Babel support for the Latin language

Claudio Beccari Keno Wehr*

v. 4.1 29th April 2025

Abstract

This manual documents the babel-latin package, which defines all language-specific macros for the babel languages `latin`, `classicallatin`, `medievallatin`, and `ecclesiasticallatin`. These languages are usable with pdfLaTeX, XₑLaTeX, and LuaLaTeX. The `latin` language is even usable with plain TeX (with some restrictions).

> See section 2.5 on how to update from outdated modifiers and the ecclesiastic package.

Contents

---

*Current maintainer. Please report errors to `https://github.com/wehro/babel-latin/issues`.

latin	classicallatin	medievallatin	ecclesiasticallatin
Novembris	Nouembris	Nouembris	Novembris
Praefatio	Praefatio	Præfatio	Præfatio

`\MakeUppercase{Iulius}` yields:

IULIUS	IVLIVS	IVLIVS	IULIUS

Table 1: Spelling differences between the Latin language variants

1   Language variants

Latin has been the most important language of European intellectual life for a long time. Throughout the centuries, many different styles of Latin have been in use concerning wording, spelling, punctuation, and hyphenation. The typographical conventions of an edition of a Latin classic are quite different from those of a liturgical book, even if both have been printed in the 20th century. And even the same Latin text may look quite differently depending on the preferences of the editor and the typographical customs of his country. Latin is supranational, but its typography is not.

To fit all needs, the babel-latin package defines four different language variants of Latin, i. e., four different babel languages. Table 1 shows some differences between the language variants. It is no problem to use different variants of Latin within the same document. If you need classical and modern Latin, just say

```
\usepackage[classicallatin,latin]{babel}
```

and switch the language using the commands described in the babel manual.

The `latin` language – modern Latin   This language variant is intended for the modern usage of Latin; with this we mean the kind of Latin that is used as an official language in the State of Vatican City and in the teaching of Latin in modern schools. Typically, the following alphabet is used:

a b c d e f g h i k l m n o p q r s t u v x y z
A B C D E F G H I K L M N O P Q R S T U V X Y Z

The `classicallatin` language – classical Latin   This language variant is intended for typesetting Latin texts more or less according to the ancient usage of Latin. However, the use of lower-case letters, which are not of ancient origin, is not excluded. The following alphabet is used:

a b c d e f g h i k l m n o p q r s t u x y z
A B C D E F G H I K L M N O P Q R S T V X Y Z

Note that 'V' corresponds to 'u' in lower case. This habit came up in the Middle Ages and is still in use in many text editions. It must be noted that babel-latin does not make any spelling correction in order to use only 'u' in lower case and only 'V' in upper case: if the input text is wrongly typed in, it remains as such; this means it's the typesetter's responsibility to correctly input the source text to be typeset; in spite of this, when the transformation from lower to upper case is performed (such as, for example, while typesetting headers with some document classes) the correct capitalization is performed and 'u' is capitalized to 'V'; the reverse takes place when transforming to lower case.

The `medievallatin` language – medieval/humanist Latin   The spelling is similar to the classical one, but the ligatures æ, Æ, œ, and Œ are used for the respective (former) diphthongs. Again, it is the typesetter's responsibility to input the text to be typeset in a correct way. The following alphabet is used:

a æ b c d e f g h i k l m n o œ p q r s t u x y z
A Æ B C D E F G H I K L M N O Œ P Q R S T V X Y Z

As far as the current maintainer can judge it, the consequent use of 'æ' and 'œ' ligatures came up in 15th century manuscripts in Italy. So this language variant rather reflects the Latin of the humanist/Renaissance period than that of the Middle Ages. However, we stick to the *medieval* name chosen in earlier versions of babel-latin.

The `ecclesiasticallatin` language – ecclesiastical Latin   Ecclesiastical Latin is a spelling variety of modern Latin, which is used above all in liturgical books of the Roman Catholic Church, where the ligatures æ and œ are widely used and where acute accents are used in order to mark the tonic vowel of words with more than two syllables to make sure the correct stress. The following alphabet is used:

a æ b c d e f g h i k l m n o œ p q r s t u v x y z
A Æ B C D E F G H I K L M N O Œ P Q R S T U V X Y Z

This language variant also contains a certain degree of "Frenchization" of spaces around some punctuation marks and guillemets: 1/12 of a quad is inserted before '!', '?', ':', ';', '»', and '›' as well as after '«' and '‹'. The spacing of guillemets does not work with pdfTEX except when using the shorthands "< and "> (see section 4).

For what concerns babel and typesetting with TEX, the differences between the language variants reveal themselves in the strings used to name, for example, the "Preface", that becomes "Praefatio" or "Præfatio", respectively. Hyphenation rules are also different, cf. section 3.

The name strings for chapters, figures, tables, et cetera, have been suggested by prof. Raffaella Tabacco, a latinist of the University of Vercelli, Italy, to whom we address our warmest thanks. The names suggested by Krzysztof Konrad Żelechowski, when different, are used as the names for the medieval variety, since he made a word and spelling choice more suited for this variety.

2 Modifiers

The four language variants described above do not cover all variations of Latin typography. Additionally there are several *modifiers*: usej, lowercasemonth, withprosodicmarks, and ecclesiasticfootnotes. The meaning of these modifiers is explained below.

To apply a modifier you have to append it (prefixed with a dot) to the language name when loading babel:

    \usepackage[ecclesiasticallatin.lowercasemonth]{babel}

If you need two modifiers or more, just concatenate them in arbitrary order:

    \usepackage[latin.usej.withprosodicmarks]{babel}

2.1 The letter *j*

The letter *j* is not of ancient origin. In early modern times, it was used to distinguish the consonantic *i* from the vocalic *i*. In liturgical books *j* was in use until the 1960s. Nowadays, the use of *j* has disappeared from most Latin publications. This is why babel-latin does not use *j* in predefined terms by default. Use the usej modifier if you prefer *Januarii* and *Maji* to *Ianuarii* and *Maii*.

2.2 Case of month names

Traditionally, Latin month names are capitalized: *Ianuarii, Februarii, Martii, …* (We state the genitive forms here as this is what we need for Latin dates.) So babel-latin capitalizes the month names for all four language variants. However, in recent liturgical books month names are written in lower case (as in Romance languages). Use the lowercasemonth modifier if you prefer not to capitalize the month names printed by the \today command: *ianuarii, februarii, martii, …*

2.3 Shorthands for prosodic marks

Textbooks, grammars, and dictionaries often use letters with prosodic marks (macrons and breves) like 'ā' and 'ă' to mark long and short vowels. On modern systems, the required characters can be input directly thanks to Unicode. For backwards compatibility and as an perhaps more comfortable alternative even today, babel-latin provides shorthands for prosodic marks if you load the language with the withprosodicmarks modifier.

Note that these shorthands may interfere with other packages. The active = character used for macrons will cause problems with commands using key=value interfaces, such as the command \includegraphics[scale=2]{...}. Therefore, the shorthands are disabled by default. You have to use dedicated commands to turn them on and off. Use \ProsodicMarksOn to enable them an \ProsodicMarksOff to disable them again. To get "Găllĭă ĕst ŏmnĭs dīvīsă ĭn părtēs trēs", type:

    \ProsodicMarksOn
    G^all^i^a ^est ^omn^is d=iv=is^a ^in p^art=es tr=es
    \ProsodicMarksOff

The following shorthands are available:

=a   for ā (a with macron), also available for ē, ī, ō, ū, and ȳ

=A   for Ā (A with macron), also available for Ē, Ī, Ō, Ū, V̄, and Ȳ. Note that a macron above the letter V is only displayed if your font supports the Unicode character U+0304 (*combining macron*).

=ae   for a͞e (ae diphthong with macron, for `latin` and `classicallatin`) or æ͞ (ae ligature with macron, for `medievallatin` and `ecclesiasticallatin`), respectively; also available for a͞u, e͞u, and o͞e/œ͞. Note that macrons above diphthongs are only displayed if your font supports the Unicode character U+035E (*combining double macron*), which always requires X͟ΗLA͟TEX or LuaLA͟TEX.[1]

=Ae   for A͞e (Ae diphthong with macron, for `latin` and `classicallatin`) or Æ͞ (AE ligature with macron, for `medievallatin` and `ecclesiasticallatin`), respectively; also available for A͞u, E͞u, and O͞e/Œ͞.

=AE   for A͞E (AE diphthong with macron, for `latin` and `classicallatin`) or Æ͞ (AE ligature with macron, for `medievallatin` and `ecclesiasticallatin`), respectively; also available for A͞U, E͞U, and O͞E/Œ͞.

^a   for ă (a with breve), also available for ĕ, ĭ, ŏ, ŭ, and y̆. Note that a breve above the letter y is only displayed if your font supports the Unicode character U+0306 (*combining breve*).

^A   Ă (A with breve), also available for Ĕ, Ĭ, Ŏ, Ŭ, V̆, and Y̆. Note that breves above the letters V and Y are only displayed if your font supports the Unicode character U+0306 (*combining breve*).

Note the incompatibilities described in section 5.

2.4   Ecclesiastical footnotes

The ecclesiastic package, an outdated extension of former versions of babel-latin, typeset footnotes with ordinary instead of superior numbers and without indentation.

