Babel support for the Latin language

Claudio Beccari Keno Wehr*

v. 4.2 August 11, 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See section 2.5 on how to update from outdated modifiers and the ecclesiastic package.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 ǣ (ae ligature with macron, for `medievallatin` and `ecclesiasticallatin`), respectively; also available for a͞u, e͞u, and o͞e/œ̄. Note that macrons above diphthongs are only displayed if your font supports the Unicode character `U+035E` (*combining double macron*), which always requires XeLaTeX or LuaLaTeX.[1]

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

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

`^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, Ý, 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.

5.5 ucharclasses

The XeTeX package ucharclasses can be used with babel-latin, but the punctuation spacing for ecclesiastical Latin does not work with this package.

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.

```
\ProvidesLanguage{latin}[2025-08-11 v4.2 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`.

```
\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).

```
\def\babellatin@classical{classicallatin}
\def\babellatin@classic{classiclatin}
\def\babellatin@medieval{medievallatin}
\def\babellatin@ecclesiastical{ecclesiasticallatin}
\def\babellatin@ecclesiastic{ecclesiasticlatin}
```

7.1 Hyphenation patterns

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

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

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

```
\def\babellatin@test@modern@patterns{%
  \ifx\l@latin\undefined
    \@nopatterns{latin}%
    \adddialect\l@latin0
  \fi}%
\def\babellatin@test@classical@patterns{%
  \ifx\l@classiclatin\undefined
    \PackageWarningNoLine{babel-latin}{%
      No hyphenation patterns were found for the\MessageBreak
      classicallatin language. Now I will use the\MessageBreak
      patterns for modern Latin instead}%
    \babellatin@test@modern@patterns
    \adddialect\l@classiclatin\l@latin
  \fi}%
```

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

```
\ifx\CurrentOption\babellatin@classical
  \babellatin@test@classical@patterns
```

```
  \adddialect\l@classicallatin\l@classiclatin
\else
  \ifx\CurrentOption\babellatin@classic
    \babellatin@test@classical@patterns
  \else
    \ifx\CurrentOption\babellatin@ecclesiastical
      \babellatin@test@modern@patterns
      \adddialect\l@ecclesiasticallatin\l@latin
    \else
      \ifx\CurrentOption\babellatin@ecclesiastic
        \babellatin@test@modern@patterns
        \adddialect\l@ecclesiasticlatin\l@latin
      \else
        \ifx\CurrentOption\babellatin@medieval
          \babellatin@test@modern@patterns
          \adddialect\l@medievallatin\l@latin
        \else
          \babellatin@test@modern@patterns
        \fi
      \fi
    \fi
  \fi
\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.

```
\newif\ifbabellatin@useligatures
\addto\extrasmedievallatin{\babellatin@useligaturestrue}%
\addto\noextrasmedievallatin{\babellatin@useligaturesfalse}%
\addto\extrasecclesiasticallatin{\babellatin@useligaturestrue}%
\addto\noextrasecclesiasticallatin{\babellatin@useligaturesfalse}%
\addto\extrasecclesiasticlatin{\babellatin@useligaturestrue}%
\addto\noextrasecclesiasticlatin{\babellatin@useligaturesfalse}%
```

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

```
\StartBabelCommands*{\CurrentOption}{captions}
  \SetString\prefacename{\ifbabellatin@useligatures Pr\ae fatio\else Praefatio\fi}
  \SetString\refname{Conspectus librorum}
  \SetString\abstractname{Summarium}
  \SetString\bibname{Conspectus librorum}
  \SetString\chaptername{Caput}
  \SetString\appendixname{Additamentum}
  \SetString\contentsname{Index}
  \SetString\listfigurename{Conspectus descriptionum}
  \SetString\listtablename{Conspectus tabularum}
  \SetString\indexname{Index rerum notabilium}
  \SetString\figurename{Descriptio}
  \SetString\tablename{Tabula}
  \SetString\partname{Pars}
  \SetString\enclname{Adduntur}%   Or "Additur"? Or simply Add.?
```

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

```
  \SetString\ccname{Exemplar}%     Use the recipient's dative
  \SetString\headtoname{\ignorespaces}% Use the recipient's dative
  \SetString\pagename{Charta}
  \SetString\seename{cfr.}
  \SetString\alsoname{cfr.}% Tabacco never saw "cfr" + "atque" or similar forms
  \SetString\proofname{Demonstratio}
  \SetString\glossaryname{Glossarium}
\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".

```
\def\babellatin@latex{LaTeX2e}%
\ifx\fmtname\babellatin@latex
  \DeclareUppercaseMapping[la-x-classic]{`u}{V}
  \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. It's not enough to test for XeLaTeX or LuaLaTeX, because LuaLaTeX might be used with the luainputenc package. We test for Unicode input the same way the packages hyph-utf8 and dehyph-exptl do in their hyphenation pattern loader files.

