Grzegorz Murzynowski

The gmutils Package*

Written by Grzegorz Murzynowski,
natror at o2 dot pl

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{gmutils}
      [2008/11/22 v0.97 some rather TeXnical macros, some of them
           tricky (GM)]
```

Contents

* This file has version number v0.97 dated 2008/11/22.

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 3749.
\fi

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

A couple of abbreviations

\@xa
\@nx

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

\@xau

```
\def\@xau{\@xa\unexpanded\@xa}
```

\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}}
```

`\firstofone` and the queer `\catcode`s

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

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

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

`\foone`
```
\long\def\foone#1{\begingroup#1\egfirstofone}
\long\def\egfirstofone#1{\endgroup#1}
```

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

Global Boolean switches

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

`\newgif`
```
\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 \expandafters e.g.

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

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

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

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

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

And for use in the very document,

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

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 as \relax
      \afterfifi\@firstoftwo%
    \else% defined and not \relax
      \afterfifi\@secondoftwo%
    \fi
  \else% not defined
    \afterfi\@firstoftwo%
  \fi}
```

`\gm@testdefined`
```
\long\def\gm@testdefined#1\iftrue{% This is a macro that expands to \iftrue
      or \iffalse depending on whether it's argument is defined in the LaTeX
      sense. Its syntax requires an \iftrue to balance \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}
```

Storing and restoring the catcodes of specials

\gmu@storespecials
```
\newcommand*\gmu@storespecials[1][]{% we provide a possibility of adding stuff. For usage see line ??.
  \def\do##1{\catcode`\@nx##1=\the\catcode`##1\relax}%
  \edef\gmu@restorespecials{\dospecials\do\^^M#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}
```

From the ancient xparse of TEXLive 2007

The code of this section is rewritten contents of the xparse package version 0.17 dated 1999/09/10, the version available in TEX Live 2007-13, in Ubuntu packages at least. It's a stub 'im Erwartung' (Schönberg) for the LATEX3 bundle and it does what I want, namely defines `\DeclareDocumentCommand`. I rewrote the 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`).

Well, I add the `\protected` prefix to the first macro.

After exchange of some mails with Morten Høgholm and trying xparse of 2008/08/03 svn 748 (which beautifully spoils the catcodes) I wrap the ancient code in a conditional to avoid name collision if someone loads xparse before gmutils

```
\gm@testundefined\DeclareDocumentCommand\iftrue
\unless\ifdefined\@temptokenb
```

\@temptokenb
```
\newtoks\@temptokenb
\fi
```

\xparsed@args
```
\newtoks\xparsed@args
```

\DeclareDocumentCommand
```
\long\def\DeclareDocumentCommand#1#2#3{%
    % #1  command to be defined,
    % #2  arguments specification,
    % #3  definition body.
  \@tempcnta\z@
  \toks@{}%
  \@temptokena\toks@
  \@temptokenb\toks@
  \@ddc#2X% X is the guardian of parsing.
  \protected\edef#1{% The \protected prefix is my (GM) addition.
    \@nx\@ddc@
    {\the\toks@}%
    \@xa\@nx\csname\string#1\endcsname
    \@nx#1%
  }%
  \long\@xa\def\csname\string#1\@xa\endcsname
  \the\@temptokena{#3}}
```

\DeclareDocumentEnvironment
```
\long\def\DeclareDocumentEnvironment#1#2#3#4{%
  \@xa\DeclareDocumentCommand\csname#1\endcsname{#2}{%
    \xparsed@args\toks@
    #3}%
```

```
    \@xa\let\csname␣end#1\endcsname\@parsed@endenv
    \long\@xa\def\csname␣end\string\\#1\@xa\endcsname
    \the\@temptokena{#4}}
\@parsed@endenv
\def\@parsed@endenv{%
    \@xa\@parsed@endenv@\the\xparsed@args}
\@parsed@endenv@
\def\@parsed@endenv@#1{%
    \csname␣end\string#1\endcsname}
\@ddc@
\def\@ddc@#1#2#3{%
    \ifx\protect\@typeset@protect
    \@xa\@firstofone
    \else
    \protect#3\@xa\@gobble
    \fi
    {\toks@{#2}#1\the\toks@}}
\@ddc
\def\@ddc#1{%
    \ifx#1X%
    \else
    \ifx#1m%
    \addto@hook\@temptokenb␣m%
    \else
    \toks@\@xa{%
      \the\@xa\toks@
      \csname␣@ddc@\the\@temptokenb\@xa\endcsname
      \csname␣@ddc@#1\endcsname}%
    \@temptokenb{}%
    \fi
    \advance\@tempcnta\@ne
    \@temptokena\@xa{%
      \the\@xa\@temptokena\@xa##\the\@tempcnta}%
    \@xa
    \@ddc
    \fi}
\@ddc@s
\long\def\@ddc@s#1\toks@{%
    \@ifstar
    {\addto@hook\toks@\BooleanTrue#1\toks@}%
    {\addto@hook\toks@\BooleanFalse#1\toks@}}
\@ddc@m
\long\def\@ddc@m#1\toks@#2{%
    \addto@hook\toks@{{#2}}#1\toks@}%
\@ddc@o
\long\def\@ddc@o#1\toks@{%
    \@ifnextchar[%
    {\@ddc@o@{#1}}
    {\addto@hook\toks@\NoValue#1\toks@}}
\@ddc@o@
\long\def\@ddc@o@#1[#2]{%
    \addto@hook\toks@{{#2}}#1\toks@}
\@ddc
\def\@ddc#1{%
    \ifx#1X%
    \perhaps@grab@ms
    \else
    \ifx#1m%
```

```
    \addto@hook\@temptokenb␣m%
    \else
    \toks@\@xa{%
      \the\@xa\toks@
      \csname␣@ddc@x\the\@temptokenb\@xa\endcsname
      \csname␣@ddc@#1\endcsname}%
    \@temptokenb{}%
    \ifx#1O%
    \let\next@ddc\grab@default
    \else
    \ifx#1C%
    \let\next@ddc\grab@default
    \fi
    \fi
    \fi
    \advance\@tempcnta\@ne
    \@temptokena\@xa{%
      \the\@xa\@temptokena\@xa##\the\@tempcnta}%
    \@xa
    \next@ddc
    \fi
}%
\let\next@ddc\@ddc
\def\grab@default#1{%
    \toks@\@xa{%
      \the\toks@
      {#1}}%
    \let\next@ddc\@ddc
    \@ddc
}
\long\def\@ddc@O#1#2\toks@{%
    \@ifnextchar[%
    {\@ddc@o@{#2}}%
    {\addto@hook\toks@{{#1}}#2\toks@}}
\long\def\@ddc@c#1\toks@{%
    \@ifnextchar(%
    {\@ddc@c@#1}%
    {\PackageError{gmutils/xparse}{Missing~coordinate~argument}%
      {A~value~of~(o,o)~is~assumed}%
      \addto@hook\toks@{{oo}}#1\toks@}%
}
\long\def\@ddc@c@#1(#2,#3){%
    \addto@hook\toks@{{{#2}{#3}}}#1\toks@}
\long\def\@ddc@C#1#2\toks@{%
    \@ifnextchar(%
    {\@ddc@c@#2}%
    {\addto@hook\toks@{{#1}}#2\toks@}}
\let\perhaps@grab@ms\relax
\def\grab@ms{%
    \toks@\@xa{%
      \the\@xa\toks@
```

```
    \csname␣@ddc@x\the\@temptokenb\endcsname
  }}
\let\@ddc@m\undefined
\long\def\@ddc@xm#1\toks@#2{%
  \addto@hook\toks@{{#2}}#1\toks@}
\long\def\@ddc@xmm#1\toks@#2#3{%
  \addto@hook\toks@{{#2}{#3}}#1\toks@}
\long\def\@ddc@xmmm#1\toks@#2#3#4{%
  \addto@hook\toks@{{#2}{#3}{#4}}#1\toks@}
\long\def\@ddc@xmmmm#1\toks@#2#3#4#5{%
  \addto@hook\toks@{{#2}{#3}{#4}{#5}}#1\toks@}
\long\def\@ddc@xmmmmm#1\toks@#2#3#4#5#6{%
  \addto@hook\toks@{{#2}{#3}{#4}{#5}{#6}}#1\toks@}
\long\def\@ddc@xmmmmmm#1\toks@#2#3#4#5#6#7{%
  \addto@hook\toks@{{#2}{#3}{#4}{#5}{#6}{#7}}#1\toks@}
\long\def\@ddc@xmmmmmmm#1\toks@#2#3#4#5#6#7#8{%
  \addto@hook\toks@{{#2}{#3}{#4}{#5}{#6}{#7}{#8}}#1\toks@}
\long\def\@ddc@xmmmmmmmm#1\toks@#2#3#4#5#6#7#8#9{%
  \addto@hook\toks@{{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}#1\toks@}
\long\def\@ddc@xmmmmmmmmm\the\toks@#1#2#3#4#5#6#7#8#9{%
  \addto@hook\toks@{{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}\the%
       \toks@}
\let\@ddc@x\relax
\long\def\DeclareDocumentEnvironment#1#2#3#4{%
  \@xa\DeclareDocumentCommand\csname#1\endcsname{#2}{%
    #3}%
  \@namedef{end#1}{#4}%
}
\let\@parsed@endenv\undefined
\let\@parsed@endenv@\undefined
\def\IfSomethingTF#1{\def\something@in{#1}\If@SomethingTF}
\def\IfSomethingT#1#2#3{\def\something@in{#1}%
  \If@SomethingTF{#2}{#3}\@empty}
\def\IfSomethingF#1#2#3{\def\something@in{#1}%
  \If@SomethingTF{#2}\@empty{#3}}
\def\If@SomethingTF#1{%
  \def\something@tmp{#1}%
  \ifx\something@tmp\something@in
  \@xa\@secondofthree
  \else
  \@xa\def\@xa\something@tmpb\@xa{#1}%
  \ifx\something@tmp\something@tmpb
  \@xa\@xa\@xa\@thirdofthree
  \else
  \@xa\@xa\@xa\@firstofone
  \fi
  \fi
  {\@xa\If@SomethingTF\@xa{#1}}%
```

```
690 }
\@secondofthree 692 \long\def\@secondofthree#1#2#3{#2}
\@thirdofthree 693 \long\def\@thirdofthree#1#2#3{#3}
\NoValue 694 \def\NoValue{-NoValue-}
\NoValueInIt 695 \def\NoValueInIt{\NoValue}
\IfNoValueTF 696 \def\IfNoValueTF{\IfSomethingTF\NoValue}
\IfNoValueT 697 \def\IfNoValueT{\IfSomethingT\NoValue}
\IfNoValueF 698 \def\IfNoValueF{\IfSomethingF\NoValue}
\IfValueTF 699 \def\IfValueTF#1#2#3{\IfNoValueTF{#1}{#3}{#2}}
700 \let\IfValueT\IfNoValueF
701 \let\IfValueF\IfNoValueT
\BooleanFalse 702 \def\BooleanFalse{TF}
\BooleanTrue 703 \def\BooleanTrue{TT}
\IfBooleanTF 704 \def\IfBooleanTF#1{%
705   \if#1%
706   \@xa\@firstoftwo
707   \else
708   \@xa\@secondoftwo
709   \fi
710 }
\IfBooleanT 712 \def\IfBooleanT#1#2{%
713   \IfBooleanTF{#1}{#2}\@empty
714 }
\IfBooleanF 716 \def\IfBooleanF#1{%
717   \IfBooleanTF{#1}\@empty
718 }
720 \fi% of \unless\ifdefined\DeclareDocumentCommand.
```

