gmv@hyphenchar
    \else"A6␣\fi}% broken bar, ¦ or the same as provided in gmverb for verba-
        tims. You can redefine it as you please. This char is used as the hyphen-
        ation char in URLs and therefore should be different from - (hyphen),
        which is often a part of an URL. The broken bar seems to be quite unlikely
        in URLs and/or file names.
```

`\verbatim@mathhack`

```
  \def\verbatim@mathhack{%
    \ifdefined\verbatim@specials@list
```

```
      \@xa\verbatim@mathhack@\verbatim@specials@list
    \fi
  }%
\verbatim@mathhack@
  \def\verbatim@mathhack@##1##2##3##4##5##6{%
    \IfValueT{##4}{%
      \edef\gmu@thinmuskip{\the\thinmuskip}%
      \edef\gmu@medmuskip{\the\medmuskip}%
      \edef\gmu@thickmuskip{\the\thickmuskip}%
      \begingroup
      \lccode`\~=`##4\lowercase{%
        \endgroup\def~####1~}%
\thinmuskip
      {$\thinmuskip\gmu@thinmuskip\relax
        \medmuskip\gmu@medmuskip\relax
        \thickmuskip\gmu@thickmuskip\relax
        ####1%
        $}%
      \catcode`##4\active
    }%
  }%
\gm@UrlSetup
  \def\gm@UrlSetup{%
    \medmuskip\Urlmuskip␣\thickmuskip\medmuskip␣%
        \thinmuskipomu%
    \relpenalty\UrlBigBreakPenalty␣\binoppenalty%
        \UrlBreakPenalty
    \def\do{\gmu@doUrlMath\UrlBreakPenalty}\UrlBreaks␣% bin ( \hy|
        phenpenalty anyway)
    \def\do{\gmu@doUrlMath\UrlBigBreakPenalty}\UrlBigBreaks␣% rel
        ( \hyphenpenalty anyway)
    \def\do{\gmu@doUrlMath\@M}\UrlNoBreaks␣% open (no break)
    \def\do{\gmu@doUrlMathAc\UrlBreakPenalty}% ( \hyphenpenalty)
    \UrlSpecials
    \if␣\iffontchar\font"2329␣1\else0\fi\iffontchar%
        \font"232A␣1\else2\fi
```

we check whether the font provides both left and right angle brackets.

```
      \gmu@measurewd{^^^^2329}%
      \edef\gmu@tempa{%
        \@nx\gmu@doUrlMathAc\@M\@nx\<{%
          \hbox␣to\gmu@tempb{\unexpanded{\hss\char"2329␣%
              \hss}}}%
        }\gmu@tempa
      \gmu@measurewd{^^^^232a}%
      \edef\gmu@tempa{%
        \@nx\do\@nx\>{%
          \hbox␣to\gmu@tempb{\unexpanded{\hss\char"232A␣%
              \hss}}}%
        }\gmu@tempa
    \else
      \gmu@doUrlMathAc\@M\<{\langle}\do\>{\rangle}%
    \fi
    \iffontchar\font"22C6␣% low star
      \do\*{\hbox{\char"22C6␣}}%
    \else␣\do\**%
```

```
    \fi
    \ifx\do@url@hyp\@empty
      \gmu@measurewd{-}% this macro is defined in line 5197.
      \edef\gmu@tempa{%
        \unexpanded{\gmu@doUrlMathAc\@M\-}%
        {\hbox to \gmu@tempb{\unexpanded{\hss-\hss}}%
          \@nx\-}% hyphen is a good point for hyphenation, but the hyphenation char should be sth. else, and it is indeed: ¦ (broken bar, \char"A6). See also line 6765
      }\gmu@tempa
    \fi
    \addfontfeature{Ligatures=NoCommon, Mapping=none}% instead of 'doing' \verbatim@nolig@list.
  }% of \gm@UrlSetup.