```
  \def\babellatin@testengine#1#2!{\def\babellatin@secondarg{#2}}%
  \babellatin@testengine χ!\relax % that's chi, a 2-byte UTF-8 sequence
```

```
  \ifx\babellatin@secondarg\empty
    \DeclareUppercaseMapping[la-x-classic]{`ú}{\a'{V}}
    \DeclareUppercaseMapping[la-x-classic]{`ū}{\a={V}}
    \DeclareUppercaseMapping[la-x-classic]{`ŭ}{\u{V}}
    \DeclareUppercaseMapping[la-x-medieval]{`ú}{\a'{V}}
    \DeclareUppercaseMapping[la-x-medieval]{`ū}{\a={V}}
    \DeclareUppercaseMapping[la-x-medieval]{`ŭ}{\u{V}}
  \fi
\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”.

```
\StartBabelCommands*{classicallatin,classiclatin,medievallatin}{}
  \SetHyphenMap{\BabelLower{`V}{`u}}
\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.

```
\newif\ifbabellatin@usev
\newif\ifbabellatin@usej
\newif\ifbabellatin@lowercasemonth
\babellatin@usevtrue
\addto\extrasclassicallatin{\babellatin@usevfalse}%
\addto\noextrasclassicallatin{\babellatin@usevtrue}%
\addto\extrasclassiclatin{\babellatin@usevfalse}%
\addto\noextrasclassiclatin{\babellatin@usevtrue}%
\addto\extrasmedievallatin{\babellatin@usevfalse}%
\addto\noextrasmedievallatin{\babellatin@usevtrue}%
```

The Latin month names are needed in the genitive case.

```
\def\babellatin@monthname{%
  \ifcase\month\or\ifbabellatin@usej Januarii\else Ianuarii\fi
    \or Februarii%
    \or Martii%
    \or Aprilis%
    \or\ifbabellatin@usej Maji\else Maii\fi
    \or\ifbabellatin@usej Junii\else Iunii\fi
    \or\ifbabellatin@usej Julii\else Iulii\fi
    \or Augusti%
    \or Septembris%
    \or Octobris%
    \or\ifbabellatin@usev Novembris\else Nouembris\fi
    \or Decembris%
  \fi}%
```

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

```
\expandafter\def\csname date\CurrentOption\endcsname{%
```

```
\def\today{%
  \uppercase\expandafter{\romannumeral\day}~%
  \ifbabellatin@lowercasemonth
    \lowercase\expandafter{\babellatin@monthname}%
  \else
    \babellatin@monthname
  \fi
  \space
  \uppercase\expandafter{\romannumeral\year}%
}%
}%
```

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.

```
\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.

```
\initiate@active@char{"}%
\@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.

```
\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.

```
\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.

```
\declare@shorthand {latin} {"}
  {
    \mode_if_math:TF { \token_to_str:N " }
      {
        \texorpdfstring { \babellatin_apply_quotemark:N } { }
      }
  }
```

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.

```
\cs_set_protected:Npn \babellatin@declare@apostrophe@shorthands
  {
    \declare@shorthand {latin} {'}
      {
```

```
          \mode_if_math:TF { \active@math@prime }
            {
              \texorpdfstring { \babellatin_put_acute:N } { \' }
            }
        }
    }
```

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.

```
\cs_set_protected:Npn \babellatin@declare@prosodic@shorthands
  {
    \declare@shorthand {latin} {=}
      {
        \mode_if_math:TF { \token_to_str:N = }
          {
            \texorpdfstring { \babellatin_put_macron:N } { \= }
          }
      }
    \declare@shorthand {latin} {^}
      {
        \mode_if_math:TF { \token_to_str:N ^ } { \babellatin_put_breve:N }
      }
  }
```

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.

```
\cs_set_protected:Npn \babellatin_apply_quotemark:N #1
  {
    \str_case:nnF {#1}
      {
        {A} { \babellatin_ligature_shorthand:Nnn E { \AE }
                {
                  \babellatin_ligature_shorthand:Nnn e { \AE }
                    {
                      \babellatin_allowhyphens: A
                    }
                }
            }
        {a} { \babellatin_ligature_shorthand:Nnn e { \ae }
                {
                  \babellatin_allowhyphens: a
                }
            }
```

```
          {O} { \babellatin_ligature_shorthand:Nnn E { \OE }
                  {
                    \babellatin_ligature_shorthand:Nnn e { \OE }
                      {
                        \babellatin_allowhyphens: O
                      }
                  }
              }
          {o} { \babellatin_ligature_shorthand:Nnn e { \oe }
                  {
                    \babellatin_allowhyphens: o
                  }
              }
          {|} { \babellatin_allowhyphens: }
          {<} { \babellatin@guillemetleft }
          {>} { \babellatin@guillemetright }
        }
        {
          \tl_if_single_token:nTF {#1}
            {
              \token_if_letter:NTF #1 { \babellatin_allowhyphens: }
                {
                  \babellatin_if_ligature_command:NTF #1 { \babellatin_allowhyphens: }
                    {
                      \token_to_str:N "
                    }
                }
            }
            {
              \token_to_str:N "
            }
            #1
        }
    }
```

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.