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 1718.

```
\ampulexdef 755 \DeclareDocumentCommand\ampulexdef{O{}mO{}O{}mmm}{%
        % [#1] definition prefix, empty by default,
        %  #2  macro to be redefined,
        % [#3] \def parameters string, empty by default,
        % [#4] definition body parameters to be taken in a one-step expansion of the
            redefined macro, empty by default,
        %  #5  start token(s),
        %  #6  end token(s)
        %  #7  replacement.
```

For the example of usage see 1718.

```
\gmu@tempa 770  \def\gmu@tempa{#5}%
\gmu@tempb 771  \def\gmu@tempb{#6}%
\gmu@tempc 772  \def\gmu@tempc{#7}% we wrap the start, end and replacement tokens in macros
                    to avoid unbalanced \ifs.
           774  \edef\gmu@tempd{%
           775    \long\def\@nx\gmu@tempd
           776    ####1\all@other\gmu@tempa
           777    ####2\all@other\gmu@tempb
           778    ####3\@nx\gmu@tempd{%
           779      \@ifempty{####3}{\hidden@iffalse}{\hidden@iftrue}}}%
           781  \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 #2.
           785  \edef\gmu@tempe{%
           786    \@nx\gmu@tempd\all@other#2%
           787    \all@other\gmu@tempa
           788    \all@other\gmu@tempb\@nx\gmu@tempd
           789  }%
           791  \gmu@tempe% we apply the checker and it produces an open \if.
           793  \edef\gmu@tempd{%
           794    \long\def\@nx\gmu@tempd
           795    ####1\@xa\unexpanded\@xa{\gmu@tempa}%
           796    ####2\@xa\unexpanded\@xa{\gmu@tempb}%
           797    ####3\@nx\ampulexdef{% we define a temporary macro with the parameters
                      delimited with the 'start' and 'end' parameters of \ampulexdef.
           800      \@nx\unexpanded{####1}%
           801      \@nx\@xa\@nx\unexpanded
           802      \@nx\@xa{\@nx\gmu@tempc}% we replace the part of the redefined macro's
                        meaning with the replacement text.
           804      \@nx\unexpanded{####3}}}%
           806  \gmu@tempd
           809  \edef\gmu@tempf{#4}%
           810  \edef\gmu@tempe{%
           811    #1\def\@nx#2#3{%
           812      \@xa\@xa\@xa\gmu@tempd\@xa#2\gmu@tempf\ampulexdef}}%
           813  \gmu@tempe
           814  \fi}
\ampulexhash 816 \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 824 \def\gmshowlists{\showboxdepth=1000␣\showboxbreadth=1000␣%
                   \showlists}
     \nameshow 828 \newcommand\nameshow[1]{\@xa\show\csname#1\endcsname}
\nameshowthe 829 \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
```
\def\xiistring#1{%
  \if\@nx#1\xiispace
    \xiispace
  \else
    \string#1%
  \fi}
```

\@ifnextcat, \@ifnextac

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

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

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

```
{\def\:{\let\@sptoken=␣}␣\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.

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

\@ifnif
```
\def\@ifnif{%
  \ifx\@let@token\@sptoken
    \let\reserved@c\@xifnif
```

```
  \else
    \if\reserved@d\@nx\@let@token
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}

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

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

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

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

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

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

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

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

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

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}}}
```

`\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`.

```
\longafterfi 1001 \def\longafterfi{%
\afterfi     1002   \long\def\afterfi##1##2\fi{\fi##1}}
             1003 \longafterfi
```

And two more of that family:

```
\afterfifi      1005 \long\def\afterfifi#1#2\fi#3\fi{\fi\fi#1}
\afteriffifi    1007 \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  1011 \long\def\afterififfifi#1#2\fi#3\fi#4\fi{\fi#1}
\afteriffifi    1012 \long\def\afteriffifi#1#2\fi#3\fi#4\fi{\fi\fi#1}
\afterfifi      1013 \long\def\afterfifi#1#2\fi#3\fi#4\fi{\fi\fi\fi#1}
```

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 1045 \long\def\@begnamedgroup#1{%
                1046   \@ignorefalse% not to ignore blanks after group
                1047   \begingroup\@endpefalse
                1048   \edef\@currenvir{#1}% We could do recatcoding through \string but all the
                           name 'other' could affect a thousand packages so we don't do that and we'll
                           recatcode in a testing macro, see line 1093.
```

```
\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{\@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 evironments such as `\string\par`.

`\@ifenvir`

```
\long\def\@ifenvir#1#2#3{%
  \edef\gmu@reserveda{\@xa\string\csname\@currenvir\endcsname}%
  \edef\gmu@reservedb{\@xa\string\csname#1\endcsname}%
  \ifx\gmu@reserveda\gmu@reservedb\afterfi{#2}%
  \else\afterfi{#3}%
  \fi}
```

`\@checkend`

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

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

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

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

From relsize

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

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

`\relsize` `\smaller` `\larger` `\smallerr` `\largerr`

You declare the font size with `\relsize{`⟨*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
 is␣unknown!␣(Why?!?)\MessageBreak␣Assuming␣\string\normalsize}}
```

And a handful of shorthands:

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

Some ‘other’ stuff

Here I define a couple of macros expanding to special chars made ‘other’. It’s important the cs are expandable and therefore they can occur e.g. inside `\csname...\endcsname` unlike e.g. cs’es `\chardefed`.

```
\foone{\catcode`\_=8␣}%
\subs {\let\subs=_}

\foone{\@makeother\_}%
\xiiunder {\def\xiiunder{_}}

\ifdefined\XeTeXversion
\xiiunder   \def\xiiunder{\char"005F␣}%
            \let\_\xiiunder
\fi

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

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

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

```
\foone{\catcode`\_=\active}
{%
\smartunder   \newcommand*\smartunder{%
                \catcode`\_=\active
                \def_{\ifmmode\subs\else\_\fi}}}%
```

We define it as `\_` not just as `\xiiunder` because some font encodings don’t have _ at the `\char`_` position.

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

\bslash   \let\bslash=\xiibackslash

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

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

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

\foone{\@makeother\#}%
\xiihash  {\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 recognized the difference between `\xiispace` which has to be plain ‘other’ char (used in `\xiistring`) and something visible to be printed in any font.

```
\AtBeginDocument{%
```

```
  \ifdefined\xxt@visiblespace
    \let\visiblespace\xxt@visiblespace
  \else
    \let\visiblespace\xiispace
  \fi}
```

Metasymbols

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

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

`\meta`
```
\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."

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

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

```
    #1\/%
  }\ensuremath\rangle
}
```

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

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

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

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

Macros for printing macros and filenames

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

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

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

```
\discretionary{#1}{#1}{#1}\penalty10000\hskipposp\relax}
```

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

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

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

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

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

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

\env
```
\pdef\env#1{\cs[]{#1}}
```

And for the special sequences like ^^A:

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

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

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

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

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

```
\cmd 1415 \def\cmd#1{\cs{\@xa\cmd@to@cs\string#1}}
\cmd@to@cs 1417 \def\cmd@to@cs#1#2{\char\number`#2\relax}
```

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

```
\marg 1421 \def\marg#1{{\ttfamily\char`\{}\meta{#1}{\ttfamily\char`\}}}
```

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

```
\oarg 1426 \def\oarg{\@ifnextchar[\@oargsq\@oarg}
\@oarg 1428 \def\@oarg#1{{\ttfamily[}\meta{#1}{\ttfamily]}}
\@oargsq 1429 \def\@oargsq[#1]{\@oarg{#1}}
```

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

```
\parg 1433 \def\parg{\@ifnextchar(\@pargp\@parg}
\@parg 1435 \def\@parg#1{{\ttfamily(}\meta{#1}{\ttfamily)}}
\@pargp 1436 \def\@pargp(#1){\@parg{#1}}
```

But we can have all three in one command.

