# Babel

## User guide

**Javier Bezos**
Current maintainer

**Johannes L. Braams**
Original author

Localization and internationalization

Unicode
TeX
pdfTeX
LuaTeX
XeTeX

# Contents

# Troubleshoooting

**What is this document about?** This user guide focuses on internationalization and localization with LaTeX and pdftex, xetex and luatex with the babel package. There are also some notes on its use with e-Plain and pdf-Plain TeX.

**What if I'm interested only in the latest changes?** Changes and new features with relation to version 3.8 are highlighted with New X.XX (⊕ is a link to the babel site), and there are some notes for the latest versions in the babel site. The most recent features can be still unstable.

**Can I help?** Sure! If you are interested in the TeX multilingual support, please join the kadingira mail list. You can follow the development of babel in GitHub and make suggestions; feel free to fork it and make pull requests. If you are the author of a package, send to me a few test files which I'll add to mine, so that possible issues can be caught in the development phase.

**It doesn't work for me!** You can ask for help in some forums like tex.stackexchange, but if you have found a bug, I strongly beg you to report it in GitHub, which is much better than just complaining on an e-mail list or a web forum. Remember *warnings are not errors* by themselves, they just warn about possible problems or incompatibilities. Hyphenation rules are maintained separately here.

**How can I contribute a new language?** See section 3.1 for contributing a language.

**I only need learn the most basic features.** The first subsections (1.1-1.3) describe the traditional way of loading a language (with `ldf` files), which is usually all you need. The alternative way based on `ini` files, which complements the previous one (it does *not* replace it, although it is still necessary in some languages), is described below; go to 1.13.

**I don't like manuals. I prefer sample files.** This manual contains lots of examples and tips, but in GitHub there are many sample files.

**Where is the code?** Run

```
lualatex --jobname=babel-code \let\babelcode\relax\input{babel.dtx}.
```

# 1 The user interface

## 1.1 Monolingual documents

In most cases, a single language is required, and then all you need in LaTeX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in LaTeX for an option – in this case a language – to be recognized by several packages.

Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current LaTeX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading fontspec. You may want to set the font attributes with fontspec, too.

**EXAMPLE** Here is a simple full example for "traditional" TeX engines (see below for xetex and luatex). The packages `fontenc` and `inputenc` do not belong to babel, but they are included in the example because typically you will need them. It assumes UTF-8, the default encoding:

PDFTEX
```
\documentclass{article}

\usepackage[T1]{fontenc}
```

```
\usepackage[french]{babel}

\begin{document}

Plus ça change, plus c'est la même chose!

\end{document}
```

Now consider something like:

```
\documentclass[french]{article}
\usepackage{babel}
\usepackage{varioref}
```

With this setting, the package `varioref` will also see the option `french` and will be able to use it.

**EXAMPLE**  And now a simple monolingual document in Russian (text from the Wikipedia) with xetex or luatex. Note neither fontenc nor inputenc are necessary, but the document should be encoded in UTF-8 and a so-called Unicode font must be loaded (in this example `\babelfont` is used, described below).

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

Россия, находящаяся на пересечении множества культур, а также
с учётом многонационального характера её населения, — отличается
высокой степенью этнокультурного многообразия и способностью к
межкультурному диалогу.

\end{document}
```

**TROUBLESHOOTING**  A common source of trouble is a wrong setting of the input encoding. Depending on the LaTeX version you can get the following somewhat cryptic error:

```
! Paragraph ended before \UTFviii@three@octets was complete.
```

Or the more explanatory:

```
! Package inputenc Error: Invalid UTF-8 byte ...
```

Make sure you set the encoding actually used by your editor.

**NOTE**  Because of the way babel has evolved, "language" can refer to (1) a set of hyphenation patterns as preloaded into the format, (2) a package option, (3) an `ldf` file, and (4) a name used in the document to select a language or dialect. So, a package option refers to a language in a generic way – sometimes it is the actual language name used to select it, sometimes it is a file name loading a language with a different name, sometimes it is a file name loading several languages. Please, read the documentation for specific languages for further info.

**TROUBLESHOOTING** The following warning is about hyphenation patterns, which are not under the direct control of babel:

```
Package babel Warning: No hyphenation patterns were preloaded for
(babel)                the language `LANG' into the format.
(babel)                Please, configure your TeX system to add them and
(babel)                rebuild the format. Now I will use the patterns
(babel)                preloaded for \language=0 instead on input line 57.
```

The document will be typeset, but very likely the text will not be correctly hyphenated. Some languages in some system may be raising this warning wrongly (because they are not hyphenated) – just ignore it. See the manual of your distribution (MacTeX, MikTeX, TeXLive, etc.) for further info about how to configure it.

**NOTE** With hyperref you may want to set the document language with something like:

```
\usepackage[pdflang=es-MX]{hyperref}
```

This is not currently done by babel and you must set it by hand. The document language can be also set with \DocumentMetadata, before \documentclass; for example:

```
\DocumentMetadata{lang=es-MX}
```

**NOTE** Although it has been customary to recommend placing \title, \author and other elements printed by \maketitle after \begin{document}, mainly because of shorthands, it is advisable to keep them in the preamble. Currently there is no real need to use shorthands in those macros.

**NOTE** Babel does not make any readjustments by default in font size, vertical positioning or line height by default. This is on purpose because the optimal solution depends on the document layout and the font, and very likely the most appropriate one is a combination of these settings.

## 1.2 Multilingual documents

In multilingual documents, just use a list of the required languages as package or class options. The last language is considered the main one, activated by default. Sometimes, the main language changes the document layout (eg, spanish and french).

**EXAMPLE** In LaTeX, the preamble of the document:

```
\documentclass{article}
\usepackage[dutch,english]{babel}
```

would tell LaTeX that the document would be written in two languages, Dutch and English, and that English would be the first language in use, and the main one.

You can also set the main language explicitly, but it is discouraged except if there is a real reason to do so:

```
\documentclass{article}
\usepackage[main=english,dutch]{babel}
```

Examples of cases where main is useful are the following.

**EXAMPLE** Some classes load babel with a hardcoded language option. Sometimes, the main language can be overridden with something like that before `\documentclass`:

```
\PassOptionsToPackage{main=english}{babel}
```

**NOTE** Languages may be set as global and as package option at the same time, but in such a case you should set explicitly the main language with the package option `main`:

```
\documentclass[italian]{book}
\usepackage[ngerman,main=italian]{babel}
```

**WARNING** In the preamble the main language has *not* been selected, except hyphenation patterns and the name assigned to `\languagename` (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the language selectors described below.

To switch the language there are two basic macros, described below in detail: `\selectlanguage` is used for blocks of text, while `\foreignlanguage` is for chunks of text inside paragraphs.

**EXAMPLE** A full bilingual document with pdftex follows. The main language is `french`, which is activated when the document begins. It assumes UTF-8:

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[english,french]{babel}

\begin{document}

Plus ça change, plus c'est la même chose!

\selectlanguage{english}

And an English paragraph, with a short text in
\foreignlanguage{french}{français}.

\end{document}
```

**EXAMPLE** With xetex and luatex, the following bilingual, single script document in UTF-8 encoding just prints a couple of 'captions' and `\today` in Danish and Vietnamese. No additional packages are required, because the default font supports both languages.

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename, \alsoname, \today.

\selectlanguage{vietnamese}

\prefacename, \alsoname, \today.

\end{document}
```

**NOTE** Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.23 for further details.

## 1.3 Mostly monolingual documents

New 3.39 ⊕  Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of \babelfont, if used).

This is particularly useful, too, when there are short texts of this kind coming from an external source whose contents are not known on beforehand (for example, titles in a bibliography). At this regard, it is worth remembering that \babelfont does *not* load any font until required, so that it can be used just in case.

New 3.84 ⊕  With pdftex, when a language is loaded on the fly (actually, with \babelprovide, because this is the macro used internally to load it) selectors now set the font encoding based on the list provided when loading fontenc. Not all scripts have an associated encoding, so this feature works only with Latin, Cyrillic, Greek, Arabic, Hebrew, Cherokee, Armenian, and Georgian, provided a suitable font is found.

**EXAMPLE**  A trivial document with the default font in English and Spanish, and FreeSerif in Russian is:

LUATEX/XETEX

```
\documentclass[english]{article}
\usepackage{babel}

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

English. \foreignlanguage{russian}{Русский}.
\foreignlanguage{spanish}{Español}.

\end{document}
```

**NOTE**  Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or tree-letter word is a valid name for a language (eg, lu can be the locale name with tag khb or the tag for lubakatanga). See section 1.23 for further details.

## 1.4 Modifiers

New 3.9c  The basic behavior of some languages can be modified when loading babel by means of *modifiers*. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the main key accepts them). An example is (spaces are not significant and they can be added or removed):[1]

```
\usepackage[latin.medieval, spanish.notilde.lcroman, danish]{babel}
```

Attributes (described below) are considered modifiers, ie, you can set an attribute by including it in the list of modifiers.

New 3.89 ⊕  Alternatively, modifiers can be set with a separate option, with the keyword modifiers followed by a dot and the language name (note the language is not selected or loaded with this option). It is useful to activate some feature when the language is declared as a class option:

---

[1]No predefined "axis" for modifiers are provided because languages and their scripts have quite different needs.

```
\documentclass[spanish]{report}
\usepackage[modifiers.spanish = notilde.lcroman]{babel}
```

## 1.5  Troubleshooting

- Loading directly `sty` files in LaTeX (ie, `\usepackage{`⟨*language*⟩`}`) is deprecated and you will get the error:[2]

```
! Package babel Error: You are loading directly a language style.
(babel)                This syntax is deprecated and you must use
(babel)                \usepackage[language]{babel}.
```

- Another typical error when using babel is the following:[3]

```
! Package babel Error: Unknown language `#1'. Either you have
(babel)                misspelled its name, it has not been installed,
(babel)                or you requested it in a previous run. Fix its name,
(babel)                install it or just rerun the file, respectively. In
(babel)                some cases, you may need to remove the aux file
```

The most frequent reason is, by far, the latest (for example, you included `spanish`, but you realized this language is not used after all, and therefore you removed it from the option list). In most cases, the error vanishes when the document is typeset again, but in more severe ones you will need to remove the aux file.

## 1.6  Plain

In e-Plain and pdf-Plain, load languages styles with `\input` and then use `\begindocument` (the latter is defined by babel):

```
\input estonian.sty
\begindocument
```

**WARNING**  Not all languages provide a `sty` file and some of them are not compatible with those formats. Please, refer to Using babel with Plain for further details.

## 1.7  Basic language selectors

This section describes the commands to be used in the document to switch the language in multilingual documents. In most cases, only the two basic macros `\selectlanguage` and `\foreignlanguage` are necessary. The environments `otherlanguage`, `otherlanguage*` and `hyphenrules` are auxiliary, and described in the next section.
The main language is selected automatically when the `document` environment begins.

`\selectlanguage` {⟨*language*⟩}

When a user wants to switch from one language to another he can do so using the macro `\selectlanguage`. This macro takes the language, defined previously by a language definition file, as its argument. It calls several macros that should be defined in the language definition files to activate the special definitions for the language chosen:

---

[2]In old versions the error read "You have used an old interface to call babel", not very helpful.
[3]In old versions the error read "You haven't loaded the language LANG yet".

```
\selectlanguage{german}
```

This command can be used as environment, too.

**NOTE**  For "historical reasons", a macro name is converted to a language name without the leading \; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a "real" name is deprecated. New 3.43 ⊕  However, if the macro name does not match any language, it will get expanded as expected.

**NOTE**  Bear in mind `\selectlanguage` can be automatically executed, in some cases, in the auxiliary files, at heads and foots, and after the environment `otherlanguage*`.

**WARNING**  If used inside braces there might be some non-local changes, as this would be roughly equivalent to:

```
{\selectlanguage{<inner-language>} ...}\selectlanguage{<outer-language>}
```

If you want a change which is really local, you must enclose this code with an additional grouping level.

**WARNING**  There are a couple of issues related to the way the language information is written to the auxiliary files:

- `\selectlanguage` should not be used inside some boxed environments (like floats or `minipage`) to switch the language if you need the information written to the `aux` be correctly synchronized. This rarely happens, but if it were the case, you must use `otherlanguage` instead.

- In addition, this macro inserts a `\write` in vertical mode, which may break the vertical spacing in some cases (for example, between lists). New 3.64 ⊕  The behavior can be adjusted with `\babeladjust{select.write=⟨mode⟩}`, where ⟨*mode*⟩ is `shift` (which shifts the skips down and adds a `\penalty`); `keep` (the default – with it the `\write` and the skips are kept in the order they are written), and `omit` (which may seem a too drastic solution, because nothing is written, but more often than not this command is applied to more or less shorts texts with no sectioning or similar commands and therefore no language synchronization is necessary).

`\foreignlanguage` [⟨*option-list*⟩]{⟨*language*⟩}{⟨*text*⟩}

The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one.
This command (1) only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates, (2) does not send information about the language to auxiliary files (i.e., the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the `bidi` option, it also enters in horizontal mode (this is not done always for backwards compatibility), and since it is meant for phrases only the text direction (and not the paragraph one) is set.
New 3.44 ⊕  As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

```
\foreignlanguage[date]{polish}{\today}
```

In addition, captions can be switched with `captions` (or both, of course, with `date, captions`). Until 3.43 you had to write something like `{\selectlanguage{..} ..}`, which was not always the most convenient way.

## 1.8  Auxiliary language selectors

`\begin{otherlanguage}` {⟨*language*⟩} ... `\end{otherlanguage}`

The environment `otherlanguage` does basically the same as `\selectlanguage`, except that language change is (mostly) local to the environment.

Actually, there might be some non-local changes, as this environment is roughly equivalent to:

```
\begingroup
\selectlanguage{<inner-language>}
...
\endgroup
\selectlanguage{<outer-language>}
```

If you want a change which is really local, you must enclose this environment with an additional grouping, like braces {}.

Spaces after the environment are ignored.

`\begin{otherlanguage*}` [⟨*option-list*⟩]{⟨*language*⟩} ... `\end{otherlanguage*}`

Same as `\foreignlanguage` but as environment. Spaces after the environment are *not* ignored.

This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a line. However, by default it never complied with the documented behavior and it is just a version as environment of `\foreignlanguage`, except when the option `bidi` is set – in this case, `\foreignlanguage` emits a `\leavevmode`, while `otherlanguage*` does not.

### 1.9 More on selection

`\babeltags` {⟨*tag1*⟩ = ⟨*language1*⟩, ⟨*tag2*⟩ = ⟨*language2*⟩, ...}

New 3.9i In multilingual documents with many language-switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.

It defines `\text`⟨*tag1*⟩{⟨*text*⟩} to be `\foreignlanguage`{⟨*language1*⟩}{⟨*text*⟩}, and `\begin`{⟨*tag1*⟩} to be `\begin{otherlanguage*}`{⟨*language1*⟩}, and so on. Note `\`⟨*tag1*⟩ is also allowed, but remember to set it locally inside a group.

**WARNING** There is a clear drawback to this feature, namely, the 'prefix' `\text...` is heavily overloaded in LaTeX and conflicts with existing macros may arise (`\textlatin`, `\textbar`, `\textit`, `\textcolor` and many others). The same applies to environments, because `arabic` conflicts with `\arabic`. Furthermore, and because of this overloading, detecting the language of a chunk of text by external tools can become unfeasible. Except if there is a reason for this 'syntactical sugar', the best option is to stick to the default selectors or to define your own alternatives.

**EXAMPLE** With

```
\babeltags{de = german}
```

you can write

```
text \textde{German text} text
```

and

```
   text
\begin{de}
  German text
\end{de}
   text
```

**NOTE** Something like \babeltags{finnish = finnish} is legitimate – it defines
\textfinnish and \finnish (and, of course, \begin{finnish}).

\babelensure [include=⟨*commands*⟩,exclude=⟨*commands*⟩,fontenc=⟨*encoding*⟩]{⟨*language*⟩}

New 3.9i Except in a few languages, like russian, captions and dates are just strings, and
do not switch the language. That means you should set it explicitly if you want to use them,
or hyphenation (and in some cases the text itself) will be wrong. For example:

```
\foreignlanguage{russian}{text \foreignlanguage{polish}{\seename} text}
```

Of course, TeX can do it for you. To avoid switching the language all the while,
\babelensure redefines the captions for a given language to wrap them with a selector:

```
\babelensure{polish}
```

By default only the basic captions and \today are redefined, but you can add further
macros with the key include in the optional argument (without commas). Macros not to
be modified are listed in exclude. You can also enforce a font encoding with the option
fontenc.[4] A couple of examples:

```
\babelensure[include=\Today]{spanish}
\babelensure[fontenc=T5]{vietnamese}
```

They are activated when the language is selected (at the afterextras event), and it makes
some assumptions which could not be fulfilled in some languages. Note also you should
include only macros defined by the language, not global macros (eg, \TeX of \dag).
With ini files (see below), captions are ensured by default.

## 1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary TeX code.
Shorthands can be used for different kinds of things; for example: (1) in some languages
shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1;
(2) in some languages shorthands such as ! are used to insert the right amount of white
space; (3) several kinds of discretionaries and breaks can be inserted easily with "-, "=, etc.
The package inputenc as well as xetex and luatex have alleviated entering non-ASCII
characters, but minority languages and some kinds of text can still require characters not
directly available on the keyboards (and sometimes not even as separated or precomposed
Unicode characters). As to the point 2, now pdfTeX provides \knbccode, and luatex can
manipulate the glyph list. Tools for point 3 can be still very useful in general.
There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of
precedence). In most cases, you will use only shorthands provided by languages.

**NOTE** Keep in mind the following:

1. Activated chars used for two-char shorthands cannot be followed by a closing brace
   } and the spaces following are gobbled. With one-char shorthands (eg, :), they are
   preserved.

---

[4]With it, encoded strings may not work as expected.

2. If on a certain level (system, language, user, language user) there is a one-char shorthand, two-char ones starting with that char and on the same level are ignored.

3. Since they are active, a shorthand cannot contain the same character in its definition (except if deactivated with, eg, \string).

**TROUBLESHOOTING** A typical error when using shorthands is the following:

```
! Argument of \language@active@arg" has an extra }.
```

It means there is a closing brace just after a shorthand, which is not allowed (eg, "}). Just add {} after (eg, "{}}).

\shorthandon {⟨*shorthands-list*⟩}
\shorthandoff *{⟨*shorthands-list*⟩}

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands \shorthandoff and \shorthandon are provided. They each take a list of characters as their arguments. The command \shorthandoff sets the \catcode for each of the characters in its argument to other (12); the command \shorthandon sets the \catcode to active (13). Both commands only work on 'known' shorthand characters, and an error will be raised otherwise. You can check if a character is a shorthand with \ifbabelshorthand (see below). New 3.9a However, \shorthandoff does not behave as you would expect with characters like ~ or ^, because they usually are not "other". For them \shorthandoff* is provided, so that with

```
\shorthandoff*{~^}
```

~ is still active, very likely with the meaning of a non-breaking space, and ^ is the superscript character. The catcodes used are those when the shorthands are defined, usually when language files are loaded.
If you do not need shorthands, or prefer an alternative approach of your own, you may want to switch them off with the package option shorthands=off, as described below.

**WARNING** It is worth emphasizing these macros are meant for temporary changes. Whenever possible and if there are not conflicts with other packages, shorthands must be always enabled (or disabled).

\useshorthands *{⟨*char*⟩}

The command \useshorthands initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands. New 3.9a User shorthands are not always alive, as they may be deactivated by languages (for example, if you use " for your user shorthands and switch from german to french, they stop working). Therefore, a starred version \useshorthands*{⟨*char*⟩} is provided, which makes sure shorthands are always activated.
Currently, if the package option shorthands is used, you must include any character to be activated with \useshorthands. This restriction will be lifted in a future release.

\defineshorthand [⟨*language*⟩,⟨*language*⟩,...]{⟨*shorthand*⟩}{⟨*code*⟩}

The command \defineshorthand takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to. New 3.9a An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add

12

\languageshorthands{⟨*lang*⟩} to the corresponding \extras⟨*lang*⟩, as explained below). By default, user shorthands are (re)defined.

User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over "normal" user shorthands.

**EXAMPLE**  Let's assume you want a unified set of shorthand for discretionaries (languages do not define shorthands consistently, and "-, \-, "= have different meanings). You can start with, say:

```
\useshorthands*{"}
\defineshorthand{"*}{\babelhyphen{soft}}
\defineshorthand{"-}{\babelhyphen{hard}}
```

However, the behavior of hyphens is language-dependent. For example, in languages like Polish and Portuguese, a hard hyphen inside compound words are repeated at the beginning of the next line. You can then set:

```
\defineshorthand[*polish,*portuguese]{"-}{\babelhyphen{repeat}}
```

Here, options with * set a language-dependent user shorthand, which means the generic one above only applies for the rest of languages; without * they would (re)define the language shorthands instead, which are overridden by user ones.

Now, you have a single unified shorthand ("-), with a content-based meaning ('compound word hyphen') whose visual behavior is that expected in each context.

**\languageshorthands**  {⟨*language*⟩}

The command \languageshorthands can be used to switch the shorthands on the language level. It takes one argument, the name of a language or none (the latter does what its name suggests).[5] Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

```
\addto\extrasenglish{\languageshorthands{ngerman}}
```

(You may also need to activate them as user shorthands in the preamble with, for example, \useshorthands or \useshorthands*.)

**EXAMPLE**  Very often, this is a more convenient way to deactivate shorthands than \shorthandoff, for example if you want to define a macro to easy typing phonetic characters with tipa:

```
\newcommand{\myipa}[1]{{\languageshorthands{none}\tipaencoding#1}}
```

**\babelshorthand**  {⟨*shorthand*⟩}

With this command you can use a shorthand even if (1) not activated in shorthands (in this case only shorthands for the current language are taken into account, ie, not user shorthands), (2) turned off with \shorthandoff or (3) deactivated with the internal \bbl@deactivate; for example, \babelshorthand{"u} or \babelshorthand{:}. (You can conveniently define your own macros, or even your own user shorthands provided they do not overlap.)

---

[5]Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.

**EXAMPLE** Since by default shorthands are not activated until `\begin{document}`, you may use this macro when defining the `\title` in the preamble:

```
\title{Documento científico\babelshorthand{"-}técnico}
```

For your records, here is a list of shorthands, but you must double check them, as they may change:[6]

**Languages with no shorthands** Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh

**Languages with only " as defined shorthand character** Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

**Basque** " ' ~
**Breton** : ; ? !
**Catalan** " ' `
**Czech** " -
**Esperanto** ^
**Estonian** " ~
**French** (all varieties) : ; ? !
**Galician** " . ' ~ < >
**Greek** ~
**Hungarian** `
**Kurmanji** ^
**Latin** " ^ =
**Slovak** " ^ ' -
**Spanish** " . < > ' ~
**Turkish** : ! =

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.[7]

`\ifbabelshorthand` {⟨*character*⟩}{⟨*true*⟩}{⟨*false*⟩}

New 3.23 Tests if a character has been made a shorthand.

### 1.11  Package options

New 3.9a These package options are processed before language options, so that they are taken into account irrespective of its order. The first three options have been available in previous versions.

`KeepShorthandsActive` Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.

`activeacute` For some languages babel supports this options to set ' as a shorthand in case it is not done by default.

`activegrave` Same for `.

`shorthands=` ⟨*char*⟩⟨*char*⟩... | off

The only language shorthands activated are those given, like, eg:

---

[6]Thanks to Enrico Gregorio
[7]This declaration serves to nothing, but it is preserved for backward compatibility.

```
\usepackage[esperanto,french,shorthands=:;!?]{babel}
```

If ' is included, `activeacute` is set; if ` is included, `activegrave` is set. Active characters (like ~) should be preceded by `\string` (otherwise they will be expanded by LaTeX before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With `shorthands=off` no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro `\babelshorthand` is defined, which allows using them; see above.

safe= none | ref | bib

Some LaTeX macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from varioref and ifthen).
With `safe=none` no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of New 3.34 , in $\epsilon$TeX based engines (ie, almost every engine except the oldest ones) shorthands can be used in these macros (formerly you could not).

math= active | normal

Shorthands are mainly intended for text, not for math. By setting this option with the value `normal` they are deactivated in math mode (default is `active`) and things like `${a'}$` (a closing brace after a shorthand) are not a source of trouble anymore.

config= ⟨*file*⟩

Load ⟨*file*⟩`.cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).

main= ⟨*language*⟩

Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.

headfoot= ⟨*language*⟩

By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots. An alternative is to set the language explicitly when heads and foots are redefined.

noconfigs Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. However, if the key `config` is set, this file is loaded.

showlanguages Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.

silent New 3.9l No warnings and no *infos* are written to the log file.[8]

hyphenmap= off | first | select | other | other*

New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.[9] It can take the following values:

---

[8]You can use alternatively the package silence.
[9]Turned off in plain.

off  deactivates this feature and no case mapping is applied;

first  sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first \selectlanguage in the preamble), and it's the default if a single language option has been stated;[10]

select  sets it only at \selectlanguage;

other  also sets it at otherlanguage;

other*  also sets it at otherlanguage* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other* for monolingual documents.[11]

**bidi=**  default | basic | basic-r | bidi-l | bidi-r

New 3.14  Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.25.

**layout=**

New 3.16  Selects which layout elements are adapted in bidi documents. See sec. 1.25.

**provide=**  *

New 3.49 ⊕  An alternative to \babelprovide for languages passed as options. See section 1.13, which describes also the variants provide+= and provide*=.

### 1.12  The base option

With this package option babel just loads some basic macros (those in switch.def), defines \AfterBabelLanguage and exits. It also selects the hyphenation patterns for the last language passed as option (by its name in language.dat). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

**\AfterBabelLanguage**  {⟨option-name⟩}{⟨code⟩}

This command is currently the only provided by base. Executes ⟨code⟩ when the file loaded by the corresponding package option is finished (at \ldf@finish). The setting is global. So

```
\AfterBabelLanguage{french}{...}
```

does ... at the end of french.ldf. It can be used in ldf files, too, but in such a case the code is executed only if ⟨option-name⟩ is the same as \CurrentOption (which could not be the same as the option name as set in \usepackage!).

**EXAMPLE**  Consider two languages foo and bar defining the same \macro with \newcommand. An error is raised if you attempt to load both. Here is a way to overcome this problem:

```
\usepackage[base]{babel}
\AfterBabelLanguage{foo}{%
  \let\macroFoo\macro
  \let\macro\relax}
\usepackage[foo,bar]{babel}
```

---

[10] Duplicated options count as several ones.

[11] Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn't really useful, but who knows.

**NOTE**  With a recent version of LaTeX, an alternative method to execute some code just after an ldf file is loaded is with `\AddToHook` and the hook `file/<language>.ldf/after`. Babel does not predeclare it, and you have to do it yourself with `\ActivateGenericHook`.

**WARNING**  Currently this option is not compatible with languages loaded on the fly.

### 1.13  `ini` files

An alternative approach to define a language (or, more precisely, a *locale*) is by means of an `ini` file. Currently babel provides about 360 of these files containing the basic data required for a locale, covering about 270 languages, plus basic templates for about 400 locales.
`ini` files are not meant only for babel, and they has been devised as a resource for other packages. To easy interoperability between TeX and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the `\...name` strings).
Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them by means of `\babelprovide`. In other words, `\babelprovide` is mainly meant for auxiliary tasks, and as alternative when the `ldf`, for some reason, does not exists or does not work as expected.

**EXAMPLE**  Although Georgian has its own `ldf` file, here is how to declare this language with an `ini` file in Unicode engines.

LUATEX/XETEX

```
\documentclass{book}

\usepackage{babel}
\babelprovide[import, main]{georgian}

\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეუღო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეუღო ერთ-ერთი უძღიდრესია მთელ მსოფლიოში.

\end{document}
```

**MORE**  There is an example of how to use a template `ini` file here, for Phoenician.

New 3.49 ⊕  Alternatively, you can tell babel to load all or some languages passed as options with `\babelprovide` and not from the `ldf` file in a few few typical cases. Thus, `provide=*` means 'load the main language with the `\babelprovide` mechanism instead of the `ldf` file' applying the basic features, which in this case means `import, main`. There are (currently) three options:

- `provide=*` is the option just explained, for the main language;

- `provide+=*` is the same for additional languages (the main language is still the `ldf` file);

- `provide*=*` is the same for all languages, ie, main and additional.

**EXAMPLE**  The preamble in the previous example can be more compactly written as:

17

```
\documentclass{book}
\usepackage[georgian, provide=*]{babel}
\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}
```

Or also:

```
\documentclass[georgian]{book}
\usepackage[provide=*]{babel}
\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}
```

**NOTE**  The ini files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follow (which could no longer be valid when you read this manual, if the packages involved han been updated). The Harfbuzz renderer has still some issues, so as a rule of thumb prefer the default renderer, and resort to Harfbuzz only if the former does not work for you. Fortunately, fonts can be loaded twice with different renderers; for example:

```
\babelfont[spanish]{rm}{FreeSerif}
\babelfont[hindi]{rm}[Renderer=Harfbuzz]{FreeSerif}
```

**Arabic**  Monolingual documents mostly work in luatex, but it must be fine tuned, particularly math and graphical elements like `picture`. In xetex babel resorts to the bidi package, which seems to work.

**Hebrew**  Niqqud marks seem to work in both engines, but depending on the font cantillation marks might be misplaced (xetex or luatex with Harfbuzz seems better).

**Devanagari**  In luatex and the the default renderer many fonts work, but some others do not, the main issue being the 'ra'. You may need to set explicitly the script to either deva or dev2, eg:

```
\newfontscript{Devanagari}{deva}
```

Other Indic scripts are still under development in the default luatex renderer, but should work with `Renderer=Harfbuzz`. They also work with xetex, although unlike with luatex fine tuning the font behavior is not always possible.

**Southeast scripts**  Thai works in both luatex and xetex, but line breaking differs (rules are hard-coded in xetex, but they can be modified in luatex). Lao seems to work, too, but there are no patterns for the latter in luatex. Khemer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and lualatex also applies here. Some quick patterns can help, with something similar to:

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1ກ 1ຸ 1ະ 1ງ 1ກ 1ຈ} % Random
```

**East Asia scripts**  Settings for either Simplified of Traditional should work out of the box, with basic line breaking with any renderer. Although for a few words and shorts texts the ini files should be fine, CJK texts are best set with a dedicated framework (CJK, luatexja, kotex, CTeX, etc.). This is what the class `ltjbook` does with luatex, which can be used in conjunction with the `ldf` for `japanese`, because the following piece of code loads luatexja:

```
\documentclass[japanese]{ltjbook}
\usepackage{babel}
```

**Latin, Greek, Cyrillic** Combining chars with the default luatex font renderer might be wrong; on then other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenations points are discarded (this bug is related to kerning, so it depends on the font). With xetex both combining characters and hyphenation work as expected (not quite, but in most cases it works; the problem here are font clusters).

**NOTE** Wikipedia defines a *locale* as follows: "In computing, a locale is a set of parameters that defines the user's language, region and any special variant preferences that the user wants to see in their user interface. Usually a locale identifier consists of at least a language code and a country/region code." Babel is moving gradually from the old and fuzzy concept of *language* to the more modern of *locale*. Note each locale is by itself a separate "language", which explains why there are so many files. This is on purpose, so that possible variants can be created and/or redefined easily.