```
  \cs_set_protected:Npn \babellatin_put_acute:N #1
    {
      \str_case:nnF {#1}
        {
          {A} { \babellatin_ligature_shorthand:Nnn E { \a'\AE }
                  {
                    \babellatin_ligature_shorthand:Nnn e { \a'\AE } { Á }
                  }
              }
          {a} { \babellatin_ligature_shorthand:Nnn e { \a'\ae } { á } }
          {E} { É }
          {e} { é }
          {I} { Í }
```

```
        {i} { í }
        {O} { \babellatin_ligature_shorthand:Nnn E { \a'\OE }
                {
                  \babellatin_ligature_shorthand:Nnn e { \a'\OE } { Ó }
                }
            }
        {o} { \babellatin_ligature_shorthand:Nnn e { \a'\oe } { ó } }
        {U} { Ú }
        {u} { ú }
        {V} { \a'V }
        {Y} { \a'Y }
        {y} { \a'y }
        {Æ} { \a'\AE }
        {æ} { \a'\ae }
        {Œ} { \a'\OE }
        {œ} { \a'\oe }
      }
      {
        \tl_if_single_token:nTF {#1}
          {
            \babellatin_if_ligature_command:NTF #1 { \a' }
              {
                \token_to_str:N '
              }
          }
          {
            \token_to_str:N '
          }
        #1
      }
  }
```

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.

```
\cs_set_protected:Npn \babellatin_put_macron:N #1
  {
    \str_case:nnF {#1}
      {
        {A} { \babellatin_ligature_macron:NNnn AE { \a=\AE }
                {
                  \babellatin_ligature_macron:NNnn Ae { \a=\AE }
                    {
                      \babellatin_diphthong_macron:NNn AU
                        {
                          \babellatin_diphthong_macron:NNn Au { \a=A }
                        }
                    }
                }
            }
```

```
          {a} { \babellatin_ligature_macron:NNnn ae { \a=\ae }
                  {
                    \babellatin_diphthong_macron:NNn au { \a=a }
                  }
              }
          {E} { \babellatin_diphthong_macron:NNn EU
                  {
                    \babellatin_diphthong_macron:NNn Eu { \a=E }
                  }
              }
          {e} { \babellatin_diphthong_macron:NNn eu { \a=e } }
          {I} { \a=I }
          {i} { \a=\i }
          {O} { \babellatin_ligature_macron:NNnn OE { \a=\OE }
                  {
                    \babellatin_ligature_macron:NNnn Oe { \a=\OE } { \a=O }
                  }
              }
          {o} { \babellatin_ligature_macron:NNnn oe { \a=\oe } { \a=o } }
          {U} { \a=U }
          {u} { \a=u }
          {V} { \a=V }
          {Y} { \a=Y }
          {y} { \a=y }
        }
        {
          \tl_if_single_token:nTF {#1}
            {
              \babellatin_if_ligature_command:NTF #1 { \a= }
                {
                  \token_to_str:N =
                }
            }
            {
              \token_to_str:N =
            }
          #1
        }
    }
```

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.

```
\cs_set:Npn \babellatin_put_breve:N #1
  {
    \str_case:nnF {#1}
      {
        {A} { \u{A} }
        {a} { \u{a} }
        {E} { \u{E} }
        {e} { \u{e} }
        {I} { \u{I} }
        {i} { \u{\i} }
```

```
          {O} { \u{O} }
          {o} { \u{o} }
          {U} { \u{U} }
          {u} { \u{u} }
          {V} { \u{V} }
          {Y} { \u{Y} }
          {y} { \u{y} }
        }
        {
          \token_to_str:N ^
          #1
        }
    }
```

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.

```
\cs_set:Npn \babellatin_allowhyphens:
  {
    \bbl@allowhyphens
    \discretionary {-} {} {}
    \bbl@allowhyphens
  }
```

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`.

