The Italian support for babel

Claudio Beccari — mail: claudio dot beccari at gmail dot com

v.1.4.01 — 2019/01/05

Contents

1 The Italian language **1**
1.1 The commented code . . . 4
1.2 Traditionally labelled enumerate environment . 6
1.3 Support for etymological hyphenation 8
1.3.1 Some history . . . 8
1.3.2 The current solution 8
1.4 Facilities required by the ISO 31/XI regulations . . 11
1.5 Intelligent comma 12

1 The Italian language

Important notice: This language description file relies on functionalities provided by a modern TeX system distribution with pdfLaTeX working in extended mode (eTeX commands available); it should perform correctly also with XeLaTeX and LuaLaTeX; tests have been made also with the latter programs, but it was really tested in depth with `babel` and pdfLaTeX.

The file `italian.dtx`[1] defines all the required and some optional language-specific macros for the Italian language.

The features of this language definition file are the following:

1. The Italian hyphenation is invoked, provided that the Italian hyphenation pattern files were loaded when the specific format file was built.
2. The language dependent infix words to be inserted by such commands as `\chapter`, `\caption`, `\tableofcontents`, etc. are redefined in accordance with the Italian typographical practice.
3. Since Italian can be easily hyphenated and Italian practice allows to break a word before the last two letters, hyphenation parameters have been set accordingly, but a very high demerit value has been set in order to avoid word breaks in the penultimate line of a paragraph. Specifically

[1] The file described in this section has version number v.1.4.01 and was last revised on 2019/01/05. The original author is Maurizio Codogno. It was initially revised by Johannes Braams and then completely rewritten by Claudio Beccari

`"`	inserts a compound word mark where hyphenation is legal; it allows etymological hyphenation which is recommended for technical terms, chemical names and the like; it does not work if the next character is represented with a control sequence or is an accented character.
`"|`	the same as the above without the limitation on characters represented with control sequences or accented ones.
`""`	inserts open quotes “.
`"<`	inserts open guillemets without trailing space.
`">`	inserts closed guillemets without leading space.
`"/`	allows hyphenation of both words connected with slash.
`"-`	allows hyphenation of both words connected with a short dash (*trattino copulativo*, in Italian)

Table 1: shorthands for the Italian language. These shorthands are available only if command `\setactivedoublequote` is given after loading `babel` and before `\begin{document}`.

the `\clubpenalty`, and the `\widowpenalty` are set to rather high values and `\finalhyphendemerits` is set to such a high value that hyphenation is strongly discouraged between the last two lines of a paragraph.

4. Some language specific shorthands have been defined so as to allow etymological hyphenation, specifically `"` inserts a break point at any word boundary that the typesetter chooses, provided it is not followed by an accented letter (very unlikely in Italian, where compulsory accents fall only on the last and ending vowel of a word, but it may take place with compound words that include foreign roots), and `"|` when the desired break point falls before an accented letter. As you can read in table 1, these shorthands are available only if they get activated with `\setactivedoublequote` after loading `babel` but before the `\begin{docuemnt}` statement. This is done in order to preserve the user from package conflicts: if s/he wants to use these facilities s/he must remember that conflicts may arise unless active characters are deactivated; this can be done with the `babel` command `\shorthadsoff{"}` (and reactivated with `\shorthandson{"}`) when its wise to do so; conflicts have been reported with package `xypic` and with `TikZ`, but the latter has its own library to deactivate all active characters, not just the double quotes, the only Italian language possibly activated character.

5. Some Italian compound words have a connecting short dash (a hyphen sign) between them without any space between the component words and the short dash; in this situation standard LaTeX allows a line break only just after the short dash; this may lead to paragraphs with protruding lines or with ugly looking wide inter word spaces. If a break point is desired in the second word, one may use a `"` sign just after the short dash; but if a line break is required in the first word, them the `"-` shorthand comes in

handy; pay attention though, that if you use an en-dash or an em-dash (both should not be used in Italian as compound words connectors, but...) then the `"-` shorthand might impeach the `--` or `---` ligatures, thus producing an unacceptable appearance.

6. The shorthand `""` introduces the raised (English) opening double quotes; this shorthand proves its usefulness when one reminds that the Italian keyboard misses the backtick key, and the backtick on a Windows based platform may be obtained only by pressing the `Alt` key while keying the numerical code 0096 in the numeric keypad; very, very annoying!

7. The shorthands `"<` and `">` insert the guillemets sometimes used also in Italian typography; with the T1 font encoding the ligatures `<<` and `>>` should insert such signs directly, but not all the virtual fonts that claim to follow the T1 font encoding actually contain the guillemets; with the OT1 encoding the guillemets are not available and must be faked in some way. By using the `"<` and `">` shorthands (even with the T1 encoding) the necessary tests are performed and in case the guillemets are faked by means of the special LaTeX math symbols. At the same time if OpenType fonts are being used with XeLaTeX or LuaLaTeX, there are no problems with guillemets.

8. Three new specific commands `\unit`, `\ped`, and `\ap` are introduced so as to enable the correct composition of technical mathematics according to the ISO 31/XI recommendations. The definition of `\unit` takes place only at "begin document" so that it is possible to verify if some other similar functionalities have already been defined by other packages, such as `units.sty` or `siunitx.sty`. In particular command `\unit` is deactivated by default; the user can activate it by entering the command `\setISOcompliance` after loading the `babel` package and before the `\begin{document}` statement. The above checks will enter into action even if this ISO compliance is set, in order to avoid conflicts with the above named packages. The `\ap` and `\ped` commands remain available because up to now no specific conflicts have been reported.