\gmu@doUrlMath
  \def\gmu@doUrlMath##1##2{%
      % #1 value of the penalty (used as a Boolean: if < 10 000,
        % \hyphenpenalty will be used anyway, if ≥ 10 000, there will be no
        % \discretionary),
      % #2 the char, given as \⟨char⟩.
    \begingroup
    \lccode`\~=`##2\lowercase{%
      \endgroup\def~{\@ifnextchar~}%
      \@xa\addtomacro\@xa~}% of \lowercase.
    \ifnum##1<\@M
    {%
      {\char`##2\csname gmu@dbl\string##2kern\endcsname}% if next is the same char
      {\ifmmode\char`##2% else
        \else\gmu@urlbreakable{##1}{##2}%
        \fi}%
    }% of \addtomacro's argument \ifnum true.
    \else
    {%
      {\char`##2\csname gmu@dbl\string##2kern\endcsname}{%
          \char`##2}%
    }% of \addtomacro's argument \ifnum false.
    \fi
      \catcode`##2=\active
    }% of \gmu@doUrlMath.

\gmu@doUrlMathAc
  \def\gmu@doUrlMathAc##1##2##3{%
      % #1 (value of) a penalty (see the remark to ##1 of the previous macro),
      % #2 the char (as \⟨char⟩),
      % #3 the definition.
    \begingroup
    \lccode`\~=`##2\lowercase{%
      \endgroup\def~{\@ifnextchar~}%
      \@xa\addtomacro\@xa~}% of \lowercase.
    \ifnum ##1<\@M
    {%
      {\ifmmode\char`##2\else$##3\m@th$\fi}%
      {\ifmmode\char`##2%
```

```
          \else\discretionary{\hbox{$##3\m@th$}}{}{\hbox{$##3%
               \m@th$}}%
          \fi}%
      }% of \addtomacro's argument if num true.
      \else
      {%
        {\ifmmode\char`##2\else$##3\m@th$\fi}{\ifmmode%
             \char`##2\else$##3\m@th$\fi}%
      }% of \addtomacro's argument if num false.
      \fi
      \catcode`##2=\active
    }% of \gmu@doUrlMathAc.
\gmu@url@rigidbreak  \pdef\gmu@url@rigidbreak##1##2{\discretionary{\char`##2}{%
         }{\char`##2}}%
\gmu@url@flexbreak   \pdef\gmu@url@flexbreak##1##2{\penalty\@M␣\hskip\z@␣
         pluso,o3em
      \char`##2\penalty##1\hskip\z@␣pluso,o3em\relax}%
    \let\gmu@urlbreakable\gmu@url@flexbreak
\Url@z              \def\Url@z##1{%
```

Do any hyper referencing due to hyperref (or perform a url-def)

```
      \Url@HyperHook
```

Now do the formatting in a group (can also have `\Url@HyperHook` take this as an argument).

```
      {\Url@Format{##1}}%
      \endgroup}%
    \DeclareUrlCommand\file{\urlstyle{sf}}%
    \emptify\Url@moving%
```

with our settings `\url` is pretty allowed in moving arguments, I hope.

```
}% of \gmu@UrlFix.
\UrlSlashKern  \DeclareCommand\UrlSlashKern{O{tt}m}%
{\AtBeginDocument{%
      \@nameedef{url@#1style}{\def\@nx\UrlFont{%
           \@xa\@nx\csname#1family\endcsname
           \def\@xa\@nx\csname␣gmu@dbl\string\/kern\endcsname
           {\kern#2\relax}%
        }% of \UrlFont
      }% of \url#1style
      \urlstyle{#1}%
    }% of \AtBeginDocument
}% of \UrlSlashKern
```

Conditional tilde

Polish typesetting standards say that for 12 dd and 10 dd ⟨*czcionki*⟩ if leading is narrower that 3½ ⟨*kwadratu*⟩, 3½ × 48dd, then hanging letters are allowed, which also applies to 8 and 6 dd ⟨*czcionki*⟩ in less than 3 ⟨*kwadraty*⟩ leading. I treat this recommendation not strictly but as an inspiration, that is I translate »dd« to »pt«.

```
\TrzaskaTilde 6952 \def\TrzaskaTilde{%
6953   \@xa\DeclareCommand\@xa\gm@smarttilde
6954   \@xa{\@xa␣S\@xa{\all@stars~}}{%
6955     \IfValueTF{##1}{\nobreakspace{}}%
6956     {\ifdim\dimexpr\hsize-\leftskip-\rightskip
6957       -\ifdim\hangindent<\z@-\fi\hangindent␣% the last parameter is
                used with respect to the floatflt package.
6959       >%
6960         \ifdim\f@size␣pt>\dimexpr10opt-1sp\relax
6961           168dd
6962         \else
6963           144dd
6964         \fi
6965         \nobreakspace␣{}%
6966       \else
6967         \␣%
6968       \fi
6969     }% of \gm@ifstar's else,
6970   }% of \gm@smarttilde,
6971   \let~\gm@smarttilde
6972 }% of \TrzaskaTilde.