As many ecclesiastical documents and liturgical books use footnotes that are very similar to the ordinary LATEX ones, we do not use this footnote style as default even for the `ecclesiasticallatin` language variant. But you may use the `ecclesiasticfootnotes` modifier (with any variant of Latin) if you prefer that footnote style.

Note that this modifier affects the entire document. It can only be applied to the document's main language.

2.5   Legacy modifiers and language options

babel-latin defined only one single babel language up to v. 3.5. Language variants used to be accessible via modifiers. This approach has proved to be disadvantageous concerning compatibility with other language-specific packages like biblatex. That's why v. 4.0 introduced the `classiclatin`, `medievallatin`, and `ecclesiasticlatin` languages. `classiclatin` and `ecclesiasticlatin` have been renamed to `classicallatin` and `ecclesiasticallatin` in v. 4.1 for the sake of philological correctness.

---

[1] A good choice for a font supporting the combining double macron might be *Libertinus Serif*, the font of this manual.

Language variant	Hyphenation style	Name of patterns
`latin`	modern	`latin`
`classicallatin`	classical	`classiclatin`
`medievallatin`	modern	`latin`
`ecclesiasticallatin`	modern	`latin`
–	liturgical	`liturgicallatin`

Table 2: Latin hyphenation styles

The legacy modifiers `classic`, `medieval`, and `ecclesiastic` as well as the language options `classiclatin` and `ecclesiasticlatin` are still available and backwards compatibility is made sure. However, a warning is issued if you use one of these modifiers or language options. They may be dropped from babel-latin in a future version.

For maximum compatibility, replace

- `\usepackage[classiclatin]{babel}` by `\usepackage[classicallatin]{babel}`,

- `\usepackage[ecclesiasticlatin]{babel}` by `\usepackage[ecclesiasticallatin]{babel}`,

- `\usepackage[latin.classic]{babel}` by `\usepackage[classicallatin]{babel}`,

- `\usepackage[latin.medieval]{babel}` by `\usepackage[medievallatin]{babel}`,

- `\usepackage[latin.ecclesiastic]{babel}` by
  `\usepackage[ecclesiasticallatin.ecclesiasticfootnotes,activeacute]{babel}`.

The last replacement is also recommended if you have been loading the ecclesiastic package so far. This package is no longer necessary as its functionality is provided by babel-latin now.

3 Hyphenation

There are three different sets of hyphenation patterns for Latin, reflecting three different styles of hyphenation: *classical*, *modern*, and *liturgical*. Separate documention for these hyphenation styles is available on the Internet.[2] Each of the four Latin language variants has its default hyphenation style as indicated by table 2. Use the `\babelprovide` command with the `hyphenrules` option if the default style does not fit your needs.

To typeset a liturgical book in the recent "Solesmes style" say

```
\usepackage[ecclesiasticallatin.lowercasemonth]{babel}
\babelprovide[hyphenrules=liturgicallatin]{ecclesiasticallatin}
```

The typical commands for a Latin text edition in the German-speaking world will be

```
\usepackage[latin]{babel}
\babelprovide[hyphenrules=classiclatin]{latin}
```

Note that the liturgical hyphenation patterns are the default of none of the language variants. To use them, you have to load them explicitly in any case.

---

[2] `https://github.com/gregorio-project/hyphen-la/blob/master/doc/README.md#hyphenation-styles`

4 Shorthands

The following shorthands are available for all variants of Latin. Note that shorthands beginning with ' are only available if you load babel with the `activeacute` option.

- `"<` for « (left guillemet)

- `">` for » (right guillemet)

- `"` If no other shorthand applies, `"` before any letter character defines an optional break point allowing further break points within the same word (as opposed to the `\-` command).

- `"|` the same as `"`, but also possible before non-letter characters

- `'a` for á (a with acute), also available for é, í, ó, ú, ý, ǽ, and ǿ

- `'A` for Á (A with acute), also available for É, Í, Ó, Ú, V́, Ý, Ǽ, and Ǿ

The following shorthands are only available for the `medievallatin` and the `ecclesiasticallatin` languages. Again, the shorthands beginning with ' only work with babel's `activeacute` option.

- `"ae` for æ (ae ligature), also available for œ

- `"Ae` for Æ (AE ligature), also available for Œ

- `"AE` for Æ (AE ligature), also available for Œ

- `'ae` for ǽ (ae ligature with acute), also available for ǿ

- `'Ae` for Ǽ (AE ligature with acute), also available for Ǿ

- `'AE` for Ǽ (AE ligature with acute), also available for Ǿ

Furthermore, there are shorthands for prosodic marks; see section 2.3. Note the incompatibilities described in section 5.

5 Incompatibilities with other packages

5.1 unicode-math

Loading the Latin language together with the `activeacute` babel option may cause error messages if the unicode-math package is loaded. Do not use `activeacute` if you need unicode-math, even if Latin is only a secondary language of your document.[3]

---

[3]See `https://github.com/wspr/unicode-math/issues/462` and `https://github.com/reutenauer/polyglossia/issues/394` for related discussions.

5.2 LuaTeX

The " character is made active by babel-latin; its use within the `\directlua` command will lead to problems (except in the preamble). Switch the shorthand off for such commands:

```
\shorthandoff{"}
\directlua{tex.print("Salve")}
\shorthandon{"}
```

You may avoid the shorthand switching by using single instead of double quotes. However, note that this will not work if the `activeacute` option is used, as ' is active in this case as well.

Furthermore, beware of using `\directlua` commands containing the = character between `\ProsodicMarksOn` and `\ProsodicMarksOff` if you load the Latin language with the `withprosodicmarks` modifier.

5.3 babel-turkish

Both Turkish and Latin (when loaded with the `withprosodicmarks` modifier) make the = character active. However, babel-latin takes care the active behaviour of this character is only enabled between `\ProsodicMarksOn` and `\ProsodicMarksOff` to avoid conflicts with packages using `key=value` interfaces.

If you need Latin with prosodic shorthands and Turkish with active = character in one document, you have to say `\shorthandon{=}` before the first occurence of = in each Turkish text part.

5.4 babel-esperanto, babel-kurmanji, and babel-slovak

Esperanto, Kurmanji, Slovak, and Latin (when loaded with the `withprosodicmarks` modifier) make the ^ character active. However, babel-latin takes care the active behaviour of this character is only enabled between `\ProsodicMarksOn` and `\ProsodicMarksOff` to avoid conflicts with TeX's ^^xx convention.

If you need Latin with prosodic shorthands and Esperanto/Kurmanji/Slovak with active ^ character in one document, you have to say `\shorthandon{^}` before the first occurence of ^ in each Esperanto/Kurmanji/Slovak text part.

6 Plain TeX

According to the babel manual, the recommended way to load the Latin language in plain TeX is:

```
\input latin.sty
\begindocument
```

The modifiers `usej` and `lowercasemonth` may be accessed by means of the `\languageattribute` command:

```
\input latin.sty
\languageattribute{latin}{usej,lowercasemonth}
\begindocument
```

babel does not provide sty files for classicallatin, medievallatin, and ecclesiasticallatin. It should be possible to create them locally if needed.

Note that no Latin shorthands are available in plain TeX.

7 The code

We identify the language definition file.

```
1 \ProvidesLanguage{latin}[2025-04-29 v4.1 Latin support from the babel system]
```

The macro `\LdfInit` takes care of preventing that this file is loaded more than once with the same option, checking the category code of the @ sign, etc. `\CurrentOption` is the language requested by the user, i. e., latin, classicallatin, medievallatin, or ecclesiasticallatin.

```
2 \LdfInit\CurrentOption{captions\CurrentOption}
```

For tests, we need variables containing several possible values of the language name (including classiclatin and ecclesiasticlatin, which are outdated, but still supported).

```
3 \def\babellatin@classical{classicallatin}
4 \def\babellatin@classic{classiclatin}
5 \def\babellatin@medieval{medievallatin}
6 \def\babellatin@ecclesiastical{ecclesiasticallatin}
7 \def\babellatin@ecclesiastic{ecclesiasticlatin}
```

7.1 Hyphenation patterns

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

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

We define macros for testing if the required hyphenation patterns are available.

```
 9 \def\babellatin@test@modern@patterns{%
10   \ifx\l@latin\undefined
11     \@nopatterns{latin}%
12     \adddialect\l@latin0
13   \fi}%
14 \def\babellatin@test@classical@patterns{%
15   \ifx\l@classiclatin\undefined
16     \PackageWarningNoLine{babel-latin}{%
17       No hyphenation patterns were found for the\MessageBreak
18       classicallatin language. Now I will use the\MessageBreak
19       patterns for modern Latin instead}%
20     \babellatin@test@modern@patterns
21     \adddialect\l@classiclatin\l@latin
22   \fi}%
```

We use the classiclatin hyphenation patterns for classical Latin and the (modern) latin hyphenation patterns for all other varieties of Latin.

```
23 \ifx\CurrentOption\babellatin@classical
24   \babellatin@test@classical@patterns
25   \adddialect\l@classicallatin\l@classiclatin
26 \else
27   \ifx\CurrentOption\babellatin@classic
28     \babellatin@test@classical@patterns
```

```
29  \else
30    \ifx\CurrentOption\babellatin@ecclesiastical
31      \babellatin@test@modern@patterns
32      \adddialect\l@ecclesiasticallatin\l@latin
33    \else
34      \ifx\CurrentOption\babellatin@ecclesiastic
35        \babellatin@test@modern@patterns
36        \adddialect\l@ecclesiasticlatin\l@latin
37      \else
38        \ifx\CurrentOption\babellatin@medieval
39          \babellatin@test@modern@patterns
40          \adddialect\l@medievallatin\l@latin
41        \else
42          \babellatin@test@modern@patterns
43        \fi
44      \fi
45    \fi
46  \fi
47 \fi
```