9. Since in all languages different from English the decimal separator according to the ISO regulations *must* be a comma[2]; since no language description file nor the `babel` package itself provides for this functionality, a not so simple intelligent comma definition is provided such that at least in mathematics it behaves correctly. There are other packages that provide a similar functionality, for example `icomma` and `ncccomma`; `icomma`, apparently is not in conflict with `dcolomn`, but requires a space after the comma all the times it plays the rôle of a punctuation mark; `ncccomma`, checks if the next token is a digit, but it repeated ten tests every time it meets a comma, irrespective from what it is followed by. I believe that my solution is better than that

[2]Actually the Bureau International des Pois et Mésures allows also the point as a decimal separator without mentioning any language, but recommends to follow the national typographical traditions

provided by both those packages; but I assume that if the user loads on of those packages, it prefers to use that functionality; In case one of those pachages is loaded, this module excludes its intelligent comma functionality. By default this functionality is turned *off*, therefore the user should turn it on by means of the `\IntelligentComma` command; it can turn it off by means of `\NoIntelligentComma`. Please, read subsection 1.5 to see the various situations where a mathematical comma may be used and how to overcome the few cases when the macros of this file don't behave as expected. The section describes also some limitations when some cong=flicting packages are being loaded.

10. In Italian legal documents it is common to tag list-items with the old fashioned 21-letter Italian alphabet, that differs from the Latin one by the omission of the letters 'j', 'k', 'w','x', and 'y'. This applies for both upper and lower case tags. This feature is obtained by using the commands `\XXIletters` and `\XXVIletters` that allow to switch back and forth between 21- and 26-letter tagging.

For this language a few shorthands have been defined, table 1, some of which are introduced to overcome certain limitations of the Italian keyboard; in section 1.5 there are other comments and hints in order to overcome some other keyboard limitations.

Acknowlegements

It is my pleasure to acknowledge the contributions of Giovanni Dore, Davide Liessi, Grazia Messineo, Giuseppe Toscano, who spotted some bugs or conflicts with other packages, mainly `amsmath` and `icomma`, and with digits hidden inside macros or control sequences representing implicit characters. Testing by real users and their feedback is essential with open software such as the uncountable contributions to the TEX system. Thank you very much.

References

[1] Beccari C., "Computer Aided Hyphenation for Italian and Modern Latin", TUGboat vol. 13, n. 1, pp. 23-33 (1992).

[2] Beccari C., "Typesetting mathematics for science and technology according to ISO 31/XI", TUGboat vol. 18, n. 1, pp. 39-48 (1997).

1.1 The commented code

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
1 \LdfInit{italian}{captionsitalian}%
```

When this file is read as an option, i.e. by the `\usepackage` command, `italian` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@italian` to see whether we have to do something here.

```
\ifx\l@italian\@undefined
    \@nopatterns{Italian}%
    \adddialect\l@italian0\fi
```

The next step consists of defining commands to switch to (and from) the Italian language.

`\captionsitalian` The macro `\captionsitalian` defines all strings used in the four standard document classes provided with LaTeX.

```
\addto\captionsitalian{%
  \def\prefacename{Prefazione}%
  \def\refname{Riferimenti bibliografici}%
  \def\abstractname{Sommario}%
  \def\bibname{Bibliografia}%
  \def\chaptername{Capitolo}%
  \def\appendixname{Appendice}%
  \def\contentsname{Indice}%
  \def\listfigurename{Elenco delle figure}%
  \def\listtablename{Elenco delle tabelle}%
  \def\indexname{Indice analitico}%
  \def\figurename{Figura}%
  \def\tablename{Tabella}%
  \def\partname{Parte}%
  \def\enclname{Allegati}%
  \def\ccname{e~p.~c.}%
  \def\headtoname{Per}%
  \def\pagename{Pag.}%
  \def\seename{vedi}%
  \def\alsoname{vedi anche}%
  \def\proofname{Dimostrazione}%
  \def\glossaryname{Glossario}%
  }%
```

`\dateitalian` The macro `\dateitalian` redefines the command `\today` to produce Italian dates.

```
\def\dateitalian{%
  \def\today{\number\day~\ifcase\month\or
    gennaio\or febbraio\or marzo\or aprile\or maggio\or giugno\or
    luglio\or agosto\or settembre\or ottobre\or novembre\or
    dicembre\fi\space \number\year}}%
```

`\italianhyphenmins` The italian hyphenation patterns can be used with both `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

`\extrasitalian` `\noextrasitalian` Lower the chance that clubs or widows occur.

```
\addto\extrasitalian{%
  \babel@savevariable\clubpenalty
  \babel@savevariable\widowpenalty
  \babel@savevariable\@clubpenalty
  \clubpenalty3000\widowpenalty3000\@clubpenalty\clubpenalty}%
```

Never ever break a word between the last two lines of a paragraph in Italian texts.

```
\addto\extrasitalian{%
  \babel@savevariable\finalhyphendemerits
  \finalhyphendemerits50000000}%
```

In order to enable the hyphenation of words such as "nell'altezza" we give the ' a non-zero lower case code. When we do that TEX finds the following hyphenation points `nel-l'al-tez-za` instead of none. If this `italian.ldf` is (improperly) used with `babel` when typesetting with `xelatex` or `lualatex` the apostrophe must receive a unicode code point. Therefore we use the same test that is being used by the `iftex` package in order to check if this option to `babel` is used while typesetting with `pdflatex` or another typesetting engine.

```

\addto\extrasitalian{%
\lccode\string‘’=‘’ \unless\ifcsname pdfmatch\endcsname
\lccode\string"2019=\string"2019\fi}
\addto\noextrasitalian{%
\lccode\string‘’=0 \unless\ifcsname pdfmatch\endcsname
\lccode\string"2019=0\fi}

```

Notice, though, that if you use `babel` when typesetting with `lualatex` or `xelatex` using the `fontspec` usual commands and options may not lead to their proper font alternative variants being used. Apparently the `\balefont` command is more performant in transmitting the proper information to `fontspec`. Of course `\babelfont` must be used after the `babel` package has been invoked; while there appears to be any loading precedence requirements when `fontspec` and `polyglossa` are used

1.2 Traditionally labelled enumerate environment

In some traditional texts, especially of legal nature, enumerations labelled with lower or upper case letters use the reduced Latin alphabet that omits the so called "non Italian letters": j, k, w, x, and y.

`\XXIletters` `\XXVIletters`

At the same time it is considered useful to have the possibility of switching back and forth from the 21-letter tagging and the 26-letter one. This requires a counter that keeps the switching status (0 for 21 letters and 1 for 26 letters) and commands `\XXIletters` and `\XXVIletters` to set the switch. Default is 26 letter tagging.