```
1440 \AtBeginDocument{%
\arg 1441   \let\math@arg\arg
\arg 1442   \def\arg{\ifmmode\math@arg\else\afterfi{%
1443       \@ifnextchar[%
1444       \@oargsq{\@ifnextchar(%
1445         \@pargp\marg}}\fi}%
1446 }
```

Now you can write

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

Storing and restoring the meanings of cses

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

```
\ifgmu@SMglobal 1461 \newif\ifgmu@SMglobal
\SMglobal 1463 \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 meanig back (when grouping is not recommended):

```
\StoreMacro 1474 \pdef\StoreMacro{%
1475   \begingroup\makeatletter\@ifstar\egStore@MacroSt\egStore@Macro}
```

The unstarred version takes a cs and the starred version a text, which is intended for special control sequences. For storing environments there is a special command in line 1598.

```
\egStore@Macro 1480 \long\def\egStore@Macro#1{\endgroup\Store@Macro{#1}}
\egStore@MacroSt 1481 \long\def\egStore@MacroSt#1{\endgroup\Store@MacroSt{#1}}
\Store@Macro 1483 \long\def\Store@Macro#1{%
1484   \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 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 expand-
        ing.
    \if\gmu@reserveda.% a dot finishes storing.
      \global\gmu@SMglobalfalse
    \else
      \if\gmu@reserveda,% The list this macro is put before may contain commas
          and that's O.K., we just continue the work.
        \afterfifi\gml@storemacros
      \else% what is else this shall be stored.
        \gml@StoreCS{#1}% we use a particular cs to may \let it both to the storing
            macro as above and to the restoring one as below.
        \afterfifi{\gmu@setSMglobal\gml@storemacros}%
      \fi
    \fi}
```

And for the restoring

```
\RestoreMacro 1554 \pdef\RestoreMacro{%
1555   \begingroup\makeatletter\@ifstar\egRestore@MacroSt%
           \egRestore@Macro}
\egRestore@Macro 1557 \long\def\egRestore@Macro#1{\endgroup\Restore@Macro{#1}}
\egRestore@MacroSt 1558 \long\def\egRestore@MacroSt#1{\endgroup\Restore@MacroSt{#1}}
\Restore@Macro 1560 \long\def\Restore@Macro#1{%
1561   \escapechar92
1562   \ifgmu@SMglobal\afterfi\global\fi
1563   \@xa\let\@xa#1\csname␣/gmu/store\string#1\endcsname
1564   \global\gmu@SMglobalfalse}
\Restore@MacroSt 1566 \long\def\Restore@MacroSt#1{%
1567   \edef\gmu@smtempa{%
1568     \ifgmu@SMglobal\global\fi
1569     \@nx\let\@xa\@nx\csname#1\endcsname
1570     \@xa\@nx\csname/gmu/store\bslash#1\endcsname}% cf. the commentary
           in line 1493.
1572   \gmu@smtempa
1573   \global\gmu@SMglobalfalse}
\RestoreMacros 1576 \long\pdef\RestoreMacros{\begingroup\makeatletter\Restore@Macros}
\Restore@Macros 1578 \long\def\Restore@Macros#1{\endgroup
1579   \gmu@setsetSMglobal
1580   \let\gml@StoreCS\Restore@Macro% we direct the core cs towards restoring
         and call the same iterating macro as in line 1521.
1583   \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 1589 \def\StoredMacro{\begingroup\makeatletter\Stored@Macro}
\Stored@Macro 1590 \long\def\Stored@Macro#1{\endgroup\Restore@Macro#1#1}
```

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

```
\storedcsname 1593 \def\storedcsname#1{%
1594   \csname␣/gmu/store\bslash#1\endcsname}
```

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

```
\StoreEnvironment 1598 \pdef\StoreEnvironment#1{%
1600   \StoreMacro*{#1}\StoreMacro*{end#1}}
\RestoreEnvironment 1602 \pdef\RestoreEnvironment#1{%
1604   \RestoreMacro*{#1}\RestoreMacro*{end#1}}
```

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

```
\StoringAndRelaxingDo 1619 \long\def\StoringAndRelaxingDo{%
1620   \gmu@SMdo@setscope
1621   \long\def\do##1{%
1622     \gmu@SMdo@scope
1623     \@xa\let\csname␣/gmu/store\string##1\endcsname##1%
```

```
    \gmu@SMdo@scope\let##1\relax}}
```

\gmu@SMdo@setscope
```
\def\gmu@SMdo@setscope{%
  \ifgmu@SMglobal\let\gmu@SMdo@scope\global
  \else\let\gmu@SMdo@scope\relax
  \fi
  \global\gmu@SMglobalfalse}
```

And here is the counter-definition for restore.

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

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

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

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

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

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

Not only preamble!

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

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

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

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

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

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 in with. 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 not too large. the first is the parameters
    string and the second the argument for one-level expansion of \nocite so it
    has to consist of two times less hashes than the first. 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?

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

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

gm@PronounGender

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

\gm@atppron

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

\heshe

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

\hisher

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

\himher

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

\hishers

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

\HeShe

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

\HisHer

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

\HimHer

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

\HisHers

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

Improvements to mwcls sectioning commands

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

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

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

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

Auxiliary macros for the kernel sectioning macro:

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

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

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

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

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

The kernel of MW's sectioning commands:

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

[2] A. Berg, *Wozzeck*.

```
%     \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
    \gm@hyperrefstepcounter
  \fi% of \ifHeadingNumbered

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

```
    \fi% of \if\mw@HeadingRunIn
    }
```

An improvement of MW's \SetSectionFormatting

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

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

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

```
\relaxen\SetSectionFormatting
\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 distinctor,
          % #3 the meaning,
          % #4 the number of argument to error message.
    \@ifempty{#3}
      {\mw@secundef{#1#2}{#4}}
      {\@namedef{#1#2}{#3}}}
\mw@secundef \def\mw@secundef#1#2{%
    \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 \def\addtoheading#1#2{%
      \n@melet{gmu@reserveda}{#1@head}%
```

```
      \edef\gmu@reserveda{\unexpanded{#2}\@xa\unexpanded{%
          \gmu@reserveda}}%
      \n@melet{#1@head}{gmu@reserveda}%
  }
}% 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.

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

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

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

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

\gmu@checkaftersec
```
\def\gmu@checkaftersec{%
  \gm@ifundefined{gmu@prevsec}{}{%
    \ifgmu@postsec% an additional switch that is true by default but may be
        turned into an \ifdim in special cases, see line 2058.
    {\@xa\mw@getflags\@xa{\gmu@prevsec}%
      \glet\gmu@reserveda\mw@HeadingBreakAfter}%
    \if\mw@HeadingBreakBefore\def\gmu@reserveda{11}\fi% if the current
        heading inserts page break before itself, all the play with vskips is irrele-
        vant.
    \if\gmu@reserveda\else
    \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 spe-
      cial 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.
\def\WPheadings{%
  \SetSectionFormatting[breakbefore,wholewidth]
      {part}{\z@\@plus1fill}{}{\z@\@plus3fill}%
  \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@}%
  }%
  \SetTwoheadSkip{section}{27\p@\@pluso,5\p@}%
  \SetSectionFormatting{section}
      {24\p@\@pluso,5\p@\@minus5\p@}%
      {\FormatHangHeading␣{\Large}}
      {10\p@\@pluso,5\p@}% ed. Krajewska of "Wiedza Powszechna", as we understand her, wants the skip between a heading and text to be rigid.
  \SetTwoheadSkip{subsection}{11\p@\@pluso,5\p@\@minus1\p@}%
  \SetSectionFormatting{subsection}
      {19\p@\@pluso,4\p@\@minus6\p@}
      {\FormatHangHeading␣{\large}}% 12/14 pt
      {6\p@\@pluso,3\p@}% after-skip 6 pt due to p.12, not to squeeze the before-skip too much.
  \SetTwoheadSkip{subsubsection}{10\p@\@plus1,75\p@\@minus1\p@}%
  \SetSectionFormatting{subsubsection}
      {10\p@\@pluso,2\p@\@minus1\p@}
      {\FormatHangHeading␣{\normalsize}}
      {3\p@\@pluso,1\p@}% those little skips should be smaller than you calculate out of a geometric progression, because the interline skip enlarges them.
  \SetSectionFormatting[runin]{paragraph}
      {7\p@\@pluso,15\p@\@minus1\p@}
      {\FormatRunInHeading{\normalsize}}
      {2\p@}%
  \SetSectionFormatting[runin]{subparagraph}
      {4\p@\@plus1\p@\@minuso,5\p@}
      {\FormatRunInHeading{\normalsize}}
      {\z@}%
}% of \WPheadings
}% of \@ifnotmw
```

