The teubner package
Extensions for Greek philology[*]

Claudio Beccari[†]

Contents

Abstract

This extension package complements the greek option of the babel package so as to enable the use of the Lipsian[1] fonts and to use several macros for inserting special annotations in the written text, as well as to typeset verses with special layout. Metric sequences may be defined and typeset by means of a companion font `gmtr????` that follows the same conventions as the CB fonts that are normally used when the babel `greek` option is in force.

Examples and lists of commands are available in the file `teubner-doc.pdf` which, as a regular pdf file, embeds all the necessary fonts and may be read on screen as well as printed on paper; beware, though, that the PostScript fonts that are being used in `teubner-doc.pdf` are not installed by default if a basic installation is sought.

This version 4.0 tries to adapt to the new handling of the Greek language by babel, since it now this language script is based on the LICR (LaTeX Internal Character Representation), and source files, now can include Greek glyphs, besides the usual transliteration with Latin letters.

1 Introduction

Philologists in general have the necessity of using special alphabets and several special symbols in order to mark up their texts and to typeset them in a special way. Greek philology makes no exception, therefore I prepared this file and some extra fonts in order to complement what is already available with the `greek` option of the babel package.

I must warmly thank Paolo Ciacchi of the University of Trieste who invited me in this "adventure", since I know nothing about philology; he assisted me with all his competence, so that I could learn so many new things and I could appreciate the world of philologists.

[*]This file has version number v.4.2, last revised on 2013/12/31.

[†]claudio dot beccari at gmail dot com

[1]What here are called Lipsian fonts are a family of fonts that in Greece are called "Lipsiakos"; they are similar to the ones that were being used in the Teubner Printing Company of Lipsia from mid XIX century on.

Paolo Ciacchi's "invitation" arrived when I was almost finished with the design of the Lipsian font family; I was working on this new typeface after a kind request by Dimitri Filippou, with whom I already collaborated for other questions related to Greek typesetting. I warmly thank also Dimitri Filippou for the patience with which he revised every single glyph of the new typeface. Paolo Ciacchi added his constructive criticism to the typeface, especially for what concerns diacritical marks. At the end I think that the new typeface turned out pretty well thanks to both my friends.

The Lipsian font, also called Leipzig, or Lipsiakos in Greece, is one of the oblique fonts that used to be employed by the typesetters working in the German city of Leipzig, among which the Teubner Printing Company. This Company's classical works of ancient Greek poetry are considered among the best ever published. The name of this file and this extension package is in honour of that printing company.

This package documentation does not contain any example written in Greek, because when you process this file it is very likely that you don't have the suitable Greek fonts and you must still download all or some of them. Therefore a companion file `teubner-doc.pdf` is included in this bundle where most, if not all, new commands are documented and suitably shown.

This package contains new environments and new commands; it presumes the user invokes it after declaring the `greek` option to the babel package; should he forget, this package will complain. But once the `greek` option is properly declared, this package verifies that the `polutonikogreek` dialect (deprecated) is selected, or that the `polutoniko` attribute is set. This choice depends on the particular version of the babel package, but should not concern the user; switching back and forth between classical Greek and some modern western language is performed in a transparent way; possibly there might be some problem switching from classical to modern spelling in Greek itself, but since in modern spelling the multiplicity of Greek diacritical marks is not forbidden, it's the author choice to select classical or modern words, Lipsian or Didot fonts, polytonic or monotonic accentuation. The worst it can happen is that babel might use just one hyphenation pattern set, so that in one of the two Greek versions some words might turn out with the wrong hyphens.

The CB Greek fonts, which have been available for some years now on CTAN in the directory `/tex-archive/fonts/greek/cb` have been completed with the new files for the Lipsian fonts, and the metric symbols font `gmtr????.*`; the latter does not need a formal font definition file, because the necessary definitions are included in this package. All fonts are available also as Type 1 scalable fonts. In general, recent distributions of the TEX system already contain the necessary configuration to use the Type 1 font in one size, 10 pt, but, thanks to scaling, these can be used at any size; this version of teubner is compatible with this reduced set. If optical sizes are desired for a more professional typesetting, the CTAN archives contain also the cbgreek-full package, which includes also all the Type 1 fonts at the various standard (EC) sizes, plus other facilities that allow to use the CB fonts also in conjunction with the Latin Modern ones. Complete installations of the TEX system include the full CB fonts installation.

The CB Greek fonts allow to input Greek text with a Latin keyboard and by employing the prefix notation; with a Greek keyboard and file `iso-8859-7.def` it is possible to directly input Greek text with the monotonic spelling; if polytonic spelling is required I fear that the above file is of little help and that a Latin keyboard does the job without an excessive burden. The recent modifications to the babel package and of the Greek language description file allow to enter also polytonic Greek text, keyboard permitting, with no effort; it is necessary to use the `utf8` input encoding. It's important to notice the Apple computers have available a virtual keyboard (called keyboard viewer) that is operable with the mouse and, just by selecting the *Greek Polytonic* keyboard driver, the author can enter Greek text directly in the source file. Computer with a touch screen virtual keyboard allow to switch from on's the national to the Greek keyboard just by a single sweep of the finger. Some platforms have the possibility of switching keyboard but they don't show the new keyboard layout on the screen, but the user can generally build a personal table that describes the correspondence betwwen the key board and the physical keyboard layout.

Nevertheless there is a little point to observe; Lipsian fonts are very nice but show some kerning errors with more evidence than the traditional Didot Greek fonts. With the prefix notation in force, kerning programs may result disabled and some diphthongs and some consonant-vowel combinations appear poorly matched when the second letter caries any diacritical mark. In order to avoid this "feature", the accented vowels may be input by means of macros, that directly translate to the accented glyph, rather than invoking the ligature programs that are implied by the prefix notation; reading a Greek text on the screen while editing the input `.tex` file when a Latin keyboard and such macros are used may be very strange, but authors get used to it, and agree that the effort is worth the result.

2 Environments

I apologise if I chose Italian names for verse environments; I wanted to use names very different from the corresponding English ones, but at the same time easily recognisable; after all *versi* is the plural of *verso* and therefore is the exact Italian translation of *verses*. If you feel more comfortable with Latin, the alias environment names in Latin, `versus`, `Versus`, and `VERSUS`, are also available.

`versi` `\verso`

The environment `versi` (`versus`) is used to typeset verses in line, without an implicit end of line at the end of each verse; a vertical bar with a number on top of it marks the verse limit while allowing a numeric reference to a specific verse; the opening environment statement requires a string, a short text, in order to indent the verse lines the amount of this string width; the syntax is the following

```
\begin{versi}{⟨string⟩}
⟨verse⟩\verso[⟨starting number⟩]⟨verse⟩\verso
⟨verse⟩\verso⟨verse⟩...
\end{versi}
```

where, of course, ⟨*starting number*⟩ is required only for the first instance of `\verso` or when numbering must be restarted, for example after an ellipsis.

`Versi` The environment `Versi` (`Versus`) is similar to the standard LaTeX environment `verse`, except verse lines are numbered on multiples of 5; the opening statement requires the ⟨*starting number*⟩ as an optional argument; if this optional argument is not specified, the starting number is assumed to be 1.

```
\begin{Versi}[⟨starting number⟩]
⟨verse⟩\\⟨*⟩[⟨vertical space⟩]
⟨verse⟩\\
...
\end{Versi}
```

`VERSI` `\SubVerso` `\NoSubVerso` The environment `VERSI` (`VERSUS`) allows for two verse enumerations; the main enumeration is identical to the one performed by the previous environment `Versi`, while the secondary enumeration is in smaller digits and normally numbers consecutive verses, except that it can be turned on and off; the verses that lack the secondary enumeration are indented by moving them to the right.

```
\begin{VERSI}[⟨starting principal number⟩]
⟨verse⟩\\⟨*⟩[⟨vertical space⟩]
\SubVerso[⟨starting secondary number⟩]
⟨verse⟩\\⟨*⟩[⟨vertical space⟩]
...
\NoSubVerso
⟨verse⟩\\⟨*⟩[⟨vertical space⟩]
...
\end{VERSI}
```

where if ⟨*starting principal number*⟩ is missing, 1 is assumed, while if ⟨*starting secondary number*⟩ is missing, the enumeration is continued from the next available integer. Of course ⟨*starting secondary number*⟩ is used again when the secondary enumeration must be restarted; there are no means to restart the principal enumeration.

`bracedmetrics` The previous environments accept ⟨*verses*⟩ in any language and in any alphabet, the one that is in force before opening the environment; the language and, even less, the alphabet cannot be globally changed within the above environments; if such a change is performed, it is valid only for one verse, or for the remaining fraction of the verse after the language or font change. This means, among the other things, that if the default "alphabet" is the one that shows the metric symbols, the above environments may be used to display "metric verses", that is the pattern of long, short or ancipital symbols, together with any other metric symbol so as to display the metrics without disturbing the written text; when doing this metric typesetting, it may happen that some verse patterns exhibit some variants; in this case the `bracedmetrics` environment comes handy, because it can display such variants in separate lines but grouped with a large right brace; some commands allow to roughly align these variants, so as to allow to nest several such environments as if they were single blocks of metric symbols. The argument of the opening statement specifies the width of the block so as to align properly all the symbols even in nested environments.

```
\begin{bracedmetrics}{⟨length⟩}
⟨metric pattern⟩\\
⟨metric pattern⟩\\
...
\end{bracedmetrics}
```

\verseskip \Hfill Within the ⟨*metric pattern*⟩ it is possible to flush right the symbols by prefixing the whole string with a `\Hfill` command; the ⟨*length*⟩ may be specified as an integer multiple of a "long" symbol by means of

```
\verseskip{⟨number⟩}
```

The macro `\verseskip` can be used also within ⟨*metric pattern*⟩ in order to space out metric symbols.

3 Commands and symbols

This package defines a lot of commands for inserting special signs in the middle of regular text, for marking zeugmas and synizeses, for putting unusual accents on any symbol, for inserting special "parentheses" that are used by philologists for marking blocks of letters or blocks of text. I suggest that the user consults the documentation file `teubner-doc.pdf` for a complete list of commands and symbols.

\newmetrics Here it might be useful to describe a command for defining metric sequences, so as to shorten the definition of metric verses; this new command is `\newmetrics` and may be used for the definition of new commands whose name *may start with one digit*: precisely this digit may be one of 2, 3, 4. Even if LaTeX does not allow macros to contain both digits and letters, other service macros have been defined so as to handle these special control sequences even if they start with *one* digit strictly lower than 5. The syntax is:

```
\newmetrics{⟨control sequence⟩}{⟨definition⟩}
```

where ⟨*definition*⟩ consists in general of a sequence of metric commands such as `\longa`, `\brevis`, `\anceps`, etc.

4 Acknowledgements

I must thank with gratitude Paolo Ciacchi that urged me to prepare this extension file in order to help him typeset his master thesis of philological type in classical Greek.

I am pleased to thank Günter Milde who wrote a definition file for accessing the LGR encoded fonts in order to fetch the accented glyphs; I kindly gave me permission to use his macros, that I adapted to the conventions used within this file. These macros were saved into the definition file `LGRaccent-glyph.def`, so that it could be used also without the `teubner` package, for example for typesetting without setting the *polytoniko* language attribute. But since he became the

maintainer of the Greek language support for the babel package, he extended this support to the point that the extended macros are already part of the new Greek support files.

I got some ideas also from a paper that Werner Lemberg published on Eutypon, the magazine of the Hellenic Friends of TEX, where he discussed in a constructive critical way the problems connected with the LGR encoded fonts and the Unicode encoding.

Now the new Greek support for the babel package, thanks to Günter Milde, includes also the support for Unicode input in the source file, in spite of using LGR encoded output fonts. The actual support is partially useful also with XeLaTeX and LuaLaTeX.

5 Code

5.1 Preliminaries

In order to use the PostScript pfb fonts (CM, EC, and CB) it is necessary to know if we are dealing with LATEX or pdfLATEX; this was necessary because apparently the pfb math scalable fonts derived from the METAFONT counterparts do not have exactly the same effective dimensions; this is why the "zeugma" and the "synizesis" signs have to be corrected when the pfb fonts are used; with these, in facts, the black leader that joins the curved extremities appeared a little too fat and did not join exactly the left mark. Recently, apparently, the fonts have been corrected and this trick is not necessary any more. Nevertheless we define a new boolean that copes with the fact that at least since 2007 the TEX engine is pdftex even when DVI output is sought; the package iftex creates three `\if`s that allow to diagnose if the typesetting engine is pdftex in PDF mode, XeTeX or Luatex; since at the moment this package teubner is compatible only with pdftex in PDF mode, we equate the `\ifPDF` switch (defined in previous versions of teubner) with the switch defined by package iftex:

```
\RequirePackage{iftex}
\let\ifPDF\ifPDFTeX
```

When `teubner.sty` is input the language Greek must have been already defined; otherwise an error message is issued and processing is terminated.

```
\ifx\captionsgreek\undefined
\PackageError{teubner}{Greek language unknown!\MessageBreak
I am not going to use Lipsian fonts and Scholars' signs\MessageBreak
if Greek is unknown.\MessageBreak
Use the babel package greek option.\MessageBreak
Type X <return> to exit.}%
{Type X <return> to exit.}
\fi
```

If this test is passed, this means that not only the greek option to the babel package is set, but also that all the babel machinery is available.

Since `teubner.sty` accepts some options it is necessary to provide their definitions; in particular the `\or` control sequence conflicts with the `\or` primitive command used within the syntax of `\ifcase`[2]; `\oR` is a little exception since all the other accent-vowel macros contain only lowercase letters; `og` is another exception, and the accent macros have to be used; `\og` collided with the Frenc command for inserting the opening guillemets (see below). The point is that accent vowel sequences that directly access the accented glyph are made up as such:

\⟨*base character*⟩⟨*first diacritic*⟩⟨*second diacritic*⟩⟨*third diacritic*⟩
where
⟨*first diacritic*⟩ is `d` or `r` or `s` for diaeresis, rough or smooth breadth
⟨*second diacritic*⟩ is `c` or `a` or `g` for circumflex or acute or grave
⟨*third diacritic*⟩ is `i` for iota subscript or adscript

Evidently none of the diacritical marks is compulsory, but at least one must be present; if more than one is present it must be given in that sequence. Since `\oR` means omicron with rough breath, it is not very important that it is declared with the standard sequence `<o` or with `\oR` or with `\<o`, because it never falls after another letter, so that it never breaks any ligature or kerning command. The command is there just for completeness. More or less the same is true with the `\og` sequence; it fails to work correctly when the main language is French and the `french` option to the `babel` package is in force; matter of facts, `\og` conflicts with the homonymous command defined by that option to mean "ouvrir guillemets"; therefore it's necessary to use either the plain sequence `‘o` or the extended accent macro `\‘o`.

At the same time since all accent combinations are defined as "text commands", in LaTeX jargon, when their commands are followed by a vowel (or 'r') they define a "text symbol" i.e. they fetch directly the glyph of the accented character; therefore `<o`, `\oR` and `\r{o}` are all equivalent (at the beginning of a word where omicron with rough breath is the only place where you might find it). See more on this point in the following sections.

These glyph name macros are not defined by default, because the `GlyphNames` boolean variable is false by default; you can specify the option *GlyphNames* for activating these macro names. In any case the same result is more comfortably obtained by using the extended accent macros whose behaviour is specified by the new `babel`greek settings.

Another unusual option is set up for being used with non standard TeX system fonts; we have noticed that the Lipsian fonts appear a little too light when used together with the Times or the Palatino fonts; probably this is true also with other PostScript fonts. In this case the user might specify the option `boldLipsian` and the Lipsian fonts used in medium series will be substituted with those of the semibold one.

At the same time, from July 2005 and for a few years, the full collection of the complete size set of the CB fonts has not been available any more *as the default TeX system set up*; only the 10pt size were available unless the `cbgreek-full` font

[2]With version 2002/07/18 v.1.0d this has been eliminated; the option remains for compatibility with older versions, but the only legal command is now `\oR`.

package was loaded; for this reason a new option was needed in order to instruct `teubner.sty` to use the specific file `type1ec.sty` dated at least 2002/09/07, so as to scale up or down all the EC and CB fonts from an original 10 pt size. In order that the `10pt` plays its role correctly, it was convenient, if not compulsory, to require first the `babel` package, then the `teubner` one with the option `10pt`, then all other packages required for a specific document, in particular the `fontenc` one if the `T1` encoding is requested. All this is maintained for backward compatibility, and should not be necessary any more; now the complete set of the CB fonts gets installed with every full installation of the TeX system, but with partial or basic installations, such option might turn out to be useful even now.

This kludge is necessary for all fonts that have description files that use the `\EC@family` command for describing the available shapes and sizes; in practice this happens only with the EC fonts, even when the `cm-super` scalable implementation is used. For using the Latin Modern fonts (LM) new specific font description files for the CB fonts are part of the babel package, so this problem does not exist; when using other fonts, such as the TX, PX, ZE, . . . , other kludges are necessary, because their font family names are different from those normally used with TeX: `cmr` for serifed fonts, `cmss` for sans serif ones, and `cmtt` for monospaced ones.[3]

Notice that when using, for example, the TX fonts and no kludge is available, the CB fonts are loaded only as the "error font", since the TX fonts have different family names than the CB ones; in many cases this might pass un-noticed, but if real Greek text in different shapes and series has to be typeset, the unaware typesetter might get crazy trying to force shape and series changes in the Greek text; it would not be impossible, but it would be very, very boring. In any case see in the next sections the implemented kludges in order to run successful compilations also with non standard TeX system fonts.

```
\newif\ifor\orfalse % Compatibility with older versions
\DeclareOption{or}{\relax}
\newif\ifboldLipsian \boldLipsianfalse
\DeclareOption{boldLipsian}{\boldLipsiantrue}
\newif\ifonesizetypeone
\DeclareOption{10pt}{\onesizetypeonetrue}
\newif\ifGlyphNames \GlyphNamesfalse
\DeclareOption{NoGlyphNames}{\GlyphNamesfalse}
\DeclareOption{GlyphNames}{\GlyphNamestrue}
\ProcessOptions*
```

In the next sections we frequently use the acronym for the Greek font encoding; we hope it will eventually become `GR` or, following the actual 256 glyph font

[3]Even the Latin Modern fonts have different family names, but, due to their importance, specific font description files have been added to the babel package. The LM fonts are more comfortable than the EC ones, when scalable fonts are to be used, because they are continuously scalable and download into the produced files less font files than the EC o cm-super ones. Unless specifically requested, the LM fonts should always be preferred to the EC and cm-super ones. When using the LM fonts, its better to use the full collection of the CB fonts, even if the CB font description files are compatible with the single 10 pt size.

encodings, `T7` or `X7`[4]. Meanwhile the acronym is `LGR`, so we'd better define a symbolic name, so that we can change the definitive name in just one place.

```
\def\GRencoding@name{LGR}
```

Now the default Olga Greek fonts, used for rendering the "Greek italic shape" are an alternative with the Lipsian ones.

