The Italian support for babel

Claudio Beccari — email: `claudio.beccari at gmail.com`

v.1.5.02 — 2026/01/13

Indice

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. This is normally done with the initial installation of TEXLive, but other distributions may not do so.

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 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.

---

[1]The file described in this section has version number v.1.5.02 and was last revised on 2026/01/13. 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)

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

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 user 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, then 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 back tick key, and the back tick 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.

   Remember, though, that at least since 2022 the default input encoding is Unicode; therefore any platform has means to insert non T! encoded characters, and the necessary glyphs are taken from suitable other font encodings, for example the Text Companion fonts with encoding TS1, without the need to load the corresponding packages; on a Mac platform, for example, users can input guillemets by striking `option+\` and `option+|` respectively for « and ». Notice that this specific file is typeset with `pdflatex` and the previous guillemets have been entered directly in the source file by means of « and » while using the Latin Modern fonts.

8. Three specific commands `\unit`, `\ped`, and `\ap` are introduced so as to enable the correct composition of technical mathematics according to the ISO 80000 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`. Command `\unit` does not conflict with package `SIunitx` and it may be redefined by means of its internals. Nevertheless command `\unit` is deactivated by default; users 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`, `ncccomma` and `decimalcomma`; `icomma`, apparently is not in conflict with `dcolumn`, but requires a space after the comma all the times it plays the rôle of a punctuation mark; `ncccomma` and `decimal comma` check if the next token is a digit, but they repeat up to ten tests every time they meet a comma, irrespective from what it is followed by. I believe that

---

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

my solution is better than that provided by those packages; but I assume that if users load one of those packages, they prefer to use their functionalities; in case one of those packages is loaded, this module excludes its own intelligent comma functionality.

By default this functionality is turned *off*, therefore users should turn it on by means of the `\IntelligentComma` command; they can turn it off by means of `\NoIntelligentComma`. Please, read subsection 2.4 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 conflicting packages are being loaded. Apparently there are conflicts with package `unicode-math` because active characters are defined with different codes. Therefore this functionality is not available when typesetting with XeLaTeX or LuaLaTeX unless `decimalcomma` is used, because it is conformant with `unicode-math` settings.

10. Sometimes it is necessary to quote English texts; if such texts contain some math, the `\IntelligentComma` must be deactivated; an `english` environment (not available in `babel` but available in `polyglossia`, although without the functionalities connected with English math typesetting) is defined so that, besides locally setting the English language, issues a `\NoIntelligentComma` command; nevertheless users are still responsible to use the decimal point while typesetting English math.

11. 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 2.4 there are other comments and hints in order to overcome some other keyboard limitations.

Acknowlegements

It is my pleasure to acknowledge the contributions of Enrico Gregorio, 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 with their feedback is essential with open software such as the uncountable contributions to the TEX system. Thank you very much.

2   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}%
```

4

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.

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

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

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

```
5 \addto\captionsitalian{%
6     \def\prefacename{Prefazione}%
7     \def\refname{Riferimenti bibliografici}%
8     \def\abstractname{Sommario}%
9     \def\bibname{Bibliografia}%
10    \def\chaptername{Capitolo}%
11    \def\appendixname{Appendice}%
12    \def\contentsname{Indice}%
13    \def\listfigurename{Elenco delle figure}%
14    \def\listtablename{Elenco delle tabelle}%
15    \def\indexname{Indice analitico}%
16    \def\figurename{Figura}%
17    \def\tablename{Tabella}%
18    \def\partname{Parte}%
19    \def\enclname{Allegati}%
20    \def\ccname{e~p.~c.}%
21    \def\headtoname{Per}%
22    \def\pagename{Pag.}%
23    \def\seename{vedi}%
24    \def\alsoname{vedi anche}%
25    \def\proofname{Dimostrazione}%
26    \def\glossaryname{Glossario}%
27    }%
```

`\dateitalian` The `\dateitalian` macro redefines the command `\today` to produce Italian da-
`\Numberitoday` tes. Normally in Italian day numbers a set as cardinal numbers; some people
`\ISOtoday` prefer to use the ordinal number for day 1, and use the cardinal number with the superscript masculine 'o'. By default this functionality is turned off but the user can select it with command `\OrdinalDayNumberOn` and deselect it with `\OrdinalDayNumberOff`. The national official regulations require the cardinal numbers only with fully numerical dates; they allow, without prescribing, roman numbers and ordinal numbers only when the month name is spelled out in letters.