```
\newcount\it@lettering \it@lettering=\@ne
\newcommand*\XXIletters{\it@lettering=\z@}
\newcommand*\XXVIletters{\it@lettering=\@ne}
```

```
\let\bbl@alph\@alph \let\bbl@Alph\@Alph
\addto\extrasitalian{\babel@savevariable\it@lettering
  \let\@alph\it@alph \let\@Alph\it@Alph}
\addto\noextrasitalian{\let\@alph\bbl@alph\let\@Alph\bbl@Alph}
```

To make this feasible it's necessary to redefine the way the LaTeX `\@alph` and `\@Alph` work. Let's make the alternate definitions:

```
\def\it@alph#1{%
\ifcase\it@lettering
   \ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or
   l\or m\or n\or o\or p\or q\or r\or s\or t\or u\or v\or
   z\else\@ctrerr\fi
\or
   \ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or
   j\or k\or l\or m\or n\or o\or p\or q\or r\or s\or t\or u\or v\or
   w\or x\or y\or z\else\@ctrerr\fi
\fi}%
\def\it@Alph#1{%
\ifcase\it@lettering
   \ifcase#1\or A\or B\or C\or D\or E\or F\or G\or H\or I\or
   L\or M\or N\or O\or P\or Q\or R\or S\or T\or U\or V\or
   Z\else\@ctrerr\fi
\or
   \ifcase#1\or A\or B\or C\or D\or E\or F\or G\or H\or I\or
   J\or K\or L\or M\or N\or O\or P\or Q\or R\or S\or T\or U\or V\or
   W\or X\or Y\or Z\else\@ctrerr\fi
\fi}%
```

In order to have a complete description, the situation is as such:

1. If you want to always use the 21-letter item tagging, simply use the `\XXIletters` declaration just after `\begin{document}` and this setting remains global (provided, of course, that the declaration is defined, i.e. that the Italian language is the default one); in this way the setting is global while you use the Italian language.

2. The `XXVIletter` command, issued outside any environment sets the 26-letter item tagging in a global way; this setting is the default one.

3. If you specify `\XXIletters` just before entering an environment that uses alphabetic tagging, this environment will be tagged with the 21-letter alphabet, but this is a local setting, because the letter tagging takes place only from the second level of enumeration.

4. The declarations `\XXIletters` and `\XXVIletters` let you switch back and forth between the two kinds of tagging, But this kind of tagging, the 21-letter one, is meaningful only in Italian and when you change language, letter tagging reverts to the 26-letter one.

1.3 Support for etymological hyphenation

In Italian etymological hyphenation is desirable with technical terms, chemical names, and the like.

1.3.1 Some history

In his article on Italian hyphenation [1] Beccari pointed out that the Italian language gets hyphenated on a phonetic basis, although etymological hyphenation is allowed; this is in contrast with what happens in Latin, for example, where etymological hyphenation is always used. Since the patterns for both languages would become too complicated in order to cope with etymological hyphenation, in his paper Beccari proposed the definition of an active character '`_`' such that it could insert a "soft" discretionary hyphen at the compound word boundary. For several reasons that idea and the specific active character proved to be unpractical and was abandoned.

This problem is so important with the majority of the European languages, that babel from the very beginning developed the tradition of making the `"` character active so as to perform several actions that turned useful with every language. One of these actions consisted in defining the shorthand `"|`, that was extensively used in German and in many other languages, in order to insert a discretionary hyphen such that hyphenation would not be precluded in the rest of the word as it happens with the standard TEX command `\-`.

Meanwhile the `ec` fonts with the double Cork encoding (thus formerly called the `dc` fonts) have become more or less standard and are widely used by virtually all Europeans that write languages with many special national characters; by so doing they avoid the use of the `\accent` primitive which would be required with the standard OT1 encoded `cm` fonts; with such OT1 encoded fonts the primitive command `\accent` is such that hyphenation becomes almost impossible, in any case strongly impeached.

The T1 encoded fonts contain a special character, named "compound word mark", that occupies slot 23 (or `'27` or `"17` in the font scheme and may be input with the sequence `^^W`. Up to now, apparently, this special character has never been used in a practical way for typesetting languages rich of compound words; moreover it has never been inserted in the hyphenation pattern files of any language. Beccari modified his pattern file `ithyph.tex v4.8b` for Italian so as to contain five new patterns that involve `^^W`, and he tried to give the babel active character `"` a new shorthand definition, so as to allow the insertion of the "compound word mark" in the proper place within any word where two semantic fragments join up. With such facility for marking the compound word boundaries, etymological hyphenation becomes possible even if the patterns know nothing about etymology (but the typesetter hopefully does!).

1.3.2 The current solution

Even this solution proved to be inconvenient on certain *NIX platforms, so Beccari resorted to another approach that uses the babel active character `"` and relies on

the category code of the character that follows ".

Instead of a boolean switch we use a private counter so as to check at `\begin{document}` if this facility has to be activated. The default value is zero; anything different from zero means that the facility has to be activated; this is done with command `\setactivedoublequote` to be issued before `\begin{document}`

```
\newcount\it@doublequoteactive \it@doublequoteactive=\z@
\def\setactivedoublequote{\it@doublequoteactive=\@ne}
{\catcode`"=12 \global\let\it@doublequote"}
{\catcode`"=13 \global\let\it@@dqactive"}
\AtBeginDocument{%
  \unless\ifnum\it@doublequoteactive=\z@
  \initiate@active@char{"}%
  \addto\extrasitalian{\bbl@activate{"}\languageshorthands{italian}}%
```

`\it@cwm` The active character " is now defined for language `italian` so as to perform different actions in math mode compared to text mode; specifically in math mode a double quote is inserted so as to produce a double prime sign, while in text mode the temporary macro `\it@next` is defined so as to defer any further action until the next token category code has been tested.