\getprevdepth 6975 \pdef\getprevdepth{%
6976   \endgraf
6977   \xdef\setprevdepth{\prevdepth=\the\prevdepth\relax}%
6978 }
```

Storing the catcode of line end

```
\StoreCatM 6981 \def\StoreCatM{%
6982   \protected\edef\RestoreCatM{%
6983     \catcode`\@nx\^^M=\the\catcode`\^^M\relax}%
6984 }

\RestoreCatM 6986 \pdef\RestoreCatM{\PackageE{gmutils}{first␣store␣the␣catcode␣
        of
6987     ^\empty^\empty␣M␣with␣\string\StoreCatM.}%
6988 }
```

A really empty page

Copied from Marcin Woliński's macros.

```
\clearemptydoublepage 6994 \newcommand{\clearemptydoublepage}{%
6995           \newpage{\pagestyle{empty}\cleardoublepage}}

\setspaceskip 6998 \DeclareCommand\setspaceskip{%
6999   A{}% optional factor for all three components
7000   O{\fontdimen2\font}%
7001  >is
7002  O{\fontdimen3\font}%
7003  >is
7004  O{\fontdimen4\font}}
```

```
{\spaceskip=#1#2plus#1#3minus#1#4\relax}

\foone\obeylines{%
\disobeylines   \def\disobeylines{%  for arguments in which line end is active to simulate normal behaviour
      \ifnum\catcode`\^^M=\active%
\@ifnextgroup       \pdef^^M{\@ifnextgroup{\ifhmode\unskip\space\fi}{%
                        \gmu@disMinner}}%
\gmu@disMinner      \def\gmu@disMinner##1{%
                      \ifx^^M##1\endgraf%
                      \else\afterfi{\ifhmode\unskip\space\fi}\fi##1}%
      \fi}%
}

\makestarlow    \def\makestarlow{%
      \begingroup\lccode`\~=`\*\lowercase{%
*       \endgroup\def~{\gmu@lowstar}}%  2009/10/19 \let changed to \def
\gmu@lowstar          to allow redefinitions of \gmu@lowstar.
      \catcode`\*=\active
      \defLowStarFake
}

\defLowStarFake \DeclareCommand\defLowStarFake{%
      Q{+-0123456789,.}{0,5}%  fraction of fontchar depth of the star glyph
}%
{%
\gmu@lowstarfake    \def\gmu@lowstarfake{%
        \leavevmode\vbox{\hbox{*}\kern#1\fontchardp\font`*}%
      }%
}

\gmu@lowstarfake    \def\gmu@lowstarfake{*}␣%  useful for next command where normal star is low.
```

The * CS and active * should be defined different to make them distinguishable by tests, especially with \gm@ifstar in mind.

```
\*  \DeclareCommand\*{Q{0123456789}{1}}
    {\gmu@star@loop0{#1}}

\gmu@star@loop  \def\gmu@star@loop#1#2{%  this is an expandable loop as in The ε-TEX Manual p. 9.
      \ifnum#1<\numexpr#2\relax%
        \gmu@lowstar
        \@xa\gmu@star@loop
        \@xa{\number\numexpr#1+1\@xa}%
        \@xa{\number#2\@xa}%
      \fi}