If the `10pt` option was specified it is necessary to load also the package texttt-type1ec.sty. In any case we load also packages `graphicx` and `ifthen` that shall be useful for some commands.

```
\ifonesizetypeone
  \RequirePackage[10pt]{type1ec}[2002/09/07]
\fi
\RequirePackage{graphicx}
\RequirePackage{ifthen}
```

`\metricsfont` Similarly the metric symbol font is declared together with a command for selecting it:

```
\DeclareFontFamily{U}{mtr}{\hyphenchar\font\m@ne}
%\EC@family{U}{mtr}{m}{n}{gmtr}
\ifonesizetypeone
\DeclareFontShape{U}{mtr}{m}{n}{<-> gmtr1000}{}%
\else
\DeclareFontShape{U}{mtr}{m}{n}{%
      <-5.5>    gmtr0500     <5.5-6.5> gmtr0600
      <6.5-7.5> gmtr0700     <7.5-8.5> gmtr0800
      <8.5-9.5> gmtr0900     <9.5-11>  gmtr1000
      <11-15>   gmtr1200     <15->     gmtr1728}{}%
\fi
\DeclareFontShape{U}{mtr}{m}{it}{<->ssub*mtr/m/n}{}%
\DeclareFontShape{U}{mtr}{b}{it}{<->ssub*mtr/m/n}{}%
\DeclareFontShape{U}{mtr}{b}{n}{<->ssub*mtr/m/n}{}%
\newcommand*\metricsfont{\fontencoding{U}\fontfamily{mtr}\upshape}
```

Next we require the package for extensible math fonts; it might be strange to use extensible math fonts in Greek philology, but a certain glyph must be picked up from such fonts, with the assurance that it changes size together with the current font size.

```
\RequirePackage{exscale}
```

Some macros are necessary to switch languages; such macros must be independent (at least for now) from the particular babel version, whether it be version 3.6, 3.7, or 3.8; in the former the concept of "language attribute" is unknown, while the latter recognises varieties of the same language by the attribute setting. With babel version 3.9g things have further changed; the attribute to a language may

[4]Apparently `T7` has been reserved to define an encoding where the first 128 glyphs are the standard `OT1` encoded Latin fonts, and the second group, again of 128 glyphs, contains the Greek characters; therefore, since polytonic spelling requires more than 128 glyphs, the extended encoding `X7` will probably become the one applicable to the whole set of the CB fonts. Time passing by, the `X7` encoding name apparently has been used for some nordic language.

be appended to the language name with an interposed dot; for example for Greek it might be `greek.polutoniko`. Such macros, besides being as robust as possible, must provide the alphabet changes as required.

`\GreekName` During the language switching operations `\GreekName` distinguishes the dialect or the main language whose attribute gets set and, evidently, becomes effective when the main language `greek` is in force.

```
\ifx\languageattribute\undefined
 \def\GreekName{polutonikogreek}%
\else
 \languageattribute{greek}{polutoniko}\def\GreekName{greek}%
\fi
```

5.2 Compatibility with Latin fonts

`\previouslanguage`
`\previousencoding` The "default" language is defined as the "previous" language; similarly the "default" encoding is defined as the "previous" encoding; these are the language and the encoding in force when the document starts; this is why such macros are defined at the beginning of the document. At the same time we assure that if the CM (or EC) or the LM fonts are the default ones, nothing special is done, while if the default fonts are, say, the TX ones, they are correctly restored, but the CM families are used for the CB ones.

`\substitutefontfamily`
`\ifLipsian` The font macro `\substitutefontfamily` is already present in the babel kernel, but with version 3.9g it is deprecated, although maintained for backwards compatibility; it copes only with the standard families, series and shapes, therefore it does not consider the Lipsian shape and its series. I had to redefine it together with a new conditional macro in order to do the same job as the original one but taking into consideration also the Lipsian shape; the purpose of this macro is to write in the working directory a number of font description files that refer to the LGR Greek encoding, but have the names of the Latin font families; such font description files, simply substitute these non existent encoding-family series and shapes with the existing series and shapes of any other LGR encoded Greek font, in particular the CB ones. Things might change in the future, so as to use the package by Günter Milde `substitutefont` package (already present in the CTAN archive) or other solutions by the core of babel. Meanwhile we cope with what is available right now.

By issuing a command such as:

```
\ifFamily{pxr}{cmr}
```

an association is made with all the series and shapes of the Palatino serifed fonts to the corresponding CB serifed series and shapes; therefore when a language shift changes the default encoding from, say, `T1` to `LGR` the font family `LGR+pxr` is mapped to the font family `LGR+cmr` and everything is supposed to work fine; when another language change resets the encoding to `T1`, the original Latin script is used again. The redefined `\substitutefontfamily` macro is as such:

```
\newif\ifLipsian

\providecommand*\substitutefontfamily{}%
\renewcommand*\substitutefontfamily[3]{{%
  \edef\@tempA{#1#2.fd}%
  \lowercase\expandafter{\expandafter\def\expandafter\@tempA\expandafter{\@tempA}}%
  \expandafter\IfFileExists\expandafter{\@tempA}{}{%
  \immediate\openout15=\@tempA
  \typeout{Writing file #1#2.fd}
  \immediate\write15{%
    \string\ProvidesFile{#1#2.fd}^^J
    [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
     \space generated font description file]^^J
    \string\DeclareFontFamily{#1}{#2}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
    \string\DeclareFontShape{#1}{#2}{bx}{n}{<->ssub * #3/bx/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{bx}{it}{<->ssub * #3/bx/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{bx}{sl}{<->ssub * #3/bx/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{bx}{sc}{<->ssub * #3/bx/sc}{}^^J
    }%
    \ifLipsian
    \immediate\write15{%
    \string\DeclareFontShape{#1}{#2}{m}{li}{<->ssub * #3/m/li}{}^^J %<- Lipsian
    \string\DeclareFontShape{#1}{#2}{b}{li}{<->ssub * #3/b/li}{}^^J %<- Lipsian
    \string\DeclareFontShape{#1}{#2}{bx}{li}{<->ssub * #3/bx/li}{}^^J %<-Lipsian
    \string\DeclareFontShape{#1}{#2}{m}{ui}{<->ssub * #3/m/ui}{}^^J %<- upright Olga
    \string\DeclareFontShape{#1}{#2}{b}{ui}{<->ssub * #3/m/ui}{}^^J %<- upright Olga
    \string\DeclareFontShape{#1}{#2}{bx}{ui}{<->ssub * #3/bx/ui}{}^^J%<-upright Olga
    \string\DeclareFontShape{#1}{#2}{m}{rs}{<->ssub * #3/m/rs}{}^^J %<-serifed lc
    \string\DeclareFontShape{#1}{#2}{b}{rs}{<->ssub * #3/m/rs}{}^^J %<-serifed lc
    \string\DeclareFontShape{#1}{#2}{bx}{rs}{<->ssub * #3/bx/rs}{}^^J%<-serifed lc
    }%
    \global\Lipsianfalse\fi
  \closeout15}%
  }}

```

Notice that together with the Lipsian fonts the upright italics (Olga) and upright serifed lowercase alphabets are defined. In a while there are the definition for selecting these shapes. Of course you are not obliged to use them, but in case you wanted. . .

These results are obtained by means of the following macros.

\ifCMLM
\ifFamily

The `\ifCMLM` processes the necessary test in order to set the auxiliary macro `\n@xt` to be an alias to `\iftrue` or `iffalse` depending on the fact that the CM (or EC) fonts or the LM fonts are the default Latin ones, in this case it sets the `\n@xt` macro equivalent to `\iftrue`, otherwise it sets it to `\iffalse`. In order to succeed, it requires to analyse the first two letters of the default family name; if these letters form one of the sequences `cm` or `lm`, the CM or LM fonts have been loaded, otherwise some other fonts are in force. We need therefore a macro with delimited arguments in order to extract the first two letters of the family name.

```
\def\ifCMLM#1#2#3!{\edef\f@milyprefix{#1#2}%
   \ifthenelse{\(\equal{\f@milyprefix}{cm}\OR\equal{\f@milyprefix}{lm}\)}%
   {\let\n@xt\iftrue}{\def\f@milyprefix{cmr}\let\n@xt\iffalse}\n@xt}

```

The other macro `\ifFamily` uses the previous macro and according to the test result, possibly runs the `\substitutefontfamily` macro that, if necessary, creates the description file that map the specified family font description file to the second specified font family, both connected to the LGR encoding. Therefore, after these font definition files exist, LaTeX can fetch the Greek fonts by way of substitution. Let's explain again: if you specify

```
\Lipsiantrue\ifFamily{pxr}{lmr}
```

you state that you want to run the macro on the serifed Palatino font family, by associating the `pxr` family to the `lmr` one[5]; by specifying `\Lipsiantrue` you state that you want to create entries also for the Lipsian series and shape; the macro provides to reset `\Lipsianfalse` in order to avoid that other calls of that macro on non serifed or monospaced fonts try to create entries that in any case do not exist: the Lipsian font comes only as a serifed font! In this way, if you are using Palatino fonts through the pxfonts package, the `teubner` macros provide to create the necessary font description files so that while you are typesetting in medium normal Latin Palatino and you switch to Greek, the built in macros change the encoding to LGR; the LGR Palatino serifed medium normal Greek font does not exist, but that family, series and shape are mapped by the font description file to the corresponding LGR encoded Latin Modern CB fonts in medium series and normal shape, and typesetting goes on with the right Greek fonts.

```
\newcommand*\ifFamily[2]{%
\expandafter\ifCMLM#1!\else\substitutefontfamily{LGR}{#1}{#2}\fi}

```

You don't actually need to use that macro for the Times or the Palatino eXtended fonts loaded by means of the corresponding packages txfonts or pxfonts, because a hook is set up so that "at begin document" the loading of those packages is tested, and if the test is true, the necessary font description files are possibly created. If

[5]If you have the full CB Greek font collection it's more convenient to map the missing fonts to the Latin Modern Greek ones, while if you need to use the *10pt* option, you'd better map the missing family to the ordinary Computer Modern ones; the actual fonts are the same, but the latter font definition files cope with the *10pt* option, while the former don't.

you load the Times or the Palatino or any other non standard font by means of other packages, it's up to you to issue the `\substitutefontfamily` macro right after calling that font package and by using the correct family names; similarly you might substitute the new Latin font family names to other Greek font family names, if you have other fonts available. At the same time at begin document we memorise the name and encoding of the Latin font used for the default language, so that when returning to Latin font typesetting after Greek font typesetting, the proper language typesetting rules and encoding are restored.

```
\AtBeginDocument{%
\@ifpackageloaded{pxfonts}{\typeout{Palatino fonts loaded}%
\Lipsiantrue\ifFamily{pxr}{cmr}\Lipsianfalse
\ifFamily{pxss}{cmss}\ifFamily{pxtt}{cmtt}}{\relax}}

\AtBeginDocument{%
\@ifpackageloaded{txfonts}{\typeout{Times fonts loaded}%
\RequirePackage{teubnertx}}{}}

\AtBeginDocument{%
    \edef\previouslanguage{\languagename}%
    \edef\previousencoding{\f@encoding}}
```

Nevertheless all this requires a minimum of attention in specifying the options for the babel package and in the order extensions packages are input. The `teubner.sty` package should be read *after* any other package that sets or resets the Latin font encoding; for example if the T1 encoding is selected as the default one, in place of the OT1 encoding, then this choice must be made before this package is read in. Similarly when the babel options are specified, remember that the last language name becomes the default language at begin document; never specify `greek` as the last language option!

`\Lipsiakostext` `\lishape` `\textli`

`\lishape` is the normal declaration, modelled on the other similar macros in the LaTeX kernel, made up to chose the Lipsian shape. Nevertheless since it is a light character, if it must blend well with the other PostScript fonts, not only the CM and LM, but also the other ones available for typesetting with the TeX system, it is necessary to chose the `b` (bold) series in place of the `m` (medium) one, while maintaining the `bx` (bold extended) series when the other fonts are set with the blacker and larger series. This is why the `\lishape` declaration is a little more complicate than normal, since it has to test the value of the current series. The text command `\textli` matches the similar commands for Latin fonts. But the `\lishape` declaration is used also within the more complicated macros for declaring or setting the Lipsian font.

`\Lipsiakostext` is a *declaration* stating that from now on typesetting will be done with the Lipsian fonts; notice that the encoding and the language name in force before this declaration are memorised, then the current Greek version is selected; the `\let\~\accperispomeni` is required because switching on and off may reset the active tilde and connected macros definitions. `\~` in Greek must set the circumflex accent, so we make sure that this really occurs.

```
\DeclareRobustCommand{\lishape}{%
\not@math@alphabet\lishape\relax
\ifthenelse{\equal{\f@encoding}{\GRencoding@name}}{%
\ifboldLipsian
\ifthenelse{\equal{\f@series}{m}}%
{\fontseries{b}\fontshape{li}\rmfamily}%
{\fontshape{li}\rmfamily}\else
\fontshape{li}\rmfamily\fi}%
{\fontshape{it}\selectfont}}%

\DeclareTextFontCommand{\textli}{\lishape}%
\DeclareRobustCommand\Lipsiakostext{%
    \expandafter\select@language\expandafter{\GreekName}%
    \let\~\accperispomeni\let~\accperispomeni\lishape}

```

`\textLipsias` `\textLipsias` is a command that typesets its argument with the `\Lipsiakostext` declaration in force. The LaTeX command declaration used here makes sure that possible italic corrections are taken into account; the actual font switching is made through the same `\Lipsiakostext` declaration, but the inner working maintain local this declaration; for this reason we suggest to use this text command rather than the font declaration.

```
\DeclareTextFontCommand{\textLipsias}{\Lipsiakostext}

```

`\NoLipsiakostext` `\NoLipsiakostext` is the opposite declaration that undoes everything that was done with `\Lipsiakostext`. Probably it is superfluous, but it has been asked for. If `\Lipsiakostext` is delimited within a scope by means of an explicit group or an environment, it stops its effectiveness with the end of its scope.

It is worth noting that, in order to delimit within a scope the action of this and of the other declarations, it is possible to use them as environments with the same name without the backslash. for example one might input in the source file something as:

```
\begin{Lipsiakostext}
⟨Greek text to be typeset with the Lipsian font⟩
\end{Lipsiakostext}
```

Remember also that these Greek text declarations may be issued while typesetting with Latin fonts; they provide also the language switch, so that they do not require the typesetter to first switch to Greek and then to choose a certain Greek font.

```
\DeclareRobustCommand\NoLipsiakostext{%
    \ifthenelse{\equal{\f@series}{b}}{\fontseries{m}}{\relax}%
    \fontshape{n}\selectfont
    \expandafter\select@language\expandafter{\previouslanguage}%
    \rmfamily\bbl@activate{~}}

```

\textDidot `\textDidot` is a similar macro where the common upright Greek characters are selected; it goes by itself that if `\textli` is specified within the `\textDidot` argument, the typesetting is or becomes identical with what one can obtain with the `\textLipsias` command.

```
\DeclareRobustCommand\textDidot[1]{{%
    \expandafter\select@language\expandafter{\GreekName}%
    \let\~\accperispomeni\let~\accperispomeni
    \fontencoding{LGR}\rmfamily#1}}

```

\textlatin `\textlatin` is a redefinition of the standard babel macro that is adapted to the present situation, where it may be called behind the scenes in certain situations that are beyond the control of the typesetter. Therefore every precaution is taken in order to be sure that the composition of the command argument is really done with the default encoding and font families, but maintaining the current series and shape; of course, if the shape is that related to the Lipsian font, then the italic shape is temporarily restored (local definition). Moreover, with the (default) Latin fonts the tilde is restored to a non breaking space by simply making it an active character.

```
\DeclareRobustCommand\textlatin[1]{\edef\externalencoding{\f@encoding}{%
    \def\itdefault{it}\def\@tempA{li}\ifx\@tempA\f@shape\def\f@shape{it}\fi%
    \expandafter\select@language\expandafter{\previouslanguage}%
    \fontencoding{\previousencoding}%
    \fontfamily{\rmdefault}\selectfont
    \bbl@activate{~}#1}%
    \expandafter\fontencoding\expandafter{\externalencoding}\rmfamily}

```

\uishape
\textui
\rsshape
\textrs

The other switching font macros for using the other shapes that are available with the CB fonts are working only when typesetting in Greek and the default encoding is therefore LGR.

```
\DeclareRobustCommand\uishape{%
\ifthenelse{\equal{\f@encoding}{\GRencoding@name}}%
{\fontshape{ui}\selectfont}{\relax}}
\DeclareTextFontCommand{\textui}{\uishape}

\DeclareRobustCommand\rsshape{%
\ifthenelse{\equal{\f@encoding}{\GRencoding@name}}%
{\fontshape{rs}\selectfont}{\relax}}
\DeclareTextFontCommand{\textrs}{\rsshape}

```

5.3 Service macros

Now we start the specific additions introduced with this package.

\strip@pt The LaTeX kernel has the macro `\strip@pt` that strips off the pt part from the expanded value of a dimension register and makes available the measure in pt of the contained length (the register contains the length measure in scaled points; the expansion performed by TeX with the command `\the` converts the scaled points to printer points and shows the result with a string of decimal digits with, possibly, a decimal fraction); its argument is supposed to be a dimension register name, not its expanded contents. The `\strip@pt` command eliminates the decimal point and the fractional part if the latter is nought.

With the help of such service macro we are going to define a certain number of "lift accent" macros or "put cedilla" macros that work with both upright and slanted fonts, although they contain different parameters for Latin compared to Greek alphabets.