```
  \declare@shorthand{italian}{"}{%
    \ifmmode
      \def\it@next{''}%
    \else
      \def\it@next{\futurelet\it@temp\it@cwm}%
    \fi
    \it@next
  }%
\fi
```

The following statement must be conditionally executed after the above modification of the `\extraasitalian` list; in facts at the "begin document" execution the main language has already been set without the above modifications; therefore nothing takes place unless the Italian main language is selected again with the explicit command `\selectlanguage` without this trick the active double quotes would remain inactive; of course `\languagename` contains the string `italian` if this language was the main one; by testing this string, the suitable command may be issued again with the new settings and the double quotes become really active. Thanks to Davide Liessi for reporting this bug.

```
\ifdefstring{\languagename}{italian}{\selectlanguage{italian}}{\relax}
}%
```

`\it@cwm` The `\it@next` service control sequence is such that upon its execution a temporary variable `\it@temp` is made equivalent to the next token in the input list without actually removing it. Such temporary token is then tested by the macro `\it@cwm` and if it is found to be a letter token (cathode=11), then it introduces a compound word separator control sequence `\it@allowhyphens` whose expansion introduces a discretionary hyphen and an unbreakable zero space; otherwise the token is not a letter; then it is therefore tested against $|_{12}$: if so a compound word separator

is inserted and the | token is removed; otherwise two other tests are performed to see if guillemets have to be inserted, and in case a suitable intelligent guillemet macro is introduced that gobbles unwanted leading or trailing spaces; otherwise a test is made to see if the next char is a slash character, and in case a special discretionary break is inserted such as to maintain the slash while allowing the hyphenation of both words before and after the slash; otherwise another test is performed to see if another double quote sign follows: in this case a double open quote mark is inserted; otherwise another test is made to see if a connecting hyphen char follows, and in this case the hyphen char is substituted with a discretionary break that allows hyphenation of both words before and after the hyphen char; otherwise nothing is done.

The double quote shorthand for inserting a double open quote sign is useful for people who are inputting Italian text by means of an Italian keyboard which unfortunately misses the grave or backtick key. The shorthand "" becomes equivalent to `` for inserting raised open high double quotes.

```
\def\it@@cwm{\bbl@allowhyphens\discretionary{-}{}{}\bbl@allowhyphens}%
\def\it@@slash{\bbl@allowhyphens\discretionary{/}{}{/}\bbl@allowhyphens}%
\def\it@@trattino{\bbl@allowhyphens\discretionary{-}{}{-}\bbl@allowhyphens}%
\def\it@@ocap#1{\it@ocap}\def\it@@ccap#1{\it@ccap}%
\DeclareRobustCommand*{\it@cwm}{\let\it@@next\it@doublequote
\ifcat\noexpand\it@temp a%
    \def\it@@next{\it@@cwm}%
\else
    \if\noexpand\it@temp \string|%
        \def\it@@next{\it@@cwm\@gobble}%
    \else
        \if\noexpand\it@temp \string<%
            \def\it@@next{\it@@ocap}%
        \else
            \if\noexpand\it@temp \string>%
                \def\it@@next{\it@@ccap}%
            \else
                \if\noexpand\it@temp\string/%
                    \def\it@@next{\it@@slash\@gobble}%
                \else
                    \ifcat\noexpand\it@temp\noexpand\it@@dqactive
                        \def\it@@next{``\@gobble}%
                    \else
                        \if\noexpand\it@temp\string-%
                            \def\it@@next{\it@@trattino\@gobble}%
                        \fi
                    \fi
                \fi
            \fi
        \fi
    \fi
\fi
\it@@next}%
```

By this definition of " if one types `macro"istruzione` the possible break points become ma-cro-istru-zio-ne, while without the " mark they would be ma-croi-stru-zio-ne, according to the phonetic rules such that the `macro` prefix is not taken as a unit. A chemical name such as `des"clor"fenir"amina"cloridrato` is breakable as des-clor-fe-nir-ami-na-clo-ri-dra-to instead of de-sclor-fe-ni-ra-mi-na-...

In other language description files a shorthand is defined so as to allow a break point without actually inserting any hyphen sign; examples are given such as entrada/salida; actually if one wants to allow a breakpoint after the slash, it is much clearer to type `\slash` instead of `/` and LaTeX does everything by itself; here the shorthand `"/` was introduced to stand for `\slash` so that one can type `input"/output` and allow a line break after the slash. This shorthand works only for the slash, since in Italian such constructs are extremely rare.

Attention: the expansion of " takes place before the actual expansion of OT1 or T1 accented sequences such as `\‘{a}`; therefore this etymological hyphenation facility works as it should only when the semantic word fragments *do not start* with an accented letter; this in Italian is always avoidable, because compulsory accents fall only on the last vowel, but it may be necessary to mark a compound word where one or more components come from a foreign language and contain diacritical marks according to the spelling rules of that language. In this case the special shorthand `"|` may be used that performs exactly as " normally does, except that the | sign is removed from the token input list: `kilo"|\"orsted` gets hyphenated as `ki-lo-ör-sted`; but also `kilo"|örsted` gets hyphenated correctly as `ki-lo-ör-sted` The `"|` macro is necessary because, even with a suitable option specified to the `inputenc` package, the letter ‘ö’ does not have category code 11, as the ASCII letters do, because of the LICR (LaTeX Internal Character Representation), i.e. the set of intermediate macros that have to be expanded in order to fetch the proper glyph in the output font.

1.4 Facilities required by the ISO 31/XI regulations

The ISO 31/XI regulations require that units of measure are typeset in upright font in both math and text, and that in text mode they are separated from the numerical indication of the measure with an unbreakable (thin) space. The command `\unit` that was defined for achieving this goal happened to conflict with the homonymous command defined by the `units.sty` package; we therefore need to test if that package has already been loaded so as to avoid conflicts; we assume that if the user loads that package, s/he wants to use that package facilities and command syntax.

Actually there are around several packages that help to typeset units of measure in the proper way; besides `units.sty` there are also `SIunits` and `siunitx.sty`; the latter nowadays offers the best performances in this domain. Therefore we keep controlling the possibility that `units.sty` has been loaded just for backwards compatibility, but we must do the same with `SIunits` and `siunitx.sty`. In order to avoid the necessity o loading packages in a certain order, we delay the test until `\begin{document}`.

The same ISO regulations require also that super and subscripts (apices and

pedices) are in upright font, *not in math italics*, when they represent "adjectives" or appositions to mathematical or physical variables that do not represent countable or measurable entities: for example, $V_{\max}$ or V_{rms} for a maximum voltage or a root mean square voltage, compared to V_i or V_T as the i-th voltage in a set, or a voltage that depends on the thermodynamic temperature T. See [2] for a complete description of the ISO regulations in connection with typesetting.

More rarely it happens to use superscripts that are not mathematical variables, such as the notation $\mathbf{A}^{\mathrm{T}}$ to denote the transpose of matrix $\mathbf{A}$; text superscripts are useful also as ordinals or in old fashioned abbreviations in text mode; for example the feminine ordinal 1ª or the old fashioned obsolete abbreviation F^lli for Fratelli in company names (compare with "Bros." for Brothers in American English); text subscripts are mostly used in logos.

\unit
\ap
\ped
\setISOcompliance

First we define the new (internal) commands `\bbl@unit`, `\bbl@ap`, and `\bbl@ped` as robust ones. This facility is deactivated by default according to the contents of an internal counter and the setting of the activation command by the user; commands for apices and pedices remain available in any case.