7.2 Latin captions

We need a conditional governing the spelling of the captions. Medieval and ecclesiastical Latin use the ligatures æ and œ, classical and modern Latin do not.

```
48 \newif\ifbabellatin@useligatures
49 \addto\extrasmedievallatin{\babellatin@useligaturestrue}%
50 \addto\noextrasmedievallatin{\babellatin@useligaturesfalse}%
51 \addto\extrasecclesiasticallatin{\babellatin@useligaturestrue}%
52 \addto\noextrasecclesiasticallatin{\babellatin@useligaturesfalse}%
53 \addto\extrasecclesiasticlatin{\babellatin@useligaturestrue}%
54 \addto\noextrasecclesiasticlatin{\babellatin@useligaturesfalse}%
```

We define the Latin captions using the commands recommended by the babel manual.[4]

```
55 \StartBabelCommands*{\CurrentOption}{captions}
56   \SetString\prefacename{\ifbabellatin@useligatures Pr\ae fatio\else Praefatio\fi}
57   \SetString\refname{Conspectus librorum}
58   \SetString\abstractname{Summarium}
59   \SetString\bibname{Conspectus librorum}
60   \SetString\chaptername{Caput}
61   \SetString\appendixname{Additamentum}
62   \SetString\contentsname{Index}
63   \SetString\listfigurename{Conspectus descriptionum}
64   \SetString\listtablename{Conspectus tabularum}
65   \SetString\indexname{Index rerum notabilium}
66   \SetString\figurename{Descriptio}
67   \SetString\tablename{Tabula}
68   \SetString\partname{Pars}
69   \SetString\enclname{Adduntur}%   Or "Additur"? Or simply Add.?
70   \SetString\ccname{Exemplar}%     Use the recipient's dative
71   \SetString\headtoname{\ignorespaces}% Use the recipient's dative
72   \SetString\pagename{Charta}
73   \SetString\seename{cfr.}
```

---

[4]Most of these names were kindly suggested by Raffaella Tabacco.

```
74  \SetString\alsoname{cfr.}% Tabacco never saw "cfr" + "atque" or similar forms
75  \SetString\proofname{Demonstratio}
76  \SetString\glossaryname{Glossarium}
77 \EndBabelCommands
```

In the above definitions there are some points that might change in the future or that require a minimum of attention from the typesetter.

1. The `\enclname` is translated by a passive verb, that literally means "(they) are being added"; if just one enclosure is joined to the document, the plural passive is not suited any more; nevertheless a generic plural passive might be incorrect but suited for most circumstances. On the opposite "Additur", the corresponding singular passive, might be more correct with one enclosure and less suited in general: what about the abbreviation "Add." that works in both cases, but certainly is less elegant?

2. The `\headtoname` is empty and gobbles the possible following space; in practice the typesetter should use the dative of the recipient's name; since nowadays not all such names can be translated into Latin, they might result indeclinable. The clever use of a dative appellative by the typesetter such as "Domino" or "Dominae" might solve the problem, but the header might get too impressive. The typesetter must make a decision on his own.

3. The same holds true for the copy recipient's name in the "Cc" field of `\ccname`.

7.3   Mapping between upper and lower case

For classical and medieval Latin we need the suitable correspondence between uppercase V and lower-case u since in that spelling there is only one letter for the vowel and the consonant, and the u shape is an (uncial) variant of the capital V.

We set the mapping only if the LaTeX format is used because we need commands for it that are not available in plain TeX. The following commands take care for the correct behaviour of the `\MakeUppercase` and the `\MakeLowercase` command. They make sure that `\MakeUppercase{Heluetia}` yields "HELVETIA" and that `\MakeLowercase{LVDVS}` yields "ludus".

```
78 \def\babellatin@latex{LaTeX2e}%
79 \ifx\fmtname\babellatin@latex
80   \DeclareUppercaseMapping[la-x-classic]{`u}{V}
81   \DeclareLowercaseMapping[la-x-classic]{`V}{u}
```

The mapping for medieval Latin is part of the l3text module, which is part of the LaTeX format (see `source3.pdf`).

For Unicode-based engines, we also have to take into account characters with diacritics. We map ú, ū, and ŭ to V with the respective diacritic.

```
82   \ExplSyntaxOn
83   \sys_if_engine_opentype:T
84     {
85       \DeclareUppercaseMapping[la-x-classic]{`ú}{\a'{V}}
86       \DeclareUppercaseMapping[la-x-classic]{`ū}{\a={V}}
87       \DeclareUppercaseMapping[la-x-classic]{`ŭ}{\u{V}}
88       \DeclareUppercaseMapping[la-x-medieval]{`ú}{\a'{V}}
89       \DeclareUppercaseMapping[la-x-medieval]{`ū}{\a={V}}
90       \DeclareUppercaseMapping[la-x-medieval]{`ŭ}{\u{V}}
```

11

```
91      }
92    \ExplSyntaxOff
93 \fi
```

The following `\BabelLower` command takes care for the correct hyphenation of words written in capital letters. It makes sure that "LVDVS" is hyphenated the same way as "ludus".

```
94 \StartBabelCommands*{classicallatin,classiclatin,medievallatin}{}
95   \SetHyphenMap{\BabelLower{`V}{`u}}
96 \EndBabelCommands
```

7.4   The Latin date

We need three conditionals governing the spelling of the month names. Ecclesiastical and modern Latin use the character v, classical and medieval Latin use only u. This affects the month of November. The user may demand to use the letter j where suitable or to lowercase month names using the respective modifiers.

```
 97 \newif\ifbabellatin@usev
 98 \newif\ifbabellatin@usej
 99 \newif\ifbabellatin@lowercasemonth
100 \babellatin@usevtrue
101 \addto\extrasclassicallatin{\babellatin@usevfalse}%
102 \addto\noextrasclassicallatin{\babellatin@usevtrue}%
103 \addto\extrasclassiclatin{\babellatin@usevfalse}%
104 \addto\noextrasclassiclatin{\babellatin@usevtrue}%
105 \addto\extrasmedievallatin{\babellatin@usevfalse}%
106 \addto\noextrasmedievallatin{\babellatin@usevtrue}%
```

The Latin month names are needed in the genitive case.

```
107 \def\babellatin@monthname{%
108   \ifcase\month\or\ifbabellatin@usej Januarii\else Ianuarii\fi
109     \or Februarii%
110     \or Martii%
111     \or Aprilis%
112     \or\ifbabellatin@usej Maji\else Maii\fi
113     \or\ifbabellatin@usej Junii\else Iunii\fi
114     \or\ifbabellatin@usej Julii\else Iulii\fi
115     \or Augusti%
116     \or Septembris%
117     \or Octobris%
118     \or\ifbabellatin@usev Novembris\else Nouembris\fi
119     \or Decembris%
120   \fi}%
```

Depending on the chosen language, we have to define a `\latindate`, `\classicallatindate`, `\medievallatindate`, or `\ecclesiasticallatindate` command. The date format is "XXXI Decembris MMXXI".

```
121 \expandafter\def\csname date\CurrentOption\endcsname{%
122   \def\today{%
123     \uppercase\expandafter{\romannumeral\day}~%
124     \ifbabellatin@lowercasemonth
125       \lowercase\expandafter{\babellatin@monthname}%
126     \else
127       \babellatin@monthname
```

```
128    \fi
129    \space
130    \uppercase\expandafter{\romannumeral\year}%
131  }%
132 }%
```

7.5 Shorthands

We define shorthands only if the LaTeX format is used because we need commands for
them that are not available in plain TeX.

```
133 \ifx\fmtname\babellatin@latex
```

Every shorthand character needs an `\initiate@active@char` command, which makes
the respective character active, but expanding to itself as long as no further definitions
occur. The apostrophe (acute) is only made active if babel has been called with the
`activeacute` option.

```
134    \initiate@active@char{"}%
135    \@ifpackagewith{babel}{activeacute}{\initiate@active@char{'}}{}%
```

The following command is defined by the hyperref package. We use a dummy definition
if this package is not loaded.

```
136    \providecommand\texorpdfstring[2]{#1}%
```

A peculiarity of the babel-latin package are shorthands of different lengths. `"` before a
letter character defines an additional hyphenation point, but `"ae` is a shorthand for the
ligature 'æ' in medieval and ecclesiastical Latin. So the shorthands definitions are rather
complex and we need expl3 syntax for them.

```
137    \ExplSyntaxOn
```

The character `"` is used as a shorthand unconditionally. In math mode it expands to
itself. In text mode it is defined as a macro with one parameter. This makes it possible
to read the following token, on which the actual meaning of the shorthand depends.

```
138    \declare@shorthand {latin} {"}
139      {
140        \mode_if_math:TF { \token_to_str:N " }
141          {
142            \texorpdfstring { \babellatin_apply_quotemark:N } { }
143          }
144      }
```