`\lift@accent` The first "lift accent" macro just puts an accent over a letter, without inserting any space between them; the first argument is the accent code (decimal, hexadecimal or octal; I prefer decimal), while the second argument is the letter – any letter, even if it is not a vowel!

```
\newcommand*\lift@accent[2]{\leavevmode
{\edef\slant@{\strip@pt\fontdimen1\font}%
\dimen@=\z@\setbox\z@\hbox{\char#1}\advance\dimen@-.5\wd\z@
\setbox\tw@\hbox{i}\setbox\z@\hbox{#2}%
\ifdim\wd\z@>\wd\tw@\advance\dimen@ .5\wd\z@
    \else\advance\dimen@ .3\wd\z@\fi
\ifx#2h\advance\dimen@.05\wd\z@\fi
\@tempdima\ht\z@\advance\@tempdima-1ex\relax
\advance\dimen@\slant@\@tempdima
\raise\@tempdima\hbox to\z@{\kern\dimen@\char#1\relax\hss}\box\z@}}

```

`\Lift@accent` The second "lift accent" macro behaves as the first one except it interposes a small vertical distance between the accent and the letter:

```
\newcommand*\Lift@accent[2]{\leavevmode
{\edef\slant@{\strip@pt\fontdimen1\font}%
\dimen@=\z@\setbox\z@\hbox{\char#1}\advance\dimen@-.5\wd\z@
\setbox\tw@\hbox{i}\setbox\z@\hbox{#2}%
\ifdim\wd\z@>\wd\tw@\advance\dimen@ .5\wd\z@
\else\advance\dimen@ .3\wd\z@\fi
\ifx#2a\advance\dimen@-.1\wd\z@\fi
\ifx#2h\advance\dimen@.05\wd\z@\fi
\@tempdima\ht\z@\advance\@tempdima-1ex\advance\@tempdima.1ex\relax
\advance\dimen@\slant@\@tempdima
\raise\@tempdima\hbox to\z@{\kern\dimen@\char#1\relax\hss}\box\z@}}

```

`\LIFT@accent` The third "lift accent" macro behaves as the first one, except it interposes a specified vertical space between the letter and the accent; this space is specified as the second argument:

```
\newcommand*\LIFT@accent[3]{\leavevmode
{\edef\slant@{\strip@pt\fontdimen1\font}%
\dimen@=\z@\setbox\z@\hbox{\char#1}\advance\dimen@-.5\wd\z@
\setbox\tw@\hbox{i}\setbox\z@\hbox{#3}%
\ifdim\wd\z@>\wd\tw@\advance\dimen@ .5\wd\z@
\else\advance\dimen@ .3\wd\z@\fi
\ifx#2a\advance\dimen@-.1\wd\z@\fi
\ifx#2h\advance\dimen@.05\wd\z@\fi
\@tempdima\ht\z@\advance\@tempdima-1ex\relax
\def\@tempA{#2}\ifx\@tempA\undefined\else
\advance\@tempdima#2\fi\let\@tempA\undefined
\advance\dimen@\slant@\@tempdima
\raise\@tempdima\hbox to\z@{\kern\dimen@\char#1\relax\hss}\box\z@}}

```

All these macros will be used in subsequent “put accent” macros, that will stack also several accents one above the other; the necessity arises for example when the macron or breve diacritical marks have to be put over accented letters; according to typographical practice the accents must go over the macron or the breve. In a similar way philologists often must use other diacritical marks in addition to the traditional Greek ones, therefore these macros will be used, for example, for setting the Scandinavian ring (from a Latin font) over a Greek letter (from a Greek font).

`\cap@` The first such unusual diacritical mark is a small cap, a small upside down breve sign, that is in position 1 of the Greek font table.

```
\DeclareRobustCommand{\cap@}[1]{\leavevmode
{\edef\slant@{\strip@pt\fontdimen1\font}%
\setbox\tw@\hbox{\fontencoding{\GRencoding@name}\selectfont
    \char1}\dimen@-.5\wd\tw@
\setbox\z@\hbox{#1}%
\advance\dimen@ .5\wd\z@
\@tempdima\ht\z@\advance\@tempdima.55ex\relax
\advance\dimen@\slant@\@tempdima
\ifx\cf@encoding\GRencoding@name\else
\ifx#1k\advance\dimen@-.3\wd\tw@\fi\fi
\raise\@tempdima\hbox to\z@{\kern\dimen@\box\tw@\relax\hss}\box\z@}}

```

The `\ifx\cf@encoding\GRencoding@name` conditional construct shows that this macro behaves differently with different font encodings; the following `\ifx#1k` checks the argument against the Greek letter kappa, which shows very clearly that these macros operate on any letter, not only on vowels.

`\cap` By means of the above `\cap@` macro we can define three equivalent commands to be used either when the Greek encoding is in force, or when one of the Latin encodings is in force; but we must pay attention, because there exist already the command `\cap` to be used in mathematics; therefore we better exclude this possibility through a clever use of the `\textormath` macro. Therefore we first save the math command into an alia `\mcap`; then we define three textual commands for the various encodings `\tcap`; finnally we use `\textormath`:

```
\let\mcap\cap
```

```
\DeclareTextCommand{\tcap}{\GRencoding@name}{\cap@}
\DeclareTextCommand{\tcap}{OT1}{\cap@}
\DeclareTextCommand{\tcap}{T1}{\cap@}
\DeclareRobustCommand*\cap{\textormath{\tcap}{\mcap}}
```

Probably one definition would be sufficient, but on one side the presence of three encoding dependent macros are the remains of initial works, while on the other side they prevent to use these macros with encodings for which the macro might not work well, because it was not tested with them.

`\cap@cedilla` Similarly a small cap can be put under another letter as it was a cedilla; for this task another macro is defined, which makes use of the same glyph in position 1 in the Greek font table:

```
\newcommand*\cap@cedilla[1]{\leavevmode
{\setbox4\hbox{\fontencoding{\GRencoding@name}\selectfont\char1}%
\dimen@-.5\wd4
\setbox\z@\hbox{#1}%
\ifx\cf@encoding\GRencoding@name
   \ifx#1i\advance\dimen@ .65\wd\z@\else\advance\dimen@ .5\wd\z@\fi
\else
   \ifx#1i\advance\dimen@ .55\wd\z@\else\advance\dimen@ .5\wd\z@\fi
\fi
\hbox to\z@{\kern\dimen@\box4\hss}\unhbox\z@}}

```

`\ring@cedilla` Another cedilla like diacritical mark is the Scandinavian ring put under a letter; the ring is taken from the metrics font, so its slot position does not depend on the various Latin encodings; the correct positioning requires careful examination of the letter under which it is to be placed, distinguishing the Greek from the Latin encodings:

```
\newcommand*\ring@cedilla[1]{\leavevmode
{\setbox4\hbox{\metricsfont\char26}%
\edef\slant@{\strip@pt\fontdimen1\font}%
\dimen@-.5\wd4\ifdim\slant@\p@>\z@\advance\dimen@-.04ex\fi
\setbox\z@\hbox{#1}%
\ifx\cf@encoding\GRencoding@name
    \advance\dimen@ .45\wd\z@
    \ifx#1h\advance\dimen@-.13\wd\z@\fi
    \ifx#1a\advance\dimen@-.07\wd\z@\fi
    \ifx#1o\advance\dimen@-.07\wd\z@\fi
    \ifx#1u\advance\dimen@+.07\wd\z@\fi
    \ifx#1w\advance\dimen@+.03\wd\z@\fi
\else
    \ifx#1i\advance\dimen@.55\wd\z@\else
    \ifx#1r\advance\dimen@.38\wd\z@\else
    \ifx#1o\advance\dimen@.47\wd\z@\else
            \advance\dimen@0.5\wd\z@
    \fi\fi\fi
\fi
\hbox to\z@{\kern\dimen@\box4\hss}\unhbox\z@}}

```

`\dot@cedilla` Even the standard LaTeX macro `dot` must be redefined with a cedilla like macro, so as to make use of a special dot from the metric symbols font:

```
\newcommand*\dot@cedilla[1]{\leavevmode
{\setbox4\hbox{\metricsfont\char27}%
\dimen@-.5\wd4
\setbox\tw@\hbox{i}\setbox\z@\hbox{#1}%
\ifx\cf@encoding\GRencoding@name
    \advance\dimen@ .5\wd\z@
    \ifx#1h\advance\dimen@-.13\wd\z@\fi
\else
    \ifdim\wd\z@>\wd\tw@\advance\dimen@.55\wd\z@
        \else\advance\dimen@.5\wd\tw@\fi
\fi
\setbox\tw@\hbox{o}\ifdim\wd\z@=\wd\tw@\advance\dimen@-.05\wd\z@\fi
\hbox to\z@{\kern\dimen@\box4\hss}\unhbox\z@}}

```

`\tie@cedilla` LaTeX has the macro `\t` for placing a “tie” over two letters; philologists require also a tie under two letters; this is why another cedilla like macro is needed:

```
\newcommand*\tie@cedilla[1]{\leavevmode
{\setbox4\hbox{\fontencoding{\GRencoding@name}\selectfont\char20}%
\dimen@-.5\wd4
\setbox\tw@\hbox{i}\setbox\z@\hbox{#1}%
\ifx\cf@encoding\GRencoding@name
    \advance\dimen@.5\wd\z@
    \ifx#1h\advance\dimen@-.1\wd\z@\fi
    \ifx#1u\advance\dimen@.15\wd\z@\fi
\else
    \ifdim\wd\z@>\wd\tw@\advance\dimen@ .55\wd\z@
        \else\advance\dimen@ .5\wd\tw@\fi
\fi
\setbox\tw@\hbox{o}\ifdim\wd\z@=\wd\tw@\advance\dimen@-.05\wd\z@\fi
\hbox to\z@{\kern\dimen@\box4\hss}\unhbox\z@}}

```

5.4 Extended accent definitions

We will use those service macros in the definition of several accent like macros that keep all the intricacies away from the user.

In particular the LaTeX kernel macros are used in order to declare accents, composite glyphs, composite commands, and the like; these are used as the default definitions; afterwards other definitions will be given that work when these composite macros don’t work.

In other words, while `\~` and `u` in Greek form the composite glyph “upsilon with circumflex” that exists in the Greek font table, the same macro `\~` and the letter `k` produce the superposition of a circumflex on top of a "kappa" glyph, since this glyph does not exist in the Greek font table. Notice that all these declarations are restricted to the Greek font encoding so they are usable only when such encoding is in force. See the `teubner-doc.pdf` file for more details concerning the usefulness of the extended accent macros vs. the ligature mechanism. In any case, with version 3.9g of babel and the modifications introduced by Günter Milde, the actual babelgreek maintainer, during the year 2013, such extended accent macros are LICR encoded and may always be used when typesetting in Greek; see file `greek-fontenc.def` and `lgrenc.def` for further details, remembering that such files are always loaded when typesetting Greek texts, irrespective of the input encoding; if the encoding is `utf8` direct Greek glyph input is possible (if your keyboard allows you to do such input).

5.5 Special accent macros

Now we come back to the "accent like" and "cedilla like" general macros we defined above, and that will be extensively used in the following definitions. Note that for what the circumflex is concerned, when `teubner` is in effect it is not defined as an active character and does not work as a non breaking space. The command `\~` is just an accent macro; how do you put a non breaking space in a Greek context? By simply using the LaTeX kernel macro `\nobreakspace`; when typesetting with non-Greek fonts the `~` is certainly handy to insert a non breaking space (a tie), but for polytonic Greek spelling in the past 15 years or so the Greek language definition file has always used the `~` sign a letter, not as an active character. If you look in the babel package documentation related to the Greek language, you find that for what concerns the `~` with polytonic spelling a number of "dirty tricks" have been used, but nothing has been done to replace the "tie" function of this character when typesetting in languages that use the Latin script; the only action related to this point has been to redefine the kernel macros for typesetting figure and table captions so as to substitute the `~` character with its explicit definition `\nobreakspace`. It is necessary to do the same when this package is used, although a shorter alias command `\nbs` is provided in order to simplify the input keying.

```
\let\nbs\nobreakspace
```

Before defining the Greek accents with the extended macros input with the LICR accent macros, we have to define the accent superpositioning macros that with teubner allow to stack from one to more accent over the base letter, taking into account the slant of the font from which the base letter is taken. Notice that these macros allow to set an accent on any base latter, even if that might be meaningless. At the same time we redefine the standard macros so as to let them do the same work when the LGR encoding is not in force. This duality is necessary, not only for backward compatibility, but also for avoiding that the normal redefined macros highjack the LICR facility.

We firs define aliases for the standard LaTeX accents, so that when entering Greek typesetting modo we can save the LaTeX macros, and restore them upon exiting:

```
\let\accacuto\'
\let\accgrave\`
\let\acccircon\~
\let\accdieresi\"
\let\accbreve\u
\let\accmacron\=
```

Then we define alternate macros fors these accents, as “lifting” accent macros, so that they can put their respective accent over any letter. For the diaeresis we need to put an invisible character (`v` in the LGR encoded CB fonts, that with LICR becomes `\textcompwordmark`) in order to avoid any ligature with an implied end of word (boundarychar) that turns the diaeresis into an apostrophe.

```
\DeclareTextCommand{\G}{\GRencoding@name}[1]{\lift@accent{96}{#1}}
\DeclareTextCommand{\A}{\GRencoding@name}[1]{\lift@accent{39}{#1}}
\DeclareTextCommand{\C}{\GRencoding@name}[1]{\lift@accent{126}{#1}}
\DeclareTextCommand{\D}{\GRencoding@name}[1]{\lift@accent{34\textcompwordmark}{#1}}
\DeclareTextCommand{\B}{\GRencoding@name}[1]{\lift@accent{30}{#1}}
\DeclareTextCommand{\M}{\GRencoding@name}[1]{\lift@accent{31}{#1}}
%
```

But we have to provide also the means for disabling the `~` shorthand that is reset every time the Greek language is selected again in a multi language document where language shifts take place quite often; we must also counteract the resetting of the `\~` definition performed by the `greek.ld` file in every language shift; we add the accent definition to the `\extrasgreek` macro, and we reset them at `\noextrasgreek` execution.

```
\addto\extrasgreek{\shorthandoff{~}\let\~\accperispomeni
\let\"\accdialytika\let\'\acctonos\let\`\accvaria}
\addto\noextrasgreek{\shorthandon{~}\let\~\acccircon
\let\"\accdieresi\let\'\accacuto\let\`\accgrave}

```

Besides the normal `\B` command for setting a breve command, another “large breve” is required by philologists, who need to mark a diphthong, or in general two letters; the macro `\U` does the job, but it is the typesetter’s responsibility to input the macro argument as made of two letters (possibly with their own accents):

```
\DeclareTextCommand{\U}{\GRencoding@name}[1]{\lift@accent{151}{#1}}
% rough
\DeclareTextCommand{\r}{\GRencoding@name}[1]{\lift@accent{60}{#1}}
% smooth
\DeclareTextCommand{\s}{\GRencoding@name}[1]{\lift@accent{62}{#1}}
% acute+diaeresis
\DeclareTextCommand{\Ad}{\GRencoding@name}[1]{\lift@accent{35}{#1}}
% grave+diaeresis
\DeclareTextCommand{\Gd}{\GRencoding@name}[1]{\lift@accent{36}{#1}}
% circumflex+diaeresis
\DeclareTextCommand{\Cd}{\GRencoding@name}[1]{\lift@accent{32}{#1}}
% acute+rough
\DeclareTextCommand{\Ar}{\GRencoding@name}[1]{\lift@accent{86}{#1}}
% grave+rough
```

```
\DeclareTextCommand{\Gr}{\GRencoding@name}[1]{\lift@accent{67}{#1}}
% circumflex+rough
\DeclareTextCommand{\Cr}{\GRencoding@name}[1]{\lift@accent{64}{#1}}
% acute+smooth
\DeclareTextCommand{\As}{\GRencoding@name}[1]{\lift@accent{94}{#1}}
% grave+smooth
\DeclareTextCommand{\Gs}{\GRencoding@name}[1]{\lift@accent{95}{#1}}
% circumflex+smooth
\DeclareTextCommand{\Cs}{\GRencoding@name}[1]{\lift@accent{92}{#1}}
```

Most of the above accent commands are used again in order to tie a text symbol meaning to certain combinations, that is when they receive as argument a vowel whose accented glyph is present in the font; in this way in order to type "alpha with rough breath, acute accent and iota subscript" you can type `<'a|`, or `\Ar{a}|` or `\arai` or `\<'a|`, if you use the new standard accent macros; the advantage of using the first notation is its short string; the advantage of the second is that it does not break kerning commands with a preceding letter; the advantage of the third is that it does not break any kerning either before or after; the fourth solution produces the same result as the third, but it's easier to make up and you don't have to memorise any specific naming rule for accented glyphs. With the Lipsian font this trick is particularly useful for any sequence of alpha and upsilon each one with its own accents and/or diaeresis.

In Greek the regular cedilla is meaningless, so that `\c` may be redefined as a semivowel command; at the same time the typesetter might be more comfortable if he could always use the same, although longer, macro for marking a vowel as a semivowel one; therefore `\c` plays the same role in Greek as `\semiv`.

```
% cap cedilla
\DeclareTextCommand{\c}{\GRencoding@name}[1]{\cap@cedilla{#1}}
\DeclareTextCommand{\semiv}{\GRencoding@name}[1]{\cap@cedilla{#1}}
\DeclareTextCommand{\semiv}{OT1}[1]{\cap@cedilla{#1}}
\DeclareTextCommand{\semiv}{T1}[1]{\cap@cedilla{#1}}
% ring cedilla
\DeclareTextCommand{\ring}{\GRencoding@name}[1]{\ring@cedilla{#1}}
\DeclareTextCommand{\ring}{OT1}[1]{\ring@cedilla{#1}}
\DeclareTextCommand{\ring}{T1}[1]{\ring@cedilla{#1}}
% dot cedilla
\DeclareTextCommand{\Dot}{\GRencoding@name}[1]{\dot@cedilla{#1}}
\DeclareTextCommand{\Dot}{OT1}[1]{\dot@cedilla{#1}}
\DeclareTextCommand{\Dot}{T1}[1]{\dot@cedilla{#1}}
% tie cedilla
\DeclareTextCommand{\ut}{\GRencoding@name}[1]{\tie@cedilla{#1}}
\DeclareTextCommand{\ut}{OT1}[1]{\tie@cedilla{#1}}
\DeclareTextCommand{\ut}{T1}[1]{\tie@cedilla{#1}}
%
% Acute breve
\DeclareTextCommand{\Ab}{\GRencoding@name}[1]%
    {\LIFT@accent{39}{-.15ex}{\lift@accent{30}{#1}}}
% Grave breve
\DeclareTextCommand{\Gb}{\GRencoding@name}[1]%
```

```
    {\LIFT@accent{96}{-.15ex}{\lift@accent{30}{#1}}}
% Acute rough breve
\DeclareTextCommand{\Arb}{\GRencoding@name}[1]%
    {\LIFT@accent{86}{-.15ex}{\lift@accent{30}{#1}}}
% Grave rough breve
\DeclareTextCommand{\Grb}{\GRencoding@name}[1]%
    {\LIFT@accent{67}{-.15ex}{\lift@accent{30}{#1}}}
% Acute smooth breve
\DeclareTextCommand{\Asb}{\GRencoding@name}[1]%
    {\LIFT@accent{94}{-.15ex}{\lift@accent{30}{#1}}}
% Grave smooth breve
\DeclareTextCommand{\Gsb}{\GRencoding@name}[1]%
    {\LIFT@accent{95}{-.15ex}{\lift@accent{30}{#1}}}
%
% Acute macron
\DeclareTextCommand{\Am}{\GRencoding@name}[1]%
    {\Lift@accent{39}{\lift@accent{31}{#1}}}
% Grave macron
\DeclareTextCommand{\Gm}{\GRencoding@name}[1]%
    {\Lift@accent{96}{\lift@accent{31}{#1}}}
% Circumflex macron
\DeclareTextCommand{\Cm}{\GRencoding@name}[1]%
    {\Lift@accent{126}{\lift@accent{31}{#1}}}
% Acute rough macron
\DeclareTextCommand{\Arm}{\GRencoding@name}[1]%
    {\Lift@accent{86}{\lift@accent{31}{#1}}}
% Grave rough macron
\DeclareTextCommand{\Grm}{\GRencoding@name}[1]%
    {\Lift@accent{67}{\lift@accent{31}{#1}}}
% Circumflex rough macron
\DeclareTextCommand{\Crm}{\GRencoding@name}[1]%
    {\Lift@accent{64}{\lift@accent{31}{#1}}}
% Acute smooth macron
\DeclareTextCommand{\Asm}{\GRencoding@name}[1]%
    {\Lift@accent{94}{\lift@accent{31}{#1}}}
% Grave smooth macron
\DeclareTextCommand{\Gsm}{\GRencoding@name}[1]%
    {\Lift@accent{95}{\lift@accent{31}{#1}}}
% Circumflex smooth macron
\DeclareTextCommand{\Csm}{\GRencoding@name}[1]%
    {\Lift@accent{92}{\lift@accent{31}{#1}}}
% smooth macron
\DeclareTextCommand{\Sm}{\GRencoding@name}[1]%
    {\Lift@accent{62}{\lift@accent{31}{#1}}}
% rough macron
\DeclareTextCommand{\Rm}{\GRencoding@name}[1]%
    {\Lift@accent{60}{\lift@accent{31}{#1}}}
% smooth breve
\DeclareTextCommand{\Sb}{\GRencoding@name}[1]%
    {\LIFT@accent{62}{-0.15ex}{\lift@accent{30}{#1}}}
```

```
% rough breve
\DeclareTextCommand{\Rb}{\GRencoding@name}[1]%
    {\LIFT@accent{60}{-0.15ex}{\lift@accent{30}{#1}}}
% breve and dieresis
\DeclareTextCommand{\bd}{\GRencoding@name}[1]%
    {\LIFT@accent{30}{-.1ex}{\lift@accent{34v}{#1}}}
%
% iota subscript
\DeclareTextCommand{\iS}{\GRencoding@name}[1]
   {\ooalign{#1\crcr\hidewidth\char124\hidewidth}}

```

`\d` The `\d` macro must be made available also with the Greek encoding

```
\DeclareTextCommand{\d}{\GRencoding@name}[1]%
   {\leavevmode\bgroup\o@lign{\relax#1\crcr
    \hidewidth\sh@ft{10}.\hidewidth}\egroup}

```

Some other philologist diacritical marks are needed.