```
\ExplSyntaxOff
\def\babellatin@test@for@ligatures{%
  \ifbabellatin@useligatures
    \chardef\babellatin@useligatures@bool=1
  \else
    \chardef\babellatin@useligatures@bool=0
  \fi
}%
\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).

```
\cs_set_protected:Npn \babellatin_ligature_shorthand:Nnn #1#2#3
  {
    \babellatin@test@for@ligatures
    \bool_if:NTF \babellatin@useligatures@bool
      {
        \peek_meaning_remove:NTF #1 {#2} {#3}
      }
      {
        #3
      }
  }
```

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.

```
\cs_set_protected:Npn \babellatin_diphthong_macron:NNn #1#2#3
  {
    \peek_meaning:NTF #2
      {
        #1
        \bool_lazy_or:nnTF { \sys_if_engine_xetex_p: } { \sys_if_engine_luatex_p: }
          {
            \iffontchar \font "35E \relax
              \char "35E \relax
            \else
              \msg_warning:nn {babel-latin} {no-double-macron-font}
            \fi
          }
          {
            \msg_warning:nn {babel-latin} {no-double-macron-engine}
          }
      }
      {
        #3
      }
  }
\msg_set:nnn {babel-latin} {no-double-macron-font}
  {
    The~combining~double~macron~(U+035E)~is~not~available~in~the~current~
    font.~The~diphthong~is~typeset~without~macron~ \msg_line_context: .
  }
\msg_set:nnn {babel-latin} {no-double-macron-engine}
  {
    The~combining~double~macron~(U+035E)~is~not~available~with~
    \c_sys_engine_str . ~ The~diphthong~is~typeset~without~macron~
    \msg_line_context: .
  }
```

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.

```
\cs_set_protected:Npn \babellatin_ligature_macron:NNnn #1#2#3#4
  {
    \babellatin_ligature_shorthand:Nnn #2 {#3}
      {
        \babellatin_diphthong_macron:NNn #1 #2 {#4}
      }
  }
```

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

```
\prg_set_conditional:Npnn \babellatin_if_ligature_command:N #1 {TF}
```

```
  {
    \token_if_eq_meaning:NNTF #1 \AE { \prg_return_true: }
      {
        \token_if_eq_meaning:NNTF #1 \ae { \prg_return_true: }
          {
            \token_if_eq_meaning:NNTF #1 \OE { \prg_return_true: }
              {
                \token_if_eq_meaning:NNTF #1 \oe { \prg_return_true: }
                  {
                    \prg_return_false:
                  }
              }
          }
      }
  }
\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.

```
\let\babellatin@guillemetleft\guillemetleft
\let\babellatin@guillemetright\guillemetright
```

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

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@activate{"}%
  \languageshorthands{latin}%
}%
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
  \bbl@deactivate{"}%
}%
\@ifpackagewith{babel}{activeacute}{%
  \babellatin@declare@apostrophe@shorthands
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \bbl@activate{'}%
  }%
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
    \bbl@deactivate{'}%
  }%
}{}%
\fi
```

7.6 Ecclesiastical punctuation spacing

We define some conditionals concerning the engine used.

```
\newif\ifbabellatin@luatex
\newif\ifbabellatin@xetex
\ifnum\bbl@engine=1
  \babellatin@luatextrue
\else
  \ifnum\bbl@engine=2
    \babellatin@xetextrue
  \fi
```

```
\fi
```

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

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

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

```
  \ifbabellatin@luatex
    \directlua{ecclesiasticallatin=require('ecclesiasticallatin')}%
  \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.

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

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.

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

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.

```
    \def\babellatin@replace@following@space{%
      \babellatin@insert@punctuation@space
      \ignorespaces
    }%
```

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, respectively, 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.

```
    \ifbabellatin@xetex
      \newXeTeXintercharclass\babellatin@qmark@class
      \newXeTeXintercharclass\babellatin@colon@class
      \newXeTeXintercharclass\babellatin@oguill@class
      \newXeTeXintercharclass\babellatin@cguill@class
      \newXeTeXintercharclass\babellatin@obracket@class
```

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

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

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

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

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

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

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

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

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

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

A space is inserted between two opening guillemets.

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

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

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

A space is inserted between two closing guillemets.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In pdfTEX active characters are needed for punctuation spacing.

```
      \initiate@active@char{;}%
      \initiate@active@char{:}%
      \initiate@active@char{!}%
      \initiate@active@char{?}%
      \declare@shorthand{latin}{;}{%
        \ifhmode
          \babellatin@replace@preceding@space
          \string;%
        \else
          \string;%
        \fi
      }%
      \declare@shorthand{latin}{:}{%
        \ifhmode
          \babellatin@replace@preceding@space
          \string:%
        \else
          \string:%
        \fi
      }%
      \declare@shorthand{latin}{!}{%
        \ifhmode
          \babellatin@replace@preceding@space
          \string!%
        \else
          \string!%
        \fi
      }%
      \declare@shorthand{latin}{?}{%
        \ifhmode
          \babellatin@replace@preceding@space
          \string?%
        \else
          \string?%
        \fi
      }%
    \fi
  \fi
}%
```

We call the previously defined command for ecclesiastical Latin.