```
28 \edef\Numbertoday{\the\day/\the\month/\the\year}%
29 %
30 \newcommand\ISOtoday{\number\year-%
31    \ifnum\number\month<10\relax 0\fi\number\month-%
32    \ifnum\number\day<10\relax 0\fi\number\day}%
33 %
34 \newif\ifOrdinalDayNumber \OrdinalDayNumberfalse
35 \def\OrdinalDayNumberOn{\OrdinalDayNumbertrue}
```

```
36 \def\OrdinalDayNumberOff{\OrdinalDayNumberfalse}
37 \def\dateitalian{%
38   \def\today{\ifnum\number\day=\@ne
39       \ifOrdinalDayNumber1\ap{o}\else1\fi
40     \else
41       \number\day
42     \fi~\ifcase\month\or
43     gennaio\or febbraio\or marzo\or aprile\or maggio\or
44     giugno\or luglio\or agosto\or settembre\or ottobre\or
45     novembre\or dicembre\fi\space \number\year}}%
```

Therefore by writing

> \today\qquad\Numbertoday\qquad\ISOtoday

so that, while writing in Italian, we get:

<div align="center">

14 gennaio 2026   14/1/2026   2026-01-14

</div>

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

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

\extrasitalian Lower the chance that clubs or widows occur.

\noextrasitalian
```
47 \addto\extrasitalian{%
48   \babel@savevariable\clubpenalty
49   \babel@savevariable\widowpenalty
50   \babel@savevariable\@clubpenalty
51   \clubpenalty3000\widowpenalty3000\@clubpenalty\clubpenalty}%
```

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

```
52 \addto\extrasitalian{%
53   \babel@savevariable\finalhyphendemerits
54   \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 used with `babel` when typesetting with `xelatex` or `lualatex` the apostrophe must receive a unicode code point. Therefore we use a special test that was suggested by Hironobu Yamashita[3] if this `babel` language option is used while using either `pdflatex` or a Unicode aware typesetting engine.

```
55
56 \addto\extrasitalian{%
57 \lccode\string''=' \ifcsname Umathcode\endcsname
58 \lccode\string"2019=\string"2019\fi}
59 \addto\noextrasitalian{%
60 \lccode\string''=0 \ifcsname Umathcode\endcsname
61 \lccode\string"2019=0\fi}
62
```

---

[3]Thanks to Hironobu Yamashita <h.y.acetaminophen@gmail.com>

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 `\babelfont` 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 no loading precedence requirements when `fontspec` and `polyglossia` are used.

2.1 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.

```
63 \newcount\it@lettering \it@lettering=\@ne
64 \newcommand*\XXIletters{\it@lettering=\z@}
65 \newcommand*\XXVIletters{\it@lettering=\@ne}
66 \let\bbl@alph\@alph \let\bbl@Alph\@Alph
67 \addto\extrasitalian{\babel@savevariable\it@lettering
68   \let\@alph\it@alph \let\@Alph\it@Alph}
69 \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:

```
70 \def\it@alph#1{%
71 \ifcase\it@lettering
72   \ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or
73   l\or m\or n\or o\or p\or q\or r\or s\or t\or u\or v\or
74   z\else\@ctrerr\fi
75 \or
76   \ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or
77   j\or k\or l\or m\or n\or o\or p\or q\or r\or s\or t\or u\or
78   v\or w\or x\or y\or z\else\@ctrerr\fi
79 \fi}%
80 \def\it@Alph#1{%
81 \ifcase\it@lettering
82   \ifcase#1\or A\or B\or C\or D\or E\or F\or G\or H\or I\or
83   L\or M\or N\or O\or P\or Q\or R\or S\or T\or U\or V\or
84   Z\else\@ctrerr\fi
85 \or
86   \ifcase#1\or A\or B\or C\or D\or E\or F\or G\or H\or I\or
87   J\or K\or L\or M\or N\or O\or P\or Q\or R\or S\or T\or U\or
88   V\or W\or X\or Y\or Z\else\@ctrerr\fi
89 \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.

2.2 Support for etymological hyphenation

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

2.2.1 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 in languages that use 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.

In practice the OT1 encoded fonts are sufficient for modern English only; medieval and renaissance English used accents. In modern English accents are not used anymore, but it is frequent to use foreign names and, in reference lists, foreign authors's names and document titles written in foreign languages, so that accents are necessary also with modern English. Therefore OT1 encoded fonts

should never be used in this XXI century. At the same time, as already remarked, the Unicode encoding is become the standard one for all TEX input files, that accented characters are not a problem any more.

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 user hopefully does!).

2.2.2 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}`

```
90 \newcount\it@doublequoteactive \it@doublequoteactive=\z@
91 \def\setactivedoublequote{\it@doublequoteactive=\@ne}
92 {\catcode'"=12 \global\let\it@doublequote"}
93 {\catcode'"=13 \global\let\it@@dqactive"}
94 \AtBeginDocument{%
95   \unless\ifnum\it@doublequoteactive=\z@
96   \initiate@active@char{"}%
97   \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.

```
98    \declare@shorthand{italian}{"}{%
99      \ifmmode
100       \def\it@next{''}%
101      \else
102       \def\it@next{\futurelet\it@temp\it@cwm}%
103      \fi
104      \it@next
105    }%
106 \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.

```
107 \ifdefstring{\languagename}{italian}{\selectlanguage{italian}}{\relax}
108 }%
```

`\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; therefore it is 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 guillemets 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 dash char follows, and in this case the dash char is substituted with a discretionary break that allows hyphenation of both words before and after that dash 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.

```
109 \def\it@@cwm{\bbl@allowhyphens\discretionary{-}{}{}\bbl@allowhyphens}%
110 \def\it@@slash{\bbl@allowhyphens\discretionary{/}{}{/}\bbl@allowhyphens}%
111 \def\it@@trattino{\bbl@allowhyphens\discretionary{-}{}{-}\bbl@allowhyphens}%
112 \def\it@@ocap#1{\it@ocap}\def\it@@ccap#1{\it@ccap}%
113 \DeclareRobustCommand*{\it@cwm}{\let\it@@next\it@doublequote
114 \ifcat\noexpand\it@temp a%
115     \def\it@@next{\it@@cwm}%
116 \else
117   \if\noexpand\it@temp \string|%
118       \def\it@@next{\it@@cwm\@gobble}%
119   \else
120     \if\noexpand\it@temp \string<%
121         \def\it@@next{\it@@ocap}%
122     \else
123       \if\noexpand\it@temp \string>%
124           \def\it@@next{\it@@ccap}%
125       \else
126         \if\noexpand\it@temp\string/%
127             \def\it@@next{\it@@slash\@gobble}%
128         \else
```

```
129            \ifcat\noexpand\it@temp\noexpand\it@@dqactive
130                \def\it@@next{"\@gobble}%
131            \else
132              \if\noexpand\it@temp\string-%
133                  \def\it@@next{\it@@trattino\@gobble}%
134              \fi
135            \fi
136          \fi
137        \fi
138      \fi
139    \fi
140 \fi
141 \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 in Spanish 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 `entrata"/uscita` 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 explicitly 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. Users should realise that with `pdfLaTeX` all accented letters are LICR macros, not plain characters as they are in `xelatex` and `lualatex`.

2.3 Facilities required by the ISO 80000 regulations

The ISO 80000[4] 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 value 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` and `siunitx.sty` packages; we therefore need to test if that package has already been loaded so as to avoid conflicts; we assume that if users load that package, they want 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 superscripts and subscripts (apices and pedices) are typeset in upright font, *not in math italics*, when they represent "adjectives" or appositions to mathematical or physical symbols 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 symbols, such as the notation $\mathbf{A}^T$ 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` First we define the new (internal) commands `\bbl@unit`, `\bbl@ap`, and `\bbl@ped`
`\ap` as robust ones. This facility is deactivated by default according to the contents
`\ped` of an internal counter and the setting of the activation command by the user;
`\setISOcompliance` commands for apices and pedices remain available in any case.