Compatibilising standard and mwcls sectionings

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

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

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

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

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

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

```
\@ifnotmw{% we are not in mwcls and want to handle mwcls-like sectionings i.e., those written with two optionals.
  \def\gm@secini{gm@la}%
  \def\gm@secxx#1#2[#3]#4{%
    \ifx\gm@secstar\@empty
      \n@melet{gm@true@#1mark}{#1mark}% a little trick to allow a special version of the heading just to the running head.
      \@namedef{#1mark}##1{% we redefine \⟨sec⟩mark to gobble its argument and to launch the stored true marking command on the appropriate argument.
        \csname gm@true@#1mark\endcsname{#2}%
        \n@melet{#1mark}{gm@true@#1mark}% after we've done what we wanted we restore original \#1mark.
      }%
      \def\gm@secstar{[#3]}% if \gm@secstar is empty, which means the sectioning command was written starless, we pass the 'true' sectioning command #3 as the optional argument. Otherwise the sectioning command was written with star so the 'true' s.c. takes no optional.
    \fi
    \@xa\@xa\csname\gm@secini#1\endcsname
    \gm@secstar{#4}}%
}{% we are in mwcls and want to reverse MW's optionals order i.e., if there's just one optional, it should go both to toc and to running head.
  \def\gm@secini{gm@mw}%
  \let\gm@secmarkh\@gobble% in mwcls there's no need to make tricks for special version to running headings.
  \def\gm@secxx#1#2[#3]#4{%
    \@xa\@xa\csname\gm@secini#1\endcsname
    \gm@secstar[#2][#3]{#4}}%
}

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

\def\gm@straightensec#1{% the parameter is for the command's name.
  \gm@ifundefined{#1}{}{% we don't change the ontological status of the command because someone may test it.
```

```
    \n@melet{\gm@secini#1}{#1}%
    \@namedef{#1}{%
\gm@secstar   \@ifstar{\def\gm@secstar{*}\gm@sec{#1}}{%
\gm@secstar     \def\gm@secstar{}\gm@sec{#1}}}%
}%
\let\do\gm@straightensec
\do{part}\do{chapter}\do{section}\do{subsection}\do{%
     subsubsection}
\@ifnotmw{}{\do{paragraph}}% this 'straightening' of \paragraph with the standard article caused the 'TeX capacity exceeded' error. Anyway, who on Earth wants paragraph titles in toc or running head?
```

`enumerate*` and `itemize*`

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

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

```
enumerate*  \@namedef{enumerate*}{%
              \ifnum\@enumdepth>\thr@@
                \@toodeep
              \else
                \advance\@enumdepth\@ne
                \edef\@enumctr{enum\romannumeral\the\@enumdepth}%
                \@xa\list\csname␣label\@enumctr\endcsname{%
                  \partopsep\topsep␣\topsep\z@␣\leftmargin\z@
                  \itemindent\@parindent␣% %\advance\itemindent\labelsep
                  \labelwidth\@parindent
                  \advance\labelwidth-\labelsep
                  \listparindent\@parindent
                  \usecounter␣\@enumctr
                  \def\makelabel##1{##1\hfil}}%
              \fi}
            \@namedef{endenumerate*}{\endlist}

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

The logos

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

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

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

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

\gmu@reserveda
```
    \def\gmu@reserveda{#1}%
  \fi
  \edef\gmu@reserveda{%
    \@nx\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 −, 15 em for T.
  {%
    \ifdim\fontdimen1\font=\z@
    \else
      \count\z@=\fontdimen5\font
      \multiply\count\z@ by 64\relax
      \divide\count\z@ by\p@
      \count\tw@=\fontdimen1\font
      \multiply\count\tw@ by\count\z@
      \divide\count\tw@ by 64\relax
      \divide\count\tw@ by\tw@
      \kern-\the\count\tw@ sp\relax
    \fi}%
  \TeX}
```

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

```
\StoreMacro\LaTeX
\StoreMacro*{LaTeX }
```

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

```
\LaTeXpar 2360 \DeclareLogo[(La)TeX]{\LaTeXpar}{%
          2361   {%
          2362     \setbox\z@\hbox{(}%)
          2363     \copy\z@
          2364     \kern-.2\wd\z@␣L%
          2365     \setbox\z@\hbox{\check@mathfonts
          2366       \fontsize\sf@size\z@
          2367       \math@fontsfalse\selectfont
          2368       A}%
          2369     \kern-.57\wd\z@
          2370     \sbox\tw@␣T%
          2371     \vbox␣to\ht\tw@{\box\z@%
          2372       \vss}%
          2373   }%
          2374   \kern-.07em% originally −, 15 em for T.
          2375   {%(
          2376     \sbox\z@)%
          2377     \kern-.2\wd\z@\copy\z@
          2378     \kern-.2\wd\z@}\TeX
          2379 }
```

"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."

```
          2386 \gm@ifundefined{AmSTeX}
  \AmSTeX 2387   {\def\AmSTeX{\leavevmode\hbox{$\mathcal␣A\kern-.2em%
                      \lower.376ex%
          2388          \hbox{$\mathcal␣M$}\kern-.2em\mathcal␣S$-\TeX}}}{}
  \BibTeX 2390 \DeclareLogo\BibTeX{{\rmfamily␣B\kern-.05em%
          2391     \textsc{i{\kern-.025em}b}\kern-.08em% the kern is wrapped in braces
                       for my \fakescaps' sake.
          2393     \TeX}}
  \SliTeX 2396 \DeclareLogo\SliTeX{{\rmfamily␣S\kern-.06emL\kern-.18em%
                   \raise.32ex\hbox
          2397        {\scshape␣i}\kern␣-.03em\TeX}}
\PlainTeX 2399 \DeclareLogo\PlainTeX{\textsc{Plain}\kern2pt\TeX}
    \Web  2401 \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 2404 \DeclareLogo[The␣TeX␣book]\TeXbook{\textsl{The␣\TeX␣book}}
         2405 \let\TB\TeXbook% TUG Boat uses this.
   \eTeX 2407 \DeclareLogo[e-TeX]\eTeX{%
         2408   \iffontchar\font"03B5{\itshape␣ }\else
         2409   \ensuremath{\varepsilon}\fi-\kern-.125em\TeX}% definition sent by Karl
                      Berry from TUG Boat itself.
         2412 \StoreMacro\eTeX
\pdfeTeX 2414 \DeclareLogo[pdfe-TeX]\pdfeTeX{pdf\eTeX}
 \pdfTeX 2416 \DeclareLogo\pdfTeX{pdf\TeX}
\pdfLaTeX 2417 \DeclareLogo\pdfLaTeX{pdf\LaTeX}
```

```
\gm@ifundefined{XeTeX}{%
  \DeclareLogo\XeTeX{X\kern-.125em\relax
    \gm@ifundefined{reflectbox}{%
      \lower.5ex\hbox{E}\kern-.1667em\relax}{%
      \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
    \TeX}}{}
\gm@ifundefined{XeLaTeX}{%
  \DeclareLogo\XeLaTeX{X\kern-.125em\relax
    \gm@ifundefined{reflectbox}{%
      \lower.5ex\hbox{E}\kern-.1667em\relax}{%
      \lower.5ex\hbox{\reflectbox{E}}\kern-.1667em\relax}%
    \LaTeX}}
```

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

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

Expandable turning stuff all into 'other'

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

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

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

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

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

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

Brave New World of XETEX

```
\newcommand\@ifXeTeX[2]{%
  \ifdefined\XeTeXversion
  \unless\ifx\XeTeXversion\relax\afterfifi{#1}\else\afterfifi{%
      #2}\fi
  \else\afterfi{#2}\fi}
\DeclareDocumentCommand\XeTeXthree{o}{%
  \@ifXeTeX{%
    \IfValueT{#1}{\PassOptionsToPackage{#1}{fontspec}}%
    \@ifpackageloaded{gmverb}{\StoreMacro\verb}{}%
    \RequirePackage{xltxtra}% since v 0.4 (2008/07/29) this package redefines \verb and verbatim*, and quite elegantly provides an option to suppress the redefinitions, but unfortunately that option excludes also a nice definition of \xxt@visiblespace which I fancy.
```

```
    \@ifpackageloaded{gmverb}{\RestoreMacro\verb}{}%
    \AtBeginDocument{%
      \RestoreMacro\LaTeX\RestoreMacro*{LaTeX }% my version of the
            LaTeX logo has been stored just after defining, in line 2354.
    \RestoreMacro\eTeX}%
  }{}}
```

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

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

Fractions

```
\Xedekfracc \def\Xedekfracc{\@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 `\@namedefs` below) and the `numr` and `dnom` features with the fractional slash otherwise (via `\gmu@dekfracc`).

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

```
\gmu@xedekfraccstar \def\gmu@xedekfraccstar{%
  \gmu@xefraccdef  \def\gmu@xefraccdef##1##2{%
      \iffontchar\font ##2
        \@namedef{gmu@xefracc##1}{\char##2 }%
      \else
        \n@melet{gmu@xefracc##1}{relax}%
      \fi}%
\gmu@dekfracc    \def\gmu@dekfracc##1/##2{%
      {\addfontfeature{VerticalPosition=Numerator}##1}%
          \gmu@numeratorkern
      \char"2044 \gmu@denominatorkern
      {\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.