\endinput
```

Change History

v0.74 \@begnamedgroup@ifcs:

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`, 2683

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, 7063

v0.75

`\@ifnextcat`:

`\let` for `#1` changed to `\def` to allow things like `\noexpand ~`, 794

`\@ifnextif`:

`\let` for `#1` changed to `\def` to allow things like `\noexpand ~`, 836

`\@ifnif`:

added, 876

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, 7063

`\freeze@actives`:

added, 5536

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, 7063

v0.78

General:

`\@namelet` renamed to `\n@melet` to solve a conflict with the beamer class. The package contents regrouped, 7063

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`, 3312

v0.80

General:

CheckSum 1689 , 0

`\hfillneg`:

added, 5433

v0.81

`\dekfraccslash`:

moved here from pmlectionis.cls, 5728

`\ifSecondClass`:

moved here from pmlectionis.cls, 5697

v0.82

`\ikern`:

added, 5736

v0.83

`\texttilde`:

postponed to `\begin{document}` to avoid overwriting by a text command and made sensible to a subsequent `/`, 5389

v0.84

General:

CheckSum 2684 , 0

v0.85

General:

CheckSum 2795 , 0

fixed behaviour of too clever headings with gmdoc by adding an `\ifdim` test, 7063

v0.86

`\texttilde`:

renamed from `\~` since the latter is one of LaTeX's accents, 5389

v0.87

General:

CheckSum 4027 , 0

the package goes ε-TeX even more, making use of `\ifdefined` and the code using UTF-8 chars is wrapped in a XeTeX-condition, 7063

v0.88

General:

CheckSum 4040 , 0

`\RestoreEnvironment`:

added, 1181

`\storedcsname`:

added, 1172

`\StoreEnvironment`:

added, 1177

v0.89

General:

removed obsolete adjustment of pgf for XeTeX, 7063

v0.90

General:

CheckSum 4035 , 0

`\XeTeXthree`:

adjusted to the redefinition of `\verb` in xlxtra 2008/07/29, 4158

v0.91

General:

CheckSum 4055 , 0

removed \jobnamewoe since \jobname is always without extension. \xiispace forked to \visiblespace \let to \xxt@visiblespace of xltxtra if available. The documentation driver integrated with the .sty file, 7063

v0.92

\@checkend:
shortened thanks to \@ifenvir, 2771

\@gif:
added redefinition so that now switches defined with it are \protected so they won't expand to a further expanding or unbalanced \iftrue/false in an \edef, 489

\@ifenvir:
added, 2707

\@ifprevenvir:
added, 2748

General:
CheckSum 4133 , 0

v0.93

\@nameedef:
added, 282

General:
A couple of \DeclareRobustCommand* changed to \pdef, 7063
CheckSum 4140 , 0
CheckSum 4501 , 0
The numerical macros commented out as obsolete and never really used, 7063

\ampulexdef:
added, 2523

\em:
added, 5110, 5119

\gmu@RPfor:
renamed from \gmu@RPif and #3 changed from a csname to CS, 5600

\litshape:
copied here from E. Szarzyński's *The Letters*, 5068

\lsl:
copied here from E. Szarzyński's *The Letters*, 5093

\nocite:
a bug fixed: with natbib an 'extra }' error. Now it fixes only the standard version of \nocite, 3349

\pdef:
added, 240

\pprovide:
added, 269

\provide:
added, 254

\textlit:
added, 5085

\thousep:
added, 6513

v0.94

\@xau:
added, 225

General:
\bgroup and \egroup in the macro storing commands and in \foone changed to \begingroup and \endgroup since the former produce an empty \mathord in math mode while the latter don't, 7063
CheckSum 4880 , 0
removed \unex@namedef and \unex@nameuse, probably never really used since they were incomplete: \edef@other undefined, 7063
The code from ancient xparse (1999) of TeXLive 2007 rewritten here, 7063

\afterfifi:
\if removed from parameters' string, 357

\ampulexdef:
made xparse-ish and \ampulexset removed, 2523

\dekfracc:
made to work also in math mode, even with math-active digits, 4265

\gm@ifundefined:
added. All \@ifundefineds used by me changed to this, 623
made robust to unbalanced \ifs and \fis the same way as LaTeX's \@ifundefined (after a heavy debug :-), 623

\gmathfurther:
removed definition of \⟨*letter*⟩s and \⟨*digit*⟩s, 4844

\ldate:
\leftline replaced with \par…\par to work well with floatflt, 6303

\prependtomacro:
order of arguments reversed, 589

\resizegraphics:
\includegraphics works well in XeTeX so I remove the complicated version with \XeTeXpicfile, 4294

\textbullet:
the XeTeX version enriched with \iffontchar due to lack of bullets with the default settings reported by Morten Høgholm and Edd Barrett, 6193

v0.95

General:
CheckSum 4908 , 0

\gm@testdefined:
added, 641
\gm@testundefined:
added, 656
v0.96
General:
CheckSum 5363 , 0
put to CTAN on 2008/11/21, 0, 7063
\glyphname:
moved here from my private document class, 6198
\gmathfurther:
Greek letters completed. Wrapped with \addtotoks to allow using in any order with \garamath and others, 4844
the \everymath's left brace moved here: earlier all the stuff was put into \everymath, 4913
\gmathscripts:
added, 4959
\LuaTeX:
added, 4109
\pk:
\textup removed to allow slanting the name in titles (that are usually typeset in Italic font), 3053
v0.97
*:
removed (it was a text tilde, available as \texttilde), 5381
\@allbutfirstof:
added, 322
General:
CheckSum 5375 , 0
put to CTAN on 2008/11/22, 0
\pdfLaTeX:
added, 4086
v0.98
\@allbutfirstof:
renamed from \@secondofmany, 322
\@ifedetokens:
rewritten not to make entries in the hash table, thanks to \detokenize and made robust to open \ifs in the arguments thanks to substitution of explicit parameters with \@firstoftwo and \@secondoftwo, 2722
\@ifinmeaning:
moved here from gmdoc and rewritten, including split to \IfAmong because the latter is needed in \DeclareCommand, 442
\@ifnextsingle:
added test for \egroup, 956
extracted from \@ifnextac, 951
General:
CheckSum 5429 , 0
CheckSum 5577 because of \DeclareCommand, 0
CheckSum 5627 because of \fakeonum, 0
CheckSum 6035 because of \DeclareCommand, \arg, 0
CheckSum 6129 because of \DeclareEnvironment, 0
CheckSum 6147 because of \arg in verbatims, 0
CheckSum 6535 because of \UrlFix, 0
CheckSum 6656 because of type settings for gmdoc (\narrativett) and for \verbatimspecials, 0
\addto@macro:
\toks@ replaced with some \expandafters because use of \toks@ here caused a disaster in \DeclareCommand, 576
\ampulexdef:
a bug fixed: added \unexpanded, 2539
first argument (the prefix) made of the Q type, 2523
\arg:
made iterating thanks to \DeclareCommand\arg@dc, 3299
\ATfootnotes:
moved here from my own private macro package to allow the beauty of these footnotes for the general audience, 6611
\balsmiley:
moved here from my personal macro package since I needed it in the documentation of gmutils, 6674
\cs:
added \verbatim@specials, 3067
made \- switch to \normalfont because IMHO this underlines the fact that a CS belongs to the narrative. \hyphenchar set to 45 as in usual texts, 3067
\DeclareCommand:
added the \@bsphack—\@esphack option, 2262
added the Q{⟨*tokens*⟩} argument type (a word over the alphabet ⟨*tokens*⟩), 2262
added the S/T spec parsing, the s spec parsing rewritten to be a shorthand for S{*}, unused code removed, 1605
\enoughpage:
\par removed since to let it be used for \pagebreak inside a paragraph, 5495

made ε-TEX-ish, 5495
made ifthenelse-like with 'then' and 'else' optional default ⟨*nothing*⟩ and `\newpage` resp., 5495
`\fakeonum`:
added, 5148
`\gmu@tempb`:
renamed from `\@ifstar` since something redefines `\@ifstar`, 935
`\IfAmong`:
split from `\IfAmong`, 399
`\IfNoValueTF`:
the xparse tests for the presence of value redefined and much simplified (43 CS'es less). Moreover, they are now fully expandable: `\IfNoValueTF`, `\IfNoValueT`, `\IfNoValueF`, `\IfValueTF`, `\IfValueT`, `\IfValueF`, 2162
`\napapierkicore`:
added optional star controlling globalness, 6003
`\rrthis`:
added, 6661
`\scantnoline`:
added, 6682
v0.99
`\@ifedetokens`:
moved to a separate macro from `\@ifenvir` and made symmetric to both arguments, 2722
v0.991
`\@ifnextif`:
inner macro fixed to handle active chars more properly (more as `\if` would do it), 836
`\@xau`:
made 1-parameter, 225
General:
CheckSum 8257 because of some shorthands and major development of `\DeclareCommand`, including 'Knuthian' and general optional arguments and '`\afterassignment`' pseudo-argument, 0
put to CTAN on 2010/03/04, 0

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.