`\Open` The `\Open` macro sets a special sign under a letter in order to mark it with an open pronunciation.

```
\DeclareRobustCommand{\Open}[1]{\leavevmode
{\setbox4\hbox{\raise-.33ex\hbox{\metricsfont\char14}}%
\dimen@-.5\wd4
\setbox\tw@\hbox{i}\setbox\z@\hbox{#1}%
\ifx\cf@encoding\GRencoding@name
    \advance\dimen@ .5\wd\z@
    \setbox\tw@\hbox{h}\ifdim\wd\z@=\wd\tw@\advance\dimen@-.13\wd\z@\fi
\else
    \ifdim\wd\z@>\wd\tw@\advance\dimen@ .55\wd\z@
        \else\advance\dimen@ .5\wd\tw@\fi
\fi
\setbox\tw@\hbox{o}\ifdim\wd\z@=\wd\tw@\advance\dimen@-.05\wd\z@\fi
\hbox to\z@{\kern\dimen@\box4\hss}\unhbox\z@}}

```

`\nasal` The macro `\nasal` marks a letter for a nasal pronunciation.

```
\DeclareRobustCommand{\nasal}[1]{\leavevmode
{\setbox4\hbox{\raise-1.7ex\hbox{\GEcq}}%
\dimen@-.5\wd4
\setbox\tw@\hbox{i}\setbox\z@\hbox{#1}%
\ifx\cf@encoding\GRencoding@name
    \advance\dimen@ .5\wd\z@
    \setbox\tw@\hbox{h}\ifdim\wd\z@=\wd\tw@\advance\dimen@-.13\wd\z@\fi
\else
    \ifdim\wd\z@>\wd\tw@\advance\dimen@ .55\wd\z@
        \else\advance\dimen@ .5\wd\tw@\fi
\fi
\setbox\tw@\hbox{o}\ifdim\wd\z@=\wd\tw@\advance\dimen@-.05\wd\z@\fi
\hbox to\z@{\kern\dimen@\box4\hss}\unhbox\z@}}

```

`\tenaspir`

Similarly `\tenaspir` marks a “tenuis aspiratio”

```
\DeclareRobustCommand{\tenaspir}[1]{#1\/%
    {\fontencoding{\GRencoding@name}\selectfont<v}}
```

`\palat`

`\palat` marks a palatal pronunciation of some consonants.

```
\DeclareRobustCommand{\palat}[1]{#1{%
    \expandafter\fontencoding\expandafter{\GRencoding@name}\selectfont
    \anwtonos}}

```

With the help of the previous macros some new commands get defined so as to use a simple more or less mnemonic macro instead of having the typesetter type in nested macros; these macros are valid only in the Greek and Latin encodings.

```
% dot and breve
\DeclareTextCommand{\Ud}{\GRencoding@name}[1]{\d{\u{#1}}}
% dot and macron
\DeclareTextCommand{\md}{\GRencoding@name}[1]{\d{\={#1}}}
% open and breve
\DeclareTextCommand{\UO}{\GRencoding@name}[1]{\Open{\u{#1}}}
% open and macron
\DeclareTextCommand{\mO}{\GRencoding@name}[1]{\Open{\={#1}}}

%
\DeclareTextCommand{\Ud}{T1}[1]{\d{\u{#1}}}
\DeclareTextCommand{\md}{T1}[1]{\d{\={#1}}}
\DeclareTextCommand{\UO}{T1}[1]{\Open{\u{#1}}}
\DeclareTextCommand{\mO}{T1}[1]{\Open{\={#1}}}
%
\DeclareTextCommand{\Ud}{OT1}[1]{\d{\u{#1}}}
\DeclareTextCommand{\md}{OT1}[1]{\d{\={#1}}}
\DeclareTextCommand{\UO}{OT1}[1]{\Open{\u{#1}}}
\DeclareTextCommand{\mO}{OT1}[1]{\Open{\={#1}}}

```

5.6 Some text commands

`\greekquoteleft` `\greekquoteright` `\textguillemotleft` `\textguillemotright` `\textcompwordmark` `\textemdash` `emdash` `\textendash`

All these macros are defined by the new `lgrenc.def` file, so it is not necessary to redefine them.

`\stigma` `\varstigma` `\koppa` `\coppa` `\varkoppa` `\sampi` `\Coppa` `\Stigma` `\Sampi` `\Euro` `\permill`

S These macros, too, are defined by the `lgrenc.def` file and subsidiaries, but with different names; therefore we make some aliases:

```
\let\stigma\textstigma
\let\varstigma\textvarstigma
\let\koppa\textkoppa
\let\qoppa\textqoppa
```

```
\let\coppa\textqoppa
\let\Koppa\textQoppa
\let\Coppa\textQoppa
\let\varkoppa\textqoppa
\let\sampi\textsampi
\let\Stigma\textStigma
\let\Sampi\textsampi
\let\f\textdigamma
\let\Digamma\textDigamma
\let\Euro\texteuro
\let\permill\textperthousand
\let\schwa\textschwa
```

`\textdollar`
`\textsection`
`\textsterling`
`\textunderscore`
`\textvisiblespace`

More important, although unlikely to be found in a philological text, is the question of standard LaTeX commands that are defined with reference to some encoding; if Greek text is being typeset, the Greek encoding is being used and such symbols would not be available any more; LaTeXwould issue warning messages complaining for their absence. Therefore we redefined them also for the Greek encoding.

```
\DeclareTextCommand{\textdollar}{\GRencoding@name}%
    {{\fontencoding{T1}\selectfont\char36}}
\DeclareTextCommand{\textsection}{\GRencoding@name}%
    {{\fontencoding{T1}\selectfont\char159}}
\DeclareTextCommand{\textsterling}{\GRencoding@name}%
    {{\fontencoding{T1}\selectfont\char191}}
\DeclareTextCommand{\textunderscore}{\GRencoding@name}%
    {{\fontencoding{T1}\selectfont\char95}}
\DeclareTextCommand{\textvisiblespace}{\GRencoding@name}%
    {{\fontencoding{T1}\selectfont\char32}}

```

5.7 Accent macros and glyph names

Now come dozens of macros that allow to access Greek accented vowels (plus rho with rough and smooth breaths) with macros instead of ligatures; such macros allow the kerning information to be used by TeX, while the ligature mechanism would sometimes impeach the use of such kerning information. Notice that the same glyphs are often accessed with a "text symbol" or a "text composite symbol"; as explained above the opportunity of using either one derives from the necessity of maintaining the kerning mechanism embedded in the font; if the CB fonts had a postfixed accent notation, instead of a prefixed one, none of these macros would be necessary (probably. . . !), but there would be other inconveniences.

Notice that the following code is subject to the boolean variable `GlyphNames` which is set to *false* by default, just for compatibility with the past; I suggest to use the *GlyphNames* option when when you really want to use such macros; remember though that all these glyphs are more easily specified by means of the extended accent macros that are also less restricted in their names; for a letter marked with a smooth breath and an acute accent you can indifferently type before the letter one of the following `\>\'`, `\>'`, `\'\>`, `\'>` at your choice. Moreover you can always postfix the mark for the iota subscribed at the right of the letter, without any need o memorising complicated names. Notice the macros `\oR` and `\oG` that have the second letter capitalised in order to avoid conflicts with otherwise homonymous macros defined in the LaTeX kernel or in other packages; by maintaining the false value to the boolean variable `GlyphNames` you are sure to avoid conflicts.

```
\ifGlyphNames
\DeclareTextSymbol{\ag}{\GRencoding@name}{128}
\DeclareTextSymbol{\ar}{\GRencoding@name}{129}
\DeclareTextComposite{\r}{\GRencoding@name}{a}{129}
\DeclareTextSymbol{\as}{\GRencoding@name}{130}
\DeclareTextComposite{\s}{\GRencoding@name}{a}{130}
\DeclareTextSymbol{\aa}{\GRencoding@name}{136}
\DeclareTextSymbol{\ac}{\GRencoding@name}{144}
\DeclareTextSymbol{\ai}{\GRencoding@name}{248}
\DeclareTextSymbol{\aai}{\GRencoding@name}{140}
\DeclareTextSymbol{\aci}{\GRencoding@name}{148}
\DeclareTextSymbol{\agi}{\GRencoding@name}{132}
\DeclareTextSymbol{\ara}{\GRencoding@name}{137}
\DeclareTextComposite{\Ar}{\GRencoding@name}{a}{137}
\DeclareTextSymbol{\arc}{\GRencoding@name}{145}
\DeclareTextComposite{\Cr}{\GRencoding@name}{a}{145}
\DeclareTextSymbol{\arg}{\GRencoding@name}{131}
\DeclareTextComposite{\Gr}{\GRencoding@name}{a}{131}
\DeclareTextSymbol{\ari}{\GRencoding@name}{133}
\DeclareTextSymbol{\asa}{\GRencoding@name}{138}
\DeclareTextComposite{\As}{\GRencoding@name}{a}{138}
\DeclareTextSymbol{\asc}{\GRencoding@name}{146}
\DeclareTextComposite{\Cs}{\GRencoding@name}{a}{146}
\DeclareTextSymbol{\asg}{\GRencoding@name}{139}
\DeclareTextComposite{\Gs}{\GRencoding@name}{a}{139}
\DeclareTextSymbol{\asi}{\GRencoding@name}{134}
\DeclareTextSymbol{\argi}{\GRencoding@name}{135}
\DeclareTextSymbol{\arai}{\GRencoding@name}{141}
\DeclareTextSymbol{\arci}{\GRencoding@name}{149}
\DeclareTextSymbol{\asai}{\GRencoding@name}{142}
\DeclareTextSymbol{\asgi}{\GRencoding@name}{143}
\DeclareTextSymbol{\asci}{\GRencoding@name}{150}
\DeclareTextSymbol{\hg}{\GRencoding@name}{152}
\DeclareTextSymbol{\hr}{\GRencoding@name}{153}
\DeclareTextComposite{\r}{\GRencoding@name}{h}{153}
\DeclareTextSymbol{\hs}{\GRencoding@name}{154}
\DeclareTextComposite{\s}{\GRencoding@name}{h}{154}
\DeclareTextSymbol{\hrg}{\GRencoding@name}{163}
\DeclareTextComposite{\Gr}{\GRencoding@name}{h}{163}
\DeclareTextSymbol{\hgi}{\GRencoding@name}{156}
\DeclareTextSymbol{\hri}{\GRencoding@name}{157}
\DeclareTextSymbol{\hsi}{\GRencoding@name}{158}
\DeclareTextSymbol{\hrgi}{\GRencoding@name}{167}
```

```
\DeclareTextSymbol{\ha}{\GRencoding@name}{160}
\DeclareTextSymbol{\hra}{\GRencoding@name}{161}
\DeclareTextComposite{\Ar}{\GRencoding@name}{h}{161}
\DeclareTextSymbol{\hsa}{\GRencoding@name}{162}
\DeclareTextComposite{\As}{\GRencoding@name}{h}{162}
\DeclareTextSymbol{\hsg}{\GRencoding@name}{171}
\DeclareTextComposite{\Gs}{\GRencoding@name}{h}{171}
\DeclareTextSymbol{\hai}{\GRencoding@name}{164}
\DeclareTextSymbol{\hrai}{\GRencoding@name}{165}
\DeclareTextSymbol{\hsai}{\GRencoding@name}{166}
\DeclareTextSymbol{\hsgi}{\GRencoding@name}{175}
\DeclareTextSymbol{\hc}{\GRencoding@name}{168}
\DeclareTextSymbol{\hrc}{\GRencoding@name}{169}
\DeclareTextComposite{\Cr}{\GRencoding@name}{h}{169}
\DeclareTextSymbol{\hsc}{\GRencoding@name}{170}
\DeclareTextComposite{\Cs}{\GRencoding@name}{h}{170}
\DeclareTextSymbol{\hci}{\GRencoding@name}{172}
\DeclareTextSymbol{\hrci}{\GRencoding@name}{173}
\DeclareTextSymbol{\hsci}{\GRencoding@name}{174}
\DeclareTextSymbol{\hi}{\GRencoding@name}{249}
\DeclareTextSymbol{\wg}{\GRencoding@name}{176}
\DeclareTextSymbol{\wr}{\GRencoding@name}{177}
\DeclareTextComposite{\r}{\GRencoding@name}{w}{177}
\DeclareTextSymbol{\ws}{\GRencoding@name}{178}
\DeclareTextComposite{\s}{\GRencoding@name}{w}{178}
\DeclareTextSymbol{\wrg}{\GRencoding@name}{179}
\DeclareTextComposite{\Gr}{\GRencoding@name}{w}{179}
\DeclareTextSymbol{\wgi}{\GRencoding@name}{180}
\DeclareTextSymbol{\wri}{\GRencoding@name}{181}
\DeclareTextSymbol{\wsi}{\GRencoding@name}{182}
\DeclareTextSymbol{\wrgi}{\GRencoding@name}{183}
\DeclareTextSymbol{\wa}{\GRencoding@name}{184}
\DeclareTextSymbol{\wra}{\GRencoding@name}{185}
\DeclareTextComposite{\Ar}{\GRencoding@name}{w}{185}
\DeclareTextSymbol{\wsa}{\GRencoding@name}{186}
\DeclareTextComposite{\As}{\GRencoding@name}{w}{186}
\DeclareTextSymbol{\wsg}{\GRencoding@name}{187}
\DeclareTextComposite{\Gs}{\GRencoding@name}{w}{187}
\DeclareTextSymbol{\wai}{\GRencoding@name}{188}
\DeclareTextSymbol{\wrai}{\GRencoding@name}{189}
\DeclareTextSymbol{\wsai}{\GRencoding@name}{190}
\DeclareTextSymbol{\wsgi}{\GRencoding@name}{191}
\DeclareTextSymbol{\wc}{\GRencoding@name}{192}
\DeclareTextSymbol{\wrc}{\GRencoding@name}{193}
\DeclareTextComposite{\Cr}{\GRencoding@name}{w}{193}
\DeclareTextSymbol{\wsc}{\GRencoding@name}{194}
\DeclareTextComposite{\Cs}{\GRencoding@name}{w}{194}
\DeclareTextSymbol{\wci}{\GRencoding@name}{196}
\DeclareTextSymbol{\wrci}{\GRencoding@name}{197}
\DeclareTextSymbol{\wsci}{\GRencoding@name}{198}
```

```
\DeclareTextSymbol{\wi}{\GRencoding@name}{250}
\DeclareTextSymbol{\ig}{\GRencoding@name}{200}
\DeclareTextSymbol{\ir}{\GRencoding@name}{201}
\DeclareTextComposite{\r}{\GRencoding@name}{i}{201}
\DeclareTextSymbol{\is}{\GRencoding@name}{202}
\DeclareTextComposite{\s}{\GRencoding@name}{i}{202}
\DeclareTextSymbol{\irg}{\GRencoding@name}{203}
\DeclareTextComposite{\Gr}{\GRencoding@name}{i}{203}
\DeclareTextSymbol{\ia}{\GRencoding@name}{208}
\DeclareTextSymbol{\ira}{\GRencoding@name}{209}
\DeclareTextComposite{\Ar}{\GRencoding@name}{i}{209}
\DeclareTextSymbol{\isa}{\GRencoding@name}{210}
\DeclareTextComposite{\As}{\GRencoding@name}{i}{210}
\DeclareTextSymbol{\isg}{\GRencoding@name}{211}
\DeclareTextComposite{\Gs}{\GRencoding@name}{i}{211}
\DeclareTextSymbol{\ic}{\GRencoding@name}{216}
\DeclareTextSymbol{\irc}{\GRencoding@name}{217}
\DeclareTextComposite{\Cr}{\GRencoding@name}{i}{217}
\DeclareTextSymbol{\isc}{\GRencoding@name}{218}
\DeclareTextComposite{\Cs}{\GRencoding@name}{i}{218}
\DeclareTextSymbol{\id}{\GRencoding@name}{240}
\DeclareTextSymbol{\idg}{\GRencoding@name}{241}
\DeclareTextComposite{\Gd}{\GRencoding@name}{i}{241}
\DeclareTextSymbol{\ida}{\GRencoding@name}{242}
\DeclareTextComposite{\Ad}{\GRencoding@name}{i}{242}
\DeclareTextSymbol{\idc}{\GRencoding@name}{243}
\DeclareTextComposite{\Cd}{\GRencoding@name}{i}{243}
\DeclareTextSymbol{\ug}{\GRencoding@name}{204}
\DeclareTextSymbol{\ur}{\GRencoding@name}{205}
\DeclareTextComposite{\r}{\GRencoding@name}{u}{205}
\DeclareTextSymbol{\us}{\GRencoding@name}{206}
\DeclareTextComposite{\s}{\GRencoding@name}{u}{206}
\DeclareTextSymbol{\urg}{\GRencoding@name}{207}
\DeclareTextComposite{\Gr}{\GRencoding@name}{u}{207}
\DeclareTextSymbol{\ua}{\GRencoding@name}{212}
\DeclareTextSymbol{\ura}{\GRencoding@name}{213}
\DeclareTextComposite{\Ar}{\GRencoding@name}{u}{213}
\DeclareTextSymbol{\usa}{\GRencoding@name}{214}
\DeclareTextComposite{\As}{\GRencoding@name}{u}{214}
\DeclareTextSymbol{\usg}{\GRencoding@name}{215}
\DeclareTextComposite{\Gs}{\GRencoding@name}{u}{215}
\DeclareTextSymbol{\uc}{\GRencoding@name}{220}
\DeclareTextSymbol{\urc}{\GRencoding@name}{221}
\DeclareTextComposite{\Cr}{\GRencoding@name}{u}{221}
\DeclareTextSymbol{\usc}{\GRencoding@name}{222}
\DeclareTextComposite{\Cs}{\GRencoding@name}{u}{222}
\DeclareTextSymbol{\ud}{\GRencoding@name}{244}
\DeclareTextSymbol{\udg}{\GRencoding@name}{245}
\DeclareTextComposite{\Gd}{\GRencoding@name}{u}{245}
\DeclareTextSymbol{\uda}{\GRencoding@name}{246}
```

```
\DeclareTextComposite{\Ad}{\GRencoding@name}{u}{246}
\DeclareTextSymbol{\udc}{\GRencoding@name}{247}
\DeclareTextComposite{\Cd}{\GRencoding@name}{u}{247}
\DeclareTextSymbol{\eg}{\GRencoding@name}{224}
\DeclareTextSymbol{\er}{\GRencoding@name}{225}
\DeclareTextComposite{\r}{\GRencoding@name}{e}{225}
\DeclareTextSymbol{\es}{\GRencoding@name}{226}
\DeclareTextComposite{\s}{\GRencoding@name}{e}{226}
\DeclareTextSymbol{\erg}{\GRencoding@name}{227}
\DeclareTextComposite{\Gr}{\GRencoding@name}{e}{227}
\DeclareTextSymbol{\ea}{\GRencoding@name}{232}
\DeclareTextSymbol{\era}{\GRencoding@name}{233}
\DeclareTextComposite{\Ar}{\GRencoding@name}{e}{233}
\DeclareTextSymbol{\esa}{\GRencoding@name}{234}
\DeclareTextComposite{\As}{\GRencoding@name}{e}{234}
\DeclareTextSymbol{\esg}{\GRencoding@name}{235}
\DeclareTextComposite{\Gs}{\GRencoding@name}{e}{235}
\DeclareTextSymbol{\oR}{\GRencoding@name}{229}
\DeclareTextComposite{\r}{\GRencoding@name}{o}{229}
\DeclareTextSymbol{\oG}{\GRencoding@name}{228}
\DeclareTextSymbol{\os}{\GRencoding@name}{230}
\DeclareTextComposite{\s}{\GRencoding@name}{o}{230}
\DeclareTextSymbol{\org}{\GRencoding@name}{231}
\DeclareTextComposite{\Gr}{\GRencoding@name}{o}{231}
\DeclareTextSymbol{\oa}{\GRencoding@name}{236}
\DeclareTextSymbol{\ora}{\GRencoding@name}{237}
\DeclareTextComposite{\Ar}{\GRencoding@name}{o}{237}
\DeclareTextSymbol{\osa}{\GRencoding@name}{238}
\DeclareTextComposite{\As}{\GRencoding@name}{o}{238}
\DeclareTextSymbol{\osg}{\GRencoding@name}{239}
\DeclareTextComposite{\Gs}{\GRencoding@name}{o}{239}
\DeclareTextSymbol{\rr}{\GRencoding@name}{251}
\DeclareTextComposite{\r}{\GRencoding@name}{r}{251}
\DeclareTextSymbol{\rs}{\GRencoding@name}{252}
\DeclareTextComposite{\s}{\GRencoding@name}{r}{252}
\DeclareTextSymbol{\Id}{\GRencoding@name}{219}
\DeclareTextSymbol{\Ud}{\GRencoding@name}{223}
\DeclareTextComposite{\"}{\GRencoding@name}{U}{223}
\fi

%
```