The character `'` is used as a shorthand if the `activeacute` option is used. So we have
to use a macro for the declaration, which can be called if necessary. In math mode the
shorthand expands to `\active@math@prime` as defined in `latex.ltx`. In text mode it is a
macro with one argument to read the following token.

```
145    \cs_set_protected:Npn \babellatin@declare@apostrophe@shorthands
146      {
147        \declare@shorthand {latin} {'}
148          {
149            \mode_if_math:TF { \active@math@prime }
150              {
151                \texorpdfstring { \babellatin_put_acute:N } { \' }
152              }
153          }
154      }
```

The characters = and ^ are only used as shorthands if the `withprosodicmarks` modifier is used. So we have to use a macro for the declaration, which can be called if necessary. In math mode both shorthands expand to themselves. In text mode they are macros with one argument to read the following token.

```
155  \cs_set_protected:Npn \babellatin@declare@prosodic@shorthands
156    {
157      \declare@shorthand {latin} {=}
158        {
159          \mode_if_math:TF { \token_to_str:N = }
160            {
161              \texorpdfstring { \babellatin_put_macron:N } { \= }
162            }
163        }
164      \declare@shorthand {latin} {^}
165        {
166          \mode_if_math:TF { \token_to_str:N ^ } { \babellatin_put_breve:N }
167        }
168    }
```

The following macro defines the behaviour of the active " character. The shorthands "AE, "Ae, "ae, "OE, "Oe, and "oe are used for ligatures if the current variety of Latin uses them. In other cases " before any letter character or before \AE, \ae, \OE, and \oe defines an additional hyphenation point. "| defines an additional hyphenation point as well. The shorthands "< and "> are used for guillemets. In other cases the active " character expands to itself and the token read as argument is reinserted.

If the argument is a braced group (e. g. if the user has typed "{ab}, unexpected behaviour may occur as the conditionals \token_if_letter:NTF and \babellatin_if_ligature_command:NTF expect a single token as first argument. Therefore we need to check if the argument is a single token using the \tl_if_single_token:nTF command before using those conditionals.

```
169  \cs_set_protected:Npn \babellatin_apply_quotemark:N #1
170    {
171      \str_case:nnF {#1}
172        {
173          {A} { \babellatin_ligature_shorthand:Nnn E { \AE }
174                {
175                  \babellatin_ligature_shorthand:Nnn e { \AE }
176                    {
177                      \babellatin_allowhyphens: A
178                    }
179                }
180              }
181          {a} { \babellatin_ligature_shorthand:Nnn e { \ae }
182                {
183                  \babellatin_allowhyphens: a
184                }
185              }
186          {O} { \babellatin_ligature_shorthand:Nnn E { \OE }
187                {
188                  \babellatin_ligature_shorthand:Nnn e { \OE }
189                    {
190                      \babellatin_allowhyphens: O
191                    }
```

```
192                         }
193                     }
194             {o} { \babellatin_ligature_shorthand:Nnn e { \oe }
195                     {
196                         \babellatin_allowhyphens: o
197                     }
198                 }
199         {|} { \babellatin_allowhyphens: }
200         {<} { \babellatin@guillemetleft }
201         {>} { \babellatin@guillemetright }
202         }
203         {
204             \tl_if_single_token:nTF {#1}
205                 {
206                     \token_if_letter:NTF #1 { \babellatin_allowhyphens: }
207                         {
208                             \babellatin_if_ligature_command:NTF #1 { \babellatin_allowhyphens: }
209                                 {
210                                     \token_to_str:N "
211                                 }
212                         }
213                 }
214                 {
215                     \token_to_str:N "
216                 }
217             #1
218         }
219     }
```

The following macro defines the behaviour of the active ' character. The shorthands 'AE, 'Ae, 'ae, 'OE, 'Oe, and 'oe are used for accented ligatures if the current variety of Latin uses them. In other cases ' before any vowel or before \AE, \ae, \OE, and \oe defines an accented character. The character V is treated as a vowel here as it may represent the vowel U, but v is not, as it is never used for a vowel. In other cases the active ' character expands to itself and the token read as argument is reinserted.

```
220   \cs_set_protected:Npn \babellatin_put_acute:N #1
221     {
222         \str_case:nnF {#1}
223             {
224                 {A} { \babellatin_ligature_shorthand:Nnn E { \a'\AE }
225                         {
226                             \babellatin_ligature_shorthand:Nnn e { \a'\AE } { Á }
227                         }
228                     }
229                 {a} { \babellatin_ligature_shorthand:Nnn e { \a'\ae } { á } }
230                 {E} { É }
231                 {e} { é }
232                 {I} { Í }
233                 {i} { í }
234                 {O} { \babellatin_ligature_shorthand:Nnn E { \a'\OE }
235                         {
236                             \babellatin_ligature_shorthand:Nnn e { \a'\OE } { Ó }
237                         }
238                     }
```

```
239          {o} { \babellatin_ligature_shorthand:Nnn e { \a'\oe } { ó } }
240          {U} { Ú }
241          {u} { ú }
242          {V} { \a'V }
243          {Y} { \a'Y }
244          {y} { \a'y }
245          {Æ} { \a'\AE }
246          {æ} { \a'\ae }
247          {Œ} { \a'\OE }
248          {œ} { \a'\oe }
249        }
250        {
251          \tl_if_single_token:nTF {#1}
252            {
253              \babellatin_if_ligature_command:NTF #1 { \a' }
254                {
255                  \token_to_str:N '
256                }
257            }
258            {
259              \token_to_str:N '
260            }
261          #1
262        }
263    }
```

The following macro defines the behaviour of the active = character. The shorthands =AE, =Ae, =ae, =AU, =Au, =au, =EU, =Eu, =eu, =OE, =Oe, and =oe are used for diphthongs with a combining double macron (U+035E) or ligatures with a macron if the current variety of Latin uses them. In other cases = before any vowel puts a macron above the vowel. The character V is treated as a vowel here as it may represent the vowel U, but v is not, as it is never used for a vowel. In other cases the active = character expands to itself and the token read as argument is reinserted.

```
264    \cs_set_protected:Npn \babellatin_put_macron:N #1
265      {
266        \str_case:nnF {#1}
267          {
268            {A} { \babellatin_ligature_macron:NNnn AE { \a=\AE }
269                {
270                    \babellatin_ligature_macron:NNnn Ae { \a=\AE }
271                      {
272                        \babellatin_diphthong_macron:NNn AU
273                          {
274                            \babellatin_diphthong_macron:NNn Au { \a=A }
275                          }
276                      }
277                }
278            }
279            {a} { \babellatin_ligature_macron:NNnn ae { \a=\ae }
280                {
281                    \babellatin_diphthong_macron:NNn au { \a=a }
282                }
283            }
284            {E} { \babellatin_diphthong_macron:NNn EU
```

```
285                        {
286                            \babellatin_diphthong_macron:NNn Eu { \a=E }
287                        }
288                    }
289            {e} { \babellatin_diphthong_macron:NNn eu { \a=e } }
290            {I} { \a=I }
291            {i} { \a=\i }
292            {O} { \babellatin_ligature_macron:NNnn OE { \a=\OE }
293                        {
294                            \babellatin_ligature_macron:NNnn Oe { \a=\OE } { \a=O }
295                        }
296                    }
297            {o} { \babellatin_ligature_macron:NNnn oe { \a=\oe } { \a=o } }
298            {U} { \a=U }
299            {u} { \a=u }
300            {V} { \a=V }
301            {Y} { \a=Y }
302            {y} { \a=y }
303        }
304        {
305            \tl_if_single_token:nTF {#1}
306              {
307                \babellatin_if_ligature_command:NTF #1 { \a= }
308                  {
309                    \token_to_str:N =
310                  }
311              }
312              {
313                \token_to_str:N =
314              }
315            #1
316        }
317    }
```

The following macro defines the behaviour of the active ^ character. ^ before any vowel puts a breve above the vowel. The character V is treated as a vowel here as it may represent the vowel U, but v is not, as it is never used for a vowel. In other cases the active ^ character expands to itself and the token read as argument is reinserted.

```
318    \cs_set:Npn \babellatin_put_breve:N #1
319    {
320        \str_case:nnF {#1}
321          {
322            {A} { \u{A} }
323            {a} { \u{a} }
324            {E} { \u{E} }
325            {e} { \u{e} }
326            {I} { \u{I} }
327            {i} { \u{\i} }
328            {O} { \u{O} }
329            {o} { \u{o} }
330            {U} { \u{U} }
331            {u} { \u{u} }
332            {V} { \u{V} }
333            {Y} { \u{Y} }
```

```
334        {y} { \u{y} }
335      }
336      {
337        \token_to_str:N ^
338        #1
339      }
340    }
```

We define a macro for an additional hyphenation point that does not suppress other hyphenation points within the word. This macro is used by the " and the "| shorthand.

```
341  \cs_set:Npn \babellatin_allowhyphens:
342    {
343      \bbl@allowhyphens
344      \discretionary {-} {} {}
345      \bbl@allowhyphens
346    }
```

The conditional \ifbabellatin@useligatures cannot be used within a expl3 context. So we have to define a macro testing if ligatures are enabled outside the expl3 code part. The result is stored in the variable \babellatin@useligatures@bool. We define this variable analogously to expl3's \c_true_bool and \c_false_bool.

```
347  \ExplSyntaxOff
348  \def\babellatin@test@for@ligatures{%
349    \ifbabellatin@useligatures
350      \chardef\babellatin@useligatures@bool=1
351    \else
352      \chardef\babellatin@useligatures@bool=0
353    \fi
354  }%
355  \ExplSyntaxOn
```

The following macro is intended for defining a shorthand for a ligature where useful. The first argument is the expected second character after " (e. g. e if "a has been read). The second argument is the true code, that applies if this character is found (the ligature command). The third argument is the false code (some other command).

```
356  \cs_set_protected:Npn \babellatin_ligature_shorthand:Nnn #1#2#3
357    {
358      \babellatin@test@for@ligatures
359      \bool_if:NTF \babellatin@useligatures@bool
360        {
361          \peek_meaning_remove:NTF #1 {#2} {#3}
362        }
363        {
364          #3
365        }
366    }
```

The following macro is intended for defining a shorthand for a diphthong with a combining double macron (U+035E). The first argument is the first character of the diphthong, which has already been read. The second argument is the second character of the diphthong, which is expected to be read. The third argument is the false code, that applies if the second character is not found as expected.