```
142 \newcount\it@ISOcompliance \it@ISOcompliance=\z@
143 \def\setISOcompliance{\it@ISOcompliance=\@ne}
144 \AtBeginDocument{%
145 \unless\ifnum\it@ISOcompliance=\z@%
146 \def\activate@it@unit{\DeclareRobustCommand*{\bbl@it@unit}[1]{%
147     \textormath{\,\textup{##1}}{\,\mathrm{##1}}}}
148 \@ifpackageloaded{units}{}{\@ifpackageloaded{siunitx}{}{%
```

---

[4]When this file was initially written, only the ISO 31/XI regulations existed and it was possible to consult them. In 2009 the new regulations ISO 80000 were published; they were divided in several "parts" to be sold individually; they were translated in other languages, and in Europe and Italy they had prefixes such as EN or UNI; in general the title numerical part remained the same. They are extremely expensive for a private retired citizen, who therefore has access to only a small parts of such regulations. Admittedly they are not so different from the previous edition, but there are several new regulated items, that were not present in the previous edition. An important change is their title: the previous edition was *Mathematical symbols and units for physics and technology* while now it is *Mathematical symbols and units for natural sciences and technology*.

```
149    \@ifpackageloaded{SIunits}{}{%
150    \activate@it@unit\addto\extrasitalian{%
151      \babel@save\unit\let\unit\bbl@it@unit}%
152    }}}%
153  \ifcsstring{bbl@main@language}{italian}{\selectlanguage{italian}}{}%
154  \fi}
155  \DeclareRobustCommand*{\bbl@it@ap}[1]{%
156    \textormath{\textsuperscript{#1}}{^{\mathrm{#1}}}}%
157  \DeclareRobustCommand*{\bbl@it@ped}[1]{%
158    \textormath{$_{\mbox{\fontsize\sf@size\z@
159        \selectfont#1}}$}{_\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 meanings so as to restore them on exit.

```
160  \addto\extrasitalian{%
161    \babel@save\ap\let\ap\bbl@it@ap
162    \babel@save\ped\let\ped\bbl@it@ped
163    }%
```

2.4   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.

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

\IntelligentComma   This feature is optional, in the sense that it is necessary to issue a specific
\NoIntellgentComma   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 action if packages icomma. ncccomma or decimalcomma have been loaded, since it is assumed that the user wants to use their functionality, not the functionality defined in this file.

We need a command to set the comma as an active character 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 comma, but we let \IntelligentComma and \NoIntelligentComma to \relax so that their use is neutralised when one of the named packages is loaded. In this way all known con-

```
165 \AtEndOfPackage{%
166 %
167 \AtEndPreamble{%
168 \newcommand*\IntelligentComma{\mathcode`\,=\string"8000}%
169 \newcommand*\NoIntelligentComma{\mathcode`\,=\string"613B}%
170 \@ifpackageloaded{icomma}{%
171   \let\IntelligentComma\relax
172   \let\NoIntelligentComma\relax}{%
173   \@ifpackageloaded{ncccomma}{%
174     \let\IntelligentComma\relax
175     \let\NoIntelligentComma\relax}{%
176     \@ifpackageloaded{dcolumn}{%
177       \let\IntelligentComma\relax
178       \let\NoIntelligentComma\relax}{%
179       \ProvideDocumentEnvironment{english}{}%
180         {\begin{otherlanguage}{english}%
181           \NoIntelligentComma}%
182         {\end{otherlanguage}}%
183   }}}%
184 }}
```