5.8 Text philological symbols and macros

Next come some short macros for inserting special symbols that philologists use quite often in Greek.

`\h` Macro `\h` is used to insert a Latin "h" while typesetting in Greek.

`\q` Macro `\q` is used to insert a Latin "q" while typesetting in Greek.

`\yod` `\iod` Macros `\yod` and `\iod` are used to insert a Latin "j" while typesetting in Greek; the control sequence `\jod` was avoided in order to reduce the possibility of typing `\jot` which is a TeX internal dimension.

```
\DeclareTextCommand{\h}{\GRencoding@name}%
    {{\fontencoding{OT1}\selectfont h}}
\DeclareTextCommand{\q}{\GRencoding@name}%
    {{\fontencoding{OT1}\selectfont q}}
\DeclareTextCommand{\yod}{\GRencoding@name}%
    {{\fontencoding{OT1}\selectfont j}}%
\let\iod\yod

```

`\f` `\F` `\digamma` `\Digamma` At the same time it was believed that for inserting lower and upper case "digamma" it was preferable to use short macros and to avoid the dilemma between the `\ddigamma` and the `\digamma` macros, the former being the one defined in the `greek` option to babel, the latter being a standard mathematical symbol; initially I believed that philologists do not use mathematical symbols so we made `\digamma` an alias for `\f`; afterwards I found out that mathematicians, physicists, engineers, ... use the teubner.sty package and that the `\digamma` is a symbol already defined in the package amssymb.sty; therefore I made a conditional creation of this alias; this trick is delayed to the beginning of the document, so as to make it independent on the order with which packages are loaded.

```
\DeclareTextSymbol{\f}{\GRencoding@name}{147}
\AtBeginDocument{\@ifpackageloaded{amssymb}%
{\let\AMSdigamma\digamma\def\digamma{\textormath{\f}{\AMSdigamma}}}% amssymb loaded
{\let\digamma\f}% amssymb not loadedloaded
}
\DeclareTextSymbol{\F}{\GRencoding@name}{195}\let\Digamma\F

```

`\fLow` `\fHigh` The digamma glyphs set forth another question because, according to Paolo Ciacchi, a different glyph should be used for typesetting text compared with the one that is used as a variant in Milesian numerals in place of the standard stigma symbol. By means of macros `\fLow` or `\fHigh` it is possible to chose the raised or the lowered digamma glyphs; Greek numerals always use the lowered one, while when text is being typeset the typesetter can chose the version he likes best.

```
\DeclareRobustCommand{\fLow}%
    {{\setbox\z@\hbox{\f}\dimen@\ht\z@
    \advance\dimen@-1ex\raise-\dimen@\hbox{\box\z@}}}
\DeclareRobustCommand{\fHigh}%
    {{\setbox\z@\hbox{\f}\dimen@\dp\z@\raise\dimen@\hbox{\box\z@}}}

```

`\qmark` `\lpar` `\rpar` `\frapar` Here we start a set of miscellaneous macros. We begin with some parentheses that should turn out in upright shape, even if the default font is the Lipsian one which is oblique; its parentheses are oblique as in all oblique fonts, therefore we need to quietly change fonts behind the scenes. The same is true with the question mark that, philologically speaking, represents an uncertain element, not the termination of a real question; it should therefore always come out between parentheses and in upright shape from a Latin font. While the parenthesized question mark comes from the OT1 Latin upright font, the parentheses obtained with `\lpar` and `\rpar` are taken from the metric symbols font, as well as the parentheses used in the parenthesized text processed with macro `\frapar`.

```
\DeclareRobustCommand\qmark{\hskip.16ex{\fontencoding{OT1}\upshape(?)}}
\DeclareRobustCommand\lpar{{\metricsfont(}}
\DeclareRobustCommand\rpar{{\metricsfont)}}
\DeclareRobustCommand\frapar[1]{\lpar#1\rpar}

```

`\ap` The apex/superscript macro `\ap` does not differ much from the plain standard LaTeX macro `\textsuperscript`, the only difference being the italic correction that precedes `\textsuperscript`.

```
\DeclareRobustCommand{\ap}[1]{\/\textsuperscript{#1}}

```

`\Dots` `\DOTS` `\Dashes` `\DASHES` Four macros are defined so as to insert a certain number of dots or dashes as specified in the optional command argument; `\Dots` and `\Dashes` fit the dots or the dashes pretty close together, while `\DOTS` and `\DASHES` fit them more loosely apart.

```
\newcommand\Dots[1][1]{{\count255=#1\@whilenum\count255>\z@
        \do{\kern.4ex\d{v}\kern.4ex\advance\count255\m@ne}}}
\newcommand\DOTS[1][1]{{\count255=#1\@whilenum\count255>\z@
        \do{\kern.8ex\d{v}\kern.8ex\advance\count255\m@ne}}}
\newcommand\Dashes[1][1]{{\count255=#1\@whilenum\count255>\z@
        \do{\kern.4ex--\kern.4ex\advance\count255\m@ne}}}
\newcommand\DASHES[1][1]{{\count255=#1\@whilenum\count255>\z@
        \do{\kern.8ex--\kern.8ex\advance\count255\m@ne}}}

```

`\:` `\;` `\?` `\MutPers` Greek text or poetry sometimes requires some stacked dots; here we prepared macros for two (`\:`), three (`\;`), and four (`\?`) stacked dots. Two stacked dots in a row indicate that the speaker of a drama or comedy has changed (*mutatio personae*). For `\:` and `\;` it is necessary to preserve the mathematical meaning, while `\?` apparently does not have any previous use in standard LaTeX. The real macros are `\tw@dots`, `\thre@dots`, and `\f@urdots`.

```
\DeclareRobustCommand{\:}{\textormath{\tw@dots}{\mskip\medmuskip}}
\DeclareRobustCommand{\;}{\textormath{\thre@dots}{\mskip\thickmuskip}}
\DeclareRobustCommand{\?}{\f@urdots}
\DeclareRobustCommand{\mutpers}{\makebox[1ex]{\:\hfill\:}\space}
\let\MutPers\mutpers\let\antilabe\mutpers
\def\tw@dots{\mbox{\kern1\p@\vbox to1ex{\hbox{.}\vss\hbox{.}}}}
\def\thre@dots{\mbox{\kern1\p@\vbox to 2ex{\hbox{.}\vss
    \hbox{.}\vss\hbox{.}}}}
\def\f@urdots{\mbox{\kern1\p@\vbox to 2ex{\hbox{.}\vss
    \hbox{.}\vss\hbox{.}\vss\hbox{.}}}}

```

\| \dBar \tBar Similarly Greek text and poetry require certain *cesurae* indicated with vertical bars; we provided commands for one (`\|`), two (`\dBar`), and three (`\tBar`) vertical bars.

```
\DeclareRobustCommand{\|}{\relax\ensuremath{\mskip2mu\vert}}
\DeclareRobustCommand{\dBar}{\ensuremath{\vert\vert}}
\DeclareRobustCommand{\tBar}{\ensuremath{\vert\vert\vert}}

```

\negthinspace \posthinspace \posthindspace \, \! The following are mostly service macros for adjusting the spacing within macro definitions. Nevertheless they are available also to the typesetter, because sometimes certain glyph combinations require a little adjustment. Of course the typesetter will not use them at the very beginning, but only during proof revision, so as to introduce them only where really necessary.

```
\def\negthinspace{\nobreak\hskip-0.07em}
\def\posthinspace{\nobreak\hskip0.07em}
\def\posthindspace{\nobreak\hskip0.14em}
\renewcommand{\,}{\textormath{\posthinspace}{\mskip\thinmuskip}}
\renewcommand{\!}{\textormath{\negthinspace}{\mskip-\thinmuskip}}

```

\lbrk \rbrk \lmqi \rmqi \lmqs \rmqs \mqi \mqs \Ladd \LLadd \ladd \lladd \lesp \ldel Philologists require a certain number of special parentheses in order to enclose parts of text that are doubtful or that have been added although they are missing from the original manuscripts; even letter strings that have been modified under the assumption that the copyist made some error. Such enclosing marks include angle brackets, square brackets, upper part of square brackets, lower part of square brackets. Such symbols may even appear doubled. Most of these glyphs have been designed anew, because they are missing or are inadequate if they are taken from the usual CM fonts (either text or math fonts). Brackets for example have been designed as to be higher and deeper than the font total height, so as not to interfere with Greek accents and to accomodate for at least one level of nesting (for example square brackets enclosing lower part of square brackets. The single glyphs may be used directly by the typesetter, but we think that the commands requiring some text are far more useful. `\Ladd` and its double version `\LLadd` enclose text that should be added for sure. `\ladd` and its double version `\lladd` enclose text that probably should be added. `\lesp` and its synonymous `\ldel` enclose text that should be deleted. `\mqi` surrounds some text with the lower part of open and closed square brackets. `\mqs` surrounds some text with the upper part of open and closed square brackets. See `teubenr-doc.pdf` for samples of such commands.

```
\DeclareRobustCommand{\lbrk}{{\metricsfont\posthindspace[\negthinspace}}
\DeclareRobustCommand{\rbrk}{{\metricsfont]}}
\DeclareRobustCommand\lmqi{{\metricsfont!}}
\DeclareRobustCommand\rmqi{{\metricsfont:}}
\DeclareRobustCommand\lmqs{{\metricsfont?}}
\DeclareRobustCommand\rmqs{{\metricsfont;}}
\DeclareRobustCommand\mqi[1]{\posthinspace\lmqi\negthinspace
    {#1\/}\rmqi}\let\mezzeq\mqi
\DeclareRobustCommand\mqs[1]{\lmqs{#1\/}\rmqs}
```

```
\DeclareRobustCommand{\Ladd}[1]{{\metricsfont<}{\!\!#1\/}%
    {\metricsfont>}}%                              litterae certe addendae
\DeclareRobustCommand{\LLadd}[1]{{\metricsfont<\kern-.3ex<}
    {\!\!#1\/}{\metricsfont>\kern-.3ex>}}%     litterae certe addendae
\DeclareRobustCommand{\ladd}[1]{{\metricsfont\kern.15ex[\negthinspace}%
    {#1\/}{\metricsfont]\kern-.15ex}}%         litterae addendae
\DeclareRobustCommand{\lladd}[1]{{\metricsfont\kern.15ex[\kern-.3ex[%
    \negthinspace}{#1\/}{\metricsfont]\kern-.3ex]%
    \kern-.15ex}}%                             litterae addendae
\DeclareRobustCommand{\lesp}[1]%
    {\mbox{$\{\kern-.20ex$#1\kern.16ex$\}$}}%   litterae delendae
\let\ldel\lesp

```

5.9 Greek, English, and German quotes

\itopenquotes \itclosedquotes \itoq \itcq

The following macros allow to set Italian/English high quotes even while typing in Greek; such quotes are standard in Italian and in English typesetting and their commands preserve the font family shape and series of the surrounding font. In French typography, as well in the typographic traditions of other countries, different quotes are used. In that case the typesetter must resort to a change of language, for example returning to German, inputting the German quotes, then turning back to Greek. He might as well define his own macros, or he might clone the following definitions and change them according to his country typographic traditions. If he decides to modify these definitions he should either rename this file or he should put his redefinitions in a private package to be input *after* `teubner.sty`.

```
\DeclareTextCommand{\itopenquotes}{\GRencoding@name}%
    {{\fontencoding{OT1}\selectfont\char92}}%
\DeclareTextCommand{\itclosedquotes}{\GRencoding@name}%
    {{\fontencoding{OT1}\selectfont\char34}}%
\let\itoq\itopenquotes
\let\itcq\itclosedquotes

```

\GEodq \GEcdq \GEdqtext \GEoq \GEcq \GEqtext \ENodq \ENcdq \ENdqtext

On the opposite the following German and English quotes are redesigned and included in the metric symbols font. Since this font is in one shape and one series, these quotes do not change as the outside font does, but remain fixed; the most useful commands are `\GEdqtext` for enclosing some text within German double quotes, `\GEqtext` for enclosing some text within German single quotes, and `\ENdqtext` for enclosing some text in English double quotes. Apparently while setting Greek poetry in stacked, possibly enumerated, verses, German double or single quotes are often used, since they cannot be misunderstood with Greek diacritical marks. Modern Greek double quotes apparently are not appreciated by philologists, at least outside Greece.

```
\newcommand\GEodq{\bgroup\futurelet\@tempA\GE@dq}
\def\GE@dq{{\metricsfont\char18}\ifx\@tempA m\posthinspace\fi\egroup}
```

```
\newcommand\GEcdq{{\metricsfont\char16}}
\newcommand\GEdqtext[1]{\GEodq\posthinspace#1\/\posthinspace\GEcdq}
\newcommand\GEoq{\bgroup\futurelet\@tempA\GE@q}
\def\GE@q{{\metricsfont\char13}\ifx\@tempA m\posthinspace\fi\egroup}
\newcommand\GEcq{{\metricsfont\char19}}
\newcommand\GEqtext[1]{\GEoq\posthinspace#1\/\posthinspace\GEcq}
\newcommand\ENodq{{\metricsfont\char16}}
\newcommand\ENcdq{{\metricsfont\char17}}
\newcommand\ENdqtext[1]{\ENodq\negthinspace#1\/\posthinspace\ENcdq}

```

5.10 Other philological symbols and macros

`\LitNil` `\litnil` The next synonymous macros indicate the *littera nihil.*

```
\DeclareRobustCommand\LitNil{\textbullet}
\let\litnil\LitNil
```

`\sva` `\shva` `\shwa` The CB fonts include also the letter “shwa”, the glyph that appears as a roman “e” rotated 180° around its center. Philologists need it even when writing Greek. In order to make it available also when the Latin encodings are in force, suitable definitions have been given so that the suitable CB font was changed behind the scenes without any intervention by the typesetter. With this version of `teubner.sty` a new definition is made up that uses the `\rotatebox` facility of the `graphicx` package; In a future revision of the CB fonts the `\schwa` slot shall be freed so that Greek glyphs only populate it, without extraneous presences. The `\schwa` glyph is made available also with the Latin encodings.

```
%\DeclareTextSymbol{\sva}{\GRencoding@name}{26}
\DeclareTextCommand{\sva}{\GRencoding@name}{%
\rotatebox[origin=c]{180}{\def\@tempA{li}%
\fontencoding{OT1}\ifx\f@shape\@tempA\fontshape{it}\fi\selectfont e}}
\DeclareTextCommand\sva{OT1}{{\expandafter\fontencoding
    \expandafter{\GRencoding@name}\selectfont\sva}}
\DeclareTextCommand\sva{T1}{{\expandafter\fontencoding
    \expandafter{\GRencoding@name}\selectfont\sva}}
\let\shva\sva\let\shwa\sva

```

`\skewstack` The `\skewstack` command stacks two arguments not one on top of the other, but the second argument is placed to the right and upwards relative to the first argument. The second argument is set in script font size. Although there are similarities with the `\textsuperscript` command, the exact placement of the second argument depends on the shape (height and depth) of both arguments. This command will be used for creating some philologist’s symbols, but is readily available to the typesetter both for direct use and for writing macros defining new symbols.

```
\DeclareRobustCommand\skewstack[2]{{%
\edef\slant@{\strip@pt\fontdimen1\font}%
\setbox\z@\hbox{#1}\dimen@\ht\z@\box\z@
\kern-.045em\setbox\@ne\hbox{\scriptsize#2}%
\ifdim\dimen@>1.2ex\advance\dimen@-\ht\@ne\else
    \dimen@1ex\advance\dimen@-.5\ht\@ne\fi
\kern\slant@\dimen@\raise\dimen@\hbox{\box\@ne}}}

```

`\hv` `\qw` `\gw` `\gusv` `\qusv` `\qu` Matter of fact some common Latin stacked symbols are defined here in terms of `\skewstack`. As it may bee seen, the second argument (the first as well, but here there are no examples) may in turn contain other macros for composite symbols.

```
\DeclareRobustCommand\hv{{\fontencoding{OT1}\selectfont
    \skewstack{h}{v}}}
\DeclareRobustCommand\qw{{\fontencoding{OT1}\selectfont
    \skewstack{q}{w}}}
\DeclareRobustCommand\gw{{\fontencoding{OT1}\selectfont
    \skewstack{g}{w}}}
\DeclareRobustCommand\gusv{{\fontencoding{OT1}\selectfont
    \skewstack{g}{\semiv{u}}}}
\DeclareRobustCommand\qusv{{\fontencoding{OT1}\selectfont
    \skewstack{q}{\semiv{u}}}}
\DeclareRobustCommand\qu{{\fontencoding{OT1}\selectfont
    \skewstack{q}{u}}}

```

`\dz` Without using `\skewstack` other symbols may be defined; here `\dz` is just an example, where the kerning between 'd' and 'z' has been found by cut and try. With other glyphs may be different kerning is necessary.

```
\DeclareRobustCommand\dz{{\fontencoding{OT1}\selectfont d\kern-.33ex z}}

```

Now we come to another set of commands like the ones needed to mark the syneresis or the zeugma and other similar marks.