```
\newcount\it@ISOcompliance \it@ISOcompliance=\z@
\def\setISOcompliance{\it@ISOcompliance=\@ne}
\AtBeginDocument{\unless\ifnum\it@ISOcompliance=\z@%
\def\activate@it@unit{\DeclareRobustCommand*{\bbl@it@unit}[1]{%
    \textormath{\,\textup{##1}}{\,\mathrm{##1}}}}
\@ifpackageloaded{units}{}{\@ifpackageloaded{siunitx}{}{%
  \@ifpackageloaded{SIunits}{}{%
  \activate@it@unit\addto\extrasitalian{%
    \babel@save\unit\let\unit\bbl@it@unit}%
  }}}\ifcsstring{bbl@main@language}{italian}{\selectlanguage{italian}}{}%
\fi}
\DeclareRobustCommand*{\bbl@it@ap}[1]{%
  \textormath{\textsuperscript{#1}}{^{\mathrm{#1}}}}%
\DeclareRobustCommand*{\bbl@it@ped}[1]{%
  \textormath{$_{\mbox{\fontsize\sf@size\z@
        \selectfont#1}}$}{_\mathrm{#1}}}%
```

Then we can use `\let` to define the user level commands, but in case the macros already have a different meaning before entering in Italian mode typesetting, we first save their meaning so as to restore them on exit.

```
\addto\extrasitalian{%
  \babel@save\ap\let\ap\bbl@it@ap
  \babel@save\ped\let\ped\bbl@it@ped
  }%
```

1.5 Intelligent comma

We need to perform some tests that require some smart control-sequence handling; therefore we call the `etoolbox` package that allows us more testing functionality. There are no problems with this package that can be invoked also by other ones before or after `babel` is called; the `\RequirePackage` mechanism is sufficiently

smart to avoid reloading the same package more than once. But we have to delay this call, because `italian.ldf` is being read while processing the options passed to `babel`, and while options are being scanned and processed it is forbidden to load packages; we delay it at the end of processing the `babel` package itself.

```
\AtEndOfPackage{\RequirePackage{etoolbox}}
```

`\IntelligentComma` `\NoIntellgentComma`

This feature is optional, in the sense that it is necessary to issue a specific command to activate it; actually this functionality is activated or, respectively, deactivated with the self explanatory commands `\IntelligentComma` and `\NoIntelligentComma`. They operate by setting or resetting the comma sign as an active character in mathematics. We defer the definition of the commands that turn on and off the intelligent comma feature at the end of the preamble, so as to avoid possible conflicts with other packages. It has already been pointed out that this procedure for setting up the active comma to behave intelligently in math mode, conflicts with the `dcolumn` package; therefore we assume these commands are defined when the final user typesets a document, but they will be possibly defined only at the end of the preamble when it will be known if the `dcolumn` package has been loaded. We do the same if packages `icomma` or `ncccomma` have been loaded, since that assumes that the user wants to use their functionality, not the functionality of this package.

We need a command to set the comma as an active charter only in math mode; the special `\mathcode` that classifies an active character in math is the hexadecimal value `"8000`. By default we set the punctuation type of comma, but we let `\IntelligentComma` and `\NoIntelligentComma` to `\relax` so that their use is forbidden when one of the named packages is loaded. In this way all known conflicts are avoided; should the user find out other conflicts, s/he is kindly required to notify it to the maintainer.

```
\AtEndOfPackage{\AtEndPreamble{%
\newcommand*\IntelligentComma{\mathcode`\,=\string"8000}% Active comma
\newcommand*\NoIntelligentComma{\mathcode`\,=\string"613B}% Punctuation comma
\@ifpackageloaded{icomma}{\let\IntelligentComma\relax
  \let\NoIntelligentComma\relax}{%
  \@ifpackageloaded{ncccomma}{\let\IntelligentComma\relax
  \let\NoIntelligentComma\relax}{%
    \@ifpackageloaded{dcolumn}{\let\IntelligentComma\relax
      \let\NoIntelligentComma\relax}{%
      \@ifpackageloaded{polyglossia}{%
        \ifcsstring{xpg@main@language}{english}{\relax}{%
          \mathcode`\,=\string"613B}
      }{%
        \ifcsstring{languagename}{english}{\relax}{%
          \mathcode`\,=\string"613B}
      }%
  }}}%
}}
```

These commands are defined only in the `babel` support for the Italian language (this file) and are not defined in the corresponding `polyglossia` support for the

language. In order to have this functionality work properly with pdfLaTeX, XeLaTeX, and LuaLaTeX, it is necessary to discover which engine is being used, or better, which language handling package is being used: `babel` or `polyglossia`? Let us remember that testing the actual engine, as it would be possible with package `iftex`, does not tell the whole truth, because LuaLaTeX and XeLaTeX behave in a similar way for what concerns language handling since they can use both `babel` and `polyglossia` (obviously not at the same time); so the use of `babel` or `polyglossia` is the real discriminant factor, not the typesetting engine.

`\virgola` `\virgoladecimale` We need two kinds of comma, one that is a decimal separator, and a second one that is a punctuation mark.