```
\ifx\CurrentOption\babellatin@ecclesiastical
  \babellatin@prepare@punctuation@spacing
\else
  \ifx\CurrentOption\babellatin@ecclesiastic
    \babellatin@prepare@punctuation@spacing
  \fi
\fi
```

The following function actually enables the spacing of punctuation.

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

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

```
  \ifbabellatin@luatex
```

```
    \directlua{ecclesiasticallatin.activate_spacing()}%
  \else
```

For XeTeX we have to enable the character classes functionality and assign the punctuation characters to the character classes. The character classes of the punctuation characters are saved because they have to be restored when changing to another language.

```
    \ifbabellatin@xetex
      \babel@savevariable\XeTeXinterchartokenstate
      \babel@savevariable{\XeTeXcharclass`\!}
      \babel@savevariable{\XeTeXcharclass`\?}
      \babel@savevariable{\XeTeXcharclass`\‼}
      \babel@savevariable{\XeTeXcharclass`\⁇}
      \babel@savevariable{\XeTeXcharclass`\⁈}
      \babel@savevariable{\XeTeXcharclass`\⁉}
      \babel@savevariable{\XeTeXcharclass`\❓}
      \babel@savevariable{\XeTeXcharclass`\;}
      \babel@savevariable{\XeTeXcharclass`\:}
      \babel@savevariable{\XeTeXcharclass`\«}
      \babel@savevariable{\XeTeXcharclass`\»}
      \babel@savevariable{\XeTeXcharclass`\‹}
      \babel@savevariable{\XeTeXcharclass`\›}
      \babel@savevariable{\XeTeXcharclass`\(}
      \babel@savevariable{\XeTeXcharclass`\[}
      \babel@savevariable{\XeTeXcharclass`\{}
      \babel@savevariable{\XeTeXcharclass`\⟨}
      \XeTeXinterchartokenstate = 1
      \XeTeXcharclass `\! \babellatin@qmark@class
      \XeTeXcharclass `\? \babellatin@qmark@class
      \XeTeXcharclass `\‼ \babellatin@qmark@class
      \XeTeXcharclass `\⁇ \babellatin@qmark@class
      \XeTeXcharclass `\⁈ \babellatin@qmark@class
      \XeTeXcharclass `\⁉ \babellatin@qmark@class
      \XeTeXcharclass `\❓ \babellatin@qmark@class
      \XeTeXcharclass `\; \babellatin@colon@class
      \XeTeXcharclass `\: \babellatin@colon@class
      \XeTeXcharclass `\« \babellatin@oguill@class
      \XeTeXcharclass `\» \babellatin@cguill@class
      \XeTeXcharclass `\‹ \babellatin@oguill@class
      \XeTeXcharclass `\› \babellatin@cguill@class
      \XeTeXcharclass `\( \babellatin@obracket@class
      \XeTeXcharclass `\[ \babellatin@obracket@class
      \XeTeXcharclass `\{ \babellatin@obracket@class
      \XeTeXcharclass `\⟨ \babellatin@obracket@class
    \else
```

For pdfTeX we activate the shorthands.

```
      \bbl@activate{;}%
      \bbl@activate{:}%
      \bbl@activate{!}%
      \bbl@activate{?}%
```

We also redefine the guillemet commands.

```
      \def\babellatin@guillemetleft{%
        \guillemetleft
```

```
        \babellatin@replace@following@space
      }%
      \def\babellatin@guillemetright{%
        \babellatin@replace@preceding@space
        \guillemetright
      }%
    \fi
  \fi
}%
```

The following function disables the spacing of punctuation.

```
\def\babellatin@no@punctuation@spacing{%
  \ifbabellatin@luatex
    \directlua{ecclesiasticallatin.deactivate_spacing()}%
  \else
    \ifbabellatin@xetex
    \else
      \bbl@deactivate{;}%
      \bbl@deactivate{:}%
      \bbl@deactivate{!}%
      \bbl@deactivate{?}%
      \let\babellatin@guillemetleft\guillemetleft
      \let\babellatin@guillemetright\guillemetright
    \fi
  \fi
}%
```

Punctuation is spaced in ecclesiastical Latin only.