`\Utie` This first macro sets a "smile" symbol under a couple of letters. The glyph is fine but is good only for two adjacent letters, therefore it is necessary to have a stretchable symbol.

```
\DeclareRobustCommand\Utie[1]{%
\mbox{\vtop{\ialign{##\crcr
   \hfil#1\hfil\crcr
   \noalign{\kern.3ex\nointerlineskip}%
   \hfil$\smile$\hfil\crcr}}}}

```

`\siner` `\siniz` This is why the `\siner` and `\siniz` synonymous commands have been defined; in place of or in addition to the "smile" symbol; they contain a stretchable filler `\upfill` that behaves almost as the stretchable horizontal brace that is used in the definition of the LaTeX commands `\underbrace` or `\overbrace`.

```
\DeclareRobustCommand{\siner}[1]{%
\mbox{\vtop{\ialign{##\crcr
   \hfil#1\hfil\crcr
```

```
    \noalign{\kern.6ex\nointerlineskip}%
    \upfill\crcr}}}}
\let\siniz\siner

```

\upfill The `\upfill` is defined as a leader, the same way as the corresponding LaTeX stretchable horizontal brace.

```
\def\upfill{$\m@th \scriptstyle\setbox\z@\hbox{$\scriptstyle\bracelu$}%
  \kern.16ex\bracelu\ifPDF\kern-.15ex\fi
  \leaders\vrule \@height\ht\z@ \@depth\z@\hfill
  \braceru\kern.16ex$}

```

\downfill The `\downfill` arc is totally similar to the `\upfill` one, except for its terminating elements that change the shape of the arc from “up” to “down”.

```
\def\downfill{$\m@th\scriptstyle\setbox\z@\hbox{$\scriptstyle\braceld$}%
  \kern.16ex\braceld\ifPDF\kern-.15ex\fi
  \leaders\vrule \@height\ht\z@ \@depth\z@\hfill
  \bracerd\kern.16ex$}

```

\zeugma Similarly `\zeugma` puts a stretchable arc over its argument; it must take into account the slant of the argument font so as to skew the placement of the arc.

```
\newcommand*\zeugma[1]{{\vbox{\setbox\z@\hbox{#1}\dimen@=\ht\z@
      \edef\@slant{\strip@pt\fontdimen1\font}%
      \dimen\tw@=\wd\z@
      \dimen@=\@slant\dimen@\ifmetricsfont\dimen@=\z@
        \advance\dimen\tw@-.5ex\fi
      \kern-.2ex\ialign{##\crcr
      \hbox to\z@{\ifmetricsfont\kern.25ex\fi\kern\dimen@
      \hbox to\dimen\tw@{\hss\downfill\kern.2\dimen@\hss}\hss}\crcr
      \noalign{\ifmetricsfont\kern.6ex
            \else\kern.4ex\fi\nointerlineskip}%
      \hfil{#1}\hfil\crcr}}}%
}

```

\slzeugma
\rszeugma Although the shape of oblique zeugma arcs cannot be changed depending on the width and height of the zeugma argument, in certain circumstances the philologists want to use oblique zeugma marks. This is why we defined a “sloping zeugma arc” `\slzeugma`, and a “rising zeugma arc” `\rszeugma` that can be used with poor results, if such arcs are superimposed over the “wrong” letters. There is nothing automatic in the choice of the oblique arc and is totally on the typesetter responsibility to use the correct command. These slanted zeugma signs are possibly useful only for two letters since they are not stretchable.

```
\newcommand*\slzeugma[1]{{\leavevmode
    \setbox\tw@\hbox{\metricsfont\char120}%
    \setbox\z@\hbox{#1}\dimen@.5\wd\z@\advance\dimen@-.5\wd\tw@
    \edef\@slant{\strip@pt\fontdimen1\font}%
    \advance\dimen@\@slant\ht\z@
    \hbox to\z@{\kern\dimen@\box\tw@\hss}\box\z@
    }%
}

\newcommand*\rszeugma[1]{{\leavevmode
    \setbox\tw@\hbox{\metricsfont\char122}%
    \setbox\z@\hbox{#1}\dimen@.5\wd\z@\advance\dimen@-.5\wd\tw@
    \edef\@slant{\strip@pt\fontdimen1\font}%
    \advance\dimen@\@slant\ht\z@
    \hbox to\z@{\kern\dimen@\box\tw@\hss}\box\z@
    }%
}

```

`\nexus` `\nesso` Originally I had two different macros for marking a *nexus*; one made use of a "up stretchable turtle bracket", and the user used a leader of Latin circumflex signs. Both were unsatisfactory; the latter was really ugly, but I kept the macro name as a synonym for compatibility with the past. The good looking marker is obtained from a mathematical `\widehat` sign by stretching it to the width of the string the marcher should mark; the new macro `\nexus` (that replaces the stretchable turtle bracket) relies on the facilities offered by the `\resizebox` of the package `graphicx`.

```
\newcommand*{\nexus}[1]{{\setbox\tw@\hbox{#1\/}%
        \edef\slant@{\strip@pt\fontdimen1\font}%
        \@tempdima=\slant@\ht\tw@\advance\@tempdima.45ex
        \setbox4\hbox{\resizebox{\wd\tw@}{\height}{$\widehat{\phantom{aaa}}$}}%
        \setbox4\hbox{\smash{\lower1.35ex\hbox{\box4}}}%
        \vbox{\ialign{##\crcr%
        \kern\@tempdima\box4%
        \crcr
        \noalign{\kern.15ex\nointerlineskip}%
        \hfil{#1}\hfil\crcr}}}}
\let\nesso\nexus

```

`\coronis` `\Coronis` `\paragr` `\dpar` While setting poetry it is necessary to mark the end of paragraphs, which do not necessarily coincide with the ends of stanzas. After the verse that concludes a logical paragraph philologists insert a mark called "coronis" (synonymous of paragraph, therefore the command `\paragr`) or a "stronger" mark called "Coronis", which differs from the common "coronis" because it bears an inverted semilunar sign on its left. Both marks are input by means of their respective commands `\paragr` (preferred to `\coronis`) or `\Coronis` inserted *at the beginning of the paragraph terminating verse.* The command `\dparagr` inserts a double coronis mark, which is sometimes required in place of the ordinary single mark.

```
\def\C@rule{\vrule\@height.45ex\@depth-.35ex\@width1.5em}
\def\coronis@rule{\hbox to\z@{\hss\C@rule\hss}}
```

```
\def\Coronis@rule{\hbox to\z@
    {\hss\hbox to\z@{\hss$\scriptstyle)$\kern-1.5\p@}\C@rule\hss}}
\DeclareRobustCommand\paragr{\raisebox{-1ex}[\z@][\z@]{\coronis@rule}}
\let\coronis\paragr
\DeclareRobustCommand\Coronis{\raisebox{-1ex}[\z@][\z@]{\Coronis@rule}}
\DeclareRobustCommand{\dparagr}%
    {\raisebox{-1.3ex}[\z@][\z@]{\coronis@rule}%
    \raisebox{-1.6ex}[\z@][\z@]{\coronis@rule}}

```

`\sinafia` `\crux` `\FinisCarmen` `\apici` `\positio` `\Int` `\star` `\dstar` `\tstar` `\responsio`

The next group of commands are intended to insert special symbols in the philological text; just the command `\apici` requires an argument, a block of text that shall be enclosed within straight vertical apices, irrespective of the font slant. The command `\FinisCarmen` although very descriptive, is long to type, therefore a shorter alias `\FinCar` has been defined. `\apex` was the initial name given to the command, but on a second time it was changed to `\positio`, and the latter should always be used in place of the former. For what concerns `\star` which is a standard LaTeX math command, the original definition is saved in the service macro `\m@thst@r` and the command is redefined so as to perform as it should both in text and in math mode. The symbol $\int$, on the contrary, was redefined so as not to mix math with text, even if its rendering resorts to mathematics.

```
\DeclareRobustCommand*\sinafia{{\metricsfont s}}
\DeclareRobustCommand*{\crux}{{\metricsfont\char'171}}
\DeclareRobustCommand*{\FinisCarmen}{\ensuremath{\otimes}}
\let\FinCar\FinisCarmen
\DeclareRobustCommand*{\apici}[1]%
    {\posthinspace{\metricsfont\char96}\negthinspace#1%
    \posthinspace{\metricsfont\char39}\negthinspace}
\DeclareRobustCommand*{\apex}%
    {\/\hskip.5ex\vrule\@height1.7ex\@depth-1ex\hskip.2ex}
\let\positio\apex
\DeclareRobustCommand*{\Int}{\ensuremath{\int}}
\let\m@thst@r\star
\DeclareRobustCommand*{\star}{\textormath{{{\upshape *}}}{\m@thst@r}}
\DeclareRobustCommand*{\dstar}{{\upshape **}}
\DeclareRobustCommand*{\tstar}{{\upshape ***}}
\DeclareRobustCommand*{\responsio}{{\boldmath\ensuremath{\sim}}}

```

`\thorn` `\Thorn`

`\thorn` and `\Thorn` are the exact equivalents of `\th` and `\Th` that are defined only for the T1 encoding. Therefore such encoding is selected in an implicit way.

```
\DeclareRobustCommand{\thorn}{{\fontencoding{T1}\selectfont\th}}
\DeclareRobustCommand{\Thorn}{{\fontencoding{T1}\selectfont\TH}}

```

5.11 Ancient Greek monetary unit symbols

`\dracma` `\hemiobelion` `\tetartemorion` `\stater` `\denarius` `\etos`

This set of symbols, taken from the metrics symbol font (which by this time is evident does not contain only metrics symbols) represents the unit symbols of some coins of ancient Greece, as they were found on many “ostraka” in several archeological sites.

```
\DeclareRobustCommand{\dracma}{{\metricsfont D}}
\DeclareRobustCommand{\hemiobelion}{{\metricsfont A}}
\DeclareRobustCommand{\tetartemorion}{{\metricsfont B}}
\DeclareRobustCommand{\stater}{{\metricsfont C}}
\DeclareRobustCommand{\denarius}{{\metricsfont E}}
\DeclareRobustCommand{\etos}{{\metricsfont G}}

```

5.12 Another set of philological symbols and macros

`\cut` `\dcutbar` `\bcutbar` `\gcutbar`

The following set of macros are all connected with the principal macro `\cut`, which should position a horizontal tie or bar across a certain number of latin letters, specifically ‘d’, ‘b’, and ‘g’; due to their different shapes, such bars are of different length and located at different heights; if they are in italics the bar position must change again. Therefore even if the user command `\cut` is the same for all these letters, its action must change depending on different circumstances. It merely checks its argument (it must be *one* letter and unpredictable results are obtained if more that one token is passed as an argument to `\cut`) and selects the proper bar. The specific bar commands `\dcutbar`, `\bcutbar`, and `\gcutbar`, are defined in such a way as to cope only with the their initial letter.

```
\DeclareRobustCommand{\cut}[1]{%
    \ifx#1d\dcutbar\else
        \ifx#1b\bcutbar\else
            \ifx#1g\gcutbar
            \fi
        \fi
    \fi}
%
\def\dcutbar{{\edef\slant@{\strip@pt\fontdimen1\font}%
    d\dimen@1.2ex\kern\slant@\dimen@
    \llap{\vrule\@height1.3ex\@depth-\dimen@
    \ifdim\slant@\p@>\z@\@width.35em\else\@width.4em\fi\kern.03em}}}
\def\bcutbar{{\edef\slant@{\strip@pt\fontdimen1\font}%
    \rlap{\dimen@1.2ex\kern\slant@\dimen@
    \ifdim\slant@\p@=\z@\kern.03em\fi
    \vrule\@height1.3ex\@depth-\dimen@
    \ifdim\slant@\p@>\z@\@width.3em\else\@width.4em\fi}b}}
\def\gcutbar{{\edef\slant@{\strip@pt\fontdimen1\font}%
    \ifdim\slant@\p@>\z@
        g\kern-.55ex\dimen@.2ex\kern-\slant@\dimen@
        \vrule\@height-.1ex\@depth\dimen@\@width.6ex
    \else
        \dimen@.2ex\kern\slant@\dimen@\vrule\@height.3ex\@depth-\dimen@
        \@width.6ex\kern-.55ex\relax g
    \fi}}

```

\OSN The next macro is just a shortcut instead of using `\oldstylenums`.

```
\let\OSN\oldstylenums

```

\splus \stimes \kclick The next three macros are used in glottology; the first two ones are used to mark special pronunciations of the sibilant, while the last one is used to mark a special pronunciation of the guttural that produces a “click”.

```
\newcommand\splus{\leavevmode{%
    \edef\slant@{\strip@pt\fontdimen1\font}%
    \setbox\z@\hbox{s}%
    \dimen@=\wd\z@
    \setbox\tw@\hbox{$\scriptscriptstyle+$}%
    \advance\dimen@.35\ht\tw@
    \raisebox{\dimen@}[\z@][\z@]{%
        \makebox[\z@][l]{\kern.5\wd\z@
         \kern\slant@\dimen@\kern-.5\wd\tw@\box\tw@}}%
    \box\z@}}%
\newcommand\stimes{\leavevmode{%
    \edef\slant@{\strip@pt\fontdimen1\font}%
    \setbox\z@\hbox{s}%
    \dimen@=\wd\z@
    \setbox\tw@\hbox{$\scriptscriptstyle\times$}%
    \advance\dimen@.2\ht\tw@
    \raisebox{\dimen@}[\z@][\z@]{%
        \makebox[\z@][l]{\kern.5\wd\z@
            \kern\slant@\dimen@\kern-.5\wd\tw@\box\tw@}}%
    \box\z@}}%
\newcommand\kclick{\leavevmode{%
    \edef\slant@{\strip@pt\fontdimen1\font}%
    \setbox\z@\hbox{k}%
    \setbox\tw@\hbox{\fontencoding\GRencoding@name\selectfont\s{v}}%
    \dimen@\wd\z@
    \ifdim\slant@\p@=\z@
        \advance\dimen@-.1\wd\z@\else\advance\dimen@\wd\tw@
    \fi
    k\makebox[\z@][r]{\unhcopy\tw@\kern.5\dimen@}%
    }}%

```

5.13 Poetry environments and macros

\verso versi Here we start with verse environments; we already explained that we defined three new verse environments that typeset poetry in “in-line” verses, “numbered by five” verses, and “numbered by five and subnumbered” verses. For the environment `versi` we first need a counter and a little macro for generating the short bar that has to receive the verse number as a “limits” superscript.

```
\newcounter{verso}\setcounter{verso}{0}
\newcommand{\smallvert}{\vrule\@height.6ex\@depth.4ex}

```

Next we define the macro `\verso` that sets the small bar with the verse number on top. Since the initial numbering might be different from 1, `\verso` accepts an optional argument, which is intended to be the initial counter value. Since `\verso` steps up the counter a different action must be taken if the optional argument is present; in order to be able to reference such verse by means of the `\label`–`\ref` cross reference mechanism, this stepping up must be done by means of `\refstepcounter`; therefore we have to leave `\refstepcounter` outside the conditional code, and step down the counter by one unit only in case the initial value is specified.

```
\DeclareRobustCommand\verso[1][]{%
    \def\@tempA{#1}\ifx\@tempA\empty
    \else
        \setcounter{verso}{#1}\addtocounter{verso}{\m@ne}%
    \fi
    \refstepcounter{verso}%
    \@killglue\space
    \ensuremath{\mathop{\smallvert}\limits^{\scriptscriptstyle\theverso}}%
    \space\ignorespaces}

```

Now that the verse separation macro is ready we can define the environment; the required opening statement argument represents a short text whose width is taken as a measure for indentation, so that verses are typeset with a left margin that leaves out this short text. Substantially this environment is a `list` one, and the left margin variable width is totally similar to the one used in `thebibiography` environment. Also the `\makelabel` command has been modified accordingly.

```
\newenvironment{versi}[1]{%
\def\makelabel##1{##1}
\setbox\z@\hbox{#1}%
\list{}{\labelwidth\wd\z@\leftmargin\labelwidth
    \advance\leftmargin\labelsep}%
    \item[\box\z@]
}{%
\endlist
}
\let\versus\versi \let\endversus\endversi
```

`Versi` The second environment `Versi` accepts an optional starting number in the opening, statement, whose default value is 1: verses are composed as in the standard LaTeX `verse` environment (with one minor difference) except they are numbered in the left margin with a progression of five; only verse numbers that are integer multiples of five are displayed. The minor difference is that stanzas cannot be marked with a blank line in the input `.tex` file, as it is customary with the standard environment, but if a visual mark is desired, such as extra vertical space, it is necessary to resort to the optional spacing parameter that can be specified to the `\\` command. This environment uses the same verse counting counter, defined for use with the `versi` environment.