```
    \gmu@xefraccdef{1/4}{"BC}%
    \gmu@xefraccdef{1/2}{"BD}%
    \gmu@xefraccdef{3/4}{"BE}%
    \gmu@xefraccdef{1/3}{"2153}%
    \gmu@xefraccdef{2/3}{"2154}%
    \gmu@xefraccdef{1/8}{"215B}%
    \gmu@xefraccdef{3/8}{"215C}%
    \gmu@xefraccdef{5/8}{"215D}%
    \gmu@xefraccdef{7/8}{"215E}%
\dekfracc@args  \pdef\dekfracc@args##1/##2{%
  \gm@duppa       \def\gm@duppa{##1/##2}%
```

```
      \gm@ifundefined{gmu@xefracc\all@other\gm@duppa}{%
        \gmu@dekfracc{##1}/{##2}}{%
        \csname␣gmu@xefracc\all@other\gm@duppa\endcsname}%
      \if@gmu@mmhbox\egroup\fi
    }% of \dekfracc@args.
    \@ifstar{\let\gmu@dekfracc\gmu@dekfraccsimple}{}%
  }
\gmu@xedekfraccplain
\dekfracc@args
\def\gmu@xedekfraccplain{% 'else' of the main \@ifstar
    \pdef\dekfracc@args##1/##2{%
      \ifmmode\hbox\fi{%
        \addfontfeature{Fractions=On}%
        ##1/##2}%
      \if@gmu@mmhbox\egroup\fi
    }% of \dekfracc@args
  }
\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
\def\gmu@numeratorkern{\kern-.05em\relax}
\let\gmu@denominatorkern\gmu@numeratorkern
```

What have we just done? We defined two versions of the `\Xefractions` declaration. The starred version is intended to make use only of the built-in fractions such as ½ or ⅞. To achieve that, a handful of macros is defined that expand to the Unicodes of built-in fractions and `\dekfracc` command is defined to use them.

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

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

A simpler version of `\dekfracc` is provided in line 3439.

`\resizegraphics`

```
\resizegraphics
\def\resizegraphics#1#2#3{%
  \resizebox{#1}{#2}{%
    \includegraphics{#3}}}
\GMtextsuperscript
\textsuperscript
\def\GMtextsuperscript{%
  \@ifXeTeX{%
    \def\textsuperscript##1{{%
        \addfontfeature{VerticalPosition=Numerator}##1}}%
  }{\truetextsuperscript}}
\truetextsuperscript
\textsuperscript
\@textsuperscript
\def\truetextsuperscript{%
  \pdef\textsuperscript##1{%
    \@textsuperscript{\selectfont##1}}%
  \def\@textsuperscript##1{%
    {\m@th\ensuremath{^{\mbox{\fontsize\sf@size\z@##1}}}}}}
```

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.

\gmu@getfontstring
```
\def\gmu@getfontstring{%
  \xdef\gmu@fontstring{%
    \gmu@fontstring@}}
```

\gmu@fontstring@
```
\def\gmu@fontstring@{%
  \@xa\@xa\@xa\gmu@quotedstring\@xa\meaning\the\font\@@nil}
```

\gmu@quotedstring
```
\def\gmu@quotedstring#1"#2"#3\@@nil{"#2"}
```

\gmu@getfontscale
```
\def\gmu@getfontscale#1Scale#2=#3,{%
  \ifx\gmu@getfontscale#3\else
  \gdef\gmu@fontscale{[#3] }%
  \afterfi\gmu@getfontscale\fi
}
```

\gmu@getfontdata
```
\def\gmu@getfontdata#1{%
  \global\emptify\gmu@fontscale
  \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
```
\def\gmath@getfamnum{%
  \edef\gmath@famnum{\@xa\gmu@stripchar\meaning\gmath@fam}%
}
```

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

\gmathbase
```
\pdef\gmathbase{%
  \gmu@getfontdata{\rmfamily\itshape}%
  \edef\gmu@tempa{%
    \@nx\DeclareSymbolFont{letters}{\encodingdefault}{gmathit}{%
        m}{it}%
    \@nx\DeclareFontFamily{\encodingdefault}{gmathit}{%
      \skewchar\font\gmu@theskewchar\space}%
    \@nx\DeclareFontShape{\encodingdefault}{gmathit}{m}{it}{%
      <-> \gmu@fontscale \gmu@fontstring}{}%
  }\gmu@tempa\typeout{@@@ gmathit (letters): \meaning\gmu@tempa}%
  \gmu@getfontdata{\rmfamily\upshape}%
  \edef\gmu@tempa{%
    \@nx\DeclareSymbolFont{gmathroman}{\encodingdefault}{%
        gmathrm}{m}{n}%
    \@nx\DeclareFontFamily{\encodingdefault}{gmathrm}{%
      \skewchar\font\gmu@theskewchar\space}%
    \@nx\DeclareFontShape{\encodingdefault}{gmathrm}{m}{n}{%
      <-> \gmu@fontscale \gmu@fontstring}{}%
```

```
}\gmu@tempa\typeout{@@@␣gmathrm␣(upright␣symbols):
  \meaning\gmu@tempa}%
\font\gmath@font=\gmu@fontstring\relax
\gmath@do \DeclareDocumentCommand\gmath@do{mom}{%
      %  #1   the character or cs to be declared,
      % [#2]  the Unicode to be assigned,
      %  #3   math type (cs like \mathord etc.)
  \gmath@getfamnum
  \IfValueTF{##2}{%
    \edef\gmu@tempa{%
      =␣\mathchar@type##3\space
      \gmath@famnum\space
      "##2\relax}%
    \if\relax\@nx##1%
      \edef\gmu@tempa{%
        \XeTeXmathchardef␣\@nx##1\gmu@tempa}%
      \else
      \edef\gmu@tempa{%
        \XeTeXmathcode␣`##1␣\gmu@tempa}
      \fi%
    }%
    {%
      \edef\gmu@tempa{%
        \XeTeXmathcode␣`##1␣=
        \mathchar@type##3\space
        \gmath@famnum\space
        `##1\relax}%
    }%
    \gmu@tempa
    \typeout{@@@@␣\@nx##1}%
    \typeout{@@@@␣\meaning\gmu@tempa}%
  }% of \gmath@do
\gmath@doif \DeclareDocumentCommand\gmath@doif{mmmoo}{%
      %  #1   the Unicode of char enquired,
      %  #2   the char or cs to be declared,
      %  #3   math type cs(\mathord etc.),
      % [#4]  second-choice Unicode (taken if first-choice is absent),
      % [#5]  third-choice Unicode (as above if second-choice is absent from
          font).
    \iffontchar\gmath@font"##1␣\gmath@do##2[##1]##3%
    \else\IfValueT{##4}{%
      \iffontchar\gmath@font"##4␣\gmath@do##2[##4]##3%
      \else\IfValueT{##5}{%
        \iffontchar\gmath@font"##5␣\gmath@do##2[##5]##3%
        \fi}%
      \fi}%
  \fi}%
\iffalse␣% doesn't work in a non-math font.
\gmath@delc \DeclareDocumentCommand\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
    \typeout{@@@@␣\@nx##1}%
    \typeout{@@@@␣\meaning\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.
\gmu@dogmathbase  \pdef\gmu@dogmathbase{%
    \let\gmath@fam\symgmathroman
    \typeout{@@@␣gmutils.sty:␣taking␣some␣math␣chars␣from␣the␣
        font^^J␣\gmu@fontstring@}%
    \gmath@do+\mathbin
    \gmath@doif{2212}-\mathbin[2013]% minus sign if present or else en dash
    \gmath@do=\mathrel
    \gmath@doo\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{2A7D}\xleq\mathrel
    \gmath@doif{2A7E}\xgeq\mathrel
    \@ifpackageloaded{polski}{%
      \ifdefined\xleq
```

```
    \let\leq=\xleq
    \let\le=\leq
    \fi
    \ifdefined\xgeq
    \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@do:\mathrel
  \gmath@doif{00B7}·\mathbin
  \gmath@doif{22C6}*\mathbin
  \gmath@doif{2300}\varnothing\mathord
  \gmath@doif{221E}\infty\mathord
  \gmath@doif{2248}\approx\mathrel
  \gmath@doif{2260}\neq\mathrel
  \let\ne\neq
  \gmath@doif{00AC}\neg\mathbin
  \gmath@do/\mathop
  \gmath@do<\mathrel
  \gmath@do>\mathrel
  \gmath@doif{2329}\langle\mathopen
  \gmath@doif{232A}\rangle\mathclose
  \gmath@doif{2202}\partial\mathord
  \gmath@doif{00B1}\pm\mathbin
  \gmath@doif{007E}\sim\mathrel
  \gmath@doif{2190}\leftarrow\mathrel
  \gmath@doif{2192}\rightarrow\mathrel
  \gmath@doif{2194}\leftrightarrow\mathrel% if not present, \gmathfurther
      will take care of it if left and right arrows are present.
  \gmath@doif{2191}\uparrow\mathrel% 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@doif{2208}\in\mathrel[03F5][0454]%
```

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]% 'medium white square (modal
        operator)' of just 'white square'.
  \fi
```

```
    \gmath@doif{EB08}\bigcircle\mathbin
    \gmath@doif{2227}\wedge\mathbin
    \gmath@doif{2228}\vee\mathbin
    \gmath@doif{0393}\Gamma\mathalpha
    \gmath@doif{0394}\Delta\mathalpha
    \gmath@doif{0398}\Theta\mathalpha
    \gmath@doif{039B}\Lambda\mathalpha
    \gmath@doif{039E}\Xi\mathalpha
    \gmath@doif{03A3}\Sigma\mathalpha
    \gmath@doif{03A5}\Upsilon\mathalpha
    \gmath@doif{03 6}\Phi\mathalpha
    \gmath@doif{03A8}\Psi\mathalpha
    \gmath@doif{03A9}\Omega\mathalpha
    \let\gmath@fam\symletters
    \gmath@doif{03B1}\alpha\mathalpha
    \gmath@doif{03B2}\beta\mathalpha
    \gmath@doif{03B3}\gamma\mathalpha
    \gmath@doif{03B4}\delta\mathalpha
    \gmath@doif{03F5}\epsilon\mathalpha
    \gmath@doif{03B5}\varepsilon\mathalpha
    \gmath@doif{03B6}\zeta\mathalpha
    \gmath@doif{03B7}\eta\mathalpha
    \gmath@doif{03B8}\theta\mathalpha
    \gmath@doif{03D1}\vartheta\mathalpha
    \gmath@doif{03B9}\iota\mathalpha
    \gmath@doif{03BA}\kappa\mathalpha
    \gmath@doif{03BB}\lambda\mathalpha
    \gmath@doif{03BC}\mu\mathalpha
    \gmath@doif{03BD}\nu\mathalpha
    \gmath@doif{03BE}\xi\mathalpha
    \gmath@doif{03C0}\pi\mathalpha
    \gmath@doif{03A0}\Pi\mathalpha
    \gmath@doif{03C1}\rho\mathalpha
    \gmath@doif{03C3}\sigma\mathalpha
    \gmath@doif{03DA}\varsigma\mathalpha% 03C2?
    \gmath@doif{03C4}\tau\mathalpha
    \gmath@doif{03C5}\upsilon\mathalpha
    \gmath@doif{03D5}\phi\mathalpha
    \gmath@doif{03C8}\psi\mathalpha
    \gmath@doif{03C9}\omega\mathalpha
    \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.
  }%
  \AtBeginDocument{\gmu@dogmathbase\let\gmathbase%
       \gmu@dogmathbase}%
  \not@onlypreamble\gmathbase
}% of \gmathbase
```