```
\addto\extrasecclesiasticallatin{\babellatin@punctuation@spacing}%
\addto\noextrasecclesiasticallatin{\babellatin@no@punctuation@spacing}%
\addto\extrasecclesiasticlatin{\babellatin@punctuation@spacing}%
\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.

```
\bbl@declare@ttribute\CurrentOption{usej}{%
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babellatin@usejtrue}%
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
    \babellatin@usejfalse}%
}%
```

7.7.2 Typesetting months in lower case

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

```
\bbl@declare@ttribute\CurrentOption{lowercasemonth}{%
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babellatin@lowercasemonthtrue}%
```

```
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
    \babellatin@lowercasemonthfalse}%
}%
```

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.

```
\ifx\fmtname\babellatin@latex
  \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.

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

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

```
      \bbl@shorthandoff\z@{=}%
      \bbl@shorthandoff\tw@{^}%
    }%
    \babellatin@declare@prosodic@shorthands
    \expandafter\addto\csname extras\CurrentOption\endcsname{%
      \bbl@activate{=}%
      \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.

```
      \def\ProsodicMarksOn{%
        \shorthandon{=}%
        \shorthandon{^}%
      }%
      \def\ProsodicMarksOff{%
        \shorthandoff{=}%
        \shorthandoff*{^}%
      }%
    }%
    \expandafter\addto\csname noextras\CurrentOption\endcsname{%
      \bbl@deactivate{=}%
      \bbl@deactivate{^}%
    }%
  }%
```

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

```
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \def\ProsodicMarksOn{%
      \PackageWarning{babel-latin}{%
        The \protect\ProsodicMarksOn\space command is only\MessageBreak
        available using the withprosodicmarks\MessageBreak
        modifier}%
    }%
    \def\ProsodicMarksOff{%
      \PackageWarning{babel-latin}{%
        The \protect\ProsodicMarksOff\space command is only\MessageBreak
        available using the withprosodicmarks\MessageBreak
        modifier}%
    }%
  }%
\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.

```
\bbl@declare@ttribute\CurrentOption{ecclesiasticfootnotes}{%
  \let\babellatin@footnote@lang\CurrentOption
  \AtEndOfPackage{%
    \ifx\bbl@main@language\babellatin@footnote@lang
      \let\@makefntext\babellatin@variant@footnote
    \else
      \PackageWarningNoLine{babel-latin}{%
        \babellatin@footnote@lang\space is not the main language.\MessageBreak
        The `ecclesiasticfootnotes' modifier\MessageBreak
        is ineffective}%
    \fi
  }%
}%
```

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

```
\def\babellatin@variant@footnote#1{%
  \parindent 1em%
  \noindent
  \hbox{\normalfont\@thefnmark.}%
  \enspace #1%
}%
```

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.

```
\def\babellatin@outdated@modifier#1#2{%
  \PackageWarningNoLine{babel-latin}{%
    The `#1' modifier is outdated. Please\MessageBreak
    consult the babel-latin manual and consider\MessageBreak
    to load the language `#2'\MessageBreak
```

```
    instead of `latin.#1'}%
}%
\def\babellatin@outdated@language#1#2{%
  \PackageWarningNoLine{babel-latin}{%
    The `#1' language is outdated.\MessageBreak
    Please load the language `#2'\MessageBreak
    instead}%
}%
\bbl@declare@ttribute{latin}{classic}{%
  \babellatin@outdated@modifier{classic}{classicallatin}%
  \addto\extraslatin{\babellatin@usevfalse}%
  \addto\noextraslatin{\babellatin@usevtrue}%
  \babellatin@test@classical@patterns
  \let\l@latin\l@classiclatin
  \DeclareUppercaseMapping[la]{`u}{V}%
  \DeclareLowercaseMapping[la]{`V}{u}%
}%
\bbl@declare@ttribute{latin}{medieval}{%
  \babellatin@outdated@modifier{medieval}{medievallatin}%
  \addto\extraslatin{%
    \babellatin@usevfalse
    \def\prefacename{Pr\ae fatio}%
  }%
  \addto\noextraslatin{%
    \babellatin@usevtrue
  }%
  \DeclareUppercaseMapping[la]{`u}{V}%
  \DeclareLowercaseMapping[la]{`V}{u}%
}%
\bbl@declare@ttribute{latin}{ecclesiastic}{%
  \babellatin@outdated@modifier{ecclesiastic}{ecclesiasticallatin}%
  \babellatin@prepare@punctuation@spacing
  \babellatin@ecclesiastic@outdated@commands
```

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

```
  \initiate@active@char{'}%
  \babellatin@declare@apostrophe@shorthands
  \addto\extraslatin{%
    \bbl@activate{'}%
    \babellatin@punctuation@spacing
    \babellatin@useligaturestrue
  }%
  \addto\noextraslatin{%
    \bbl@deactivate{'}%
    \babellatin@no@punctuation@spacing
    \babellatin@useligaturesfalse
  }%
```

We set up the footnotes like the ecclesiastic package did.

```
  \addto\extraslatin{%
    \babel@save\@makefntext
    \let\@makefntext\babellatin@variant@footnote
  }%
}%
```

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.