These commands are defined only in the `babel` support for the Italian language (this file).

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

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

Math comma activation may be 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.

```
187 {\catcode`,\active \protected\gdef,%
188   {\futurelet\tempB\it@tempcomma}}
```

In facts the above macro lets a temporary control sequence `\tempB` be an alias of the token that immediately follows `\it@tempcomma`; `\tempB` then behaves as an implicit character if that token is a single token, even a space (category code 10) or a closed brace category code 3, or an alias of a generic control sequence otherwise.

It is important to remark that `\it@tempcomma` must be a command that does not require arguments; this makes it robust when it is followed by other tokens that may play special rôles within the arguments of other macros or environments. Matter of fact the first version 1.3 of this file did accept an argument; and the result was that the active comma would "gobble" the `&` in vertical math alignments and very nasty errors took place, especially within the `amsmath` defined environments. This macro `\it@tempcomma` without arguments does not do any harm to the AMS environments and a part of the actual intelligent comma work is going to be executed by other macros that accept an argument in a safe way. Since the intelligent comma is going to be used only in math mode, it must be intelligent enough to avoid problems with commands used in math mode; in particular with commands that operate on digits, such as `\bar` and `\overline`

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, and so on); 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 within any `\if` clause, except `\ifx`, or if it is prefixed with `\noexpand`. The tests that are going to be made hereafter are therefore of different kinds, according to this scheme:

- we test if the next character is a space (category code 10) and chose at once the punctuation comma;

- we test if the next character is a closed brace (category code 3) and chose immediately the decimal comma, assuming we are in a situation such as `$2{,}619$`, just for backwards compatibility; such wrapping of the decimal comma was common when no intelligent comma macros or packages were available;

- we have to check if the next token is `bar` or `\overline` (or a `\periodo`, a new command added by this language description file); we assume that these commands are used to mark the digits of the periodic component of the fractional part of a rational number. May be this assumption is not always correct, but it should be the most common situation when one of these commands is immediately preceded by a comma.

- the `\tempB` category is tested against an asterisk to see if 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 might be represented by a control sequence; so we check this feature;

  - if that `\tempB` is a macro, we have to test its nature of a digit by testing if it represents one of the ten digits; of course we strongly discourage to let some control sequences be an alias of one or more digits;

  - in all other cases, as it has category code 12, it is an analphabetic character.

- otherwise the `\tempB` is a special character or a 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 (discouraged, as remarked above);

- if it is not a macro, it must be some other kind of token for example 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 several digits and/or other charters. If the macro represents one digit the test is correct, otherwise unexpected 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 numeric value in the range 0–9, then it may represent the fractional part of

a (single digit) decimal value, and it is correct to avoid prefixing it with a space; but the users are warned not to make use of numeric strings in the definition of parameters, unless they know exactly what they are doing.

Users, if the intelligent comma is active, may use a balanced brace comma group {,} but we discourage it for new documents; in any case such a group works fine if the intelligent comma is not active, for example because a stretch of text in a different language (except English) is being typeset.

So first we test if the comma must act intelligently; we have to perform delicate tests; as a last resource this task is demanded to another macro with arguments: `\@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 each `\if` clause is correctly exited without leaving any trace behind. Similarly the `\ifcsequal` compares two control sequence *names* and compares the equality of their meanings, just as as `\ifx` does, but they are robust in the sense that they don't leave any trace after execution: the third argument code is executed if the test is true, and the fourth one if it is false.