For pdfLaTeX a warning is issued if the diphthong is found as this engine does not support the combining double macron.

```
367  \cs_set_protected:Npn \babellatin_diphthong_macron:NNn #1#2#3
368    {
369      \peek_meaning:NTF #2
370        {
371          #1
372          \bool_lazy_or:nnTF { \sys_if_engine_xetex_p: } { \sys_if_engine_luatex_p: }
373            {
374              \iffontchar \font "35E \relax
375                \char "35E \relax
376              \else
377                \msg_warning:nn {babel-latin} {no-double-macron-font}
378              \fi
379            }
380            {
381              \msg_warning:nn {babel-latin} {no-double-macron-engine}
382            }
383        }
384        {
385          #3
386        }
387    }
388  \msg_set:nnn {babel-latin} {no-double-macron-font}
389    {
390      The~combining~double~macron~(U+035E)~is~not~available~in~the~current~
391      font.~The~diphthong~is~typeset~without~macron~ \msg_line_context: .
392    }
393  \msg_set:nnn {babel-latin} {no-double-macron-engine}
394    {
395      The~combining~double~macron~(U+035E)~is~not~available~with~
396      \c_sys_engine_str . ~ The~diphthong~is~typeset~without~macron~
397      \msg_line_context: .
398    }
```

The following macro is intended for defining a shorthand for a ligature with a macron where useful. The first argument is the first character of the diphthong, which has already been read. The second argument is the expected second character of the diphthong. The third argument is the code for the ligature with the macron. The fourth argument is the false code that applies if the second character is not found.

```
399  \cs_set_protected:Npn \babellatin_ligature_macron:NNnn #1#2#3#4
400    {
401      \babellatin_ligature_shorthand:Nnn #2 {#3}
402        {
403          \babellatin_diphthong_macron:NNn #1 #2 {#4}
404        }
405    }
```

The following conditional tests if the argument is a ligature command (\AE, \ae, \OE, or \oe).

```
406  \prg_set_conditional:Npnn \babellatin_if_ligature_command:N #1 {TF}
407    {
408      \token_if_eq_meaning:NNTF #1 \AE { \prg_return_true: }
409        {
410          \token_if_eq_meaning:NNTF #1 \ae { \prg_return_true: }
411            {
412              \token_if_eq_meaning:NNTF #1 \OE { \prg_return_true: }
```

```
413                  {
414                    \token_if_eq_meaning:NNTF #1 \oe { \prg_return_true: }
415                      {
416                        \prg_return_false:
417                      }
418                  }
419              }
420          }
421      }
422  \ExplSyntaxOff
```

For the "< and the "> shorthands we have to define the meaning of the macros used for their definition. The commands \guillemetleft and \guillemetright are provided by babel. We will have to change this definition later on for ecclesiasticallatin if pdfTEX is used.

```
423  \let\babellatin@guillemetleft\guillemetleft
424  \let\babellatin@guillemetright\guillemetright
```

Finally, we have to add the shorthand definitions to the extras of the current language.

```
425  \expandafter\addto\csname extras\CurrentOption\endcsname{%
426    \bbl@activate{"}%
427    \languageshorthands{latin}%
428  }%
429  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
430    \bbl@deactivate{"}%
431  }%
432  \@ifpackagewith{babel}{activeacute}{%
433    \babellatin@declare@apostrophe@shorthands
434    \expandafter\addto\csname extras\CurrentOption\endcsname{%
435      \bbl@activate{'}%
436    }%
437    \expandafter\addto\csname noextras\CurrentOption\endcsname{%
438      \bbl@deactivate{'}%
439    }%
440  }{}%
441 \fi
```

7.6   Ecclesiastical punctuation spacing

We define some conditionals concerning the engine used.

```
442 \newif\ifbabellatin@luatex
443 \newif\ifbabellatin@xetex
444 \ifnum\bbl@engine=1
445   \babellatin@luatextrue
446 \else
447   \ifnum\bbl@engine=2
448     \babellatin@xetextrue
449   \fi
450 \fi
```

The following command defines the preparations needed for punctuation spacing in the preamble.

```
451 \def\babellatin@prepare@punctuation@spacing{%
```

For LuaTeX we load an additional file containing some Lua code. This file is documented in section 7.9.

```
452  \ifbabellatin@luatex
453    \directlua{ecclesiasticallatin=require('ecclesiasticallatin')}%
454  \else
```

The following command inserts a kern of 1/12 of a quad. This is the only amount of space used for punctuation within this package.

```
455    \def\babellatin@insert@punctuation@space{%
456      \kern0.08333\fontdimen6\font
457    }%
```

The following command inserts the same kern, removing any positive amount of space that precedes. This is needed if a closing guillemet is preceded by a space character erroneously input by the user.

```
458    \def\babellatin@replace@preceding@space{%
459      \ifdim\lastskip>\z@\unskip\fi
460      \babellatin@insert@punctuation@space
461    }%
```

The following command inserts the same kern, removing any following space character. This is needed if an opening guillemet is followed by a space character erroneously input by the user.

```
462    \def\babellatin@replace@following@space{%
463      \babellatin@insert@punctuation@space
464      \ignorespaces
465    }%
```

For XƎTEX the punctuation spacing will be defined based on five different character classes: one for question and exclamation marks, one for colons and semicolons, one for opening and closing guillemets, respectiveley, and one for opening brackets. Concerning spacing, brackets are treated the same way as letter characters in most cases. However, in strings like "(?)" no spacing is desired before the question mark. So we need a dedicated character class for opening brackets.

```
466  \ifbabellatin@xetex
467    \newXeTeXintercharclass\babellatin@qmark@class
468    \newXeTeXintercharclass\babellatin@colon@class
469    \newXeTeXintercharclass\babellatin@oguill@class
470    \newXeTeXintercharclass\babellatin@cguill@class
471    \newXeTeXintercharclass\babellatin@obracket@class
```

Furthermore, we need a class representing the word boundary. This class has a fixed number defined in `latex.ltx`.

```
472      \let\babellatin@boundary@class\e@alloc@intercharclass@top
```

A space is inserted between a question or exclamation mark and a closing guillemet.

```
473      \XeTeXinterchartoks\babellatin@qmark@class\babellatin@cguill@class={%
474        \babellatin@insert@punctuation@space}%
```

A space is inserted between a question or exclamation mark and a colon or semicolon.

```
475      \XeTeXinterchartoks\babellatin@qmark@class\babellatin@colon@class={%
476        \babellatin@insert@punctuation@space}%
```

A space is inserted between a colon or semicolon and a closing guillemet.

```
477      \XeTeXinterchartoks\babellatin@colon@class\babellatin@cguill@class={%
478        \babellatin@insert@punctuation@space}%
```

A space character after an opening guillemet is replaced by the correct amount of space.

```
479        \XeTeXinterchartoks\babellatin@oguill@class\babellatin@boundary@class={%
480          \babellatin@replace@following@space}%
```

A space is inserted between two opening guillemets.

```
481        \XeTeXinterchartoks\babellatin@oguill@class\babellatin@oguill@class={%
482          \babellatin@insert@punctuation@space}%
```

A space is inserted between an opening guillemet and any ordinary character.

```
483        \XeTeXinterchartoks\babellatin@oguill@class\z@={%
484          \babellatin@insert@punctuation@space}%
```

A space is inserted between two closing guillemets.

```
485        \XeTeXinterchartoks\babellatin@cguill@class\babellatin@cguill@class={%
486          \babellatin@insert@punctuation@space}%
```

A space is inserted between a closing guillemet and a question or exclamation mark.

```
487        \XeTeXinterchartoks\babellatin@cguill@class\babellatin@qmark@class={%
488          \babellatin@insert@punctuation@space}%
```

A space is inserted between a closing guillemet and a colon or semicolon.

```
489        \XeTeXinterchartoks\babellatin@cguill@class\babellatin@colon@class={%
490          \babellatin@insert@punctuation@space}%
```

A space character before a question or exclamation mark is replaced by the correct amount of space.

```
491        \XeTeXinterchartoks\babellatin@boundary@class\babellatin@qmark@class={%
492          \babellatin@replace@preceding@space}%
```