```
\providecommand\SetLatinLigatures{%
  \PackageWarning{babel-latin}{%
    The \protect\SetLatinLigatures\space command is obsolete.\MessageBreak
    Please remove it}}%
\providecommand\ProsodicMarks{%
  \PackageWarning{babel-latin}{%
    The \protect\ProsodicMarks\space command is obsolete.\MessageBreak
    Please remove it}}%
```

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

```
\def\babellatin@ecclesiastic@outdated@commands{%
  \providecommand*\FrenchGuillemetsFrom[4]{%
    \PackageWarning{babel-latin}{%
      The \protect\FrenchGuillemetsFrom\space command is obsolete.\MessageBreak
      Please remove it and use \protect\usepackage[T1]{fontenc}\MessageBreak
      if compiling with pdfLaTeX}}%
  \let\FrenchGuillemotsFrom\FrenchGuillemetsFrom
  \providecommand\ToneGuillemets{%
    \PackageWarning{babel-latin}{%
      The \protect\ToneGuillemets\space command is obsolete.\MessageBreak
      Please remove it and use \protect\usepackage[T1]{fontenc}\MessageBreak
      if compiling with pdfLaTeX}}%
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babel@save\og
    \babel@save\fg
    \DeclareRobustCommand\og{%
      \babellatin@guillemetleft
      \PackageWarning{babel-latin}{%
        The \protect\og\space command is obsolete.\MessageBreak
        Please replace it by "<}}%
    \DeclareRobustCommand\fg{%
      \babellatin@guillemetright
      \PackageWarning{babel-latin}{%
        The \protect\fg\space command is obsolete.\MessageBreak
        Please replace it by ">}}%
  }%
}%
\ifx\CurrentOption\babellatin@ecclesiastic
  \babellatin@ecclesiastic@outdated@commands
\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.

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

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

```
⟨classic⟩\ProvidesLanguage{classiclatin}
⟨classical⟩\ProvidesLanguage{classicallatin}
⟨ecclesiastic⟩\ProvidesLanguage{ecclesiasticlatin}
⟨ecclesiastical⟩\ProvidesLanguage{ecclesiasticallatin}
```

```
⟨medieval⟩\ProvidesLanguage{medievallatin}
\input latin.ldf\relax
```

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

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

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.

```
luatexbase.provides_module({
    name        = "ecclesiasticallatin",
    date        = "2025-08-11",
    version     = "4.2",
    description = "babel-latin punctuation spacing for ecclesiastical Latin"
})
local add_to_callback     = luatexbase.add_to_callback
local in_callback         = luatexbase.in_callback
local new_attribute       = luatexbase.new_attribute
local node                = node
local insert_node_before  = node.insert_before
local insert_node_after   = node.insert_after
local remove_node         = node.remove
local has_attribute       = node.has_attribute
local node_copy           = node.copy
local new_node            = node.new
local end_of_math         = node.end_of_math
local get_next            = node.getnext
local get_prev            = node.getprev
local get_property        = node.getproperty
```

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

```
local glue_code    = node.id"glue"
local glyph_code   = node.id"glyph"
local penalty_code = node.id"penalty"
local kern_code    = node.id"kern"
local math_code    = node.id"math"
```

We need some node subtypes:

```
local userkern = 1
local removable_skip = {
    [0]  = true, -- userskip
    [13] = true, -- spaceskip
    [14] = true  -- xspaceskip
}
```

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

```
local kern_node   = new_node(kern_code)
kern_node.subtype = userkern
local function get_kern_node(dim)
```

```
    local n = node_copy(kern_node)
    n.kern = dim
    return n
end
```

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

```
local space_chars = {
    [0x20] = true,   -- space
    [0xA0] = true,   -- no-break space
    [0x1680] = true, -- ogham space mark
    [0x2000] = true, -- en quad
    [0x2001] = true, -- em quad
    [0x2002] = true, -- en space
    [0x2003] = true, -- em space
    [0x2004] = true, -- three-per-em-space
    [0x2005] = true, -- four-per-em space
    [0x2006] = true, -- six-per-em space
    [0x2007] = true, -- figure space
    [0x2008] = true, -- punctuation space
    [0x2009] = true, -- thin space
    [0x200A] = true, -- hair space
    [0x202F] = true, -- narrow no-break space
    [0x205F] = true, -- medium mathematical space
    [0x3000] = true  -- ideographic space
}
```

All left bracket characters, referenced by their Unicode slot:

```
local left_bracket_chars = {
    [0x28] = true,  -- left parenthesis
    [0x5B] = true,  -- left square bracket
    [0x7B] = true,  -- left curly bracket
    [0x27E8] = true -- mathematical left angle bracket
}
```

All right bracket characters, referenced by their Unicode slot:

```
local right_bracket_chars = {
    [0x29] = true,  -- right parenthesis
    [0x5D] = true,  -- right square bracket
    [0x7D] = true,  -- right curly bracket
    [0x27E9] = true -- mathematical right angle bracket
}
```

Question and exclamation marks, referenced by their Unicode slot:

```
local question_exclamation_chars = {
    [0x21] = true,   -- exclamation mark !
    [0x3F] = true,   -- question mark ?
    [0x203C] = true, -- double exclamation mark ‼
    [0x203D] = true, -- interrobang ‽
    [0x2047] = true, -- double question mark ⁇
    [0x2048] = true, -- question exclamation mark ⁈
    [0x2049] = true  -- exclamation question mark ⁉
}
```

Test for a horizontal space node to be removed:

```
local function somespace(n)
```

```
    if n then
        local id, subtype = n.id, n.subtype
        if id == glue_code then
```

It is dangerous to remove all type of glue.