Notice that the tests with macros `\periodo`, `\bar`, and `\overline` are used to verify if the fractional part after the comma sets a line over its periodic component; users are suggested to use the `\periodo` macro that is capable of expanding its argument even if it is a macro (that contains digits); for example the division by 7 yields periods that contains the periodic sequence 142857, starting with any digit of this sequence: $1/7 = 0,\overline{142857}$, $2/7 = 0,\overline{285714}$, $3/7 = 0,\overline{428571}$,.... Users might define six macros with mnemonic names (for example, `\oneoverseven`, `\twooverseven`,...) that contain the six digits in the proper order, and use such macros instead of the digit string, saving themselves possible typos; everybody knows that typos are not allowed in math. In any case these macros should be used as arguments of `periodo`, otherwise the decimal mark test fails: i.e., don't enter `$1/7=0,\oneoverseven$` but `$1/7=0,\periodo{\oneoverseven}$`.

The `\ifdigit` macros that is defined hereafter is a test made by using a LaTeX kernel internal procedure that compares its only argument with one of the ten digits at a time; if the argument is a digit choses the decimal comma, otherwise the punctuation one; the test is similar but not identical to the one performed by the `ncccomma` package; on the opposite the test against a space is performed with a test similar but not identical to the one done by the `icomma` package; all other tests are specific to this package.

```
189
190 \DeclareRobustCommand*\periodo[1]{\overline{#1}}
191
192 \newcommand\ifdigit[1]{\def\tempC{\virgola}%
193 \@tfor\@Cifra:=0123456789\do{%
194 \if\@Cifra #1\relax\def\tempC{\virgoladecimale}%
195 \@break@tfor\fi}\tempC}
196
197 \DeclareRobustCommand*\it@tempcomma{%
198   \ifx\tempB\@sptoken
199     \expandafter\@firstoftwo
200   \else
201     \expandafter\@secondoftwo
202   \fi
203   {\virgola}%
```

```
204  {%
205    \ifcat\noexpand\tempB\noexpand\egroup
206      \expandafter\@firstoftwo
207    \else
208      \expandafter\@secondoftwo
209    \fi
210  {\virgoladecimale}%
211  {%
212    \ifcsequal{tempB}{periodo}{\virgoladecimale}%
213    {%
214      \ifcsequal{tempB}{bar}{\virgoladecimale}%
215      {%
216        \ifcsequal{tempB}{overline}{\virgoladecimale}%
217        {%
218          \ifcat\noexpand\tempB*%
219            \expandafter\@firstoftwo
220          \else
221            \expandafter\@secondoftwo
222          \fi
223          {%  \tempB is of  category 12; test if it is a digit
224            \ifdigit{\tempB}%
225          }%
226          {% test if tempB is a macro
227            \ifcat\noexpand\tempB\noexpand\relax
228              \expandafter\@firstoftwo
229            \else
230              \expandafter\@secondoftwo
231            \fi
232            {\virgola}% \tempB actually is a macro
233            {\@math@@comma}% \tempB is an alias
234          }%
235        }%
236      }%
237    }%
238  }%
239  }%
240 }
241
```

In particular this macro starts testing if it is an alias for a space token (category 10); then tests if it is an alias of `\periodo`, or `\bar` or `\overline` and if it is, the decimal mark should be a decimal comma (see below); otherwise other tests are carried on. This macro then tests if the argument has category code 12, i.e. "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 its category cose is 12, it might be a digit, or another character of category code 12, for example an asterisk; so we have to test its digit nature; the simplest way that we found to test if a token is a digit, is to test if its char code lays within the range delimited by the char codes of '0' (zero) and '9'.