```
\@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{\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{\setboxo=\hbox{A}\setbox2=\hbox{%
         \scriptsize␣x}%
      \kern\dimexpr\ht2/3*2␣-\wdo/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{\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␣o\else1\fi
        \iffontchar\gmath@font"2192␣o\else2\fi
        \do\leftrightarrow{\char"2190\kern-0,1em␣\char"2192}%
             \mathrel
    \fi\fi
    \def\do##1##2##3{\def##1####1{##2{\hbox{%
            \rm
            \setboxo=\hbox{####1}%
            \edef\gma@tempa{\the\hto}%
            \edef\gma@tempb{\the\dpo}%
            ##3%
            \setboxo=\hbox{####1}%
```

```
            \lower\dimexpr(\ht\box+\dp\box)/2-\dp\box-((\gma@tempa+%
                \gma@tempb)/2-\gma@tempb)%
            \box\box}}}}%
  \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{\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{\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.

\gmath      \def\gmath{\gmathbase\gmathfurther}

\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 3073 \def\gma@dollar$#1${{\gmath$#1$}}%
  \gma@bare 3074 \def\gma@bare#1{\gma@dollar$#1$}%
\gma@checkbracket 3075 \def\gma@checkbracket{\@ifnextchar\[%
3076   \gma@bracket\gma@bare}
\gma@bracket 3077 \def\gma@bracket\[#1\]{{\gmath\[#1\]}\@ifnextchar\par{}{%
         \noindent}}
       \gma 3078 \def\gma{\@ifnextchar$%
3079   \gma@dollar\gma@checkbracket}
\garamath 3085 \def\garamath{%
3086   \addtotoks\everymath{%
3087     \quantifierhook{\addfontfeature{OpticalSize=800}}%
\gma@arrowdash 3089     \def\gma@arrowdash{{%
3090         \setboxo=\hbox{\char"2192}\copyo\kern-0,6\wdo
3091         \bgcolor\rule[-\dpo]{0,6\wdo}{\dimexpr\hto+\dpo}%
               \kern-0,6\wdo}}%
\gma@gmathhook 3093     \def\gma@gmathhook{%
3094       \def\do####1####2####3{\def####1{####3{%
  \mathchoice 3095           \mathchoice{\hbox{\rm####2}}{\hbox{\rm####2}}%
3096           {\hbox{\rm\scriptsize####2}}{\hbox{\rm%
                 \tiny####2}}}}}%
3097       \do\mapsto{\rule[0,4ex]{0,1ex}{0,4ex}\kern-0,05em%
3098         \gma@arrowdash\kern-0,05em\char"2192}\mathrel
3099       \do\cup{\scshape␣u}\mathbin
3100       \do\varnothing{\setboxo=\hbox{\gma@quantifierhook%
             \addfontfeature{Scale=1.272727}o}%
3101         \setbox2=\hbox{\char"2044}%
3102         \copyo␣\kern-0,5\wdo␣\kern-0,5\wd2␣\lowero,125\wdo␣\copy2
3103         \kerno,5\wdo\kern-0,5\wd2}{}%
3104       \do\leftarrow{\char"2190\kern-0,05em\gma@arrowdash}\mathrel
3105       \do\rightarrow{\gma@arrowdash\kern-0,05em\char"2192}%
             \mathrel
3106       \do\in{\gma@quantifierhook\char"0454}\mathbin
3107     }}%
3108   \everydisplay\everymath}
```

Minion and Garamond Premier kerning and ligature fixes

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

```
\gmu@tempa 3117 \def\gmu@tempa{\kern-0,08em\penalty10000\hskiposp\relax
3118   s\penalty10000\hskiposp\relax}
3120 \protected\edef\Vs{V\gmu@tempa}
3122 \protected\edef\Ws{W\gmu@tempa}
       \Wz 3124 \pdef\Wz{W\kern-0,05em\penalty10000\hskiposp\relax␣z}
```

Varia

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

```
\gmu@tilde 3133 \def\gmu@tilde{%
3134   \leavevmode\lower.8ex\hbox{$\,\widetilde{\mbox{␣}}\,$}}
```

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

*
```
\pdef\*{\gmu@tilde}
```

```
\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}
```

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

\dekbigskip
```
\def\dekbigskip{\vskip\bigskipamount}
```

\hfillneg
```
\def\hfillneg{\hskip␣opt␣plus␣-1fill\relax}
```

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

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

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

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

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

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

\twocoltoc
```
\newcommand*\twocoltoc{%
  \RequirePackage{multicol}%
```

\@starttoc
```
  \def\@starttoc##1{%
    \begin{multicols}{2}\makeatletter\@input␣{\jobname␣.##1}%
      \if@filesw␣\@xa␣\newwrite␣\csname␣tf@##1\endcsname
        \immediate␣\openout␣\csname␣tf@##1\endcsname␣\jobname␣
            .##1\relax
```

```
      \fi
      \@nobreakfalse\end{multicols}}}
\@onlypreamble\twocoltoc
```

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

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

An equality sign properly spaced:

\equals
```
\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 `\edefs`.

\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 be-
```

cause if we don't, then verse's \everypar would be executed in a group and thus its effect lost.

```
\*  \def\*{\czer{$*$}}
\+  \def\+{\czer{$\dag$}}
\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 nad I liked it.

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

\iteracro

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

when #2 is nonempty, it is ended with a space. Adding one more space in this line resulted in an infinite loop, 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
```
\def\acrocore{\scshape\lowercase}
```

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

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

\usc
```
\pdef\usc#1{{\addfontfeature{Letters=UppercaseSmallCaps}#1}}
```

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

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

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

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

\gmu@dekfraccsimple
```
\def\gmu@dekfraccsimple#1/#2{\leavevmode\kern.1em
  \raise.5ex\hbox{%
    \smaller[3]#1}\gmu@numeratorkern
  \dekfraccslash\gmu@denominatorkern
  {%
    \smaller[3]#2}%
  \if@gmu@mmhbox\egroup\fi}
```

\dekfraccsimple
```
\def\dekfraccsimple{%
  \let\dekfracc@args\gmu@dekfraccsimple
}
```

\dekfraccslash
\dekfraccslash
```
\@ifXeTeX{\def\dekfraccslash{\char"2044␣}}{%
  \def\dekfraccslash{/}}␣% You can define it as the fraction
      slash, \char"2044␣
\dekfraccsimple
```

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

\ikern
```
\newcommand*\ikern{\,\penalty10000\hskiposp\relax}
```

And a macro to forbid hyphenation of the next word:

\nohy
\yeshy
```
\newcommand*\nohy{\leavevmode\kernosp\relax}
\newcommand*\yeshy{\leavevmode\penalty10000\hskiposp\relax}
```

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

\@ifempty

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

\include not only .tex's

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

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

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

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

Faked small caps

```
\def\gmu@scapLetters#1{%
  \ifx#1\relax\relax\else% two \relaxes to cover the case of empty #1.
    \ifcat␣a#1\relax
      \ifnum\the\lccode`#1=`#1\relax
        {\fakescapscore\MakeUppercase{#1}}% not Plain \uppercase because that works bad with inputenc.
      \else#1%
      \fi
    \else#1%
    \fi%
    \@xa\gmu@scapLetters
  \fi}%
\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}
\def\gmu@scapss#1\@@nil{{\def~{{\nobreakspace}}%
    \gmu@scapSpaces#1␣\@@nil}}%%␣\def\\{{\newline}}\relax adding redefinition of \\ caused stack overflow. Note it disallows hyphenation except at \-.
\pdef\fakescaps#1{{\gmu@scapss#1\@@nil}}
\let\fakescapscore\gmu@scalematchX
```

Experimente z akcentami patrz no3.tex.

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

wg \zf@calc@scale pakietu fontspec.

```
\@ifpackageloaded{fontspec}{%
  \def\gmu@scalar{1.0}%
  \def\zf@scale{}%
  \def\gmu@scalematchX{%
    \begingroup
      \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 synthesized 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
\fakesc@extrascale

```
\def\fakescextrascale#1{\def\fakesc@extrascale{#1}}
```

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}
```

napapierki

```
\newenvironment*{napapierki}{%
  \par\global\napapierkicore}{%
  \par\dimen\z@=\baselineskip
  \global\baselineskip=\dimen\z@}% so that you can use \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 3649 \newenvironment*{luzniej}{\par\luzniejcore}{\par}
```

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

```
luzniej* 3654 \newenvironment*{luzniej*}[1][1]{%
         3655   \multiply\tolerance␣by␣2\relax
         3656   \everypar{\looseness=#1\relax}}{\par}
```

```
\nawj 3658 \newcommand*\nawj{\kerno,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). It begins with \ifhmode to be usable also at the beginning of a line as the mark of a dialogue.