\BreakVersotrue
\BreakVersofalse

For specific purposes it is necessary to have a boolean variable for allowing or prohibiting verses to split up at the end of line; the default is not to split.

```
\newif\ifBreakVersi
\BreakVersifalse
\newenvironment{Versi}[1][1]{%
    \setcounter{verso}{#1}%
```

An internal macro `\writ@verso` does not actually write out the complete, possibly numbered verse, but provides for checking that the verse counter contains a multiple of 5, and to write it out using old stile numbers; in case the number is not an integer multiple of 5 the number is written out as the `\empty` macro.

```
    \def\writ@verso{%
        \count255=\value{verso}\divide\count255by5\relax
        \multiply\count255by5\relax
        \advance\count255-\value{verso}%
        \ifnum\count255=\z@
          {\fontseries{m}\small\expandafter\oldstylenums\expandafter{\the\c@verso}}%
        \else
          \empty
        \fi}%
```

Since the `\\` command should provide the same functionality as the regular LaTeX command, while in this environment it should provide other functionalities, such as triggering the display of the verse number. It is necessary to define an intermediate command `\v@rscr`, that examines the possible optional arguments, such as the optional star or the brackets enclosing vertical spacing

```
    \def\\{\@ifstar{\v@rscr{\@M}}{\v@rscr{\z@}}}%
    \def\v@rscr##1{\@ifnextchar[{\wr@teverse{##1}}%
        {\wr@teverse{##1}[\z@]}}%
```

Finally the `\wr@teverse` macro does the actual typesetting of the verse. Notice that the environment opening statement and every succeeding previous verse starts an horizontal box where the contents of the current verse is stored. Therefore the first thing to do is to close the box with the `\egroup` command, then a line of text is output that contains a possibly empty box or the verse number and the command for stepping up the verse counter, followed by the verse box number 0 and an end of paragraph; in this way the `\\` operates always in vertical mode, contrary to what happens in the `verse` standard LaTeX environment. Even in this environment the actual typesetting is done within a `list` environment, whose parameters are set differently from what they are in the `verse` environment. Notice in any case that the command `\wr@teverse` reopens the 0 box, so on the last verse, upon closing the environment, it is necessary to remember to close such box, whose contents is irrelevant and can be thrown away.

I have experienced some problems in typesetting verses in two-column format; the column width might be too short for setting up verses even if verses are not that long, because in the left margin there must be room for the verse numbering; for homogeneity the spacing must conform also with the following environment `VERSI` that has a secondary verse numbering, therefore it can't be too small. The result is that there might be a test for controlling the two-column format, but I think that it is more useful for the typesetter to be able to switch on and off the possibility of breaking long verses on more lines. On two-column format in any case it is better to leave the right margin to coincide with the column right margin.

```
    \def\wr@teverse##1[##2]{\egroup
        \makebox[3em][r]{%
            \writ@verso\refstepcounter{verso}\kern1.5em}
            \ifBreakVersi
                  \begingroup\raggedright
                  \hyphenpenalty \@M
                  \unhbox\z@\par
                  \endgroup
            \else
                  \rlap{\box\z@}\par
            \fi
        \penalty##1\vskip##2\relax
        \setbox\z@\hbox\bgroup\ignorespaces}%
    \list{}{\itemsep\z@\parsep\z@
    \if@twocolumn
    \itemindent   -5.3em%
    \listparindent\itemindent
      \rightmargin\z@
      \advance\leftmargin 3.3em
    \else
     \itemindent   -1.5em%
    \listparindent\itemindent
     \rightmargin  \leftmargin
      \advance\leftmargin 1.5em
    \fi
    }%
    \item\leavevmode\setbox\z@\hbox\bgroup\ignorespaces
}{%
```

Upon closing it is necessary to activate the writing out of the last verse that is still in the 0 box, but since this box is immediately reopened, it is necessary to close it again before exiting the environment.

```
    \\%
    \egroup
    \endlist
}
\let\Versus\Versi \let\endVersus\endVersi

```

VERSI The third environment VERSI set verses in the traditional way, but numbers them with two different enumerations; the principal one is by multiples of five, while the secondary one counts by units, and may be turned on and off, or reset at will. We therefore need another counter for the secondary enumeration and commands for turning it on and off and for resetting the counter. We need also a new length and a new boolean variable in order to manage the secondary enumeration. The new length represents an indentation of those verses that do no have the secondary enumeration, while secondary enumerated verses are not indented. For `VERSI` there is the same possibility of turning on and off the possibility of breaking verses at the end of line as it happens for the environment `Versi`.

`\SubVerso` `\NoSubVerso` Macro `\NoSubVerso` turns off the secondary enumeration; macro `\SubVerso` turns on the secondary enumeration, but it accepts an optional argument for resetting the secondary counter; the default value is 0; if no optional argument is specified, and therefore if the optional argument has its default value 0, no resetting is performed and the enumeration keeps going from the last contents of the secondary counter; if the first use of `\SubVerso` does not contain the optional argument, the secondary enumeration keeps going from the old contents of the secondary counter which is unpredictable, depending upon the previous occurrences of the environment `VERSI`. The typesetter, therefore, must remember to specify the optional argument to `\SubVerso` the first time he uses it in this environment.

```
\newcounter{subverso} \setcounter{subverso}{0}
\newif\ifSubVerso
\newlength{\versoskip}
\newcommand*\NoSubVerso{\global\SubVersofalse
    \global\versoskip1.3em\ignorespaces}
\newcommand*\SubVerso[1][0]{\global\SubVersotrue
    \ifnum#1=0\else
        \setcounter{subverso}{#1}%
        \global\protected@edef\@currentlabel{\the\c@subverso}%
    \fi
    \global\versoskip.3em\ignorespaces}

```

The opening environment statement accepts an optional argument (default equals 1) which represents the primary enumeration starting number:

```
\newenvironment{VERSI}[1][1]{%
    \setcounter{verso}{#1}%
```

We need two macros `\writ@verso` and `\writ@subverso`, that typeset the primary and secondary enumeration; the first one is similar to the one used in the `Versi` environment, while the second one has no special features except the conditional construct needed to check if the secondary enumeration has to be printed out.

```
    \def\writ@verso{%
        \count255=\value{verso}\divide\count255by5\relax
        \multiply\count255by5\relax
        \advance\count255-\value{verso}%
        \ifnum\count255=0\relax
          {\fontseries{m}\small\expandafter\oldstylenums\expandafter{\the\c@verso}}%
        \else
          \empty
        \fi}%
    \NoSubVerso
    \def\writ@subverso{%
```

```
        \ifSubVerso
            {\fontseries{m}\scriptsize\expandafter\oldstylenums
                \expandafter{\the\c@subverso}}%
        \fi}%
```

Similarly to the previous environment, the `\\` command must be redefined so as to perform more or less as the standard one, while doing all the necessary actions needed in this environment. It must check the presence of the optional star and of the optional vertical skip and it has to pass control to a service macro `\v@rscr` that does the actual job; actually it passes control to a third macro `\writ@verse` that effectively outputs the current verse.

```
    \def\\{\@ifstar{\v@rscr{\@M}}{\v@rscr{\z@}}}%
    \def\v@rscr##1{\@ifnextchar[{\writ@verse{##1}}%
        {\writ@verse{##1}[\z@]}}%
    \def\writ@verse##1[##2]{\egroup
        \makebox[1.5em][r]{\writ@verso\refstepcounter{verso}}%
        \makebox[1.5em][r]{\writ@subverso\refstepcounter{subverso}}%
        \kern1.5ex\hskip\versoskip
            \ifBreakVersi
                 \begingroup
                 \hyphenpenalty \@M
                 \unhbox\z@\par
                 \endgroup
            \else
                 \rlap{\box\z@}\par
            \fi
        \penalty##1\vskip##2\relax
        \setbox\z@\hbox\bgroup\ignorespaces}%
```

For the remaining part, the environment is a normal `list` environment with specific initial parameters.

```
    \list{}{\parsep\z@\itemsep\z@
    \if@twocolumn
    \itemindent   -5.3em%
    \listparindent\itemindent
      \rightmargin\z@
      \advance\leftmargin 3.3em
    \else
     \itemindent   -1.5em%
    \listparindent\itemindent
     \rightmargin  \leftmargin
      \advance\leftmargin 1.5em
    \fi
    }%
    \item\leavevmode\setbox\z@\hbox\bgroup\ignorespaces
}{%
```

The closing statement must output the last verse, which is still contained in box 0; since box 0 is automatically reopened, it must be closed again and its contents, of no significance now, can be lost upon closing the environment group.

```
    \\%
    \egroup\endlist}
\let\VERSUS\VERSI \let\endVERSUS\endVERSUS

```

5.14 Metrics symbols, macros and environmnets

Now we start defining many macros concerned with metrics; the metric symbol font has been developed mainly for this purpose. We start defining some macros for inputting specific symbols; many such macros have their own aliases in Latin.

`\lunga` `\longa` `\breve` `\brevis` `\bbreve` `\bbrevis` `\barbreve` `\barbrevis` `\barbbrev` `\barbbrevis` `\ubarbreve` `\ubarbrevis` `\ubarbbreve` `\ubarbbrevis` `\ubarsbreve` `\ubarsbrevis` `\ubrevelunga` `\ubrevislonga`

The following definitions are straightforward; a small comment on `\breve`: since it is also a math command in standard LaTeX, its meaning is saved in a service macro `\br@ve` and the `\breve` macro is redefined taking into account whether the typesetting is being done in text or in math mode. The unusual letters that appear in the definitions of the various metric symbols have no mysterious meaning; they might have been specified by `\char`⟨*number*⟩, but it seemed shorter to specify the corresponding letters that would occupy the same slots in literal fonts.

```
\DeclareRobustCommand\lunga{{\metricsfont l}}
\let\longa\lunga
\let\br@ve\breve
\DeclareRobustCommand\breve{\textormath{{{\metricsfont b}}}{\br@ve}}
\let\brevis\breve
\DeclareRobustCommand\bbreve{{\metricsfont c}}
\let\bbrevis\bbreve
\DeclareRobustCommand\barbreve{{\metricsfont  i}}
\let\barbrevis\barbreve
\DeclareRobustCommand\barbbreve{{\metricsfont j}}
\let\barbbrevis\barbbreve
\DeclareRobustCommand\ubarbreve{{\metricsfont d}}
\let\ubarbrevis\ubarbreve
\DeclareRobustCommand\ubarbbreve{{\metricsfont e}}
\let\ubarbbrevis\ubarbbreve
\DeclareRobustCommand\ubarsbreve{{\metricsfont f}}
\let\ubarsbrevis\ubarsbreve
\DeclareRobustCommand{\ubrevelunga}{{\metricsfont\char107}}
\let\ubrevislonga\ubrevelunga

```

`\corona` `\ElemInd` `\coronainv` `\catal` `\ipercatal` `\hiatus` `\Hiatus` `\X` `\anceps` `\banceps` `\ancepsdbrevis` `\aeolicbii` `\aeolicbiii` `\aeolicbiv`

Similarly the following symbols have straightforward definitions. Only `\hiatus` and `\Hiatus` require a small explanation; `\hiatus` inserts a small capital 'H' in superscript position; in a first moment it was chosen the solution of designing a specific sans serif glyph in superscript position directly in the metric symbol font (actually this symbol is still part of the font), but while testing it, Paolo Ciacchi observed that a regular 'H' with serifs was better looking than the sans serif counterpart. Therefore the definition was changed in order to use the current font upright shape; by specifying 'H', it is irrelevant if the current one is a Latin font, and the letter is a capital 'h', or if the current one is a Greek font and the letter is a capital 'eta'. \Hiatus displays the same symbol in a zero width box so that it does not occupy any horizontal space; it is useful while writing down complicated metric sequences. Macro\X may be considered, thanks to its shape, a mnemonic shortcut in place of the full name \anceps.

```
\DeclareRobustCommand\corona{{\metricsfont\char20}}
\let\ElemInd\corona
\DeclareRobustCommand\coronainv{{\metricsfont\char21}}
\DeclareRobustCommand\catal{{\metricsfont g}}
\DeclareRobustCommand\ipercatal{{\metricsfont h}}
\DeclareRobustCommand\hiatus{\textsuperscript{\upshape H}}
\DeclareRobustCommand\Hiatus{\makebox[\z@]{\hiatus}}
\DeclareRobustCommand\X{{\metricsfont X}}
\let\anceps\X
\DeclareRobustCommand\banceps{{\metricsfont Y}}
\DeclareRobustCommand\ancepsdbrevis{{\metricsfont Z}}
\DeclareRobustCommand{\aeolicbii}{{\metricsfont I}}
\DeclareRobustCommand{\aeolicbiii}{{\metricsfont J}}
\DeclareRobustCommand{\aeolicbiv}{{\metricsfont K}}

```

\stripsl@sh
\2
\3
\4

Here we prepare for the definition of a very useful macro, \newmetrics, that should ease quite a lot writing complicated and repetitive metric sequences. We shall define \newmetrics by means of the internal LaTeX macro \@namedef which accepts a macro name containing any character, provided this name does not contain the initial back slash (if it does this back slash becomes part of the macro name; see the TeXbook where there is an example for the definition of \\TeX). Therefore we need a service macro \stripsl@sh that strips the first token from the control sequence, so that the naïf user does not have to treat the new metrics control sequence differently from the control sequences it uses for example with \newcommand. Next we define three numeric control sequences that should be followed by the rest of the macro name. The naïf user can then type in something like \2iamb␣ in order to activate a macro whose name is formed by the tokens 2iamb, which is normally impossible in LaTeX. Notice, though, the compulsory space after the macro name.

```
\newif\ifmetricsfont\metricsfontfalse
\def\stripsl@sh#1{\expandafter\@gobble\string#1}
\def\2#1 {\csname2#1\endcsname}
\def\3#1 {\csname3#1\endcsname}
\def\4#1 {\csname4#1\endcsname}

```

\newmetrics

Here is the user macro \newmetrics, to be used just as \newcommand, except it accepts a macro name starting with one of the digits '2', '3', or '4', and sets the suitable boolean variable to true so that in a long metric sequence the metric font might be selected just once.

```
\newcommand\newmetrics[2]{%
    \expandafter\@namedef\expandafter{\stripsl@sh#1}%
        {{\metricsfonttrue#2}}}
```

\iam \chor \enopl \4MACRO \aeolchorsor \hexam \pentam \2tr

Here some common metric sequences are defined; some define single measures, such as the 'iambus' or the 'choriambus', while some define complete verses such as the 'hexameter' or the 'pentameter'.

```
\newmetrics\iam{\barbreve\lunga\breve\lunga}
\newmetrics\chor{\lunga\breve\breve\lunga}
\newmetrics\enopl{\breve\lunga\breve\breve\lunga\breve\breve\lunga}
\newmetrics{\4MACRO}{\lunga\lunga\lunga\lunga}
\newmetrics{\aeolchorsor}{\lunga\zeugma{\breve\breve}\breve
    \breve\zeugma{\breve\breve}}
\newmetrics{\hexam}{\lunga\breve\breve\lunga\breve\breve
    \lunga\breve\breve\lunga\breve\breve\lunga\breve\breve
    \lunga\lunga}
\newmetrics{\pentam}{\lunga\barbbreve\lunga\barbbreve\lunga\dBar
    \lunga\breve\breve\lunga\breve\breve\lunga}
\newmetrics{\2tr}{\lunga\breve\lunga\X\ \lunga\breve\lunga\X\ }
```

As it may be seen, the definition of such metric sequences may contain almost anything; here `\zeugma` was used as well as `\␣`, but almost every macro defined in the previous parts may be freely used.

\metricstack

`\metricstack` is a command similar to `\shortstack` used to stack something over something else; specifically the second argument over the third one; it was specifically designed for use while typesetting metric sequences, but actually there is nothing that forbids to use it with any base character (typeset in text LR mode) and any superscript character belonging to a math alphabet (which is being set in script–script style, not in script style, as it happens with `\shortstack`.

```
\DeclareRobustCommand*{\metricstack}[2]%
    {$\mathord{\mathop{\hbox{#1\rule{\z@}{1ex}}}%
    \limits^{\scriptscriptstyle\relax#2\relax}}$}
```

\svert

`\svert` is a short vertical rule that may be used, for example, with `\metricstack` for putting a small number over a dividing vertical bar in metric sequences.

```
\newcommand*{\svert}{\vrule\@height.8ex\@depth.2ex\relax}
```

\textoverline

LaTeX has macro `\underline` that can be used in both text and math mode; there is nothing similar for overlining, therefore we defined a new command for this task.

```
\DeclareRobustCommand*{\textoverline}[1]{%
     \leavevmode\vbox{\setbox\z@\hbox{#1}
     \ialign{##\crcr
     \hbox to\wd\z@{\hrulefill}\crcr
     \noalign{\kern.4ex\nointerlineskip}%
     \hfil\box\z@\hfil\crcr}}}
```

`\verseskip`
`bracedmetrics`

The environment `bracedmetrics` is used primarily for setting some metric sequences one atop the other, with a certain alignment and grouped together with a right brace. We need therefore a length name `\br@cedmetrics` for measuring the width of this large metrics sequence stack; we need a command `\verseskip` for inserting a blank space before, after or in the middle of a metric sequence, that more or less is as wide as an integer number of metric symbols, and, last but not least, the environment itself for typesetting this large object containing the said metric sequences; see the documentation file `teubner-doc.pdf` for examining some examples.

```
\newlength{\br@cedmetrics}
\newcommand*{\verseskip}[1]{{%
    \setbox\z@\hbox{\longa}\dimen@\wd\z@\leavevmode\hbox to#1\dimen@{}}}

\newenvironment{bracedmetrics}[1]{\def\Hfill{\leavevmode\hfill}%
\settowidth{\br@cedmetrics}{#1}%
\ifvmode\vskip1ex\fi
$\displaystyle\left.%
\vcenter\bgroup\hsize\br@cedmetrics\parindent\z@\parskip\z@
}{\egroup\right\}$}

```

5.15 Debugging commands

`\TRON`
`\GTRON`
`\TROF`
`\GTROF`
`\treceon`
`\traceoff`

Here there are some macros for turning on and off the tracing facilities of TEX, that turn out to be useful while debugging; they are accessible also to the end user. Global settings must be turned on and off globally; local settings die out by themselves when a group is closed, but it is a good habit to explicitly turn them out regardless of groups. Attention that when the tracing facilities are on and a page ship out takes place, the `.log` file receives a lot of material, and this file gets very large. In order to avoid logging too much information the `trace` package is loaded; this package give access to the macros `\traceon` and `\traceoff` that log a lot of information, except the redundant one, specifically all the macros executed during any font change. Users don't realize the amount of processing done behind the scenes when with the New Font Selection Scheme (NFSS) a font change takes place; luckily enough modern processors are quite fast so that the compilation CPU time does not become too heavy. But if the TEX processing is logged, this amount of work implies thousands of lines of almost meaningless information when the purpose of logging depends on errors that are difficult to spot; Font changes are almost exempt from errors, so the processing of the inner workings need not be logged down.