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

We only remove user's kern.

```
            return subtype == userkern
        elseif id == glyph_code then
            return space_chars[n.char]
        end
    end
end
```

Test for a left bracket:

```
local function someleftbracket(n)
    if n then
        local id = n.id
        if id == glyph_code then
            return left_bracket_chars[n.char]
        end
    end
end
```

Test for a right bracket:

```
local function somerightbracket(n)
    if n then
        local id = n.id
        if id == glyph_code then
            return right_bracket_chars[n.char]
        end
    end
end
```

Test for two question or exclamation marks:

```
local function question_exclamation_sequence(n1, n2)
    if n1 and n2 then
        local id1 = n1.id
        local id2 = n2.id
        if id1 == glyph_code and id2 == glyph_code then
            return question_exclamation_chars[n1.char] and question_exclamation_chars[n2.char]
        end
    end
end
```

Test for a penalty node:

```
local function somepenalty(n, value)
    if n then
        local id = n.id
        if id == penalty_code then
            if value then
                return n.penalty == value
            else
                return true
```

```
            end
        end
    end
end
```

LuaTEX attribute determining whether to space punctuation or not:

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

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

```
local left_space  = {}
local right_space = {}
```

Insertion of the necessary spaces to the node list:

```
local function process(head)
    local current = head
    while current do
        local id = current.id
        if id == glyph_code then
            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.

```
                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 $10FFFF_{16}$.

```
                if not char and current.char <= 0x10FFFF then
                    char = utf8.char(current.char)
                end
                local leftspace, rightspace
                if char then
                    leftspace  = left_space[char]
                    rightspace = right_space[char]
                end
                if leftspace or rightspace then
                    local fontparameters = fonts.hashes.parameters[current.font]
                    local spacing_node
                    if leftspace and fontparameters then
                        local prev = get_prev(current)
                        local space_exception = false
                        if prev then
```

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

```
                            space_exception = someleftbracket(prev)
                                              or question_exclamation_sequence(prev, current)
                            while somespace(prev) do
                                head = remove_node(head, prev)
                                prev = get_prev(current)
                            end
                            if somepenalty(prev, 10000) then
                                head = remove_node(head, prev)
                            end
                        end
                        spacing_node = get_kern_node(leftspace * fontparameters.quad)
```

```
                        if not space_exception then
                            head = insert_node_before(head, current, spacing_node)
                        end
                    end
                    if rightspace and fontparameters then
                        local next = get_next(current)
                        local space_exception = false
                        if next then
```

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

```
                            space_exception = somerightbracket(next)
                            local nextnext = get_next(next)
                            if somepenalty(next, 10000) and somespace(nextnext) then
                                head, next = remove_node(head, next)
                            end
                            while somespace(next) do
                                head, next = remove_node(head, next)
                            end
                        end
                        spacing_node = get_kern_node(rightspace * fontparameters.quad)
                        if not space_exception then
                            head, current = insert_node_after(head, current, spacing_node)
                        end
                    end
                end
            end
        elseif id == math_code then
            current = end_of_math(current)
        end
```

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

```
        current = get_next(current)
    end
    return head
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.

```
local hairspace = 0.08333 -- 1/12
local function space_left(char)
    left_space[char] = hairspace
end
local function space_right(char, kern)
    right_space[char] = hairspace
end
space_left('!')
space_left('?')
space_left('‼')
space_left('⁇')
space_left('⁈')
space_left('⁉')
space_left('‽') -- U+203D (interrobang)
```

```
space_left(':')
space_left(';')
space_left('»')
space_left('›')
space_right('«')
space_right('‹')
```

The following functions activate and deactivate the punctuation spacing.

```
local function activate()
    tex.setattribute(punct_attr, 1)
    for _, callback_name in ipairs{ "pre_linebreak_filter", "hpack_filter" } do
        if not in_callback(callback_name, "ecclesiasticallatin-punct.process") then
            add_to_callback(callback_name, process, "ecclesiasticallatin-punct.process")
        end
    end
end
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.

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

```
ecclesiasticallatin = ecclesiasticallatin or {}
ecclesiasticallatin.activate_spacing   = activate
ecclesiasticallatin.deactivate_spacing = deactivate
return ecclesiasticallatin
```

Change History