```
         3666 \ifdefined\XeTeXversion
\pauza@skipcore 3667 \def\pauza@skipcore{\hskipo.2em␣pluso.1em\relax
   \pauzacore 3668   \pauzacore\hskip.2em␣pluso.1em\relax\ignorespaces}%
\ppauza@skipcore 3670 \def\ppauza@skipcore{\unskip\penalty10000\hskipo.2em␣pluso.1em%
                       \relax
         3671               -\hskip.2em␣pluso.1em\ignorespaces}
         3673 \AtBeginDocument{% to be independent of moment of loading of polski.
      \- 3674   \pdef\-{%
         3675     \ifhmode
         3676       \unskip\penalty10000
         3677       \afterfi{%
         3678         \@ifnextspace{\pauza@skipcore}%
         3679         {\@ifnextMac\pauza@skipcore{%
         3680             \pauzacore\penalty\hyphenpenalty\hskip\z@}}}%
         3681       \else
```

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

```
         3685       \leavevmode\pauzacore\penalty10000\hskipo,5em\ignorespaces
         3686     \fi}%
```

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

```
   \pauza 3689   \pdef\pauza{%
         3690     \ifhmode
         3691       \unskip\penalty10000
         3692       \hskipo.2em␣pluso.1em\relax
         3693       \pauzacore\hskip.2em␣pluso.1em\relax\ignorespaces%
         3694     \else
         3695       \pauzadial
         3696     \fi}%
```

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

```
\pauzadial 3701   \pdef\pauzadial{%
         3702     \leavevmode\pauzacore\penalty10000\hskipo,5em\ignorespaces}
```

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

```
  \lpauza 3706   \pdef\lpauza{%
         3707     \pauzacore\hskip.2em␣pluso.1em\ignorespaces}%
```

We define \ppauza as an en dash surrounded with thin stretchable spaces and sticking to the upper line or bare but discretionary depending on the next token being space$_1$0. 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…

```
\-    \pdef\-{%
        \ifvmode␣␣␣␣\PackageError{gmutils}{%
          command␣\bslash␣ppauza␣(en␣dash)␣not␣intended␣for␣vmode.}{%
          Use␣\bslash␣ppauza␣(en␣dash)␣only␣in␣number␣and␣numeral␣
               ranges.}%
        \else
          \afterfi{%
            \@ifnextspace{\ppauza@skipcore}{%
              \@ifnextMac\ppauza@skipcore{\unskip\discretionary{-}{%
                   -}{-}}}%
          }%
        \fi
      }%
\ppauza  \pdef\ppauza{%
        \ifvmode␣␣␣␣\PackageError{gmutils}{%
          command␣\bslash␣ppauza␣(en␣dash)␣not␣intended␣for␣vmode.}{%
          Use␣\bslash␣ppauza␣(en␣dash)␣only␣in␣number␣and␣numeral␣
               ranges.}%
        \else
          \unskip\discretionary{-}{-}{-}%
        \fi}%
\emdash   \def\emdash{\char`\—}
}% of at begin document

\longpauza  \def\longpauza{\def\pauzacore{—}}
\pauzacore  \longpauza
\shortpauza  \def\shortpauza{%
\pauzacore    \def\pauzacore{-\kern,23em\relax\llap{-}}}
\fi% of if XeTeX.
```

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

```
\ifdefined\XeTeXversion
\foone{\catcode`—\active␣\catcode`-\active␣\catcode`-\active}{%
\adashes  \def\adashes{\AtBeginDocument\adashes}% because \pauza is defined at
               begin document.
\adashes  \AtBeginDocument{\def\adashes{%
            \catcode`—\active␣\let—\—%
            \catcode`-\active␣\let-\-%
            \addtomacro\dospecials{\do\-\do\—}%
            \addtomacro\@sanitize{\@makeother\-\@makeother\—}%
            \addtomacro\gmu@septify{\do\-13\do\—13\relax}%
}}}
\else
\relaxen\adashes
\fi
```

The hyphen shouldn't be active IMO because it's used in TEX control such as \hskip-2pt. Therefore we provide the \ahyphen declaration reluctanly, because sometimes we need

it and always use it with caution. Note that my active hyphen in vertical and math modes expands to $-_{12}$.

```
\gmu@dywiz 3771 \def\gmu@dywiz{\ifmmode-\else
           3772   \ifvmode-\else\afterfifi\dywiz\fi\fi}%
           3774 \foone{\catcode`-\active}{%
  \ahyphen 3775   \def\ahyphen{\let-\gmu@dywiz\catcode`\-\active}}
```

To get current time. Works in ε-TEXs, icluding XETEX. `\czas` typesets 17.11 and `\czas[:]` typesets 17:11.

```
    \czas 3780 \newcommand*\czas[1][.]{%
           3781   \the\numexpr(\time-30)/60\relax#1%
           3782   \@tempcnta=\numexpr\time-(\time-30)/60*60\relax
           3783   \ifnum\@tempcnta<10␣0\fi\the\@tempcnta}
           3786 \@ifXeTeX{%
\textbullet 3787   \pdef\textbullet{%
           3790     \iffontchar\font"2022␣\char"2022␣\else\ensuremath{\bullet}%
                      \fi}%
\glyphname 3792   \pprovide\glyphname#1{%
           3794     \XeTeXglyph␣\numexpr\XeTeXglyphindex␣"#1"\relax\relax}% since XETEX
                      … \numexpr is redundant.
           3796 }
\textbullet 3797 {\def\textbullet{\ensuremath{\bullet}}}
  tytulowa 3799 \newenvironment*{tytulowa}{\newpage}{\par\thispagestyle{empty}%
                 \newpage}
```

To typeset peoples' names on page 4 (the editorial page):

```
\nazwired 3802 \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 3816 \newcommand*\polskadata{%
 \gmu@datef 3817   \def\gmu@datef##1-##2-##3##4,##5\gmu@datef{%
           3818     \ifx\relax##2\relax##3##4%
           3819     \else
           3820       \ifnum##3\@firstofmany##40\@@nil=0\relax
           3821       \else
           3822         \ifnum0##3=0\relax
           3823         \else##3%
           3824         \fi##4%
           3825       \fi
           3826     \ifcase##2\relax\or\␣stycznia\or\␣lutego%
           3827     \or\␣marca\or\␣kwietnia\or\␣maja\or\␣czerwca\or\␣lipca\or\␣
                      sierpnia%
           3828     \or\␣września\or\␣października\or\␣listopada\or\␣grudnia\else
           3829     {}%
           3830     \fi
           3831     \if\relax##1\relax\else\␣\fi␣##1%
```

```
    \fi
    \gmu@datecomma{##5}}% of \gmu@datef.
\gmu@datefsl  \def\gmu@datefsl##1/##2/##3##4,##5\gmu@datefsl{%
    \if\relax##2\relax##3##4%
    \else
    \ifnum##3\@firstofmany##4o\@@nil=o\relax
    \else
    \ifnumo##3=o\relax
    \else##3%
    \fi##4%
    \fi
    \ifcase##2\relax\or␣stycznia\or␣lutego%
    \or␣marca\or␣kwietnia\or␣maja\or␣czerwca\or␣lipca\or␣
        sierpnia%
    \or␣września\or␣października\or␣listopada\or␣grudnia\else
    {}%
    \fi
    \if\relax##1\relax\else␣\fi␣##1%
    \fi
    \gmu@datecomma{##5}}%
}% of \polskadata

\polskadata
```

For documentation in English:

```
\englishdate  \newcommand*\englishdate{%
  \gmu@datef    \def\gmu@datef##1-##2-##3##4,##5\gmu@datef{%
    \if\relax##2\relax##3##4%
    \else
      \ifcase##2\relax\or␣January\or␣February%
        \or␣March\or␣April\or␣May\or␣June\or␣July\or␣August%
        \or␣September\or␣October\or␣November\or␣December\else
        {}%
      \fi
    \ifnum##3\@firstofmany##4o\@@nil=o\relax
      \else
        \␣%
        \ifnumo##3=o\relax
        \else##3%
        \fi##4%
        \ifcase##3\@firstofmany##4\relax\@@nil\relax\or␣st\or␣nd%
            \or␣rd\else␣th\fi
      \fi
      \ifx\relax##1\relax\else,\␣\fi␣##1%
    \fi
  \gmu@datecomma{##5}}%
\gmu@datefsl  \def\gmu@datefsl##1/##2/##3##4,##5\gmu@datefsl{%
    \if\relax##2\relax##3##4%
    \else
      \ifcase##2\relax\or␣January\or␣February%
        \or␣March\or␣April\or␣May\or␣June\or␣July\or␣August%
        \or␣September\or␣October\or␣November\or␣December\else
        {}%
```

```
      \fi
      \ifnum##3\@firstofmany##4o\@@nil=o\relax
      \else
        \␣%
        \ifnumo##3=o\relax
        \else##3%
        \fi##4%
        \ifcase##3\@firstofmany##4\relax\@@nil\relax\or␣st\or␣nd%
             \or␣rd\else␣th\fi
      \fi
      \if\relax##1\relax\else,\␣\fi␣##1%
    \fi
    \gmu@datecomma{##5}}%
}
```

\gmu@datecomma