A space character before a colon or semicolon is replaced by the correct amount of space.

```
493        \XeTeXinterchartoks\babellatin@boundary@class\babellatin@colon@class={%
494          \babellatin@replace@preceding@space}%
```

A space character before a closing guillemet is replaced by the correct amount of space.

```
495        \XeTeXinterchartoks\babellatin@boundary@class\babellatin@cguill@class={%
496          \babellatin@replace@preceding@space}%
```

A space is inserted between any ordinary character and a question or exclamation mark.

```
497        \XeTeXinterchartoks\z@\babellatin@qmark@class={%
498          \babellatin@insert@punctuation@space}%
```

A space is inserted between any ordinary character and a colon or semicolon.

```
499        \XeTeXinterchartoks\z@\babellatin@colon@class={%
500          \babellatin@insert@punctuation@space}%
```

A space is inserted between any ordinary character and a closing guillemet.

```
501        \XeTeXinterchartoks\z@\babellatin@cguill@class={%
502          \babellatin@insert@punctuation@space}%
503      \else
```

In pdfTeX active characters are needed for punctuation spacing.

```
504        \initiate@active@char{;}%
505        \initiate@active@char{:}%
506        \initiate@active@char{!}%
507        \initiate@active@char{?}%
508        \declare@shorthand{latin}{;}{%
509          \ifhmode
510            \babellatin@replace@preceding@space
```

```
511        \string;%
512      \else
513        \string;%
514      \fi
515    }%
516    \declare@shorthand{latin}{:}{%
517      \ifhmode
518        \babellatin@replace@preceding@space
519        \string:%
520      \else
521        \string:%
522      \fi
523    }%
524    \declare@shorthand{latin}{!}{%
525      \ifhmode
526        \babellatin@replace@preceding@space
527        \string!%
528      \else
529        \string!%
530      \fi
531    }%
532    \declare@shorthand{latin}{?}{%
533      \ifhmode
534        \babellatin@replace@preceding@space
535        \string?%
536      \else
537        \string?%
538      \fi
539    }%
540    \fi
541  \fi
542 }%
```

We call the previously defined command for ecclesiastical Latin.

```
543 \ifx\CurrentOption\babellatin@ecclesiastical
544   \babellatin@prepare@punctuation@spacing
545 \else
546   \ifx\CurrentOption\babellatin@ecclesiastic
547     \babellatin@prepare@punctuation@spacing
548   \fi
549 \fi
```

The following function actually enables the spacing of punctuation.

```
550 \def\babellatin@punctuation@spacing{%
```

For LuaTEX we just have to call a function of the Lua module.

```
551   \ifbabellatin@luatex
552     \directlua{ecclesiasticallatin.activate_spacing()}%
553   \else
```

For XƎTEX we have to enable the character classes functionality and assign the punctuation characters to the character classes.

```
554     \ifbabellatin@xetex
555       \XeTeXinterchartokenstate = 1
556       \XeTeXcharclass `\! \babellatin@qmark@class
557       \XeTeXcharclass `\? \babellatin@qmark@class
```

```
558        \XeTeXcharclass `\!! \babellatin@qmark@class
559        \XeTeXcharclass `\?? \babellatin@qmark@class
560        \XeTeXcharclass `\?! \babellatin@qmark@class
561        \XeTeXcharclass `\!? \babellatin@qmark@class
562        \XeTeXcharclass `\‽ \babellatin@qmark@class
563        \XeTeXcharclass `\; \babellatin@colon@class
564        \XeTeXcharclass `\: \babellatin@colon@class
565        \XeTeXcharclass `\« \babellatin@oguill@class
566        \XeTeXcharclass `\» \babellatin@cguill@class
567        \XeTeXcharclass `\‹ \babellatin@oguill@class
568        \XeTeXcharclass `\› \babellatin@cguill@class
569        \XeTeXcharclass `\( \babellatin@obracket@class
570        \XeTeXcharclass `\[ \babellatin@obracket@class
571        \XeTeXcharclass `\{ \babellatin@obracket@class
572        \XeTeXcharclass `\( \babellatin@obracket@class
573      \else
```

For pdfTEX we activate the shorthands.

```
574        \bbl@activate{;}%
575        \bbl@activate{:}%
576        \bbl@activate{!}%
577        \bbl@activate{?}%
```

We also redefine the guillemet commands.

```
578        \def\babellatin@guillemetleft{%
579          \guillemetleft
580          \babellatin@replace@following@space
581        }%
582        \def\babellatin@guillemetright{%
583          \babellatin@replace@preceding@space
584          \guillemetright
585        }%
586      \fi
587    \fi
588 }%
```

The following function disables the spacing of punctuation.

```
589 \def\babellatin@no@punctuation@spacing{%
590   \ifbabellatin@luatex
591     \directlua{ecclesiasticallatin.deactivate_spacing()}%
592   \else
593     \ifbabellatin@xetex
594       \XeTeXcharclass `\! \z@
595       \XeTeXcharclass `\? \z@
596       \XeTeXcharclass `\!! \z@
597       \XeTeXcharclass `\?? \z@
598       \XeTeXcharclass `\?! \z@
599       \XeTeXcharclass `\!? \z@
600       \XeTeXcharclass `\‽ \z@
601       \XeTeXcharclass `\; \z@
602       \XeTeXcharclass `\: \z@
603       \XeTeXcharclass `\« \z@
604       \XeTeXcharclass `\» \z@
605       \XeTeXcharclass `\‹ \z@
606       \XeTeXcharclass `\› \z@
```

```
607        \XeTeXcharclass `\( \z@
608        \XeTeXcharclass `\[ \z@
609        \XeTeXcharclass `\{ \z@
610        \XeTeXcharclass `\( \z@
611        \XeTeXinterchartokenstate = 0
612      \else
613        \bbl@deactivate{;}%
614        \bbl@deactivate{:}%
615        \bbl@deactivate{!}%
616        \bbl@deactivate{?}%
617        \let\babellatin@guillemetleft\guillemetleft
618        \let\babellatin@guillemetright\guillemetright
619      \fi
620    \fi
621 }%
```

Punctuation is spaced in ecclesiastical Latin only.

```
622 \addto\extrasecclesiasticallatin{\babellatin@punctuation@spacing}%
623 \addto\noextrasecclesiasticallatin{\babellatin@no@punctuation@spacing}%
624 \addto\extrasecclesiasticlatin{\babellatin@punctuation@spacing}%
625 \addto\noextrasecclesiasticlatin{\babellatin@no@punctuation@spacing}%
```

7.7 Modifiers

We define some language options accessible via modifiers.

7.7.1 Using the letter *j*

The usej option sets the conditional \ifbabellatin@usej to true.

```
626 \bbl@declare@ttribute\CurrentOption{usej}{%
627   \expandafter\addto\csname extras\CurrentOption\endcsname{%
628     \babellatin@usejtrue}%
629   \expandafter\addto\csname noextras\CurrentOption\endcsname{%
630     \babellatin@usejfalse}%
631 }%
```

7.7.2 Typesetting months in lower case

The lowercasemonth option sets the conditional \ifbabellatin@lowercasemonth to true.

```
632 \bbl@declare@ttribute\CurrentOption{lowercasemonth}{%
633   \expandafter\addto\csname extras\CurrentOption\endcsname{%
634     \babellatin@lowercasemonthtrue}%
635   \expandafter\addto\csname noextras\CurrentOption\endcsname{%
636     \babellatin@lowercasemonthfalse}%
637 }%
```

7.7.3 Shorthands for prosodic marks

The withprosodicmarks option makes it possible to use shorthands like =a or ^a for vowels with macrons and breves. We define it for all four language variants of Latin, but only if the LATEX format is used.

```
638 \ifx\fmtname\babellatin@latex
639   \bbl@declare@ttribute\CurrentOption{withprosodicmarks}{%
```

Every shorthand character needs an `\initiate@active@char` command, which makes the respective character active, but expanding to itself as long as no further definitions occur. Both active characters needs to be switched off at the beginning of the document to avoid problems with commands using key=value interfaces (e. g. `\includegraphics`) and TeX's ^^xx convention.

```
640    \initiate@active@char{=}%
641    \initiate@active@char{^}%
642    \AtBeginDocument{%
```

We do not use `\shorthandoff{=}` and `\shorthandoff*{^}` in the following lines because babel-french redefines the `\shorthandoff` command for XƎLATEX and LuaLATEX. Instead, we use babel's internal definition of this command.

```
643      \bbl@shorthandoff\z@{=}%
644      \bbl@shorthandoff\tw@{^}%
645    }%
646    \babellatin@declare@prosodic@shorthands
647    \expandafter\addto\csname extras\CurrentOption\endcsname{%
648      \bbl@activate{=}%
649      \bbl@activate{^}%
```

The active = and ^ are normally turned off to avoid problems with commands using key=value interfaces and TeX's ^^xx convention. We define the commands `\ProsodicMarksOn` and `\ProsodicMarksOff` for turning them on and off within the document. We use the starred form of `\shorthandoff` when turning off ^ to keep it working within math formulas.

```
650      \def\ProsodicMarksOn{%
651        \shorthandon{=}%
652        \shorthandon{^}%
653      }%
654      \def\ProsodicMarksOff{%
655        \shorthandoff{=}%
656        \shorthandoff*{^}%
657      }%
658    }%
659    \expandafter\addto\csname noextras\CurrentOption\endcsname{%
660      \bbl@deactivate{=}%
661      \bbl@deactivate{^}%
662    }%
663  }%
```