The typesetting engines give the back tick, ', the property that when a number is required, it yields the char code if the following token is an explicit character or a macro argument number; this is why we can't use the temporary implicit token we just tested, but we must examine the first non blank token that follo-

ws 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:

```
242 \DeclareRobustCommand*\@math@@comma[1]{% argument IS of category 12
243   \ifcsundef{\expandafter\@gobble\string #1}%
244   {% test if it is a digit
245       \ifnumless{'#1}{'0}{\virgola}%
246         {\ifnumgreater{'#1}{'9}{\virgola}%
247             {\virgoladecimale}}%
248   }%
249   {% it's an implicit character of category 12
250     \let\@tempVirgola\virgola
251     \@tfor\@tempCifra:=0123456789\do{%
252       \expandafter\if\@tempCifra#1\let\@tempVirgola
253       \virgoladecimale
254       \@break@tfor\fi}\@tempVirgola
255   }#1}%
```

The service macros `\ifcsundef`, `\ifnumless`, and `\ifnumgreater` are provided by the `etoolbox` package, that shall be input 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 Italian, or another one.

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 actually it is a digit, it would use the decimal comma, which is 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 mark.

Notice that this macro may appear to be inconsistent with the contents of a language description file. We 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. This is why above we defined the environment `english` that sets `\NeIntelligentComma` at the beginning of its argument, so as to be sure that where the decimal point is the decimal mark, no active commas are are in force.

Notice that there are other packages that contain facilities for using the decimal comma as the correct decimal mark; 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 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 users can do what they desire 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.

Obsolete arrangements

In the following sections we describe some features that nowadays should be considered totally obsolete. We speak about OT1 encoded fonts (who uses them today?); about the limitations of the Italian keyboard, that with modern operating systems may be overridden easily; with French double quotes, that are being used also in Italian, although without any space separating them from their contents (as it happens in French typography); and so on. Consider that this language description file initial versions are more than 30 years old, and in this long stretch of time many legacy documents were created; the following sections describe what had to be done years ago and the information may be useful to upgrade those documents

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 to the lengthy standard TeX conventions. When an Italian keyboard is being used on a Windows based platform, it exhibits such limitations that to our 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 contains the keys with all the accented *lowercase* vowels à, è, é, ì, ò, ù, bot no *uppercase* accented vowels are directly available from the keyboard; the keyboard lacks also the tie ~ (tilde) key, while the curly braces require pressing three keys simultaneously. On the opposite the Italian keyboard has a key for the 'ç' letter that is not used in Italian.

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 working memory. Some such smart editors do exist and can be downloaded from the CTAN archives.

For what concerns the missing back tick 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 with 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 users prefer employing the old deprecated OT1 encoding, or because they selected a font set that does not completely comply with the T1 font encoding; some virtual fonts, for example, are supposed to implement the double Cork font encoding but actually they miss some glyphs; one such virtual font set was given by the `ae` virtual fonts, because they were supposed to implement such double font encoding by using virtual fonts that mapped the OT1 encoded `CM` fonts to a T1 font scheme; Such fonts are not distributed any more with any TeX system installation, therefore if some vintage source file uses them the correction is to replace them by the Latin Modern ones, that are vector fonts and cannot be used with the plain `latex` engine.

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 `AE` 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. Again the best solution is to correct the preamble of such vintage source files by replacing the call to those old not available fonts with the vector Latin Modern ones.

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 slides 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 the renewed slide fonts were created by extending the Leslie Lamport's one 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 (LXfonts), we 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 from the document main language, it is very frequent to quote text 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, we decided to simplify the previous handling of guillemets.