```
\def\gmu@datecomma#1{% sometimes we want to typeset something like '11 września, czwartek' so we add handling for comma in the \ldate's argument.
  \ifx\gmu@datecomma#1\gmu@datecomma\else
    ,\gmu@stripcomma#1%
  \fi
}% of \gmu@datecomma
```

\gmu@stripcomma

```
\def\gmu@stripcomma#1,{#1}
```

\ifgmu@dash

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

\gmu@ifnodash \gmu@tempa

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

\gmu@testdash

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

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

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

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

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

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

```
\ifdate 3957 \newif\ifdate
\bidate 3959 \pdef\bidate#1{%
        3960   \ifdate\gmu@testdash#1%
        3961     \ifgmu@dash
        3962       \gmu@datef#1,\gmu@datef
        3963     \else
        3964       \gmu@datefsl#1,\gmu@datefsl
        3965     \fi\fi}
\linedate 3967 \pdef\linedate{\@ifstar\linedate@@\linedate@}
\linedate@@ 3968 \pdef\linedate@@#1{\linedate@{--}{}{}#1}}
\linedate@ 3969 \pdef\linedate@#1{\par\ifdate\addvspace{\dateskip}%
        3970   \date@line{\footnotesize\itshape␣\bidate{#1}}%
        3971   \nopagebreak\else%%\ifnum\arabic{dateinsection}>0\dekbigskip\fi
        3972   \addvspace{\bigskipamount}%
        3973   \fi}% end of \linedate.
        3975 \let\dateskip\medskipamount
\rdate 3983 \pdef\rdate{\let\date@line\rightline␣\linedate}
\ldate 3984 \pdef\ldate{%
\date@line 3986   \def\date@line##1{\par{\raggedright##1\par}}%
        3987   \linedate}
\runindate 3988 \newcommand*\runindate[1]{%
        3989   \paragraph{\footnotesize\itshape␣\gmu@datef#1\gmu@datef}%
        3990   \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.

```
        3993 \let\thedate\ldate
\zwrobcy 3996 \pdef\zwrobcy#1{\emph{#1}}␣% ostinato, allegro con moto, garden party etc., także kompliment
\tytul 3999 \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 4005 \newenvironment{maszynopis}[1][]{#1\ttfamily
        4006   \hyphenchar\font=45\relax% this assignment is global for the font.
        4007   \@tempskipa=\glueexpr\rightskip+\leftskip\relax
        4008   \ifdim\gluestretch\@tempskipa=\z@
        4009   \tolerance900
```

it worked well with tolerance = 900.

```
        4011   \advance\rightskip␣by\z@␣pluso,5em\relax\fi
        4012   \fontdimen3\font=\z@% we forbid stretching spaces…
%␣␣\fontdimen4\font=\z@ but allow shrinking them.
        4014   \hyphenpenalty0␣% not to make TEX nervous: in a typewriting this marvellous algorithm of hyphenation should be turned off and every line broken at the last allowable point.
        4017   \StoreMacro\pauzacore
\pauzacore 4018   \def\pauzacore{-\rlap{\kern-0,3em-}-}%
        4019 }{\par}
\justified 4023 \newcommand*\justified{%
```

```
    \leftskip=1\leftskip% to preserve the natural length and discard stretch and
        shrink.
    \rightskip=1\rightskip
    \parfillskip=1\parfillskip
    \advance\parfillskip␣by␣osp␣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`
```
\newcommand*\fullpar{%
  \hunskip
  \bgroup\parfillskip\z@skip\par\egroup}
```

To conform Polish recommendation for typesetting saying 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 interword 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.
  \edef\twopar{%
    \hunskip% it's \protected, remember?
    \bgroup
    \parfillskip=\the\glueexpr
    \dimexpr\textwidth-2\parindent\relax
    minus\dimexpr\textwidth-2\parindent\relax
    \relax% to delimit \glueexpr.
    \relax% to delimit the assignment.
    \par\egroup
  }% of \gmu@twoparfill
}% of \twoparinit.
```

For dati under poems.

`\wherncore`
```
\newcommand\wherncore[1]{%
    \rightline{%
    \parbox{0,7666\textwidth}{
      \leftskiposp␣plus␣\textwidth
      \parfillskiposp\relax
      \let\\\linebreak
      \footnotesize␣#1}}}
```

`\whern`
```
\def\whern{%
  \@ifstar\wherncore{\vskip\whernskip\wherncore}}
```

`\whernskip`
```
\newskip\whernskip
\whernskip2\baselineskip␣minus␣2\baselineskip\relax
```

`\whernup`
```
\newcommand\whernup[1]{\par\wherncore{#1}}
```

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
```
\def\leftslanting{%
```
\litshape
```
  \pdef\litshape{%
    \itshape
    \@tempdima=-2\fontdimen1\font
    \advance\leftskip␣by\strip@pt\fontdimen1\font␣ex␣% to assure al 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.
    \edef\gmu@tempa{%
      \@nx\addfontfeature{FakeSlant=\strip@pt\@tempdima}}% when
          not \edefed, it caused an error, which is perfectly understandable.
    \gmu@tempa}%
```
\textlit
```
  \pdef\textlit##1{%
    {\litshape##1}}%
```
\lit
```
  \pdef\lit{\rm\litshape}%
```
\lsl
```
  \pdef\lsl{{\it
      \@tempdima=-\fontdimen1\font
      \xdef\gmu@tempa{%
        \@nx\addfontfeature{RawFeature={slant=\strip@pt%
            \@tempdima}}}}%
    \rm␣␣% Note in this declaration we left-slant the basic Roman font not the it-
        shape of the current family.
    \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
```
  \pdef\em{%
    \ifdim\fontdimen1\font=\z@␣␣\itshape
    \else
      \ifdim\fontdimen1\font>\z@␣\litshape
      \else␣\upshape
      \fi
    \fi}%
  \pdef\emph##1{%
    {\em##1}}%
}% of \leftslanting.
```

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{%
  \ifx\relax#5\relax\afterfi{#6}\else\afterfi{#7}\fi}
\gmu@thou@putter \def\gmu@thou@putter#1#2{% we are sure to have at least five tokens before the
      guardian \gmu@thou@putter.
  \advance\@tempcnta\@ne
  \@tempcntb\@tempcnta
  \divide\@tempcntb3\relax
  \@tempcnta=\numexpr\@tempcnta-\@tempcntb*3
  \edef\gmu@thou@put{\gmu@thou@put#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 \def\shortthousep{%
    \thous \pdef\thous{%
    \ifmmode\hbox\bgroup\@gmu@mmhboxtrue\fi
    \afterassignment\thous@inner
    \@tempcnta=}%
\thous@inner \def\thous@inner{%
    \ifnum\@tempcnta<0␣$-$%
      \@tempcnta=-\@tempcnta
    \fi
    \xathousep\@tempcnta
    \if@gmu@mmhbox\egroup
```

```
      \else\afterfi{\@ifnextcat␣a\space{}}%
      \fi}%
}% 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
```
\def\urladdstar{%
  \AtBeginDocument{%
    \@ifpackageloaded{hyperref}{%
      \StoreMacro\url
\url  \def\url{\@ifstar{\nolinkurl}{\storedcsname{url}}}%
    }{}}}
\@onlypreamble\urladdstar
\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, 1069
- 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, 4248

v0.75
- \@ifnextcat:
  - \let for #1 changed to \def to allow things like \noexpand~ , 862
- \@ifnextif:
  - \let for #1 changed to \def to allow things like \noexpand~ , 898
- \@ifnif:
  - added, 923

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, 4248
- \freeze@actives:
  - added, 3266

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, 4248

v0.78
- General:
  - \@namelet renamed to \n@melet to solve a conflict with the beamer class. The package contents regrouped, 4248

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, 1663

v0.80
- General:
  - CheckSum 1689, 0
- \hfillneg:
  - added, 3185

v0.81
- \dekfraccslash:
  - moved here from pmlectionis.cls, 3445
- \ifSecondClass:
  - moved here from pmlectionis.cls, 3417

v0.82
\ikern:
added, 3453
v0.83
\~:
postponed to \begin{document} to avoid overwriting by a text command and made sensible to a subsequent /, 3138
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, 4248
v0.86
\texttilde:
renamed from \~ since the latter is one of LaTeX accents, 3146
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, 4248
v0.88
General:
CheckSum 4040, 0
\RestoreEnvironment:
added, 1602
\storedcsname:
added, 1593
\StoreEnvironment:
added, 1598
v0.89
General:
removed obsolete adjustment of pgf for XeTeX, 4248
v0.90
General:
CheckSum 4035, 0
\XeTeXthree:
adjusted to the redefinition of \verb in xltxtra 2008/07/29, 2489
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, 4248
v0.92
\@checkend:
shortened thanks to \@ifenvir, 1101
\@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, 281
\@ifenvir:
added, 1093
General:
CheckSum 4133, 0
v0.93
\@nameedef:
added, 229
General:
A couple of \DeclareRobustCommand* changed to \pdef, 4248
CheckSum 4140, 0
CheckSum 4501, 0
The numerical macros commented out as obsolete and never really used, 4248
\ampulexdef:
added, 755
\em:
added, 4145, 4154
\gmu@RPfor:
renamed from \gmu@RPif and #3 changed from a csname to cs, 3328
\litshape:
copied here from E. Szarzyński's *The Letters*, 4108
\lsl:
copied here from E. Szarzyński's *The Letters*, 4129
\nocite:
a bug fixed: with natbib an 'extra }' error. Now it fixes only the standard version of \nocite, 1701
\pdef:
added, 187
\pprovide:
added, 216
\provide:
added, 201
\textlit:
added, 4123
\thousep:
added, 4160
v0.94
\@xau:
added, 183
General:

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.