```
\DeclareMathSymbol{\virgola}{\mathpunct}{letters}{"3B}
\DeclareMathSymbol{\virgoladecimale}{\mathord}{letters}{"3B}
```

Math comma activation is done only after the preamble has been completed, that is after the `\begin{document}` statement has been completely executed. Now we must give a definition to the active comma: probably it is not necessary to pass through an intermediate robust command, but certainly it is not wrong to do it.

```
\DeclareRobustCommand*\it@comma@def{\futurelet\let@token\@@math@comma}%
{\catcode `,=\active \gdef,{\it@comma@def}}%
```

The real work shall be performed by `\it@comma@def`. In facts the above macro stores the token that immediately follows `\@@math@comma` into a temporary control sequence that behaves as an implicit character if that token is a single character, even a space, or behaves as an alias of a control sequence otherwise. Actually at the end of the preamble this macro shall be `\let` to be an alias for the real `\@math@comma`.

Is is important to remark that `\@math@comma` must be a command that does not require arguments; this makes it robust when it is followed by other characters that may play special rôles within the arguments of other macros or environments. Matter of fact the first version of this file in version 1.3 did accept an argument; and the result was that the active comma would "eat up" the `&` in vertical math alignments and very nasty errors took place, especially within the `amsmath` defined ones. This macro `\@@math@comma` without arguments does not do any harm to the AMS environments and the actual intelligent comma work shall be executed by other macros.

At this point the situation may become complicated: the comma character in the input file may be followed by a real digit, by an analphabetic character of category 12 (other character), by an implicit digit, by a macro defined to be a digit, by a macro that is not defined to be a digit, by a special character (for example a closed brace, an alignment command, et cetera); therefore it is necessary to distinguish all these situations; remember that an implicit digit cannot be used as a real digit, and a macro gets expanded when used with any `\if` clause, unless it is a `\ifx` one, or is prefixed with `noexpand`. The tests that are going to be made are therefore of different kinds, according to this scheme:

- the `\let@token` is tested against an asterisk to see it it is of category 12; this is true if the token is a real digit, or an implicit digit, or an analphabetic character;
  - an implicit digit is represented by a control sequence; so we first check this feature;
  - if it is a control sequence, we have to test its nature of a digit by testing if it represented one of the ten digits;
  - otherwise it is an analphabetic character.
- otherwise the `\let@token` is a special character or a macro/command;
- a test is made to see if it is a macro; in this case we check if has been defined to be a digit,
- it is not a macro, it must be some other kind of token for example a space or another special character.

Notice that if the token is a macro, we do not test if it is defined to be a single digit or a string made up of more digits and/or other charters. If the macro represents one digit the test is correct, otherwise funny results may take place. For this reason it is always better to prefix any macro with a space, whatever its definition might be; if the macro represents a parameter defined to have a variable value in the range 0–9, then it may represent the fractional part of a (short) decimal value, and it is correct to avoid prefixing it with a space; but the user is warned not to make use of numeric strings in the definition of parameters, unless he knows what he is doing. The user may rather use a balanced brace comma group `{,}` in the input file so that the macro will not be considered by the expansion of the active comma. For example if `\x` is defined to be the numerical string `89`, the source input `$2{,}\x$` will be correctly typeset as 2,89; the input `$2,␣\x$` will be typeset as $2,\,89$ (with an unbreakable thin space after the comma) while `$2,\x$` will be typeset as 29,89, obviously wrong.

So first we test if the comma must act intelligently; if the counter `\Virgola` contains zero, we assume that the comma must always perform as a punctuation mark; but if we want to distinguish if it must behave as a decimal separator, we have to perform more delicate tests; this latter task is demanded to other macros with arguments `\@math@@comma` and `\@@math@@comma`. In order to make the various tests robust we have to resort to the usual trick of the auxiliary macros `\@firstoftwo` and `\@secondoftwo` and various `\expandafter` commands so as to be sure that every `\if` clause is correctly exited without leaving any trace behind.

```
\DeclareRobustCommand*\@math@comma{%
 \ifcat\noexpand\let@token*%
    \expandafter\@firstoftwo
 \else
    \expandafter\@secondoftwo
 \fi{%  \let@token is of  category 12
      \@math@@comma
```

```
    }{% test if \let@token is a macro
       \ifcat\noexpand\let@token\noexpand\relax
          \expandafter\@firstoftwo
       \else
          \expandafter\@secondoftwo
       \fi{% it is a macro
             \@@math@@comma
          }{% it is something else.
             \virgola
          }
    }
}
```

In particular this macro must test if the argument has category code 12, that is "other character", not a letter, nor other special signs, as & for example. In case the category code is not 12, the comma must act as a punctuation mark; but if it is, it might be a digit, or another character, an asterisk, for example; so we have to test its digit nature; the simplest that was found to test if a token is a digit, is to test its ASCII code against the ASCII codes of '0' (zero) and '9'. The typesetting engines give the backtick, `, the property that when a number is required, it yields the ASCII code if the following token in an explicit character or a macro argument; this is why we can't use the temporary implicit token we just tested, but we must examine the first non blank token that follows the `\@math@@comma` macro. Only if the token is a digit, we use the decimal comma, otherwise the punctuation mark. This is therefore the definition of the `\@math@@comma` macro which is not that simple, although the testing macros have clear meanings:

```
\DeclareRobustCommand*\@math@@comma[1]{% argument is certainly of category 12
    \ifcsundef{\expandafter\@gobble\string #1}{% test if it is a real digit
        \ifnumless{`#1}{`0}{\virgola}%
          {\ifnumgreater{`#1}{`9}{\virgola}%
             {\virgoladecimale}}%
    }{% it's an implicit character of category 12
      \let\@tempVirgola\virgola
      \@tfor\@tempCifra:=0123456789\do{%
        \expandafter\if\@tempCifra#1\let\@tempVirgola\virgoladecimale
        \@break@tfor\fi}\@tempVirgola
    }#1}