The `\ProsodicMarksOn` and `\ProsodicMarksOff` commands are useless without the withprosodicmarks modifier. They only issue warnings in this case.

```
664  \expandafter\addto\csname extras\CurrentOption\endcsname{%
665    \def\ProsodicMarksOn{%
666      \PackageWarning{babel-latin}{%
667        The \protect\ProsodicMarksOn\space command is only\MessageBreak
668        available using the withprosodicmarks\MessageBreak
669        modifier}%
670    }%
671    \def\ProsodicMarksOff{%
672      \PackageWarning{babel-latin}{%
673        The \protect\ProsodicMarksOff\space command is only\MessageBreak
674        available using the withprosodicmarks\MessageBreak
675        modifier}%
```

```
676       }%
677    }%
678 \fi
```

7.7.4 Ecclesiastical footnotes

The ecclesiasticfootnotes option sets the footnotes globally to the style defined by the (now outdated) ecclesiastic package. The definition takes place at the end of the package to be able to check babel's main language. However, the \CurrentOption has lost its value at this moment, so we have to store it.

```
679 \bbl@declare@ttribute\CurrentOption{ecclesiasticfootnotes}{%
680    \let\babellatin@footnote@lang\CurrentOption
681    \AtEndOfPackage{%
682      \ifx\bbl@main@language\babellatin@footnote@lang
683        \let\@makefntext\babellatin@variant@footnote
684      \else
685        \PackageWarningNoLine{babel-latin}{%
686          \babellatin@footnote@lang\space is not the main language.\MessageBreak
687          The `ecclesiasticfootnotes' modifier\MessageBreak
688          is ineffective}%
689      \fi
690    }%
691 }%
```

This is the footnote style as defined by the ecclesiastic package.

```
692 \def\babellatin@variant@footnote#1{%
693    \parindent 1em%
694    \noindent
695    \hbox{\normalfont\@thefnmark.}%
696    \enspace #1%
697 }%
```

7.8   Legacy modifiers and commands

We keep the modifiers classic, medieval, and ecclesiastic for backwards compatibility. We issue a warning if they are used.

```
698 \def\babellatin@outdated@modifier#1#2{%
699    \PackageWarningNoLine{babel-latin}{%
700      The `#1' modifier is outdated. Please\MessageBreak
701      consult the babel-latin manual and consider\MessageBreak
702      to load the language `#2'\MessageBreak
703      instead of `latin.#1'}%
704 }%
705 \def\babellatin@outdated@language#1#2{%
706    \PackageWarningNoLine{babel-latin}{%
707      The `#1' language is outdated.\MessageBreak
708      Please load the language `#2'\MessageBreak
709      instead}%
710 }%
711 \bbl@declare@ttribute{latin}{classic}{%
712    \babellatin@outdated@modifier{classic}{classicallatin}%
713    \addto\extraslatin{\babellatin@usevfalse}%
714    \addto\noextraslatin{\babellatin@usevtrue}%
```

```
715  \babellatin@test@classical@patterns
716  \let\l@latin\l@classiclatin
717  \DeclareUppercaseMapping[la]{`u}{V}%
718  \DeclareLowercaseMapping[la]{`V}{u}%
719  }%
720  \bbl@declare@ttribute{latin}{medieval}{%
721  \babellatin@outdated@modifier{medieval}{medievallatin}%
722  \addto\extraslatin{%
723      \babellatin@usevfalse
724      \def\prefacename{Pr\ae fatio}%
725  }%
726  \addto\noextraslatin{%
727      \babellatin@usevtrue
728  }%
729  \DeclareUppercaseMapping[la]{`u}{V}%
730  \DeclareLowercaseMapping[la]{`V}{u}%
731  }%
732  \bbl@declare@ttribute{latin}{ecclesiastic}{%
733  \babellatin@outdated@modifier{ecclesiastic}{ecclesiasticallatin}%
734  \babellatin@prepare@punctuation@spacing
735  \babellatin@ecclesiastic@outdated@commands
```

The apostrophe character becomes active, even without babel's activeacute option.

```
736  \initiate@active@char{'}%
737  \babellatin@declare@apostrophe@shorthands
738  \addto\extraslatin{%
739      \bbl@activate{'}%
740      \babellatin@punctuation@spacing
741      \babellatin@useligaturestrue
742  }%
743  \addto\noextraslatin{%
744      \bbl@deactivate{'}%
745      \babellatin@no@punctuation@spacing
746      \babellatin@useligaturesfalse
747  }%
```

We set up the footnotes like the ecclesiastic package did.

```
748  \addto\extraslatin{%
749      \babel@save\@makefntext
750      \let\@makefntext\babellatin@variant@footnote
751  }%
752  }%
```

In earlier versions of babel-latin (up to v. 3.5) a \SetLatinLigatures command and a \ProsodicMarks command have been defined. We retain them for backwards compatiblity, but they do nothing except issuing a warning.

```
753  \providecommand\SetLatinLigatures{%
754  \PackageWarning{babel-latin}{%
755      The \protect\SetLatinLigatures\space command is obsolete.\MessageBreak
756      Please remove it}}%
757  \providecommand\ProsodicMarks{%
758  \PackageWarning{babel-latin}{%
759      The \protect\ProsodicMarks\space command is obsolete.\MessageBreak
760      Please remove it}}%
```

We retain some legacy commands concerning guillemets from the ecclesiastic package, which is now outdated, but we deprecate them.

```
761 \def\babellatin@ecclesiastic@outdated@commands{%
762   \providecommand*\FrenchGuillemetsFrom[4]{%
763     \PackageWarning{babel-latin}{%
764       The \protect\FrenchGuillemetsFrom\space command is obsolete.\MessageBreak
765       Please remove it and use \protect\usepackage[T1]{fontenc}\MessageBreak
766       if compiling with pdfLaTeX}}%
767   \let\FrenchGuillemotsFrom\FrenchGuillemetsFrom
768   \providecommand\ToneGuillemets{%
769     \PackageWarning{babel-latin}{%
770       The \protect\ToneGuillemets\space command is obsolete.\MessageBreak
771       Please remove it and use \protect\usepackage[T1]{fontenc}\MessageBreak
772       if compiling with pdfLaTeX}}%
773   \expandafter\addto\csname extras\CurrentOption\endcsname{%
774     \babel@save\og
775     \babel@save\fg
776     \DeclareRobustCommand\og{%
777       \babellatin@guillemetleft
778       \PackageWarning{babel-latin}{%
779         The \protect\og\space command is obsolete.\MessageBreak
780         Please replace it by "<}}%
781     \DeclareRobustCommand\fg{%
782       \babellatin@guillemetright
783       \PackageWarning{babel-latin}{%
784         The \protect\fg\space command is obsolete.\MessageBreak
785         Please replace it by ">}}%
786   }%
787 }%
788 \ifx\CurrentOption\babellatin@ecclesiastic
789   \babellatin@ecclesiastic@outdated@commands
790 \fi
```

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.

```
791 \ldf@finish\CurrentOption
```

babel expects `ldf` files for `classicallatin`, `medievallatin` and `ecclesiasticallatin`. These files themselves only load `latin.ldf`, which does the real work:

```
792 ⟨classic⟩\ProvidesLanguage{classiclatin}
793 ⟨classical⟩\ProvidesLanguage{classicallatin}
794 ⟨ecclesiastic⟩\ProvidesLanguage{ecclesiasticlatin}
795 ⟨ecclesiastical⟩\ProvidesLanguage{ecclesiasticallatin}
796 ⟨medieval⟩\ProvidesLanguage{medievallatin}

797 \input latin.ldf\relax
```

We issue a warning if the outdated languages `classiclatin` and `ecclesiasticlatin` are called:

```
798 ⟨classic⟩\babellatin@outdated@language{classiclatin}{classicallatin}%
799 ⟨ecclesiastic⟩\babellatin@outdated@language{ecclesiasticlatin}{ecclesiasticallatin}%
```

29

7.9 The Lua module

In case LuaTEX is used for compilation, the spacing of punctuation for ecclesiastical Latin requires some Lua code, which is stored in `ecclesiasticallatin.lua`. The original version of this code has been written for the polyglossia package by É. Roux and others.