Here is the list (u means Unicode captions, and l means LICR captions):

| | | | |
|---|---|---|---|
| aa | Afar | ca | Catalan[ul] |
| ab | Abkhazian | cch | Atsam |
| af | Afrikaans[ul] | ce | Chechen |
| agq | Aghem | cgg | Chiga |
| ak | Akan | chr | Cherokee |
| am | Amharic[ul] | ckb-Arab | Central Kurdish[u] |
| ar-DZ | Arabic[u] | ckb-Latn | Central Kurdish[u] |
| ar-EG | Arabic[u] | ckb | Central Kurdish[u] |
| ar-IQ | Arabic[u] | co | Corsican |
| ar-JO | Arabic[u] | cop | Coptic |
| ar-LB | Arabic[u] | cs | Czech[ul] |
| ar-MA | Arabic[u] | cu-Cyrs | Church Slavic[u] |
| ar-PS | Arabic[u] | cu-Glag | Church Slavic |
| ar-SA | Arabic[u] | cu | Church Slavic[u] |
| ar-SY | Arabic[u] | cy | Welsh[ul] |
| ar-TN | Arabic[u] | da | Danish[ul] |
| ar | Arabic[u] | dav | Taita |
| as | Assamese[u] | de-1901 | German[ul] |
| asa | Asu | de-1996 | German[ul] |
| ast | Asturian[ul] | de-AT-1901 | Austrian German[ul] |
| az-Cyrl | Azerbaijani | de-AT-1996 | Austrian German[ul] |
| az-Latn | Azerbaijani | de-AT | Austrian German[ul] |
| az | Azerbaijani[ul] | de-CH-1901 | Swiss High German[ul] |
| bal | Baluchi | de-CH-1996 | Swiss High German[ul] |
| bas | Basaa | de-CH | Swiss High German[ul] |
| be | Belarusian[ul] | de | German[ul] |
| bem | Bemba | dje | Zarma |
| bez | Bena | dsb | Lower Sorbian[ul] |
| bg | Bulgarian[ul] | dua | Duala |
| bgc | Haryanvi | dyo | Jola-Fonyi |
| bho | Bhojpuri | dz | Dzongkha |
| bm | Bambara | ebu | Embu |
| bn | Bangla[u] | ee | Ewe |
| bo | Tibetan[u] | el-polyton | Polytonic Greek[ul] |
| br | Breton[ul] | el | Greek[ul] |
| brx | Bodo | en-AU | Australian English[ul] |
| bs-Cyrl | Bosnian | en-CA | Canadian English[ul] |
| bs-Latn | Bosnian[ul] | en-GB | British English[ul] |
| bs | Bosnian[ul] | en-NZ | English[ul] |
| byn | Blin | en-US | American English[ul] |

| Code | Language | Code | Language |
|---|---|---|---|
| en | English[ul] | kgp | Kaingang |
| eo | Esperanto[ul] | khq | Koyra Chiini |
| es-MX | Mexican Spanish[ul] | ki | Kikuyu |
| es | Spanish[ul] | kk | Kazakh |
| et | Estonian[ul] | kkj | Kako |
| eu | Basque[ull] | kl | Kalaallisut |
| ewo | Ewondo | kln | Kalenjin |
| fa | Persian[u] | km | Khmer[u] |
| ff | Fulah | kmr-Arab | Northern Kurdish[u] |
| fi | Finnish[ul] | kmr-Latn | Northern Kurdish[ul] |
| fil | Filipino | kmr | Northern Kurdish[ul] |
| fo | Faroese | kn | Kannada[u] |
| fr-BE | French[ul] | ko-Hani | Korean[u] |
| fr-CA | Canadian French[ul] | ko | Korean[u] |
| fr-CH | Swiss French[ul] | kok | Konkani |
| fr-LU | French[ul] | ks | Kashmiri |
| fr-x-acadian | Acadian[ul] | ksb | Shambala |
| fr | French[ul] | ksf | Bafia |
| fur | Friulian[ul] | ksh | Colognian |
| fy | Western Frisian | kw | Cornish |
| ga | Irish[ul] | ky | Kyrgyz |
| gaa | Ga | la-x-classic | Classical Latin[ul] |
| gd | Scottish Gaelic[ul] | la-x-ecclesia | Ecclesiastical Latin[ul] |
| gez | Geez | la-x-medieval | Medieval Latin[ul] |
| gl | Galician[ul] | la | Latin[ul] |
| gn | Guarani | lag | Langi |
| grc | Ancient Greek[ul] | lb | Luxembourgish[ul] |
| gsw | Swiss German | lg | Ganda |
| gu | Gujarati | lij | Ligurian |
| guz | Gusii | lkt | Lakota |
| gv | Manx | ln | Lingala |
| ha-GH | Hausa | lo | Lao[u] |
| ha-NE | Hausa | lrc | Northern Luri |
| ha | Hausa[ul] | lt | Lithuanian[ulll] |
| haw | Hawaiian | lu | Luba-Katanga |
| he | Hebrew[ul] | luo | Luo |
| hi | Hindi[u] | luy | Luyia |
| hr | Croatian[ul] | lv | Latvian[ul] |
| hsb | Upper Sorbian[ul] | mas | Masai |
| hu | Hungarian[ulll] | mer | Meru |
| hy | Armenian[ul] | mfe | Morisyen |
| ia | Interlingua[ul] | mg | Malagasy |
| id | Indonesian[ul] | mgh | Makhuwa-Meetto |
| ig | Igbo | mgo | Meta' |
| ii | Sichuan Yi | mk | Macedonian[ul] |
| is | Icelandic[ul] | ml | Malayalam[u] |
| it | Italian[ul] | mn | Mongolian |
| iu | Inuktitut | mr | Marathi[u] |
| ja | Japanese[u] | ms-BN | Malay |
| jgo | Ngomba | ms-SG | Malay |
| jmc | Machame | ms | Malay[ul] |
| ka | Georgian[u] | mt | Maltese |
| kab | Kabyle | mua | Mundang |
| kaj | Jju | mus | Muscogee |
| kam | Kamba | my | Burmese |
| kcg | Tyap | myv | Erzya |
| kde | Makonde | mzn | Mazanderani |
| kea | Kabuverdianu | naq | Nama |

| Code | Language | | Code | Language |
|---|---|---|---|---|
| nb | Norwegian Bokmål[ul] | | si | Sinhala[u] |
| nd | North Ndebele | | sk | Slovak[ul] |
| ne | Nepali | | sl | Slovenian[ul] |
| nl | Dutch[ul] | | smn | Inari Sami |
| nmg | Kwasio | | sn | Shona |
| nn | Norwegian Nynorsk[ul] | | so | Somali |
| nnh | Ngiemboon | | sq | Albanian[ul] |
| no | Norwegian[ul] | | sr-Cyrl-BA | Serbian[ul] |
| nqo | N'Ko | | sr-Cyrl-ME | Serbian[ul] |
| nr | South Ndebele | | sr-Cyrl-XK | Serbian[ul] |
| nso | Northern Sotho | | sr-Cyrl | Serbian[ul] |
| nus | Nuer | | sr-ijekavsk | Serbian[ul] |
| ny | Nyanja | | sr-Latn-BA | Serbian[ul] |
| nyn | Nyankole | | sr-Latn-ijekavsk | Serbian[ul] |
| oc | Occitan[ul] | | sr-Latn-ME | Serbian[ul] |
| om | Oromo | | sr-Latn-XK | Serbian[ul] |
| or | Odia | | sr-Latn | Serbian[ul] |
| os | Ossetic | | sr | Serbian[ul] |
| pa-Arab | Punjabi | | ss | Swati |
| pa-Guru | Punjabi[u] | | ssy | Saho |
| pa | Punjabi[u] | | st | Southern Sotho |
| pap | Papiamento | | sv | Swedish[ul] |
| pl | Polish[ul] | | sw | Swahili |
| pms | Piedmontese[ul] | | syr | Syriac |
| prg | Prussian | | szl | Silesian |
| ps | Pashto | | ta | Tamil[u] |
| pt-BR | Brazilian Portuguese[ul] | | te | Telugu[u] |
| pt-PT | European Portuguese[ul] | | teo | Teso |
| pt | Portuguese[ul] | | th | Thai[ul] |
| qu | Quechua | | ti | Tigrinya |
| raj | Rajasthani | | tig | Tigre |
| rm | Romansh[ul] | | tk | Turkmen[ul] |
| rn | Rundi | | tn | Tswana |
| ro-MD | Moldavian[ul] | | to | Tongan |
| ro | Romanian[ul] | | tpi | Tok Pisin |
| rof | Rombo | | tr | Turkish[ul] |
| ru | Russian[ul] | | trv | Taroko |
| rw | Kinyarwanda | | ts | Tsonga |
| rwk | Rwa | | twq | Tasawaq |
| sa-Beng | Sanskrit | | tzm | Central Atlas Tamazight |
| sa-Deva | Sanskrit | | ug | Uyghur[u] |
| sa-Gujr | Sanskrit | | uk | Ukrainian[ul] |
| sa-Knda | Sanskrit | | ur | Urdu[u] |
| sa-Mlym | Sanskrit | | uz-Arab | Uzbek |
| sa-Telu | Sanskrit | | uz-Cyrl | Uzbek |
| sa | Sanskrit | | uz-Latn | Uzbek |
| sah | Sakha | | uz | Uzbek |
| saq | Samburu | | vai-Latn | Vai |
| sbp | Sangu | | vai-Vaii | Vai |
| sc | Sardinian | | vai | Vai |
| scn | Sicilian | | ve | Venda |
| se | Northern Sami[ul] | | vi | Vietnamese[ul] |
| seh | Sena | | vo | Volapük |
| ses | Koyraboro Senni | | vun | Vunjo |
| sg | Sango | | wae | Walser |
| shi-Latn | Tachelhit | | wal | Wolaytta |
| shi-Tfng | Tachelhit | | xog | Soga |
| shi | Tachelhit | | yav | Yangben |

| | | | |
|---|---|---|---|
| yi | Yiddish | zh-Hans-SG | Chinese |
| yo | Yoruba | zh-Hans | Chinese[u] |
| yrl | Nheengatu | zh-Hant-HK | Chinese |
| yue | Cantonese | zh-Hant-MO | Chinese |
| zgh | Standard Moroccan Tamazight | zh-Hant | Chinese[u] |
| zh-Hans-HK | Chinese | zh | Chinese[u] |
| zh-Hans-MO | Chinese | zu | Zulu |

In some contexts (currently \babelfont) an ini file may be loaded by its name. Here is the list of the names currently supported. With these languages, \babelfont loads (if not done before) the language and script names (even if the language is defined as a package option with an ldf file). These are also the names recognized by \babelprovide with a valueless import.

abkhazian
acadian
afar
afrikaans
aghem
akan
albanian
american
americanenglish
amharic
ancientgreek
arabic
arabic-algeria
arabic-dz
arabic-eg
arabic-egypt
arabic-iq
arabic-iraq
arabic-jo
arabic-jordan
arabic-lb
arabic-lebanon
arabic-ma
arabic-morocco
arabic-palestinianterritories
arabic-ps
arabic-sa
arabic-saudiarabia
arabic-sy
arabic-syria
arabic-tn
arabic-tunisia
armenian
assamese
asturian
asu
atsam
australian
australianenglish
austrian
azerbaijani
azerbaijani-cyrillic
azerbaijani-cyrl

azerbaijani-latin
azerbaijani-latn
bafia
baluchi
bambara
bangla
basaa
basque
belarusian
bemba
bena
bengali
bhojpuri
blin
bodo
bosnian
bosnian-cyrillic
bosnian-cyrl
bosnian-latin
bosnian-latn
brazilian
brazilianportuguese
breton
british
britishenglish
bulgarian
burmese
canadian
canadianenglish
cantonese
catalan
centralatlastamazight
centralkurdish
centralkurdish-latin
centralkurdish-latn
chechen
cherokee
chiga
chinese
chinese-hans
chinese-hans-hk
chinese-hans-mo
chinese-hans-sg

chinese-hant
chinese-hant-hk
chinese-hant-mo
chinese-simplified
chinese-simplified-hongkongsarchina
chinese-simplified-macausarchina
chinese-simplified-singapore
chinese-traditional
chinese-traditional-hongkongsarchina
chinese-traditional-macausarchina
churchslavic
churchslavic-cyrs
churchslavic-glag
churchslavic-glagolitic
churchslavic-oldcyrillic[12]
churchsslavic-glag
churchslavonic
classicallatin
colognian
coptic
cornish
corsican
croatian
czech
danish
duala
dutch
dzongkha
ecclesiasticallatin
embu
english
english-au
english-australia
english-ca
english-canada
english-gb
english-newzealand
english-nz
english-unitedkingdom

---

[12]The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.

english-unitedstates
english-us
erzya
esperanto
estonian
europeanportuguese
ewe
ewondo
faroese
farsi
filipino
finnish
french
french-be
french-belgium
french-ca
french-canada
french-canadianfrench
french-ch
french-lu
french-luxembourg
french-swissfrench
french-switzerland
friulian
fulah
ga
galician
ganda
geez
georgian
german
german-at
german-austria
german-austria-traditional
german-ch
german-switzerland
german-switzerland-traditional
german-traditional
greek
guarani
gujarati
gusii
haryanvi
hausa
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
inuktitut
irish
italian
japanese
javanese
jju

jolafonyi
kabuverdianu
kabyle
kaingang
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
korean-han
korean-hani
koyraborosenni
koyrachiini
kurmanji
kwasio
kyrgyz
lakota
langi
lao
latin
latvian
ligurian
lingala
lithuanian
lowersorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
maithiri
makhuwameetto
makonde
malagasy
malay
malay-bn
malay-brunei
malay-sg
malay-singapore
malayalam
maltese
manx
marathi
masai
mazanderani
medievallatin
meru
meta
mexican
mexicanspanish
moldavian
mongolian
monotonicgreek
morisyen
mundang

muscogee
nama
naustrian
nepali
newzealand
ngerman
ngiemboon
ngomba
nheengatu
nigerianpidgin
nko
norsk
northernkurdish
northernkurdish-arab
northernkurdish-arabic
northernluri
northernsami
northernsotho
northndebele
norwegian
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyanja
nyankole
nynorsk
occitan
odia
oriya
oromo
ossetic
papiamento
pashto
persian
piedmontese
polish
polytonicgreek
portuguese
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
prussian
punjabi
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
quechua
rajasthani
romanian
romanian-md
romanian-moldova
romansh
rombo
rundi
russian
rwa
saho
sakha
samburu
sango

sangu
sanskrit
sanskrit-bangla
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sardinian
scottishgaelic
sena
serbian
serbian-cyrillic
serbian-cyrillic-
bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrl
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-ijekavsk
serbian-latin
serbian-latin-
bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latn
serbian-latn-ba
serbian-latn-ijekavsk
serbian-latn-me
serbian-latn-xk
serbianc

shambala
shona
sichuanyi
sicilian
silesian
sinhala
slovak
slovene
slovenian
soga
somali
sorani
southernsotho
southndebele
spanish
spanish-mexico
spanish-mx
standardmoroccantamazight
swahili
swati
swedish
swissgerman
swisshighgerman
syriac
tachelhit
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
taita
tamil
taroko
tasawaq
telugu
teso
thai
tibetan
tigre
tigrinya
tokpisin

tongan
tsonga
tswana
turkish
turkmen
tyap
ukenglish
ukrainian
uppersorbian
urdu
usenglish
uyghur
uzbek
uzbek-arab
uzbek-arabic
uzbek-cyrillic
uzbek-cyrl
uzbek-latin
uzbek-latn
vai
vai-latin
vai-latn
vai-vai
vai-vaii
venda
vietnamese
volapuk
vunjo
walser
welsh
westernfrisian
wolaytta
wolof
yangben
yiddish
yoruba
zarma
zulu

Many locale templates are quite useable, provided captions and dates are not required (which is a very frequent case, particularly in ancient languages). So, they will be included in the default babel distribution. This can serve to encourage contributions, too. A warning will remember they are 'bare minumum locales'. The locales are currently the following:

akkadian
ancientegyptian
balinese

divehi
egyptianarabic[13]
ladino

lombard
navajo
newari

oldnorse
phoenician
southernaltai

**Modifying and adding values to `ini` files**
New 3.39 ⊕   There is a way to modify the values of `ini` files when they get loaded with `\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use something like `numbers/digits.native=abcdefghij`. Keys may be added, too. Without `import` you may modify the identification keys.
This can be used to create private variants easily. All you need is to import the same `ini` file with a different locale name and different parameters.