Therefore here we 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, we will substitute the guillemets with the LaTeX special symbols reduced to script size, and we 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 `\LtxSymbCaporali` 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.

```
256 \def\LtxSymbCaporali{%
257   \DeclareRobustCommand*{\it@ocap}{\mbox{%
258     \fontencoding{U}\fontfamily{lasy}\selectfont(\kern-0.20em(}%
259       \ignorespaces}%
260   \DeclareRobustCommand*{\it@ccap}{\@killglue
261     \mbox{\fontencoding{U}\fontfamily{lasy}\selectfont)%
262     \kern-0.20em)}}}%
263 }%
264 %
265 \def\T@unoCaporali{% default setting
266   \DeclareRobustCommand*{\it@ocap}{<<\ignorespaces}%
267   \DeclareRobustCommand*{\it@ccap}{\@killglue >>}}%
268 \T@unoCaporali
```

Such macros make available the internal commands `\it@ocap` and `\it@ccap`; they become available when users specify in their preambles either the `\LtxSymbCaporali` or in a personal `.sty` file the `\T@unoCaporali` command. The former gets the real glyphs from the `lasy` font and deletes any space after the open guillemets and before the closed ones, as it si necessary in Italian typography; the latter one uses the special ligature of two pairs of angle brackets while deleting any space as the former commands do. Both sets of commands are deprecated, unless vintage files have to be recompiled. Nowadays (2026) with the default UTF-8 encoding and T1 encoded fonts, glyphs « and » may be directly entered.

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}`
or
`\CaporaliFrom{LGR}{Artemisia}{123}{125}`

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, for example, with `pdflatex`.

```
269 \def\CaporaliFrom#1#2#3#4{%
```

```
270    \DeclareFontEncoding{#1}{}{}%
271  %
272    \DeclareTextCommand{\it@ocap}{T1}{%
273      {\fontencoding{#1}\fontfamily{#2}%
274        \selectfont\char#3\ignorespaces}}%
275  %
276    \DeclareTextCommand{\it@ccap}{T1}{\@kilglue
277      {\fontencoding{#1}\fontfamily{#2}\selectfont\char#4}}%
278  }
```

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 a good idea to issue the commands shown above as an example in order to get the proper guillemets[5].

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.

```
279 \def\get@ae#1#2#3!{\def\bbl@ae{#1#2}}%
280 \def\@ifT@one@noCap{\expandafter\get@ae\f@family!%
281 \def\bbl@temp{ae}\ifx\bbl@ae\bbl@temp\expandafter\@firstoftwo\else
282    \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 if the font family is the AE one we set again the faked caporali and issue another warning[6]; otherwise we set the commands valid for the T1 encoding.

```
283 \AtBeginDocument{\normalfont\def\bbl@temp{OT1}%
284    \ifx\cf@encoding\bbl@temp
285      \LtxSymbCaporali
286      \GenericWarning{italian.ldf\space}{%
287      File italian.ldf warning: \MessageBreak\space\space\space
288      With OT1 encoding guillemets are poorly faked\MessageBreak
289      \space\space\space
290      Use T1 encoding\MessageBreak\space\space\space
291      or specify a font with command %
292      \string\CaporaliFrom\MessageBreak\space\space\space
293      See the documentation concerning the babel-italian
294      typesetting\MessageBreak\space\space}%
295    \else
296      \ifx\cf@encoding\bbl@t@one
297        \@ifT@one@noCap{%
298          \LtxSymbCaporali
299          \GenericWarning{italian.ldf\space}{%
300          File italian.ldf warning: \MessageBreak\space\space
301          \space
302          The AE font collection does not contain the guillemets
303          \MessageBreak\space\space\space
304          Use the Latin Modern font collection instead
305          \MessageBreak\space}
```

---

[5]Actually the `AE` fonts should not be used at all; the same results, 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 we restored from old versions of the language description file is acceptable

[6]Notice that 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!

```
306        }%
307     {\T@unoCaporali}\fi
308   \fi
309 }
```

Finishing commands

The `\ldf@finish` macro 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.

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

Riferimenti bibliografici

[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).

[3] Beccari C., "Regole e consigli per comporre la matematica della fisica e delle scienze sperimentali secondo le norme ISO", (2023); PDF document in Italian to be downloaded from the Documentation section of the Italian TUG site www.guitex.org.