The Lua module identifies itself using the command provided by ltluatex.

```lua
800 luatexbase.provides_module({
801     name        = "ecclesiasticallatin",
802     date        = "2025-04-29",
803     version     = "4.1",
804     description = "babel-latin punctuation spacing for ecclesiastical Latin"
805 })
806 local add_to_callback     = luatexbase.add_to_callback
807 local in_callback         = luatexbase.in_callback
808 local new_attribute       = luatexbase.new_attribute
809 local node                = node
810 local insert_node_before  = node.insert_before
811 local insert_node_after   = node.insert_after
812 local remove_node         = node.remove
813 local has_attribute       = node.has_attribute
814 local node_copy           = node.copy
815 local new_node            = node.new
816 local end_of_math         = node.end_of_math
817 local get_next            = node.getnext
818 local get_prev            = node.getprev
819 local get_property        = node.getproperty
```

Node types according to `node.types()`:

```lua
820 local glue_code    = node.id"glue"
821 local glyph_code   = node.id"glyph"
822 local penalty_code = node.id"penalty"
823 local kern_code    = node.id"kern"
824 local math_code    = node.id"math"
```

We need some node subtypes:

```lua
825 local userkern = 1
826 local removable_skip = {
827     [0]  = true, -- userskip
828     [13] = true, -- spaceskip
829     [14] = true  -- xspaceskip
830 }
```

We make a new node, so that we can copy it later on:

```lua
831 local kern_node   = new_node(kern_code)
832 kern_node.subtype = userkern
833 local function get_kern_node(dim)
834     local n = node_copy(kern_node)
835     n.kern = dim
836     return n
837 end
```

All possible space characters according to section 6.2 of the Unicode Standard (`https://www.unicode.org/versions/Unicode12.0.0/ch06.pdf`):

```lua
838 local space_chars = {
839     [0x20] = true,   -- space
```

```
840    [0xA0] = true,   -- no-break space
841    [0x1680] = true, -- ogham space mark
842    [0x2000] = true, -- en quad
843    [0x2001] = true, -- em quad
844    [0x2002] = true, -- en space
845    [0x2003] = true, -- em space
846    [0x2004] = true, -- three-per-em-space
847    [0x2005] = true, -- four-per-em space
848    [0x2006] = true, -- six-per-em space
849    [0x2007] = true, -- figure space
850    [0x2008] = true, -- punctuation space
851    [0x2009] = true, -- thin space
852    [0x200A] = true, -- hair space
853    [0x202F] = true, -- narrow no-break space
854    [0x205F] = true, -- medium mathematical space
855    [0x3000] = true  -- ideographic space
856 }
```

All left bracket characters, referenced by their Unicode slot:

```
857 local left_bracket_chars = {
858    [0x28] = true,  -- left parenthesis
859    [0x5B] = true,  -- left square bracket
860    [0x7B] = true,  -- left curly bracket
861    [0x27E8] = true -- mathematical left angle bracket
862 }
```

All right bracket characters, referenced by their Unicode slot:

```
863 local right_bracket_chars = {
864    [0x29] = true,  -- right parenthesis
865    [0x5D] = true,  -- right square bracket
866    [0x7D] = true,  -- right curly bracket
867    [0x27E9] = true -- mathematical right angle bracket
868 }
```

Question and exclamation marks, referenced by their Unicode slot:

```
869 local question_exclamation_chars = {
870    [0x21] = true,   -- exclamation mark !
871    [0x3F] = true,   -- question mark ?
872    [0x203C] = true, -- double exclamation mark ‼
873    [0x203D] = true, -- interrobang ‽
874    [0x2047] = true, -- double question mark ⁇
875    [0x2048] = true, -- question exclamation mark ⁈
876    [0x2049] = true  -- exclamation question mark ⁉
877 }
```

Test for a horizontal space node to be removed:

```
878 local function somespace(n)
879    if n then
880       local id, subtype = n.id, n.subtype
881       if id == glue_code then
```

It is dangerous to remove all type of glue.

```
882          return removable_skip[subtype]
883       elseif id == kern_code then
```

We only remove user's kern.

```
884          return subtype == userkern
885       elseif id == glyph_code then
886          return space_chars[n.char]
887       end
888    end
889 end
```

Test for a left bracket:

```
890 local function someleftbracket(n)
891    if n then
892       local id = n.id
893       if id == glyph_code then
894          return left_bracket_chars[n.char]
895       end
896    end
897 end
```

Test for a right bracket:

```
898 local function somerightbracket(n)
899    if n then
900       local id = n.id
901       if id == glyph_code then
902          return right_bracket_chars[n.char]
903       end
904    end
905 end
```

Test for two question or exclamation marks:

```
906 local function question_exclamation_sequence(n1, n2)
907    if n1 and n2 then
908       local id1 = n1.id
909       local id2 = n2.id
910       if id1 == glyph_code and id2 == glyph_code then
911          return question_exclamation_chars[n1.char] and question_exclamation_chars[n2.char]
912       end
913    end
914 end
```

Test for a penalty node:

```
915 local function somepenalty(n, value)
916    if n then
917       local id = n.id
918       if id == penalty_code then
919          if value then
920             return n.penalty == value
921          else
922             return true
923          end
924       end
925    end
926 end
```

LuaTeX attribute determining whether to space punctuation or not:

```
927 local punct_attr = new_attribute("ecclesiasticallatin_punct")
```

Tables containing the left and right space amount (in units of a quad) of every character:

```
928 local left_space  = {}
929 local right_space = {}
```

Insertion of the necessary spaces to the node list:

```
930 local function process(head)
931     local current = head
932     while current do
933         local id = current.id
934         if id == glyph_code then
935             if has_attribute(current, punct_attr) then
```

We try to obtain the character of the current node from its property table, which is the most reliable way as the same character may be rendered by different glyphs with different code numbers.

```
936                 local char = get_property(current) and get_property(current).glyph_info
```

If the glyph_info property is not available, we use the node's char field to obtain the character, which is however only possible for numbers up to $10FFFF_{16}$.

```
937                 if not char and current.char <= 0x10FFFF then
938                     char = utf8.char(current.char)
939                 end
940                 local leftspace, rightspace
941                 if char then
942                     leftspace  = left_space[char]
943                     rightspace = right_space[char]
944                 end
945                 if leftspace or rightspace then
946                     local fontparameters = fonts.hashes.parameters[current.font]
947                     local spacing_node
948                     if leftspace and fontparameters then
949                         local prev = get_prev(current)
950                         local space_exception = false
951                         if prev then
```

We do not add space after left (opening) brackets and between question/exclamation marks:

```
952                             space_exception = someleftbracket(prev)
953                                 or question_exclamation_sequence(prev, current)
954                             while somespace(prev) do
955                                 head = remove_node(head, prev)
956                                 prev = get_prev(current)
957                             end
958                             if somepenalty(prev, 10000) then
959                                 head = remove_node(head, prev)
960                             end
961                         end
962                         spacing_node = get_kern_node(leftspace * fontparameters.quad)
963                         if not space_exception then
964                             head = insert_node_before(head, current, spacing_node)
965                         end
966                     end
967                     if rightspace and fontparameters then
968                         local next = get_next(current)
969                         local space_exception = false
970                         if next then
```

We do not add space before right (closing) brackets:

```
971                            space_exception = somerightbracket(next)
972                            local nextnext = get_next(next)
973                            if somepenalty(next, 10000) and somespace(nextnext) then
974                                head, next = remove_node(head, next)
975                            end
976                            while somespace(next) do
977                                head, next = remove_node(head, next)
978                            end
979                        end
980                        spacing_node = get_kern_node(rightspace * fontparameters.quad)
981                        if not space_exception then
982                            head, current = insert_node_after(head, current, spacing_node)
983                        end
984                    end
985                end
986            end
987        elseif id == math_code then
988            current = end_of_math(current)
989        end
```

The following line does not cause an error even if `current` is `nil`.

```
990        current = get_next(current)
991    end
992    return head
993 end
```

Now we define the actual amount of space for the relevant punctuation characters. For ecclesiastical Latin (and sometimes for Italian) a very small space is used for the punctuation. The ecclesiastic package, a predecessor of the current babel-latin, used a space of 0.3\fontdimen2, where \fontdimen2 is an interword space, which is typically between 1/4 and 1/3 of a quad. We choose a half of a \thinspace here, i. e., 1/12 of a quad.

```
994 local hairspace = 0.08333 -- 1/12
995 local function space_left(char)
996     left_space[char] = hairspace
997 end
998 local function space_right(char, kern)
999     right_space[char] = hairspace
1000 end
1001 space_left('!')
1002 space_left('?')
1003 space_left('‼')
1004 space_left('⁇')
1005 space_left('⁉')
1006 space_left('⁈')
1007 space_left('‽') -- U+203D (interrobang)
1008 space_left(':')
1009 space_left(';')
1010 space_left('»')
1011 space_left('›')
1012 space_right('«')
1013 space_right('‹')
```

The following functions activate and deactivate the punctuation spacing.

```
1014 local function activate()
```

```
1015    tex.setattribute(punct_attr, 1)
1016    for _, callback_name in ipairs{ "pre_linebreak_filter", "hpack_filter" } do
1017        if not in_callback(callback_name, "ecclesiasticallatin-punct.process") then
1018            add_to_callback(callback_name, process, "ecclesiasticallatin-punct.process")
1019        end
1020    end
1021 end
1022 local function deactivate()
```

Though it would make compilation slightly faster, it is not possible to safely remove the process from the callback here. Imagine the following case: you start a paragraph by some spaced punctuation text, then, in the same paragraph, you change the language to something else, and thus call this function. This means that, at the end of the paragraph, the function won't be in the callback, so the beginning of the paragraph won't be processed by it. So we just unset the attribute.

```
1023    tex.setattribute(punct_attr, -0x7FFFFFFF) -- this value means "unset"
1024 end
```

For external access to the activation and deactivation of the punctuation spacing, we define a table ecclesiasticallatin containing two functions. We return this table.

```
1025 ecclesiasticallatin = ecclesiasticallatin or {}
1026 ecclesiasticallatin.activate_spacing   = activate
1027 ecclesiasticallatin.deactivate_spacing = deactivate
1028 return ecclesiasticallatin
```

Change History