If the user needs to trace something in order to spot errors, s/he is invited to use the commands `\traceon` and `\traceoff`; commands `\TRON` and `\TROF` do log much more material, in particular font changes, but at least they action may be confined within groups or environments; `\GTRON` and `\GTROF` are global settings and can't be confined within groups or environments; sometimes they are necessary, but it's important to turn off global tracing as soon as possible.

```
\RequirePackage{trace}
\def\GTRON{\global\tracingcommands=\tw@ \global\tracingmacros=\tw@}
\def\GTROF{\global\tracingcommands=\z@ \global\tracingmacros=\z@}
\def\TRON{\tracingcommands=\tw@ \tracingmacros=\tw@}
\def\TROF{\tracingcommands=\z@ \tracingmacros=\z@}

```

5.16 Classical Greek numerals

When typesetting Greek it may occur to specify numbers written out as Milesian numerals; the `greek` option to the babel package defines a couple of macros for transforming explicit arabic numerals or counter contents as Milesian numerals. Since this package offers more possibilities in the choice of those "non alphabetic" characters used in the Milesian notation, such macros have to be redefined. On the occasion we changed some little internal details so as to make such macros a little faster and more robust.

`\Greeknumeral` `\greeknumeral` `\@ifStar` `\grtoday`

Both `\greeknumeral` and `\Greeknumeral`, the latter producing upper case Greek numerals, while the former produces lower case ones, resort to a service macro `\gr@@numeral`. But the new definition accepts the starred version; without the star the digit value 6 is represented with a "stigma", while with the star that value is represented with a lowered "digamma". The upper case version requires intermediate macros before using `\MakeUppercase` on the result in order to convert lower to upper case Milesian value symbols. This means that `\gr@@numeral` may work only with lower case symbols. It turned out that the normal redefinition command `\renewcommand` produced fragile commands that broke out when used as arguments of other commands, specifically the Greek date was broken when it was passed as the argument to the `\date` command of the class memoir; therefore I decided to redefine the `\@ifstar` macro into another `\@ifStar` one so as not to fiddle with LaTeX kernel commands. I defined also the lowercase version of the `\grtoday` date, since the babel package provides only the `\today` command with no control over the use of which type of numerals; `\grtoday` uses the lowercase Milesian numerals through the redefined `\greeknumeral` macro.

```
\def\@ifStar#1#2{\def\@tempA{#1}\def\@tempB{#2}\futurelet\@tempC\@testStar}
\def\@testStar{\ifx\@tempC*\bbl@afterelse\expandafter\@tempA\@gobble\else
      \bbl@afterfi\@tempB\fi}
\DeclareRobustCommand*{\Greeknumeral}{%
      \let\n@vanta\Coppa\let\n@vecento\Sampi
      \@ifStar{\Gr@@kn@meral}{\Gr@@knum@ral}}
\DeclareRobustCommand*{\greeknumeral}{%
      \let\n@vanta\varkoppa\let\n@vecento\sampi
    \@ifStar{\let\s@i\stigma\gr@@numeral}{\let\s@i\fLow\gr@@numeral}}
\def\Gr@@kn@meral#1{\let\s@i\Stigma
    \expandafter\MakeUppercase\expandafter{\gr@@numeral{#1}}}
\def\Gr@@knum@ral#1{\let\s@i\Digamma
    \expandafter\MakeUppercase\expandafter{\gr@@numeral{#1}}}
\def\grtoday{{\expandafter\greeknumeral\expandafter{\the\day}}\space
\gr@c@month\space{\expandafter\greeknumeral\expandafter{\the\year}}}
```

\gr@@numeral

`\gr@@numeral` must do most of the processing; it must check that the argument is within the allowable range $1 \sim 999\,999$ and issue suitable warnings if not. On the other side, if the number is within the correct range, it must check in which decade it falls and must call other macros so as to produce the correct decimal digit ↔ Milesian symbol. Six such macros are needed because the allowable range contains at maximum six decimal places. Apparently Milesian symbology allows to go beyond one million, but Apostolos Syropoulos, who originally wrote the code thought (correctly) that Milesian numbers would not be used for "acrobatic performances" but possibly for writing the Greek date with the AD year; six decimal places are more than enough for this purpose. `\gr@ill@value` was not redefined from Apostolos Syropoulos' babel definition; it simply issues a warning message about an argument out of range. The presence of the primitive command `number` in these macros is for two purposes: (a) transforms a counter contents into a sequence of digits tokens, and (b) if the argument is already a digit string, it removes any leading zeros. No braces are present because this string is examined sequentially one digit at a time from the leading position to the least significant position; of course this means that the decimal zero is treated correctly even if Milesian symbols do not have the equivalent of a zero.

```
\def\gr@@numeral#1{%
  \ifnum#1<\@ne\space\gr@ill@value{#1}%
  \else
    \ifnum#1<10\relax\expandafter\gr@num@i\number#1%
    \else
      \ifnum#1<100\relax\expandafter\gr@num@ii\number#1%
      \else
        \ifnum#1<\@m\relax\expandafter\gr@num@iii\number#1%
        \else
          \ifnum#1<\@M\relax\expandafter\gr@num@iv\number#1%
          \else
            \ifnum#1<100000\relax\expandafter\gr@num@v\number#1%
            \else
              \ifnum#1<1000000\relax\expandafter\gr@num@vi\number#1%
              \else
                \space\gr@ill@value{#1}%
              \fi
            \fi
          \fi
        \fi
      \fi
    \fi
  \fi
}
```

\gr@num@i
\gr@num@ii
\gr@num@iii
\gr@num@iv
\gr@num@v
\gr@num@vi

The next six macros transform single decimal digits into Milesian symbols. The argument to each macro is a single decimal digit; their positional value is determined by the calling a macro that invokes a different transformation routine for every position. To the right of the least significant position there must be the symbol “anwtonos”, similar to an apostrophe, while to the left of each most significant symbol whose value is greater than 999 there must be a “katwtonos” symbol, similar to a lowered and inverted apostrophe. Zeros are examined in all macros, except the one for “units”, because their value cannot be printed but there still is the possibility that there are no more digits higher than zero, so that the anwtonos must be set. Macros `\n@vanta` and `\n@vecento` are set by the calling macros so as to be the correct lower or upper case ‘qoppa’ or sampi’ respectively.

```
\def\gr@num@i#1{%
  \ifcase#1\or a\or b\or g\or d\or e%
  \or \s@i\or z\or h\or j\fi
  \ifnum#1=\z@\else\anw@true\fi\anw@print}
\def\gr@num@ii#1{%
  \ifcase#1\or i\or k\or l\or m\or n%
  \or x\or o\or p\or \n@vanta\fi
  \ifnum#1=\z@\else\anw@true\fi\gr@num@i}
\def\gr@num@iii#1{%
  \ifcase#1\or r\or s\or t\or u\or f%
  \or q\or y\or w\or \n@vecento\fi
  \ifnum#1=\z@\anw@false\else\anw@true\fi\gr@num@ii}
\def\gr@num@iv#1{%
  \ifnum#1=\z@\else\katwtonos\fi
  \ifcase#1\or a\or b\or g\or d\or e%
  \or \s@i\or z\or h\or j\fi
  \gr@num@iii}
\def\gr@num@v#1{%
  \ifnum#1=\z@\else\katwtonos\fi
  \ifcase#1\or i\or k\or l\or m\or n%
  \or x\or o\or p\or \n@vanta\fi
  \gr@num@iv}
\def\gr@num@vi#1{%
  \katwtonos
  \ifcase#1\or r\or s\or t\or u\or f%
  \or q\or y\or w\or \n@vecento\fi
  \gr@num@v}

```

5.17 Attic numerals

It’s true that Apostolos Siropoulos wrote also the `athnum.sty` extension package in order to typeset integer numbers with the Athenian or Attic notation; this representation of integer strictly positive integers was similar in a way to the Roman notation, based on a biquinalry representation of decimal digits (taking into account that there was not a symbol for zero) so as the Romans had the symbols for 1, 5, 10, 50, 100, 500 and 1000 (I, V, X, L, C, D, M) the Attic notation has symbols for the same sequence of decimal values extended with 10 000 and 50 000. While typesetting philological texts in Greek it might be necessary to use also the Attic notation. As the original Roman notation used to be purely additive (i.e. 9 = VIIII), so is the Attic notation.

\AtticNumeral \AtticCycl@ Therefore another conversion macro was devised that receives the value to be converted as its argument and checks that it falls between the boundaries; actually the lower boundary is zero, while the upper boundary was chosen to be 99 999, for no other reason that the lack of further symbols, beyond the value 50 000, would force to long sequences of identical symbols that are difficult to read. The `athnum.sty` package allows to extend this range to 249 9999; should it be necessary, the user is invited to load that package and its transformation command `\athnum`.

The user command `\AtticNumeral` is very simple, but it must be preceded by the definitions of the quinary symbols for 50, 500, 5000, and 50 000; such symbols are present in all the CB Greek fonts in all sizes, series and shapes; therefore the definitions must be subject to the LGR enconding:

```
\DeclareTextSymbol{\Vmiria}{\GRencoding@name}{5}
\DeclareTextSymbol{\Vkilo}{\GRencoding@name}{4}
\DeclareTextSymbol{\Vetto}{\GRencoding@name}{3}
\DeclareTextSymbol{\Vdeka}{\GRencoding@name}{2}
```

The we need a command for issuing a warning message if the number to be transformed is out of range:

```
\newcommand*\attic@ill@value[1]{\PackageWarning{teubner}{%
Illegal value (\number#1) for \string\ActicNumeral\space}}
```

Finally the robust definition of the `\AtticNumeral` command"

```
\DeclareRobustCommand*\AtticNumeral[1]{%
\ifnum#1<\@ne \attic@ill@value{#1}\else
  \ifnum#1>99999\relax \attic@ill@value{#1}\else
    \AtticCycl@{#1}
  \fi
\fi}

```

The real transformation algorithm is transfered to the auxiliary macro `\AttiCycl@`, where successive division by 10 allow to extract the various decimal digits of various weights maintaining the remainder in the original counter; each decimal digit is possibly divided into the quinary value and the remaining units up to 4; the the cycle is repeated untile the decimal units, that do not require the computation of the remainder and terminate the cycle. Notice that we use also the ε-TeX extended commands for integer computations; this implies that `teubner` mus be run with a suitably recent version of the typesetting engine that embeds the above extensions.

```
\def\AtticCycl@#1{%
    \bgroup
    \countdef\valore=252\countdef\cifra=250\relax
    \valore=#1\relax
    \cifra=\valore\divide\cifra10000\relax
    \valore=\numexpr\valore-\cifra*10000\relax
    \ifnum\cifra>4\relax\Vmiria \advance\cifra-5\fi
       \@whilenum\cifra>\z@\do{M\advance\cifra\m@ne}%
    \cifra=\valore\divide\cifra1000\relax
    \valore=\numexpr\valore-\cifra*1000\relax
    \ifnum\cifra>4\relax\Vkilo \advance\cifra-5\fi
       \@whilenum\cifra>\z@\do{Q\advance\cifra\m@ne}%
    \cifra=\valore\divide\cifra100\relax
    \valore=\numexpr\valore-\cifra*100\relax
    \ifnum\cifra>4\relax\Vetto \advance\cifra-5\fi
       \@whilenum\cifra>\z@\do{H\advance\cifra\m@ne}%
    \cifra=\valore\divide\cifra10\relax
    \valore=\numexpr\valore-\cifra*10\relax
    \ifnum\cifra>4\relax\Vdeka \advance\cifra-5\fi
       \@whilenum\cifra>\z@\do{D\advance\cifra\m@ne}%
    \cifra=\valore
    \ifnum\cifra>4\relax P\advance\cifra-5\relax\fi
       \@whilenum\cifra>\z@\do{I\advance\cifra\m@ne}%
    \egroup}

```

6 Accessing the CBgreek fonts when the TX fonts are selected

During the year 2010 this package teubner.sty was upgraded in order to allow using the CBgreek fonts even when other Latin fonts, different from the "standard" CM and LM ones are selected for typesetting text with the Latin script.

At the same time Antonis Tsolomitis uploaded a new package in order to let Greek users use some Greek fonts that match the Times eXtended ones. In order to use the de facto default encoding LGR for Greek fonts, he produced the necessary `lgrtxr.fd`, `lgrtxss.fd`, `lgrtxtt.fd`, font definition files that allow the font switching implied by the `greek` option to the babel package. These files take precedence over the mechanism outlined in section **??**, because command `\substitutefontfamily` first tests the existence of `lgrtxr.fd`, and, if this is not available, it may generate a specific one suitable for working smoothly with teubner.sty.

Now if Tsolomitis' files are available on the main system tree, these take precedence and the teubner compatible files are not generated. Unfortunately Tsolomitis' fonts, although better suited to match the TX fonts, are well adapted to typeset common Greek text, but they are not adapted to typeset philological texts.

We therefore avoid this clash by creating a teubnertx.sty file. This extension defines the families and shapes available with the familiar fond definition files, but the information gets input by teubner.sty at the "begin document" time, without resorting to any `.fd` file. May be more information is loaded than is strictly necessary, but it better to do this way than to clash with other packages.

```
\DeclareFontFamily{LGR}{txr}{}
\DeclareFontShape{LGR}{txr}{m}{n}{<->ssub * cmr/m/n}{}
\DeclareFontShape{LGR}{txr}{m}{it}{<->ssub * cmr/m/it}{}
\DeclareFontShape{LGR}{txr}{m}{sl}{<->ssub * cmr/m/sl}{}
```

```
\DeclareFontShape{LGR}{txr}{m}{sc}{<->ssub * cmr/m/sc}{}
\DeclareFontShape{LGR}{txr}{b}{n}{<->ssub * cmr/bx/n}{}
\DeclareFontShape{LGR}{txr}{b}{it}{<->ssub * cmr/bx/it}{}
\DeclareFontShape{LGR}{txr}{b}{sl}{<->ssub * cmr/bx/sl}{}
\DeclareFontShape{LGR}{txr}{b}{sc}{<->ssub * cmr/bx/sc}{}
\DeclareFontShape{LGR}{txr}{bx}{n}{<->ssub * cmr/bx/n}{}
\DeclareFontShape{LGR}{txr}{bx}{it}{<->ssub * cmr/bx/it}{}
\DeclareFontShape{LGR}{txr}{bx}{sl}{<->ssub * cmr/bx/sl}{}
\DeclareFontShape{LGR}{txr}{bx}{sc}{<->ssub * cmr/bx/sc}{}

\DeclareFontShape{LGR}{txr}{m}{li}{<->ssub * cmr/m/li}{}
\DeclareFontShape{LGR}{txr}{b}{li}{<->ssub * cmr/b/li}{}
\DeclareFontShape{LGR}{txr}{bx}{li}{<->ssub * cmr/bx/li}{}
\DeclareFontShape{LGR}{txr}{m}{ui}{<->ssub * cmr/m/ui}{}
\DeclareFontShape{LGR}{txr}{b}{ui}{<->ssub * cmr/m/ui}{}
\DeclareFontShape{LGR}{txr}{bx}{ui}{<->ssub * cmr/bx/ui}{}
\DeclareFontShape{LGR}{txr}{m}{rs}{<->ssub * cmr/m/rs}{}
\DeclareFontShape{LGR}{txr}{b}{rs}{<->ssub * cmr/m/rs}{}
\DeclareFontShape{LGR}{txr}{bx}{rs}{<->ssub * cmr/bx/rs}{}

\DeclareFontFamily{LGR}{txss}{}
\DeclareFontShape{LGR}{txss}{m}{n}{<->ssub * cmss/m/n}{}
\DeclareFontShape{LGR}{txss}{m}{it}{<->ssub * cmss/m/it}{}
\DeclareFontShape{LGR}{txss}{m}{sl}{<->ssub * cmss/m/sl}{}
\DeclareFontShape{LGR}{txss}{m}{sc}{<->ssub * cmss/m/sc}{}
\DeclareFontShape{LGR}{txss}{b}{n}{<->ssub * cmss/bx/n}{}
\DeclareFontShape{LGR}{txss}{b}{it}{<->ssub * cmss/bx/it}{}
\DeclareFontShape{LGR}{txss}{b}{sl}{<->ssub * cmss/bx/sl}{}
\DeclareFontShape{LGR}{txss}{b}{sc}{<->ssub * cmss/bx/sc}{}
\DeclareFontShape{LGR}{txss}{bx}{n}{<->ssub * cmss/bx/n}{}
\DeclareFontShape{LGR}{txss}{bx}{it}{<->ssub * cmss/bx/it}{}
\DeclareFontShape{LGR}{txss}{bx}{sl}{<->ssub * cmss/bx/sl}{}
\DeclareFontShape{LGR}{txss}{bx}{sc}{<->ssub * cmss/bx/sc}{}

\DeclareFontFamily{LGR}{txtt}{\hyphenchar=-1}
\DeclareFontShape{LGR}{txtt}{m}{n}{<->ssub * cmtt/m/n}{}
\DeclareFontShape{LGR}{txtt}{m}{it}{<->ssub * cmtt/m/it}{}
\DeclareFontShape{LGR}{txtt}{m}{sl}{<->ssub * cmtt/m/sl}{}
\DeclareFontShape{LGR}{txtt}{m}{sc}{<->ssub * cmtt/m/sc}{}
\DeclareFontShape{LGR}{txtt}{b}{n}{<->ssub * cmtt/bx/n}{}
\DeclareFontShape{LGR}{txtt}{b}{it}{<->ssub * cmtt/bx/it}{}
\DeclareFontShape{LGR}{txtt}{b}{sl}{<->ssub * cmtt/bx/sl}{}
\DeclareFontShape{LGR}{txtt}{b}{sc}{<->ssub * cmtt/bx/sc}{}
\DeclareFontShape{LGR}{txtt}{bx}{n}{<->ssub * cmtt/bx/n}{}
\DeclareFontShape{LGR}{txtt}{bx}{it}{<->ssub * cmtt/bx/it}{}
\DeclareFontShape{LGR}{txtt}{bx}{sl}{<->ssub * cmtt/bx/sl}{}
\DeclareFontShape{LGR}{txtt}{bx}{sc}{<->ssub * cmtt/bx/sc}{}
```