\DeclareRobustCommand*\@@math@@comma[1]{% argument is a macro
   \let\@tempVirgola\virgola
   \@tfor\@tempCifra:=0123456789\do{%
   \if\@tempCifra#1\let\@tempVirgola\virgoladecimale
   \@break@tfor\fi}\@tempVirgola#1
}
```

The service macros `\ifcsundef`, `\ifnumless`, and `\ifnumgreater` are provided by the `etoolbox` package, that shall be read at most at the end of the `babel` package processing; therefore we must delay the code at "end preamble" time, since only at that time it will be known if the main language is English, or any other one.

This is why we have to perform such a baroque definition as the following one:

```
209 \AtEndOfPackage{\AtEndPreamble{\let\@@math@comma\@math@comma}}
```

This intelligent comma definition is pretty intelligent, but it requires some kind of information from the context; this context does not give enough bits of information to this 'intelligence' in just one case: when the comma plays the rôle of a serial separator in expressions such as $i = 1, 2, 3, \ldots, \infty$, entered as `$i=1,␣2,␣3,\dots,\infty$`. Only in this case the comma must be followed by an explicit space; should this space be absent the macro takes the following non blank token as a digit, and since it actually is a digit, it would use the decimal comma, which would be wrong. The control sequences `\dots` and `\infty` are tested to see if they are undefined, and since they are defined and do not represent digits, the macro inserts a punctuation mark, instead of a decimal separator.

Notice that this macro may appear to be inconsistent with the contents of a language description file. I don't agree: matter of facts even math is part of typesetting a text in a certain language. Does this set of macros influence other language description files? May be, but I think that the clever use of macros `\IntelligentComma` and `\NoIntellingentComma` may solve any interference; they allow to use the proper mark even if the Italian language is not the main language, the important point is to turn the switch on and/or off. By default it is off, so there should not be any interference even with legacy documents typeset in Italian.

Notice that there are other packages that contain facilities for using the decimal comma as the correct decimal separator; for example `SIunitx` defines a command `\num` that not only correctly spaces the decimal separator, but also can change the input glyph with another one, so that it is possible to copy and paste numbers from texts in English (with the decimal point) and paste them into the argument of the `\num` macro in an Italian document where the decimal point is changed automatically into a decimal comma. Of course `SIunitx` does much more than that; if it's being loaded, then the default `\NoIntelligentComma` declaration disables the functionality defined in this language description file and the user can do what he desires with the many functionalities of that package.

Apparently a conflict with the active comma arises with the D column defined by the `dcolomn` package. Disabling the "Italian" active comma allows the D column operate correctly. Thanks to Giuseppe Toscano for telling me about this conflict.

Accents

Most of the other language description files introduce a number of shorthands for inserting accents and other language specific diacritical marks in a more comfortable way compared with the lengthy standard TEX conventions. When an Italian keyboard is being used on a Windows based platform, it exhibits such limitations that to my best knowledge no convenient shorthands have been developed; the reason lies in the fact that the Italian keyboard lacks the grave accent (also known as "backtick"), which is compulsory on all accented vowels, but, on the opposite, it carries the keys with all the accented *lowercase* vowels à, è, é, ì, ò, ù, bot no

uppercase accented vowels are directly avalaible from the keyboard; the keyboard lacks also the tie `~` (tilde) key, while the curly braces require pressing three keys simultaneously.

The best solution Italians have found so far is to use a smart editor that accepts shorthand definitions such that, for example, by striking `"(` one gets directly `{` on the screen and the same sign is saved into the `.tex` file; the same smart editor should be capable of translating the accented characters into the standard TEX sequences when writing a file to disk (for the sake of file portability), and to transform the standard TEX sequences into the corresponding signs when loading a `.tex` file from disk to memory. Such smart editors do exist and can be downloaded from the CTAN archives.

For what concerns the missing backtick key, which is used also for inputting the open quotes, it must be noticed that the shorthand `""` described above completely solves the problem for *double* raised open quotes; besides this, a single open raised quote may be input whit the little known LATEX kernel command `\lq`; according to the traditions of particular publishing houses, since there are no compulsory regulations on the matter, the guillemets may be used; in this case the T1 font encoding solves the problem by means of its built in ligatures `<<` and `>>`; such ligatures are also available when using OpenType fonts with XeLaTeX and LuaLaTeX, provided they are loaded with the option `Ligatures = TeX`. But...

Caporali or French double quotes

Although the T1 font encoding ligatures solve the problem, there are some circumstances where even the T1 font encoding cannot be used, either because the author/typesetter wants to use the OT1 encoding, or because s/he uses a font set that does not comply completely with the T1 font encoding; some virtual fonts, for example, are supposed to implement the double Cork font encoding but actually miss some glyphs; one such virtual font set is given by the `ae` virtual fonts, because they are supposed to implement such double font encoding by using virtual fonts that map the `CM` fonts to a T1 font scheme; the type 1 PostScript version of the `CM` fonts do exist, therefore one believes to be able of using them with pdfLaTeX; but since the `CM` fonts do not contain the guillemets, neither the `AM` ones do. Since guillemets (in Italian *caporali*) do not exist in any OT1 encoded `cm` Latin font, their glyphs must be substituted with something else that fakes them.

In the previous versions of this language description file the absent guillemets were faked with other fonts, by taking example from the solution the French had found for their language description file; they would get suitable guillemets from the cyrillic fonts; this solution was good in most cases, except when the "slides fonts" were used, because there is no Cyryllic slide font around.

This might seem a negligible "feature" because the modern classes or extension modules to produce slides mostly avoid the "old" fonts for slides created by Leslie Lamport when he made available the macro package LaTeX to the TeX community.

Since I designed renewed slide fonts extending those created by Leslie Lamport to the T1 encoding, the Text Companion fonts, and the most frequent "regular"

and AMS math fonts with the same graphic style and excellent legibility (LX-fonts), I thought that this feature is not so negligible. It's true that nowadays nobody should use the old OT1 encoding when typesetting in any language, English included, because independently form the document main language, it is very frequent to quote passages in other languages, or to type foreign proper names of persons or places; nevertheless having in mind a minimum of backwards compatibility and hoping that the deliberate use of OT1 encoding (still necessary to typeset mathematics) is being abandoned, I decided to simplify the previous handling of guillemets.

Therefore here I will test at "begin document" only if the OT1 encoding is the default one, while if the T1 encoding is the default one, that the font collection `AE` is not being used; should it be the case, I will substitute the guillemets with the LaTeX special symbols reduced to script size, and I will not try to fake the guillemets with better solutions; evidently if OpenType fonts are being used, nothing is done; so the tests that follow concern only typesetting old documents or the lack of a wiser choice of fonts and their encodings; an info message is issued and output to the `.log` file.

`\LtxSymbCaporali` `\it@ocap` `\it@ccap`

First the macro `\LtxSymvCaporali` is defined so as to assign a default definition of the faked guillemets: each one of these macro sets actually redefines the control sequences `\it@ocap` and `\it@ccap` that are the ones effectively activated by the shorthands `"<` and `">`. By default the caporali glyphs are taken from T1-encoded fonts; at the end of the preamble some tests are performed to control if the default fonts contain such glyphs, and in case a different font is chosen.

```
\def\LtxSymbCaporali{%
    \DeclareRobustCommand*{\it@ocap}{\mbox{%
       \fontencoding{U}\fontfamily{lasy}\selectfont(\kern-0.20em(}%
       \ignorespaces}%
    \DeclareRobustCommand*{\it@ccap}{\ifdim\lastskip>\z@\unskip\fi
    \mbox{%
       \fontencoding{U}\fontfamily{lasy}\selectfont)\kern-0.20em)}}%
}%
\def\T@unoCaporali{\DeclareRobustCommand*{\it@ocap}{<<\ignorespaces}%
    \DeclareRobustCommand*{\it@ccap}{\ifdim\lastskip>\z@\unskip\fi>>}}%
\T@unoCaporali
```

Nevertheless a macro for choosing where to get glyphs for real guillemets is offered; the syntax is the following:

`\CaporaliFrom{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*open guill. slot*⟩`} {`⟨*close guill. slot*⟩`}`

where ⟨*encoding*⟩ and ⟨*family*⟩ identify the font family name of that particular encoding from which to get the missing guillemets; ⟨*open guill. slot*⟩ and ⟨*close guill. slot*⟩ are the (preferably) decimal slot addresses of the opening and closing guillemets the user wants to use. For example if the T1-encoded Latin Modern fonts are desired, the specific command should be

```
\CaporaliFrom{T1}{lmr}{19}{20}
```

These user choices might be necessary for assuring the correct typesetting with fonts that contain the required glyphs and are available also in PostScript form so as to use them directly with pdfLaTeX, for example.

```
\def\CaporaliFrom#1#2#3#4{%
  \DeclareFontEncoding{#1}{}{}%
  \DeclareTextCommand{\it@ocap}{T1}{%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#3\ignorespaces}}%
  \DeclareTextCommand{\it@ccap}{T1}{\ifdim\lastskip>\z@\unskip\fi%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#4}}%
  \DeclareTextCommand{\it@ocap}{OT1}{%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#3\ignorespaces}}%
  \DeclareTextCommand{\it@ccap}{OT1}{\ifdim\lastskip>\z@\unskip\fi%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#4}}}
```

Notice that the above macro is strictly tied to the T1 encoding; it won't do anything if the default encoding is not the T1 one. Therefore if the AE font collection is being used it would be good idea to issue the command shown above as an example in order to get the proper guillemets[3].

Then we set a boolean variable and test the default family; if such family has a name that starts with the letters "ae" then we have no built in guillemets; of course if the AE font family is chosen after the babel package is loaded, the test does not perform as required.

```
\def\get@ae#1#2#3!{\def\bbl@ae{#1#2}}%
\def\@ifT@one@noCap{\expandafter\get@ae\f@family!%
\def\bbl@temp{ae}\ifx\bbl@ae\bbl@temp\expandafter\@firstoftwo\else
    \expandafter\@secondoftwo\fi}%
```

Now we can set some real settings; first we start by testing the encoding; if the encoding is OT1 we set the faked caporali with LaTeX symbols and issue a warning; then we test if the font family is the AE one we set again the faked caporali and issue another warning[4]; otherwise we set the commands valid for the T1 encoding, that work well also with the TeX Ligatures of the OpenType fonts.

```
\AtBeginDocument{\normalfont\def\bbl@temp{OT1}%
  \ifx\cf@encoding\bbl@temp
    \LtxSymbCaporali
    \GenericWarning{italian.ldf\space}{%
    File italian.ldf warning: \MessageBreak\space\space\space
    With OT1 encoding guillemets are poorly faked\MessageBreak
    \space\space\space
    Use T1 encoding\MessageBreak\space\space\space
    or specify a font with command \string\CaporaliFrom\MessageBreak
    \space\space\space
    See the documentation concerning the babel-italian typesetting
```

[3] Actually the AE fonts should not be used at all; the same results, more or less are obtained by using the Latin Modern ones, that are not virtual fonts and contain the whole T1 font scheme. Nevertheless the faked glyphs are not so bad, so the solution I restored from old versions of the language description file is acceptable

[4] Notice the it is impossible to check if the slots 19 and 20 of the AE fonts are defined by means of the eTeX macro `\iffontchar`, because they are actually defined as black squares!

```
    \MessageBreak\space\space}%
  \else
    \ifx\cf@encoding\bbl@t@one
      \@ifT@one@noCap{%
        \LtxSymbCaporali
        \GenericWarning{italian.ldf\space}{%
        File italian.ldf warning: \MessageBreak\space\space\space
        The AE font collection does not contain the guillemets
        \MessageBreak\space\space\space
        Use the Latin Modern font collection instead
        \MessageBreak\space}
      }%
    {\T@unoCaporali}\fi
  \fi
}
```

Finishing commands

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{italian}%
```