---

[13]Masri or Colloquial Egyptian, with tag `arz`, not to be confused with the Standard Arabic as spoken in Egypt, with tag `ar-EG`).

## 1.14 Selecting fonts

New 3.15 Babel provides a high level interface on top of `fontspec` to select fonts. There is no need to load fontspec explicitly – babel does it for you with the first `\babelfont`.[14]

`\babelfont` [⟨*language-list*⟩]{⟨*font-family*⟩}[⟨*font-options*⟩]{⟨*font-name*⟩}

**NOTE** See the note in the previous section about some issues in specific languages.

The main purpose of `\babelfont` is to define at once in a multilingual document the fonts required by the different languages, with their corresponding language systems (script and language). So, if you load, say, 4 languages, `\babelfont{rm}{FreeSerif}` defines 4 fonts (with their variants, of course), which are switched with the language by babel. It is a tool to make things easier and transparent to the user.
Here *font-family* is rm, sf or tt (or newly defined ones, as explained below), and *font-name* is the same as in fontspec and the like.
If no language is given, then it is considered the default font for the family, activated when a language is selected.
On the other hand, if there is one or more languages in the optional argument, the font will be assigned to them, overriding the default one. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, *devanagari). With this optional argument, the font is *not* yet defined, but just predeclared. This means you may define as many fonts as you want 'just in case', because if the language is never selected, the corresponding `\babelfont` declaration is just ignored.
Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need *font-options*, which is the same as in fontspec, but you may add further key/value pairs if necessary.

**EXAMPLE** Usage in most cases is very simple. Let us assume you are setting up a document in Swedish, with some words in Hebrew, with a font suited for both languages.

LUATEX/XETEX
```
\documentclass{article}

\usepackage[swedish, bidi=default]{babel}

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

If on the other hand you have to resort to different fonts, you can replace the red line above with, say:

LUATEX/XETEX
```
\babelfont{rm}{Iwona}
\babelfont[hebrew]{rm}{FreeSerif}
```

`\babelfont` can be used to implicitly define a new font family. Just write its name instead of rm, sf or tt. This is the preferred way to select fonts in addition to the three basic families.

**EXAMPLE** Here is how to do it:

---

[14]See also the package combofont for a complementary approach.

```
\babelfont{kai}{FandolKai}
```

Now, \kaifamily and \kaidefault, as well as \textkai are at your disposal.

**NOTE** You may load fontspec explicitly. For example:

```
\usepackage{fontspec}
\newfontscript{Devanagari}{deva}
\babelfont[hindi]{rm}{Shobhika}
```

This makes sure the OpenType script for Devanagari is deva and not dev2, in case it is not detected correctly. You may also pass some options to fontspec: with silent, the warnings about unavailable scripts or languages are not shown (they are only really useful when the document format is being set up).

**NOTE** Directionality is a property affecting margins, indentation, column order, etc., not just text. Therefore, it is under the direct control of the language, which applies both the script and the direction to the text. As a consequence, there is no need to set Script when declaring a font with \babelfont (nor Language). In fact, it is even discouraged.

**NOTE** \fontspec is not touched at all, only the preset font families (rm, sf, tt, and the like). If a language is switched when an *ad hoc* font is active, or you select the font with this command, neither the script nor the language is passed. You must add them by hand. This is by design, for several reasons —for example, each font has its own set of features and a generic setting for several of them can be problematic, and also preserving a "lower-level" font selection is useful.

**NOTE** The keys Language and Script just pass these values to the *font*, and do *not* set the script for the *language* (and therefore the writing direction). In other words, the ini file or \babelprovide provides default values for \babelfont if omitted, but the opposite is not true. See the note above for the reasons of this behavior.

**WARNING** Using \set*xxxx*font and \babelfont at the same time is discouraged, but very often works as expected. However, be aware with \set*xxxx*font the language system will not be set by babel and should be set with fontspec if necessary.

**TROUBLESHOOTING** *Package babel Info: The following fonts are not babel standard families.*

**This is *not* an error.** babel assumes that if you are using \babelfont for a family, very likely you want to define the rest of them. If you don't, you can find some inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used.

Actually, there is no real need to use \babelfont in a monolingual document, if you set the language system in \setmainfont (or not, depending on what you want).

As the message explains, *there is nothing intrinsically wrong* with not defining all the families. In fact, there is nothing intrinsically wrong with not using \babelfont at all. But you must be aware that this may lead to some problems.

**NOTE** \babelfont is a high level interface to fontspec, and therefore in xetex you can apply Mappings. For example, there is a set of transliterations for Brahmic scripts by Davis M. Jones. After installing them in you distribution, just set the map as you would do with fontspec.

## 1.15 Modifying a language

Modifying the behavior of a language (say, the chapter "caption"), is sometimes necessary, but not always trivial. In the case of caption names a specific macro is provided, because this is perhaps the most frequent change:

\setlocalecaption {⟨*language-name*⟩}{⟨*caption-name*⟩}{⟨*string*⟩}

New 3.51 ⊕ Here *caption-name* is the name as string without the trailing name. An example, which also shows caption names are often a stylistic choice, is:

```
\setlocalecaption{english}{contents}{Table of Contents}
```

This works not only with existing caption names, because it also serves to define new ones by setting the *caption-name* to the name of your choice (name will be postpended). Captions so defined or redefined behave with the 'new way' described in the following note.

**NOTE** There are a few alternative methods:

- With data import'ed from ini files, you can modify the values of specific keys, like:

```
\babelprovide[import, captions/listtable = Lista de tablas]{spanish}
```

(In this particular case, instead of the captions group you may need to modify the captions.licr one.)

- The 'old way', still valid for many languages, to redefine a caption is the following:

```
\addto\captionsenglish{%
  \renewcommand\contentsname{Foo}%
}
```

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do so. This redefinition is not activated until the language is selected.

- The 'new way', which is found in bulgarian, azerbaijani, spanish, french, turkish, icelandic, vietnamese and a few more, as well as in languages created with \babelprovide and its key import, is:

```
\renewcommand\spanishchaptername{Foo}
```

This redefinition is immediate.

**NOTE** Do *not* redefine a caption in the following way:

```
\AtBeginDocument{\renewcommand\contentsname{Foo}}
```

The changes may be discarded with a language selector, and the original value restored.

Macros to be run when a language is selected can be add to \extras⟨*lang*⟩:

```
\addto\extrasrussian{\mymacro}
```

There is a counterpart for code to be run when a language is unselected: \noextras⟨*lang*⟩.

**NOTE** These macros (\captions⟨*lang*⟩, \extras⟨*lang*⟩) may be redefined, but *must not* be used as such – they just pass information to babel, which executes them in the proper context.

Another way to modify a language loaded as a package or class option is by means of \babelprovide, described below in depth. So, something like:

```
\usepackage[danish]{babel}
\babelprovide[captions=da, hyphenrules=nohyphenation]{danish}
```

first loads `danish.ldf`, and then redefines the captions for `danish` (as provided by the `ini` file) and prevents hyphenation. The rest of the language definitions are not touched. Without the optional argument it just loads some aditional tools if provided by the `ini` file, like extra counters.

\BabelUppercaseMapping {⟨*locale-name*⟩}{⟨*codepoint*⟩}{⟨*output*⟩}
\BabelLowercaseMapping {⟨*locale-name*⟩}{⟨*codepoint*⟩}{⟨*output*⟩}

New 3.90 ⊕  These macros are devised as a high level interface for declaring case mappings, based on the locale name as declared by or with babel. They are the equivalent of `\DeclareUppercaseMapping` and `\DeclareLowercaseMapping` (see usrguide.pdf). The purpose is twofold: (1) a user-friendly way to declare them, because often BCP 47 tags are not known (and sometimes can be complex); (2) if for some reason the tag changes (eg, you decide to tag `english` as `en-001` instead of `en-US`), the new mappings will be still assigned to that language.

**EXAMPLE**  For Classical Latin (no need to know the tag is `la-x-classic`):

```
\BabelUppercaseMapping{classicallatin}{`u}{V}
```

**NOTE**  There are still some rough edges when declaring a mapping with the x extension, or when two babel languages share the same BCP 47 tag. These issues are expected be sorted out in future releases.

### 1.16 Creating a language

New 3.10  And what if there is no style for your language or none fits your needs? You may then define quickly a language with the help of the following macro in the preamble (which may be used to modify an existing language, too, as explained in the previous subsection).

\babelprovide [⟨*options*⟩]{⟨*language-name*⟩}

If the language ⟨*language-name*⟩ has not been loaded as class or package option and there are no ⟨*options*⟩, it creates an "empty" one with some defaults in its internal structure: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3. In either case, caption, date and language system are not defined.
If no `ini` file is imported with `import`, ⟨*language-name*⟩ is still relevant because in such a case the hyphenation and like breaking rules (including those for South East Asian and CJK) are based on it as provided in the `ini` file corresponding to that name; the same applies to OpenType language and script.
Conveniently, some options allow to fill the language, and babel warns you about what to do if there is a missing string. Very likely you will find alerts like that in the `log` file:

```
Package babel Warning: \chaptername not set for 'mylang'. Please,
(babel)                define it after the language has been loaded
(babel)                (typically in the preamble) with:
(babel)                \setlocalecaption{mylang}{chapter}{..}
(babel)                Reported on input line 26.
```

In most cases, you will only need to define a few macros. Note languages loaded on the fly are not yet available in the preamble.

**EXAMPLE**  If you need a language named `arhinish`:

```
    \usepackage[danish]{babel}
    \babelprovide{arhinish}
    \setlocalecaption{arhinish}{chapter}{Chapitula}
    \setlocalecaption{arhinish}{refname}{Refirenke}
    \renewcommand\arhinishhyphenmins{22}
```

**EXAMPLE**  Sometimes treating the IPA as a language makes sense:

```
    \documentclass{article}
    \usepackage[english]{babel}
    \babelprovide{ipa}
    \babelfont[ipa]{rm}{DejaVu Sans}
    \begin{document}
    Blah \foreignlanguage{ipa}{ɔːlˈðəʊ} Blah.
    \end{document}
```

Then you can define shorthands, transforms (with luatex), and so on.

**EXAMPLE**  Locales with names based on BCP 47 codes can be created with something like:

```
    \babelprovide[import=en-US]{enUS}
```

Note, however, mixing ways to identify locales can lead to problems. For example, is yi the name of the language spoken by the Yi people or is it the code for Yiddish?

The main language is not changed (danish in this example). So, you must add \selectlanguage{arhinish} or other selectors where necessary.
If the language has been loaded as an argument in \documentclass or \usepackage, then \babelprovide redefines the requested data.

import= ⟨*language-tag*⟩

New 3.13   Imports data from an ini file, including captions and date (also line breaking rules in newly defined languages). For example:

```
  \babelprovide[import=hu]{hungarian}
```

Unicode engines load the UTF-8 variants, while 8-bit engines load the LICR (ie, with macros like \' or \ss) ones.
New 3.23   It may be used without a value, and that is often the recommended option. In such a case, the ini file set in the corresponding babel-<language>.tex (where <language> is the last argument in \babelprovide) is imported. See the list of recognized languages above. So, the previous example is best written as:

```
  \babelprovide[import]{hungarian}
```

There are about 250 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages may show a warning about the current lack of suitability of some features.
Besides \today, this option defines an additional command for dates: \<language>date, which takes three arguments, namely, year, month and day numbers. In fact, \today calls \<language>today, which in turn calls \<language>date{\the\year}{\the\month}{\the\day}. New 3.44 ⊕   More convenient is usually \localedate, with prints the date for the current locale.

**captions=** *⟨language-tag⟩*

Loads only the strings. For example:

```
\babelprovide[captions=hu]{hungarian}
```

**hyphenrules=** *⟨language-list⟩*

With this option, with a space-separated list of hyphenation rules, babel assigns to the language the first valid hyphenation rules in the list. For example:

```
\babelprovide[hyphenrules=chavacano spanish italian]{chavacano}
```

If none of the listed hyphenrules exist, the default behavior applies. Note in this example we set chavacano as first option, which can seem redundant, but without it, it would select spanish even if chavacano exists.

A special value is +, which allocates a new language (in the TeX sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with luatex, because you can add some patterns with \babelpatterns, as for example:

```
\babelprovide[hyphenrules=+]{neo}
\babelpatterns[neo]{a1 e1 i1 o1 u1}
```

In other engines it just suppresses hyphenation (because the pattern list is empty). New 3.58 ⊕ Another special value is unhyphenated, which is an alternative to justification=unhyphenated.

**main** This valueless option makes the language the main one (thus overriding that set when babel is loaded). Only in newly defined languages.

**EXAMPLE** Let's assume your document (xetex or luatex) is mainly in Polytonic Greek with but with some sections in Italian. Then, the first attempt should be:

```
\usepackage[italian, greek.polutoniko]{babel}
```

But if, say, accents in Greek are not shown correctly, you can try

```
\usepackage[italian, polytonicgreek, provide=*]{babel}
```

Remerber there is an alternative syntax for the latter:

```
\usepackage[italian]{babel}
\babelprovide[import, main]{polytonicgreek}
```

Finally, also remember you might not need to load italian at all if there are only a few word in this language (see 1.3).

**script=** *⟨script-name⟩*

New 3.15 Sets the script name to be used by fontspec (eg, Devanagari). Overrides the value in the ini file. If fontspec does not define it, then babel sets its tag to that provided by the ini file. This value is particularly important because it sets the writing direction, so you must use it if for some reason the default value is wrong.

**language=** ⟨*language-name*⟩

New 3.15  Sets the language name to be used by fontspec (eg, Hindi). Overrides the value in the ini file. If fontspec does not define it, then babel sets its tag to that provided by the ini file. Not so important, but sometimes still relevant.

**alph=** ⟨*counter-name*⟩

Assigns to \alph that counter. See the next section.

**Alph=** ⟨*counter-name*⟩

Same for \Alph.

**casing** ⟨*rules*⟩

New 3.90 ⊕  Selects the casing rules in a few languages. They are (from interface3.pdf):

**Armenian** yiwn, which maps U+0587 to capital ech and yiwn on uppercasing.

**German** eszett, which maps the lowercase *Eszett* to a *großes Eszett*.

**Greek** iota, which converts the *ypogegrammeni* (subscript muted iota) to capital iota when uppercasing.

**EXAMPLE**  For the latter:

```
\usepackage[greek]{babel}
\babelprovide[casing=iota]{greek}
```

<p align="center">*     *     *</p>

A few options (only luatex) set some properties of the writing system used by the language. These properties are *always* applied to the script, no matter which language is active. Although somewhat inconsistent, this makes setting a language up easier in most typical cases.

**onchar=** ids | fonts | letters

New 3.38 ⊕  This option is much like an 'event' called when a character belonging to the script of this locale is found (as its name implies, it acts on characters, not on spaces). There are currently two 'actions', which can be used at the same time (separated by a space): with ids the \language and the \localeid are set to the values of this locale; with fonts, the fonts are changed to those of this locale (as set with \babelfont). Characters can be added or modified with \babelcharproperty.
New 3.81 ⊕  Option letters restricts the 'actions' to letters, in the TeX sense (i. e., with catcode 11). Digits and punctuation are then considered part of current locale (as set by a selector). This option is useful when the main script in non-Latin and there is a secondary one whose script is Latin.

**NOTE**  An alternative approach with luatex and Harfbuzz is the font option RawFeature={multiscript=auto}. It does not switch the babel language and therefore the line breaking rules, but in many cases it can be enough.

**NOTE**  There is no general rule to set the font for a punctuation mark, because it is a semantic decision and not a typographical one. Consider the following sentence: "یک, دو, and سه are Persian numbers". In this case the punctuation font must be the English one, even if the commas are surrounded by non-Latin letters. Quotation marks, parenthesis, etc., are even more complex. Several criteria are possible, like the main language (the default in babel), the first letter in the paragraph, or the surrounding letters, among others, but even so manual switching can be still necessary.

**intraspace=** ⟨*base*⟩ ⟨*shrink*⟩ ⟨*stretch*⟩

Sets the interword space for the writing system of the language, in em units (so, 0 .1 0 is 0em plus .1em). Like \spaceskip, the em unit applied is that of the current text (more precisely, the previous glyph). Currently used only in Southeast Asian scrips, like Thai, and CJK.

**intrapenalty=** ⟨*penalty*⟩

Sets the interword penalty for the writing system of this language. Currently used only in Southeast Asian scrips, like Thai. Ignored if 0 (which is the default value).

**transforms=** ⟨*transform-list*⟩

See section 1.21.

**justification=** unhyphenated | kashida | elongated | padding

New 3.59 ⊕   There are currently 4 options. Note they are language dependent, so that they will not be applied to other languages.
The first one (unhyphenated) activates a line breaking mode that allows spaces to be stretched to arbitrary amounts. Although for European standards the result may look odd, in some writing systems, like Malayalam and other Indic scripts, this has been the customary (although not always the desired) practice. Because of that, no locale sets currently this mode by default (Amharic is an exception). Unlike \sloppy, the \hfuzz and the \vfuzz are not changed, because this line breaking mode is not really 'sloppy' (in other words, overfull boxes are reported as usual).
The second and the third are for the Arabic script. It sets the linebreaking and justification method, which can be based on the the ARABIC TATWEEL character or in the 'justification alternatives' OpenType table (jalt). For an explanation see the babel site.
New 3.81 ⊕   The option padding has been devised primarily for Tibetan. It's still somewhat experimental. Again, there is an explanation in the babel site.

**linebreaking=** New 3.59 ⊕   Just a synonymous for justification. Depending on the language, this name can make more sense.

> **NOTE** (1) If you need shorthands, you can define them with \useshorthands and \defineshorthand as described above. (2) Captions and \today are "ensured" with \babelensure (this is the default in ini-based languages).

### 1.17   Digits and counters

New 3.20   About thirty ini files define a field named digits.native. When it is present, two macros are created: \<language>digits and \<language>counter (only xetex and luatex). With the first, a string of 'Latin' digits are converted to the native digits of that language; the second takes a counter name as argument. With the option maparabic in \babelprovide, \arabic is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering, and note as well dates do not rely on \arabic.)
For example:

```
\babelprovide[import]{telugu}
  % Or also, if you want:
  % \babelprovide[import, maparabic]{telugu}
\babelfont{rm}{Gautami} % With luatex, better with Harfbuzz
\begin{document}
\teludigits{1234}
\telugucounter{section}
\end{document}
```

Languages providing native digits in all or some variants are:

| | | | | |
|---|---|---|---|---|
| Arabic | Dzongkha | Lao | Odia | Thai |
| Assamese | Gujarati | Maithili | Pashto | Tibetan |
| Bangla | Haryanvi | Malayalam | Persian | Urdu |
| Bhojpuri | Hindi | Manipuri | Punjabi | Uyghur |
| Bodo | Hmong Njua | Marathi | Rajasthani | Uzbek |
| Burmese | Kannada | Mazanderani | Sanskrit | Vai |
| Cantonese | Kashmiri | Nepali | Santali | |
| Central Kurdish | Khmer | Northern | Sindhi | |
| Chinese | Konkani | Kurdish | Tamil | |
| Dogri | Korean | Northern Luri | Telugu | |

New 3.30   With luatex there is an alternative approach for mapping digits, namely, `mapdigits`. Conversion is based on the language and it is applied to the typeset text (not math, PDF bookmarks, etc.) before bidi and fonts are processed (ie, to the node list as generated by the TeX code). This means the local digits have the correct bidirectional behavior (unlike `Numbers=Arabic` in fontspec, which is deprecated).

**NOTE**  With xetex you can use the option `Mapping` when defining a font.

`\localenumeral` {⟨*style*⟩}{⟨*number*⟩}
`\localecounter` {⟨*style*⟩}{⟨*counter*⟩}

New 3.41 ⊕   Many 'ini' locale files provide information about non-positional numerical systems, based on those predefined in CSS. They only work with xetex and luatex and are fully expandable (even inside an unprotected `\edef`). Currently, they are limited to numbers below 10000.
There are several ways to use them (for the availabe styles in each language, see the list below):

- `\localenumeral{⟨style⟩}{⟨number⟩}`, like `\localenumeral{abjad}{15}`

- `\localecounter{⟨style⟩}{⟨counter⟩}`, like `\localecounter{lower}{section}`

- In `\babelprovide`, as an argument to the keys `alph` and `Alph`, which redefine what `\alph` and `\Alph` print. For example:

```
\babelprovide[alph=alphabetic]{thai}
```

The styles are:

**Ancient Greek** `lower.ancient`, `upper.ancient`
**Amharic** `afar`, `agaw`, `ari`, `blin`, `dizi`, `gedeo`, `gumuz`, `hadiyya`, `harari`, `kaffa`, `kebena`, `kembata`, `konso`, `kunama`, `meen`, `oromo`, `saho`, `sidama`, `silti`, `tigre`, `wolaita`, `yemsa`
**Arabic** `abjad`, `maghrebi.abjad`
**Armenian** `lower.letter`, `upper.letter`
**Belarusan, Bulgarian, Church Slavic, Macedonian, Serbian** `lower`, `upper`
**Bangla** `alphabetic`
**Central Kurdish** `alphabetic`
**Chinese** `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`
**Church Slavic (Glagolitic)** `letters`
**Coptic** `epact`, `lower.letters`
**French** `date.day` (mainly for internal use).
**Georgian** `letters`
**Greek** `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with keraia)
**Hebrew** `letters` ( New 3.93 ⊕  if the language is loaded explicitly, also `letters.plain`, `letters.gershayim`, `letters.final`)
**Hindi** `alphabetic`

**Italian** `lower.legal`, `upper.legal`

**Japanese** `hiragana`, `hiragana.iroha`, `katakana`, `katakana.iroha`, `circled.katakana`, `informal`, `formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

**Khmer** `consonant`

**Korean** `consonant`, `syllable`, `hanja.informal`, `hanja.formal`, `hangul.formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

**Marathi** `alphabetic`

**Persian** `abjad`, `alphabetic`

**Russian** `lower`, `lower.full`, `upper`, `upper.full`

**Syriac** `letters`

**Tamil** `ancient`

**Thai** `alphabetic`

**Ukrainian** `lower`, `lower.full`, `upper`, `upper.full`

New 3.45 ⊕  In addition, native digits (in languages defining them) may be printed with the numeral style `digits`.

## 1.18  Dates

New 3.45 ⊕  When the data is taken from an `ini` file, you may print the date corresponding to the Gregorian calendar and other lunisolar systems with the following command.

`\localedate` [⟨*calendar=.., variant=.., convert*⟩]{⟨*year*⟩}{⟨*month*⟩}{⟨*day*⟩}

By default the calendar is the Gregorian, but an `ini` file may define strings for other calendars: `am` (`ethiopic`), `ar` and `ar-*` (`islamic`), `cop` (`coptic`), `fa` (`islamic`, `persian`), `he` (`hebrew`), `hi` (`indian`), `th` (`buddhist`). In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with `calendar=hebrew` and `calendar=coptic`). However, with the option `convert` it's converted (using internally the following command).

Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like *30. Çileya Pêşîn 2019*, but with `variant=izafa` it prints *31'ê Çileya Pêşînê 2019*.

The default calendar for a language can be set in `\babelprovide`, with the key `calendar` (an empty value is the same as `gregorian`). In this case, `\today` always converts the date. Variants are preceded by a dot, so that `calendar = .genitive` in `serbian` `\today` selects the date in this variant (more explicitly is `gregorian.genitive`).

EXAMPLE  By default, `thai` prints the date with `\today` in the Buddhist calendar, but if you need a date in the Gregorian one, write:

```
\localedate[calendar=gregorian]{\year}{\month}{\day}
```

(Remember `\year`, `\month` and `\day` is the current Gregorian date, so no conversion is necessary.)

EXAMPLE  On the other hand (and following the CLDR), the preferred calendar in most locales for Arabic is `gregorian` (in `ar-SA` is `islamic-umalqura`), so to set `islamic-civil` as the default one:

```
\babelprovide[import, calendar=islamic-civil]{arabic}
```

**\babelcalendar** [⟨*date*⟩]{⟨*calendar*⟩}{⟨*year-macro*⟩}⟨*month-macro*⟩⟨*day-macro*⟩

New 3.76 ⊕ Although calendars aren't the primary concern of babel, the package should be able to, at least, generate correctly the current date in the way users would expect in their own culture. Currently, \localedate can print dates in a few calendars (provided the ini locale file has been imported), but year, month and day had to be entered by hand, which is inconvenient. With this macro, the current date is converted and stored in the three last arguments, which must be macros. Allowed calendars are:

| | | |
|---|---|---|
| buddhist | coptic | islamic-civil |
| chinese | ethiopic | islamic-umalqura |
| New 3.94 ⊕ | hebrew | persian |

The optional argument converts the given date, in the form '⟨*year*⟩-⟨*month*⟩-⟨*day*⟩', although for practical reasons most calendars accept only a restricted range of years. Please, refer to the page on the news for 3.76 in the babel site for further details.

### 1.19 Accessing language info

**\languagename** The control sequence \languagename contains the name of the current language.

WARNING Due to some internal inconsistencies in catcodes, it should *not* be used to test its value. Use iflang, by Heiko Oberdiek.

**\iflanguage** {⟨*language*⟩}{⟨*true*⟩}{⟨*false*⟩}

If more than one language is used, it might be necessary to know which language is active at a specific time. This can be checked by a call to \iflanguage, but note here "language" is used in the TeX sense, as a set of hyphenation patterns, and *not* as its babel name. This macro takes three arguments. The first argument is the name of a language; the second and third arguments are the actions to take if the result of the test is true or false respectively.

**\localeinfo** *{⟨*field*⟩}

New 3.38 ⊕ If an ini file has been loaded for the current language, you may access the information stored in it. This macro is fully expandable, and the available fields are:

name.english as provided by the Unicode CLDR.
tag.ini is the tag of the ini file (the way this file is identified in its name).
tag.bcp47 is the full BCP 47 tag (see the warning below). This is the value to be used for the 'real' provided tag (babel may fill other fields if they are considered necessary).
language.tag.bcp47 is the BCP 47 language tag.
tag.opentype is the tag used by OpenType (usually, but not always, the same as BCP 47).
script.name , as provided by the Unicode CLDR.
script.tag.bcp47 is the BCP 47 tag of the script used by this locale. This is a required field for the fonts to be correctly set up, and therefore it should be always defined.
script.tag.opentype is the tag used by OpenType (usually, but not always, the same as BCP 47).
region.tag.bcp47 is the BCP 47 tag of the region or territory. Defined only if the locale loaded actually contains it (eg, es-MX does, but es doesn't), which is how locales behave in the CLDR. New 3.75 ⊕
variant.tag.bcp47 is the BCP 47 tag of the variant (in the BCP 47 sense, like 1901 for German). New 3.75 ⊕
extension.⟨*s*⟩.tag.bcp47 is the BCP 47 value of the extension whose singleton is ⟨*s*⟩ (currently the recognized singletons are x, t and u). The internal syntax can be somewhat complex, and this feature is still somewhat tentative. An example is classicallatin which sets extension.x.tag.bcp47 to classic. New 3.75 ⊕

> **NOTE** Currently, x is used for two separate functions, namely, identifying a babel locale without a BCP47 tag and setting an alternative set of rules for casing.

> **WARNING** New 3.46 ⊕ As of version 3.46 `tag.bcp47` returns the full BCP 47 tag. Formerly it returned just the language subtag, which was clearly counterintuitive.

New 3.75 ⊕ Sometimes, it comes in handy to be able to use `\localeinfo` in an expandable way even if something went wrong (for example, the locale currently active is undefined). For these cases, `localeinfo*` just returns an empty string instead of raising an error. Bear in mind that babel, following the CLDR, may leave the region unset, which means `\getlocaleproperty*`, described below, is the preferred command, so that the existence of a field can be checked before. This also means building a string with the language and the region with `\localeinfo*{language.tab.bcp47}`-`\localeinfo*{region.tab.bcp47}` is not usually a good idea (because of the hyphen).

`\getlocaleproperty` `*{⟨macro⟩}{⟨locale⟩}{⟨property⟩}`

New 3.42 ⊕ The value of any locale property as set by the `ini` files (or added/modified with `\babelprovide`) can be retrieved and stored in a macro with this command. For example, after:

```
\getlocaleproperty\hechap{hebrew}{captions/chapter}
```

the macro `\hechap` will contain the string פרק.
If the key does not exist, the macro is set to `\relax` and an error is raised.
New 3.47 ⊕ With the starred version no error is raised, so that you can take your own actions with undefined properties.

`\localeid` Each language in the babel sense has its own unique numeric identifier, which can be retrieved with `\localeid`.
The `\localeid` is not the same as the `\language` identifier, which refers to a set of hyphenation patters (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are store in an internal macro named `\bbl@languages` (see the code for further details), but note several locales may share a single `\language`, so they are separated concepts. In luatex, the `\localeid` is saved in each node (when it makes sense) as an attribute, too.

`\LocaleForEach` `{⟨code⟩}`

Babel remembers which `ini` files have been loaded. There is a loop named `\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that `\LocaleForEach{\message{ **#1** }}` just shows the loaded `ini`'s.

`ensureinfo=off` New 3.75 ⊕ Previously, `ini` files were loaded only with `\babelprovide` and also when languages are selected if there is a `\babelfont` or they have not been explicitly declared. Now the `ini` files are loaded (and therefore the corresponding data) even if these two conditions are not met (in previous versions you had to enable it with `\BabelEnsureInfo` in the preamble). Because of the way this feature works, problems are very unlikely, but there is switch as a package option to turn the new behavior off (`ensureinfo=off`).

## 1.20 Hyphenation and line breaking

Babel deals with three kinds of line breaking rules: Western, typically the LGC group, South East Asian, like Thai, and CJK, but support depends on the engine: pdftex only deals with the former, xetex also with the second one (although in a limited way), while luatex provides basic rules for the latter, too. With luatex there are also tools for non-standard hyphenation rules, explained in the next section.

`\babelhyphen` `*{⟨type⟩}`

`\babelhyphen` *{⟨*text*⟩}

New 3.9a  It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in TeX are entered as -, and (2) *optional* or *soft hyphens*, which are entered as \-. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in TeX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.

In TeX, - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, "- in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic "hyphens" which can be used by themselves, to define a user shorthand, or even in language files.

- `\babelhyphen{soft}` and `\babelhyphen{hard}` are self explanatory.

- `\babelhyphen{repeat}` inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portuguese and Spanish.

- `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it (even if a space follows).

- `\babelhyphen{empty}` inserts a break opportunity without a hyphen at all.

- `\babelhyphen{⟨text⟩}` is a hard "hyphen" using ⟨*text*⟩ instead. A typical case is `\babelhyphen{/}`.

With all of them, hyphenation in the rest of the word is enabled. If you don't want to enable it, there is a starred counterpart: `\babelhyphen*{soft}` (which in most cases is equivalent to the original \-), `\babelhyphen*{hard}`, etc.

Note hard is also good for isolated prefixes (eg, *anti-*) and nobreak for isolated suffixes (eg, *-ism*), but in both cases `\babelhyphen*{nobreak}` is usually better.

There are also some differences with LaTeX: (1) the character used is that set for the current font, while in LaTeX it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative \hyphenchar is -, like in LaTeX, but it can be changed to another value by redefining \babelnullhyphen; (3) a break after the hyphen is forbidden if preceded by a glue $>0$ pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

`\babelhyphenation` [⟨*language*⟩,⟨*language*⟩,...]{⟨*exceptions*⟩}

New 3.9a  Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Multiple declarations work much like `\hyphenation` (last wins), but language exceptions take precedence over global ones.

It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of \lccodes's done in \extras⟨*lang*⟩ as well as the language-specific encoding (not set in the preamble by default). Multiple \babelhyphenation's are allowed. For example:

```
\babelhyphenation{Wal-hal-la Dar-bhan-ga}
```

Listed words are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.

NOTE  Using \babelhyphenation with Southeast Asian scripts is mostly pointless. But with \babelpatterns (below) you may fine-tune line breaking (only luatex). Even if there are no patterns for the language, you can add at least some typical cases.

**NOTE** Use \babelhyphenation instead of \hyphenation to set hyphenation exceptions in the preamble before any language is explicitly set with a selector. In the preamble the hyphenation rules are not always fully set up and an error can be raised.

\begin{hyphenrules} {⟨*language*⟩} … \end{hyphenrules}

The environment hyphenrules can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select 'nohyphenation', provided that in language.dat the 'language' nohyphenation is defined by loading zerohyph.tex. It deactivates language shorthands, too (but not user shorthands).
Except for these simple uses, hyphenrules is deprecated and otherlanguage* (the starred version) is preferred, because the former does not take into account possible changes in encodings of characters like, say, ' done by some languages (eg, italian, french, ukrainian).

\babelpatterns [⟨*language*⟩,⟨*language*⟩,…]{⟨*patterns*⟩}

New 3.9m *In luatex only,*[15] adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.
It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \lccodes's done in \extras⟨*lang*⟩ as well as the language-specific encoding (not set in the preamble by default). Multiple \babelpatterns's are allowed.
Listed patterns are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.
New 3.31 (Only luatex.) With \babelprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules ( New 3.32 it is disabled in verbatim mode, or more precisely when the hyphenrules are set to nohyphenation). It can be activated alternatively by setting explicitly the intraspace.
New 3.27 Interword spacing for Thai, Lao and Khemer is activated automatically if a language with one of those scripts are loaded with \babelprovide. See the sample on the babel repository. With both Unicode engines, spacing is based on the "current" em unit (the size of the previous char in luatex, and the font size set by the last \selectfont in xetex).

## 1.21 Transforms

Transforms (only luatex) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.[16]
It currently embraces \babelprehyphenation and \babelposthyphenation.
New 3.57 ⊕ Several ini files predefine some transforms. They are activated with the key transforms in \babelprovide, either if the locale is being defined with this macro or the languages has been previouly loaded as a class or package option, as the following example illustrates:

```
\usepackage[hungarian]{babel}
\babelprovide[transforms = digraphs.hyphen]{hungarian}
```

New 3.67 ⊕ Transforms predefined in the ini locale files can be made attribute-dependent, too. When an attribute between parenthesis is inserted subsequent transforms will be assigned to it (up to the list end or another attribute). For example, and provided an attribute called \withsigmafinal has been declared:

---

[15]With luatex exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and babel only provides the most basic tools.

[16]They are similar in concept, but not the same, as those in Unicode. The main inspiration for this feature is the Omega transformation processes.

```
transforms = transliteration.omega (\withsigmafinal) sigma.final
```

This applies `transliteration.omega` always, but `sigma.final` only when
`\withsigmafinal` is set.

Here are the transforms currently predefined. (A few may still require some fine-tuning.
More to follow in future releases.)

| | | |
|---|---|---|
| Arabic | `transliteration.dad` | Applies the transliteration system devised by Yannis Haralambous for dad (simple and TeX-friendly). Not yet complete, but sufficient for most texts. |
| Croatian | `digraphs.ligatures` | Ligatures *DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj*. It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry. |
| Croatian, Czech, Polish, Portuguese, Slovak, Spanish | `hyphen.repeat` | Explicit hyphens behave like `\babelhyphen {repeat}`. |
| Czech, Polish, Slovak | `oneletter.nobreak` | Converts a space after a non-syllabic preposition or conjunction into a non-breaking space. |
| Finnish | `prehyphen.nobreak` | Line breaks just after hyphens prepended to words are prevented, like in "pakastekaapit ja -arkut". |
| Greek | `diaeresis.hyphen` | Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants. |
| Greek | `transliteration.omega` | Although the provided combinations are not the full set, this transform follows the syntax of Omega: = for the circumflex, v for digamma, and so on. For better compatibility with Levy's system, ~ (as 'string') is an alternative to =. ' is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek. |
| Greek | `sigma.final` | The transliteration system above does not convert the sigma at the end of a word (on purpose). This transforms does it. To prevent the conversion (an abbreviation, for example), write `"s`. |
| Hebrew, Yiddish | `transliteration.cj` | A transliteration system based on that devised by Christian Justen for 'cjhebrew'. Final letters are not converted, and the furtive patah is not shifted. |
| Hindi, Sanskrit | `transliteration.hk` | The Harvard-Kyoto system to romanize Devanagari. |
| Hindi, Sanskrit | `punctuation.space` | Inserts a space before the following four characters: *!?:;* . |
| Hungarian | `digraphs.hyphen` | Hyphenates the long digraphs *ccs*, *ddz*, *ggy*, *lly*, *nny*, *ssz*, *tty* and *zzs* as *cs-cs*, *dz-dz*, etc. |

| | | |
|---|---|---|
| Indic scripts | `danda.nobreak` | Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Odia, Tamil, Telugu. |
| Latin | `digraphs.ligatures` | Replaces the groups *ae, AE, oe, OE* with *æ, Æ, œ, Œ*. |
| Latin | `letters.noj` | Replaces *j, J* with *i, I*. |
| Latin | `letters.uv` | Replaces *v, U* with *u, V*. |
| Sanskrit | `transliteration.iast` | The IAST system to romanize Devanagari.[17] |
| Serbian | `transliteration.gajica` | (Note `serbian` with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj. |
| Arabic, Persian | `kashida.plain` | Experimental. A very simple and basic transform for 'plain' Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59. |
| Arabic, Persian | `kashida.base` | Experimental New 3.94 ⊕ . Much like the previous one, but with diacritics stacked in the actual base character and not the kashida extension. With evenly inserted tatweels results are better. |

`\babelposthyphenation` [⟨*options*⟩]{⟨*hyphenrules-name*⟩}{⟨*lua-pattern*⟩}{⟨*replacement*⟩}

New 3.37-3.39 ⊕ *With luatex* it is possible to define non-standard hyphenation rules, like f-f → ff-f, repeated hyphens, ranked ruled (or more precisely, 'penalized' hyphenation points), and so on. A few rules are currently provided (see above), but they can be defined as shown in the following example, where {1} is the first captured char (between () in the pattern):

```
\babelposthyphenation{german}{([fmtrp]) | {1}}
{
  { no = {1}, pre = {1}{1}- }, % Replace first char with disc
  remove,                     % Remove automatic disc (2nd node)
  {}                          % Keep last char, untouched
}
```

In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads ([ïǘ]), the replacement could be {1|ïǘ|íú}, which maps ï to í, and ǘ to ú, so that the diaeresis is removed.

This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`. New 3.85 ⊕ Another option is `label`, which takes a value similar to those in `\babelprovide` key `transforms` (in fact, the latter just applies this option). This label can be used to turn on and off transforms with a higher level interface, by means of `\enablelocaletransform` and `\disablelocaletransform` (see below).
New 3.85 ⊕ When used in conjunction with `label`, this key makes a transform font dependent. As an example, the rules for Arabic kashida can differ depending on the font design. The value consists in a list of space-separated font tags:

```
\babelprehyphenation[label=transform.name, fonts=rm sf]{..}{..}
```

Tags can adopt two forms: a family, such as `rm` or `tt`, or the set family/series/shape. If a font matches one of these conditions, the transform is enabled. The second tag in `rm rm/n/it` is redundant. There are no wildcards; so, for italics you may want to write something like `sf/m/it sf/b/it`.

Transforms set for specific fonts (at least once in any language) are always reset with a font selector.

In \babelprovide, transform labels can be tagged before its name, with a list separated with colons, like:

```
transforms = rm:sf:transform.name
```

New 3.67 ⊕  With the optional argument you can associate a user defined transform to an attribute, so that it's active only when it's set (currently its attribute value is ignored). With this mechanism transforms can be set or unset even in the middle of paragraphs, and applied to single words. To define, set and unset the attribute, the LaTeX kernel provides the macros \newattribute, \setattribute and \unsetattribute. The following example shows how to use it, provided an attribute named \latinnoj has been declared:

```
\babelprehyphenation[attribute=\latinnoj]{latin}{ J }{ string = I }
```

See the babel site for a more detailed description and some examples. It also describes a few additional replacement types (string, penalty).

Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account).

You are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg.

\babelprehyphenation  [⟨options⟩]{⟨locale-name⟩}{⟨lua-pattern⟩}{⟨replacement⟩}

New 3.44-3-52 ⊕  It is similar to the latter, but (as its name implies) applied before hyphenation, which is particularly useful in transliterations. There are other differences: (1) the first argument is the locale instead of the name of the hyphenation patterns; (2) in the search patterns = has no special meaning, while | stands for an ordinary space; (3) in the replacement, discretionaries are not accepted.

See the description above for the optional argument.

This feature is activated with the first \babelposthyphenation or \babelprehyphenation.

**EXAMPLE**  You can replace a character (or series of them) by another character (or series of them). Thus, to enter ž as zh and š as sh in a newly created locale for transliterated Russian:

```
\babelprovide[hyphenrules=+]{russian-latin}   % Create locale
\babelprehyphenation{russian-latin}{([sz])h}  % Create rule
{
  string = {1|sz|šž},
  remove
}
```

**EXAMPLE**  The following rule prevent the word "a" from being at the end of a line:

```
\babelprehyphenation{english}{|a|}
  {}, {},                     % Keep first space and a
  { insert, penalty = 10000 },  % Insert penalty
  {}                          % Keep last space
}
```

**NOTE**  With luatex there is another approach to make text transformations, with the function fonts.handlers.otf.addfeature, which adds new features to an OTF font (substitution and positioning). These features can be made language-dependent, and

babel by default recognizes this setting if the font has been declared with `\babelfont`. The *transforms* mechanism supplements rather than replaces OTF features.

With xetex, where *transforms* are not available, there is still another approach, with font mappings, mainly meant to perform encoding conversions and transliterations. Mappings, however, are linked to fonts, not to languages.

`\enablelocaletransform` {⟨*label*⟩}
`\disablelocaletransform` {⟨*label*⟩}

New 3.85 ⊕ Enables and disables the transform with the given label in the current language.

### 1.22 Support for xetex interchar

New 3.97 ⊕ A few macros are provided to deal with locale dependent inter-character rules (aka 'interchar').

`\babelcharclass` {⟨*locale*⟩}{⟨*name*⟩}{⟨*char-list*⟩}

Declares a new character class, which is assigned to the characters in {⟨*char-list*⟩}, entered either as characters or in macro form (eg, \}). If you need to enter them by their numeric value, use the TeX ^-notation (eg, ^^^^1fa0). Ranges are allowed, with a hyphen (eg, .,;a-zA-Z). If you need the hyphen to be assigned a class, write it at the very beginning of the list.

There are several predefined 'global' classes, namely `default`, `cjkideogram`, `cjkleftpunctuation`, `cjkrightpunctuation`, `boundary`, and `ignore`, which are described in the xetex manual (currently CJK classes are 'empty', but they will be assigned in a future release).

`\babelinterchar` [⟨*options*⟩]{⟨*locale*⟩} {⟨*class-first*⟩}{⟨*class-second*⟩}{⟨*code*⟩}

{⟨*class-first*⟩} and {⟨*class-second*⟩} can be comma separated lists of char classes, and all combinations are defined (so that 2 first classes with 2 second classes, define 4 combinations). In the {⟨*options*⟩} field a key named `label` is available, which allows to enable or to disable the rule with the following two commands. Like prehyphenation transforms in luatex, interchars are not applied if the current hyphenation rules are `nohyphenation`.

`\enablelocaleinterchar` {⟨*label*⟩}
`\disablelocaleinterchar` {⟨*label*⟩}

Enable or disable the interchar rules with the given label for the current language.

**EXAMPLE** Not very useful, but illustrative (taken from the unfortunately obsolete interchar package, by Zou Ho), to colorize the letters 'x' and 'y' (this way to group text is usually not a good idea, however).

```
\usepackage{color}
\babelcharclass{english}{colored}{xy}
\babelinterchar{english}{default, boundary}{colored}{\bgroup\color{red}}
\babelinterchar{english}{colored}{default, boundary}{\egroup}
```

A more realistic example follows, which inserts a thin space between a digit and a percent sign. Note the former is entered as a range, and the latter in command form:

```
\babelcharclass{english}{digit}{0-9}
\babelcharclass{english}{percent}{\%}
\babelinterchar[label=percent]{english}{digit}{percent}{\,}
```

**WARNING** Keep in mind two points: (1) a character can be assigned a single class; this is a limitation in the interchar mechanims that often leads to incompatibilities; (2) since the character classes set with `\babelcharclass` are saved (so that they can be restored), there is a limit in the number of characters in the {⟨*char-list*⟩} (which, however, must be large enough for many uses).

## 1.23 Selection based on BCP 47 tags

New 3.43 ⊕ The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in documents (or parts of documents) generated by external sources, and therefore babel will provide a set of tools to select the locales in different situations, adapted to the particular needs of each case. Currently, babel provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested.

It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in babel. Instead the data is taken from the `ini` files, which means currently about 250 tags are already recognized. Babel performs a simple lookup in the following way: fr-Latn-FR → fr-Latn → fr-FR → fr. Languages with the same resolved name are considered the same. Case is normalized before, so that fr-latn-fr → fr-Latn-FR. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

\babeladjust{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

Currently the locales loaded are based on the `ini` files and decoupled from the main `ldf` files. This is by design, to ensure code generated externally produces the same result regardless of the languages requested in the document, but an option to use the `ldf` instead will be added in a future release, because both options make sense depending on the particular needs of each document (there will be some restrictions, however).

The behaviour is adjusted with `\babeladjust` with the following parameters:

`autoload.bcp47` with values `on` and `off`.

`autoload.bcp47.options`, which are passed to `\babelprovide`; empty by default, but you may add `import` (features defined in the corresponding `babel-...tex` file might not be available).

`autoload.bcp47.prefix`. Although the public name used in selectors is the tag, the internal name will be different and generated by prepending a prefix, which by default is `bcp47-`. You may change it with this key.

New 3.46 ⊕   If an `ldf` file has been loaded, you can enable the corresponding language tags as selector names with:

```
\babeladjust{ bcp47.toname = on }
```

(You can deactivate it with `off`.) So, if `dutch` is one of the package (or class) options, you can write `\selectlanguage{nl}`. Note the language name does not change (in this example is still `dutch`), but you can get it with `\localeinfo` or `\getlocaleproperty`. It must be turned on explicitly for similar reasons to those explained above.

## 1.24   Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.[18]
Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but is was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.[19]

`\ensureascii` {⟨*text*⟩}

New 3.9i   This macro makes sure ⟨*text*⟩ is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine `\TeX` and `\LaTeX` so that they are correctly typeset even with LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, LGI, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.
If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load LY1,LGR, then it is set to LY1, but if you load LY1,T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for "ordinary" text (they are stored in `\BabelNonText`, used in some special cases when no Latin encoding is explicitly set).
The foregoing rules (which are applied "at begin document") cover most of the cases. No assumption is made on characters above 127, which may not follow the LICR conventions – the goal is just to ensure most of the ASCII letters and symbols are the right ones.

## 1.25   Selecting directions

No macros to select the writing direction are provided, either – writing direction is intrinsic to each script and therefore it is best set by the language (which can be a dummy one). Furthermore, there are in fact two right-to-left modes, depending on the language, which differ in the way 'weak' numeric characters are ordered (eg, Arabic %123 *vs* Hebrew 123%).

**WARNING**   The current code for **text** in luatex should be considered essentially stable, but, of course, it is not bug-free and there can be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait. This applies to text; there is a basic support for **graphical** elements, including the `picture` environment (with pict2e) and pfg/tikz. Also, indexes and the like are under

---

[18]The so-called Unicode fonts do not improve the situation either. So, a font suited for Vietnamese is not necessarily suited for, say, the romanization of Indic languages, and the fact it contains glyphs for Modern Greek does not mean it includes them for Classic Greek.

[19]But still defined for backwards compatibility.

study, as well as math (there are progresses in the latter, including amsmath and mathtools too, but for example `gathered` may fail).

An effort is being made to avoid incompatibilities in the future (this one of the reason currently bidi must be explicitly requested as a package option, with a certain bidi model, and also the `layout` options described below).

**WARNING** If characters to be mirrored are shown without changes with luatex, try with the following line:

```
\babeladjust{bidi.mirroring=off}
```

There are some package options controlling bidi writing.

`bidi=`  `default` | `basic` | `basic-r` | `bidi-l` | `bidi-r`

New 3.14  Selects the bidi algorithm to be used. With `default` the bidi mechanism is just activated (by default it is not), but every change must be marked up. In xetex and pdftex this is the only option.

In luatex, `basic-r` provides a simple and fast method for R text, which handles numbers and unmarked L text within an R context many in typical cases. New 3.19  Finally, `basic` supports both L and R text, and it is the preferred method (support for `basic-r` is currently limited). (They are named `basic` mainly because they only consider the intrinsic direction of scripts and weak directionality.)

New 3.29  In xetex, `bidi-r` and `bidi-l` resort to the package bidi (by Vafa Khalighi). Integration is still somewhat tentative, but it mostly works. For RL documents use the former, and for LR ones use the latter.

There are samples on GitHub, under `/required/babel/samples`. See particularly `lua-bidibasic.tex` and `lua-secenum.tex`.

**EXAMPLE**  The following text comes from the Arabic Wikipedia (article about Arabia). Copy-pasting some text from the Wikipedia is a good way to test this feature. Remember `basic` is available in luatex only.

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

وقد عرفت شبه جزيرة العرب طيلة العصر الهيليني (الاغريقي) بـ
Arabia أو Aravia (بالاغريقية Αραβία)، استخدم الرومان ثلاث
بادئات بـ"Arabia" على ثلاث مناطق من شبه الجزيرة العربية، إلا أنها
حقيقةً كانت أكبر مما تعرف عليه اليوم.

\end{document}
```

**EXAMPLE**  With `bidi=basic` *both* L and R text can be mixed without explicit markup (the latter will be only necessary in some special cases where the Unicode algorithm fails). It is used much like `bidi=basic-r`, but with R text inside L text you may want to map the font so that the correct features are in force. This is accomplished with an option in `\babelprovide`, as illustrated:

```
\documentclass{book}

\usepackage[english, bidi=basic]{babel}

\babelprovide[onchar=ids fonts]{arabic}

\babelfont{rm}{Crimson}
\babelfont[*arabic]{rm}{FreeSerif}

\begin{document}

Most Arabic speakers consider the two varieties to be two registers
of one language, although the two registers can be referred to in
Arabic as فصحى العصر \textit{fuṣḥā l-ʿaṣr} (MSA) and
فصحى التراث \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

In this example, and thanks to `onchar=ids fonts`, any Arabic letter (because the language is `arabic`) changes its font to that set for this language (here defined via `*arabic`, because Crimson does not provide Arabic letters).

**NOTE** Boxes are "black boxes". Numbers inside an `\hbox` (for example in a `\ref`) do not know anything about the surrounding chars. So, `\ref{A}-\ref{B}` are not rendered in the visual order A-B, but in the wrong one B-A (because the hyphen does not "see" the digits inside the `\hbox`'es). If you need `\ref` ranges, the best option is to define a dedicated macro like this (to avoid explicit direction changes in the body; here `\texthe` must be defined to select the main language):

```
\newcommand\refrange[2]{\babelsublr{\texthe{\ref{#1}}-\texthe{\ref{#2}}}}
```

In the future a more complete method, reading recursively boxed text, may be added.

---

`layout=` `sectioning` | `counters` | `lists` | `contents` | `footnotes` | `captions` | `columns` | `graphics` | `extras`

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the `bidi` package, which provides its own mechanism to control these elements). You may use several options with a space-separated list, like `layout=counters contents sectioning` (in New 3.85 ⊕ spaces are to be preferred over dots, which was the former syntax). This list will be expanded in future releases. Note not all options are required by all engines.

`sectioning` makes sure the sectioning macros are typeset in the main language, but with the title text in the current language (see below `\BabelPatchSection` for further details).

`counters` required in all engines (except luatex with `bidi=basic`) to reorder section numbers and the like (eg, ⟨*subsection*⟩.⟨*section*⟩); required in xetex and pdftex for counters in general, as well as in luatex with `bidi=default`; required in luatex for numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it can depend on the counter format.

With `counters`, `\arabic` is not only considered L text always (with `\babelsublr`, see below), but also an "isolated" block which does not interact with the surrounding chars. So, while 1.2 in R text is rendered in that order with `bidi=basic` (as a decimal number), in `\arabic{c1}.\arabic{c2}` the visual order is *c2.c1*. Of course, you may always adjust the order by changing the language, if necessary.

New 3.84 ⊕ Since `\thepage` is (indirectly) redefined, `makeindex` will reject many entries as invalid. With `counters*` babel attempts to remove the conflicting macros.

**lists** required in xetex and pdftex, but only in bidirectional (with both R and L paragraphs) documents in luatex.

> **WARNING** As of April 2019 there is a bug with `\parshape` in luatex (a TeX primitive) which makes lists to be horizontally misplaced if they are inside a `\vbox` (like `minipage`) and the current direction is different from the main one. A workaround is to restore the main language before the box and then set the local one inside.

**contents** required in xetex and pdftex; in luatex toc entries are R by default if the main language is R.

**columns** required in xetex and pdftex to reverse the column order (currently only the standard two-column mode); in luatex they are R by default if the main language is R (including multicol).

**footnotes** not required in monolingual documents, but it may be useful in bidirectional documents (with both R and L paragraphs) in all engines; you may use alternatively `\BabelFootnote` described below (what this option does exactly is also explained there).

**captions** is similar to `sectioning`, but for `\caption`; not required in monolingual documents with luatex, but may be required in xetex and pdftex in some styles (support for the latter two engines is still experimental) New 3.18 .

**tabular** required in luatex for R `tabular`, so that the first column is the right one (it has been tested only with simple tables, so expect some readjustments in the future); ignored in pdftex or xetex (which will not support a similar option in the short term). It patches an internal command, so it might be ignored by some packages and classes (or even raise an error). New 3.18 .

**graphics** modifies the `picture` environment so that the whole figure is L but the text is R. It *does not* work with the standard `picture`, and *pict2e* is required. It attempts to do the same for pgf/tikz. Somewhat experimental. New 3.32 .

**extras** is used for miscellaneous readjustments which do not fit into the previous groups. Currently redefines in luatex `\underline` and `\LaTeX2e` New 3.19 .

**EXAMPLE** Typically, in an Arabic document you would need:

```
\usepackage[bidi=basic,
            layout=counters tabular]{babel}
```

**\babelsublr** {⟨*lr-text*⟩}

Digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic` or `bidi=basic-r` and, usually, xetex). This command is provided to set {⟨*lr-text*⟩} in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `rl` counterpart.

Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

```
RTL A ltr text \thechapter{} and still ltr RTL B
```

There are *three* R blocks and *two* L blocks, and the order is *RTL B and still ltr 1 ltr text RTL A*. This is by design to provide the proper behavior in the most usual cases — but if you need to use `\ref` in an L text inside R, the L text must be marked up explictly; for example:

```
RTL A \foreignlanguage{english}{ltr text \thechapter{} and still ltr} RTL B
```

**\localerestoredirs**

New 3.86 ⊕ *LuaTeX*. This command resets the internal text, paragraph and body directions to those of the current locale (if different). Sometimes changing directly these values can be useful for some hacks, and this command helps in restoring the directions to the correct ones. It can be used in > arguments of array, too.

**\BabelPatchSection** {⟨*section-name*⟩}

Mainly for bidi text, but it can be useful in other cases. \BabelPatchSection and the corresponding option layout=sectioning takes a more logical approach (at least in many cases) because it applies the global language to the section format (including the \chaptername in \chapter), while the section text is still the current language. The latter is passed to tocs and marks, too, and with sectioning in layout they both reset the "global" language to the main one, while the text uses the "local" language.

With layout=sectioning all the standard sectioning commands are redefined (it also "isolates" the page number in heads, for a proper bidi behavior), but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

**\BabelFootnote** {⟨*cmd*⟩}{⟨*local-language*⟩}{⟨*before*⟩}{⟨*after*⟩}

New 3.17   Something like:

```
\BabelFootnote{\parsfootnote}{\languagename}{(}{)}
```

defines \parsfootnote so that \parsfootnote{note} is equivalent to:

```
\footnote{(\foreignlanguage{\languagename}{note})}
```

but the footnote itself is typeset in the main language (to unify its direction). In addition, \parsfootnotetext is defined. The option footnotes just does the following:

```
\BabelFootnote{\footnote}{\languagename}{}{}%
\BabelFootnote{\localfootnote}{\languagename}{}{}%
\BabelFootnote{\mainfootnote}{}{}{}
```

(which also redefine \footnotetext and define \localfootnotetext and \mainfootnotetext). If the language argument is empty, then no language is selected inside the argument of the footnote. Note this command is available always in bidi documents, even without layout=footnotes.

**EXAMPLE**  If you want to preserve directionality in footnotes and there are many footnotes entirely in English, you can define:

```
\BabelFootnote{\enfootnote}{english}{}{.}
```

It adds a period outside the English part, so that it is placed at the left in the last line. This means the dot the end of the footnote text should be omitted.

## 1.26   Language attributes

**\languageattribute**

This is a user-level command, to be used in the preamble of a document (after \usepackage[...]{babel}), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

Very often, using a *modifier* in a package option is better.

Several language definition files use their own methods to set options. For example, french uses \frenchsetup, magyar (1.5) uses \magyarOptions; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, \ProsodicMarksOn in latin).

## 1.27 Hooks

New 3.9a   A hook is a piece of code to be executed at certain events. Some hooks are predefined when luatex and xetex are used.

New 3.64 ⊕   This is not the only way to inject code at those points. The events listed below can be used as a hook name in \AddToHook in the form babel/⟨*language-name*⟩/⟨*event-name*⟩ (with * it's applied to all languages), but there is a limitation, because the parameters passed with the babel mechanism are not allowed. The \AddToHook mechanism does *not* replace the current one in 'babel'. Its main advantage is you can reconfigure 'babel' even before loading it. See the example below.

\AddBabelHook   [⟨*lang*⟩]{⟨*name*⟩}{⟨*event*⟩}{⟨*code*⟩}

The same name can be applied to several events. Hooks with a certain {⟨*name*⟩} may be enabled and disabled for all defined events with \EnableBabelHook{⟨*name*⟩}, \DisableBabelHook{⟨*name*⟩}. Names containing the string babel are reserved (they are used, for example, by \useshortands* to add a hook for the event afterextras).

New 3.33   They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

Current events are the following; in some of them you can use one to three TEX parameters (#1, #2, #3), with the meaning given:

adddialect  (language name, dialect name) Used by luababel.def to load the patterns if not preloaded.

patterns  (language name, language with encoding) Executed just after the \language has been set. The second argument has the patterns name actually selected (in the form of either lang:ENC or lang).

hyphenation  (language name, language with encoding) Executed locally just before exceptions given in \babelhyphenation are actually set.

defaultcommands  Used (locally) in \StartBabelCommands.

encodedcommands  (input, font encodings) Used (locally) in \StartBabelCommands. Both xetex and luatex make sure the encoded text is read correctly.

stopcommands  Used to reset the above, if necessary.

write  This event comes just after the switching commands are written to the aux file.

beforeextras  Just before executing \extras⟨*language*⟩. This event and the next one should not contain language-dependent code (for that, add it to \extras⟨*language*⟩).

afterextras  Just after executing \extras⟨*language*⟩. For example, the following deactivates shorthands in all languages:

```
\AddBabelHook{noshort}{afterextras}{\languageshorthands{none}}
```

stringprocess  Instead of a parameter, you can manipulate the macro \BabelString containing the string to be defined with \SetString. For example, to use an expanded version of the string in the definition, write:

```
\AddBabelHook{myhook}{stringprocess}{%
  \protected@edef\BabelString{\BabelString}}
```

initiateactive  (char as active, char as other, original char) New 3.9i   Executed just after a shorthand has been 'initiated'. The three parameters are the same character with different catcodes: active, other (\string'ed) and the original one.

afterreset  New 3.9i  Executed when selecting a language just after `\originalTeX` is
run and reset to its base value, before executing `\captions`⟨*language*⟩ and
`\date`⟨*language*⟩.

begindocument  New 3.88 ⊕  Executed before the code written by ldf files with
`\AtBeginDocument`. The optional argument with the language in this particular case is
the language that wrote the code. The special value / means 'return to the core babel
definitions' (in other words, what follows hasn't been written by any language).

Four events are used in `hyphen.cfg`, which are handled in a quite different way for
efficiency reasons – unlike the precedent ones, they only have a single hook and replace a
default definition.

everylanguage  (language) Executed before every language patterns are loaded.
loadkernel  (file) By default just defines a few basic commands. It can be used to define
different versions of them or to load a file.
loadpatterns  (patterns file) Loads the patterns file. Used by `luababel.def`.
loadexceptions  (exceptions file) Loads the exceptions file. Used by `luababel.def`.

**EXAMPLE**  The generic unlocalized LaTeX hooks are predefined, so that you can write:

```
\AddToHook{babel/*/afterextras}{\frenchspacing}
```

which is executed always after the extras for the language being selected (and just
before the non-localized hooks defined with `\AddBabelHook`).

In addition, locale-specific hooks in the form babel/⟨*language-name*⟩/⟨*event-name*⟩ are
*recognized* (executed just before the localized babel hooks), but they are *not predefined*.
You have to do it yourself. For example, to set `\frenchspacing` only in bengali:

```
\ActivateGenericHook{babel/bengali/afterextras}
\AddToHook{babel/bengali/afterextras}{\frenchspacing}
```

\BabelContentsFiles  New 3.9a  This macro contains a list of "toc" types requiring a command to switch the
language. Its default value is toc,lof,lot, but you may redefine it with `\renewcommand`
(it's up to you to make sure no toc type is duplicated).

## 1.28   Languages supported by babel with ldf files

In the following table most of the languages supported by babel with and `.ldf` file are
listed, together with the names of the option which you can load babel with for each
language. Note this list is open and the current options may be different. It does not
include ini files (see below). Except in a few cases (eg, ngerman, serbianc, acadian)
names are those of the Unicode CLDR (or based on them).

**Afrikaans**  afrikaans
**Azerbaijani**  azerbaijani
**Basque**  basque
**Breton**  breton
**Bulgarian**  bulgarian
**Catalan**  catalan
**Croatian**  croatian
**Czech**  czech
**Danish**  danish
**Dutch**  dutch
**English**  english, american (*preferred to* USenglish), british (*preferred to* UKenglish),
canadian, australian, newzealand
**Esperanto**  esperanto

**Estonian**  estonian
**Finnish**  finnish
**French**  french, acadian
**Galician**  galician
**German**  ngerman, naustrian, german, austrian
**Greek**  greek, polutonikogreek
**Hebrew**  hebrew
**Icelandic**  icelandic
**Indonesian**  indonesian
**Interlingua**  interlingua
**Irish Gaelic**  irish
**Italian**  italian
**Latin**  latin
**Lower Sorbian**  lowersorbian
**Malay**  malay (*preferred to* melayu)
**Northern Sami**  northernsami
**Norwegian**  norsk, nynorsk
**Polish**  polish
**Portuguese**  portuguese, brazilian
**Romanian**  romanian
**Russian**  russian
**Scottish Gaelic**  scottishgaelic (*preferred to* scottish)
**Spanish**  spanish
**Slovakian**  slovak
**Slovenian**  slovene
**Swedish**  swedish
**Serbian**  serbian
**Turkish**  turkish
**Ukrainian**  ukrainian
**Upper Sorbian**  uppersorbian
**Welsh**  welsh

There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchle, ethiop and friulan.

**NOTE**  There are also some deprecated names (a few has been even removed): frenchb or francais, as well as canadien (french), germanb (german), bahasa, indon or bahasai (indonesian), lsorbian (lowersorbian), bahasam (malay), portuges (portuguese), brazil (brazilian), russianb (russian), usorbian (uppersorbian), vietnam (vietnamese), samin (northernsami), ukraineb (ukrainian). Deprecated names come in many cases from the times when they had to be shortened to 8 characters.

Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension `.dn`:

```
\documentclass{article}
\usepackage[hindi]{babel}
\begin{document}
{\dn devaanaa.m priya.h}
\end{document}
```

Then you preprocess it with devnag ⟨*file*⟩, which creates ⟨*file*⟩`.tex`; you can then typeset the latter with LaTeX.

## 1.29   Unicode character properties in luatex

New 3.32    Part of the babel job is to apply Unicode rules to some script-specific features based on some properties. Currently, they are 3, namely, direction (ie, bidi class), mirroring

glyphs, and line breaking for CJK scripts. These properties are stored in lua tables, which you can modify with the following macro (for example, to set them for glyphs in the PUA).

\babelcharproperty {⟨*char-code*⟩}[⟨*to-char-code*⟩]{⟨*property*⟩}{⟨*value*⟩}

New 3.32   Here, {⟨*char-code*⟩} is a number (with TEX syntax). With the optional argument, you can set a range of values. There are three properties (with a short name, taken from Unicode): direction (bc), mirror (bmg), linebreak (lb). The settings are global, and this command is allowed only in vertical mode (the preamble or between paragraphs).
For example:

```
\babelcharproperty{`¿}{mirror}{`?}
\babelcharproperty{`-}{direction}{l}  % or al, r, en, an, on, et, cs
\babelcharproperty{`)}{linebreak}{cl} % or id, op, cl, ns, ex, in, hy
```

Please, refer to the Unicode standard (Annex #9 and Annex #14) for the meaning of the available codes. For example, en is 'European number' and id is 'ideographic'.
New 3.39 ⊕   Another property is locale, which adds characters to the list used by onchar in \babelprovide, or, if the last argument is empty, removes them. The last argument is the locale name:

```
\babelcharproperty{`,}{locale}{english}
```

## 1.30   Tweaking some features

\babeladjust {⟨*key-value-list*⟩}

New 3.36 ⊕   Sometimes you might need to disable some babel features. Currently this macro understands the following keys, with values on or off:

| | | |
|---|---|---|
| autoload.bcp47 | bidi.math | layout.tabular |
| bcp47.toname | linebreak.sea | layout.lists |
| bidi.mirroring | linebreak.cjk | |
| bidi.text | justify.arabic | |

The first four are documented elsewhere. The following are by default on, but with off can disable some features: bidi.math (only preamble) changes for math or amsmath, linebreak.sea, linebreak.sea and justify.arabic the corresponding algorithms, layout.tabular and layout.lists changes for tabular and lists. Some of the are reverted only to some extent.
Other keys are:

| | | |
|---|---|---|
| autoload.options | autoload.bcp47.options | select.write |
| autoload.bcp47.prefix | prehyphenation.disable | select.encoding |

Most of them are documented elsewhere. With select.encoding=off, the encoding is not set when loading a language on the fly with pdftex (only off). prehyphenation.disable is by default nohyphenation, which means prehyphenation transforms are not applied if the current hyphenation rules are nohyphenation; with off they are never disabled.
For example, you can set \babeladjust{bidi.text=off} if you are using an alternative algorithm or with large sections not requiring it. Use with care, because these options do not deactivate other related options (like paragraph direction with bidi.text).

## 1.31 Tips, workarounds, known issues and notes

- If you use the document class book *and* you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), LaTeX will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the `safe` option to `none` or `bib`.

- Both ltxdoc and babel use `\AtBeginDocument` to change some catcodes, and babel reloads hhline to make sure `:` has the right one, so if you want to change the catcode of `|` it has to be done using the same method at the proper place, with

```
\AtBeginDocument{\DeleteShortVerb{\|}}
```

  *before* loading babel. This way, when the document begins the sequence is (1) make | active (ltxdoc); (2) make it unactive (your settings); (3) make babel shorthands active (babel); (4) reload hhline (babel, now with the correct catcodes for | and :).

- Documents with several input encodings are not frequent, but sometimes are useful. You can set different encodings for different languages as the following example shows:

```
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrasrussian{\inputencoding{koi8-r}}
```

- For the hyphenation to work correctly, lccodes cannot change, because TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.[20] So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of TeX, not of babel. Alternatively, you may use `\useshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

- `\bibitem` is out of sync with `\selectlanguage` in the `.aux` file. The reason is `\bibitem` uses `\immediate` (and others, in fact), while `\selectlanguage` doesn't. There is a similar issue with floats, too. There is no known workaround.

- Babel does not take into account `\normalsfcodes` and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the 'to do' list).

- Using a character mathematically active (ie, with math code `"8000`) as a shorthand can make TeX enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

The following packages can be useful, too (the list is still far from complete):

**csquotes** Logical markup for quotes.
**iflang** Tests correctly the current language.
**hyphsubst** Selects a different set of patterns for a language.
**translator** An open platform for packages that need to be localized.
**siunitx** Typesetting of numbers and physical quantities.
**biblatex** Programmable bibliographies and citations.
**bicaption** Bilingual captions.
**babelbib** Multilingual bibliographies.
**microtype** Adjusts the typesetting according to some languages (kerning and spacing). Ligatures can be disabled.
**substitutefont** Combines fonts in several encodings.

---

[20]This explains why LaTeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savinghyphcodes` is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.

**mkpattern**  Generates hyphenation patterns.
**tracklang**  Tracks which languages have been requested.
**ucharclasses**  (xetex) Switches fonts when you switch from one Unicode block to another.
**zhspacing**  Spacing for CJK documents in xetex.

## 1.32  Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better).

Useful additions would be, for example, time, currency, addresses and personal names.[21]. But that is the easy part, because they don't require modifying the LaTeX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ből", but "from (3)" is "(3)-ból", in Spanish an item labelled "3.º" may be referred to as either "ítem 3.º" or "3.ᵉʳ ítem", and so on.

An option to manage bidirectional document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (`xe-bidi`).

## 1.33  Tentative and experimental code

See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`). For old an deprecated functions, see the babel site.

### Options for locales loaded on the fly

New 3.51 ⊕  `\babeladjust{ autoload.options = ... }` sets the options when a language is loaded on the fly (by default, no options). A typical value would be `import`, which defines captions, date, numerals, etc., but ignores the code in the `tex` file (for example, extended numerals in Greek).

### Labels

New 3.48 ⊕  There is some work in progress for babel to deal with labels, both with the relation to captions (chapters, part), and how counters are used to define them. It is still somewhat tentative because it is far from trivial – see the babel site for further details.

# 2  Loading languages with `language.dat`

TeX and most engines based on it (pdfTeX, xetex, $\epsilon$-TeX, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, LaTeX, XeLaTeX, pdfLaTeX). babel provides a tool which has become standard in many distributions and based on a "configuration file" named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

New 3.9q  With luatex, however, patterns are loaded on the fly when requested by the language (except the "0th" language, typically english, which is preloaded always).[22] Until 3.9n, this task was delegated to the package luatex-hyphen, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must**

---

[21]See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to TeX because their aim is just to display information and not fine typesetting.

[22]This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

**rebuild the formats** if upgrading from a previous version. You may want to have a local language.dat for a particular project (for example, a book on Chemistry).[23]

## 2.1 Format

In that file the person who maintains a TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored[24]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

The file can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct LaTeX that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

```
% File    : language.dat
% Purpose : tell iniTeX what files with patterns to load.
english    english.hyphenations
=british

dutch      hyphen.dutch exceptions.dutch % Nederlands
german hyphen.ger
```

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.[25] For example:

```
german:T1 hyphenT1.ger
german hyphen.ger
```

With the previous settings, if the encoding when the language is selected is T1 then the patterns in hyphenT1.ger are used, but otherwise use those in hyphen.ger (note the encoding can be set in \extras⟨*lang*⟩).

A typical error when using babel is the following:

```
No hyphenation patterns were preloaded for
the language `<lang>' into the format.
Please, configure your TeX system to add them and
rebuild the format. Now I will use the patterns
preloaded for english instead}}
```

It simply means you must reconfigure language.dat, either by hand or with the tools provided by your distribution.

## 3 The interface between the core of babel and the language definition files

The *language definition files* (ldf) must conform to a number of conventions, because these files have to fill in the gaps left by the common code in babel.def, i.e., the definitions of the macros that produce texts. Also the language-switching possibility which has been built into the babel system has its implications.

The following assumptions are made:

- Some of the language-specific definitions might be used by plain TeX users, so the files have to be coded so that they can be read by both LaTeX and plain TeX. The current format can be checked by looking at the value of the macro \fmtname.

---

[23]The loader for lua(e)tex is slightly different as it's not based on babel but on etex.src. Until 3.9p it just didn't work, but thanks to the new code it works by reloading the data in the babel way, i.e., with language.dat.

[24]This is because different operating systems sometimes use *very* different file-naming conventions.

[25]This is not a new feature, but in former versions it didn't work correctly.

- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are \⟨*lang*⟩hyphenmins, \captions⟨*lang*⟩, \date⟨*lang*⟩, \extras⟨*lang*⟩ and \noextras⟨*lang*⟩(the last two may be left empty); where ⟨*lang*⟩ is either the name of the language definition file or the name of the LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, \date⟨*lang*⟩ but not \captions⟨*lang*⟩ does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define \l@⟨*lang*⟩ to be a dialect of \language0 when \l@⟨*lang*⟩ is undefined.

- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.

- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is /).

Some recommendations:

- The preferred shorthand is ", which is not used in LaTeX (quotes are entered as `` and ' '). Other good choices are characters which are not used in a certain context (eg, = in an ancient language). Note however =, <, >, : and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).

- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.

- Avoid adding things to \noextras⟨*lang*⟩ except for umlauthigh and friends, \bbl@deactivate, \bbl@(non)frenchspacing, and language-specific macros. Use always, if possible, \babel@save and \babel@savevariable (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in \extras⟨*lang*⟩.

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like \latintext is deprecated.[26]

- Please, for "private" internal macros do not use the \bbl@ prefix. It is used by babel and it can lead to incompatibilities.

There are no special requirements for documenting your language files. Now they are not included in the base babel manual, so provide a standalone document suited for your needs, as well as other files you think can be useful. A PDF and a "readme" are strongly recommended.

## 3.1 Guidelines for contributed languages

Currently, the easiest way to contribute a new language is by taking one the the 500 or so ini templates available on GitHub as a basis. Just make a pull request o dowonload it and then, after filling the fields, sent it to me. Fell free to ask for help or to make feature requests.

---

[26]But not removed, for backward compatibility.

As to `ldf` files, now language files are "outsourced" and are located in a separate directory (`/macros/latex/contrib/babel-contrib`), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN).

Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

- Do not hesitate stating on the file heads you are the author and the maintainer, if you actually are. There is no need to state the babel maintainer(s) as authors if they have not contributed significantly to your language files.

- Fonts are not strictly part of a language, so they are best placed in the corresponding TeX tree. This includes not only `tfm`, `vf`, `ps1`, `otf`, `mf` files and the like, but also `fd` ones.

- Font and input encodings are usually best placed in the corresponding tree, too, but sometimes they belong more naturally to the babel style. Note you may also need to define a LICR.

- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

The following page provides a starting point for `ldf` files: `http://www.texnia.com/incubator.html`. See also `https://latex3.github.io/babel/guides/list-of-locale-templates.html`. If you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

## 3.2 Basic macros

In the core of the babel system, several macros are defined for use in language definition files. Their purpose is to make a new language known. The first two are related to hyphenation patterns.

\addlanguage    The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. Here "language" is used in the TeX sense of set of hyphenation patterns.

\adddialect    The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the babel system is to define this language as a 'dialect' of the language for which the patterns were loaded as `\language0`. Here "language" is used in the TeX sense of set of hyphenation patterns.

\<lang>hyphenmins    The macro `\`⟨*lang*⟩`hyphenmins` is used to store the values of the `\lefthyphenmin` and `\righthyphenmin`. Redefine this macro to set your own values, with two numbers corresponding to these two parameters. For example:

```
\renewcommand\spanishhyphenmins{34}
```

(Assigning `\lefthyphenmin` and `\righthyphenmin` directly in `\extras<lang>` has no effect.)

\providehyphenmins    The macro `\providehyphenmins` should be used in the language definition files to set `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do *not* set them).

\captions⟨*lang*⟩    The macro `\captions`⟨*lang*⟩ defines the macros that hold the texts to replace the original hard-wired texts.

\date⟨*lang*⟩    The macro `\date`⟨*lang*⟩ defines `\today`.

\extras⟨*lang*⟩    The macro `\extras`⟨*lang*⟩ contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.

\noextras⟨*lang*⟩    Because we want to let the user switch between languages, but we do not know what state TeX might be in after the execution of `\extras`⟨*lang*⟩, a macro that brings TeX into a

predefined state is needed. It will be no surprise that the name of this macro is \noextras⟨*lang*⟩.

\bbl@declare@ttribute This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.

\main@language To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use \main@language instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.

\ProvidesLanguage The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command \ProvidesPackage.

\LdfInit The macro \LdfInit performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the .ldf file from being processed twice, etc.

\ldf@quit The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.

\ldf@finish The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.

\loadlocalcfg After processing a language definition file, LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to \captions⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.

### 3.3 Skeleton

Here is the basic structure of an ldf file, with a language, a dialect and an attribute. Strings are best defined using the method explained in sec. 3.8 (babel 3.9 and later).

```
\ProvidesLanguage{<language>}
     [2016/04/23 v0.0 <Language> support from the babel system]
\LdfInit{<language>}{captions<language>}

\ifx\undefined\l@<language>
  \@nopatterns{<Language>}
  \adddialect\l@<language>0
\fi

\adddialect\l@<dialect>\l@<language>

\bbl@declare@ttribute{<language>}{<attrib>}{%
  \expandafter\addto\expandafter\extras<language>
  \expandafter{\extras<attrib><language>}%
  \let\captions<language>\captions<attrib><language>}

\providehyphenmins{<language>}{\tw@\thr@@}

\StartBabelCommands*{<language>}{captions}
\SetString\chaptername{<chapter name>}
% More strings

\StartBabelCommands*{<language>}{date}
\SetString\monthiname{<name of first month>}
% More strings

\StartBabelCommands*{<dialect>}{captions}
\SetString\chaptername{<chapter name>}
% More strings
```

```
\StartBabelCommands*{<dialect>}{date}
\SetString\monthiname{<name of first month>}
% More strings

\EndBabelCommands


\addto\extras<language>{}
\addto\noextras<language>{}
\let\extras<dialect>\extras<language>
\let\noextras<dialect>\noextras<language>


\ldf@finish{<language>}
```

**NOTE**  If for some reason you want to load a package in your style, you should be aware it cannot be done directly in the ldf file, but it can be delayed with `\AtEndOfPackage`. Macros from external packages can be used *inside* definitions in the ldf itself (for example, `\extras<language>`), but if executed directly, the code must be placed inside `\AtEndOfPackage`. A trivial example illustrating these points is:

```
\AtEndOfPackage{%
  \RequirePackage{dingbat}%        Delay package
  \savebox{\myeye}{\eye}}%         And direct usage
\newsavebox{\myeye}
\newcommand\myanchor{\anchor}%     But OK inside command
```

## 3.4   Support for active characters

In quite a number of language definition files, active characters are introduced. To facilitate this, some support macros are provided.

\initiate@active@char  The internal macro `\initiate@active@char` is used in language definition files to instruct LaTeX to give a character the category code 'active'. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

\bbl@activate  The command `\bbl@activate` is used to change the way an active character expands.
\bbl@deactivate  `\bbl@activate` 'switches on' the active behavior of the character. `\bbl@deactivate` lets the active character expand to its former (mostly) non-active self.

\declare@shorthand  The macro `\declare@shorthand` is used to define the various shorthands. It takes three arguments: the name for the collection of shorthands this definition belongs to; the character (sequence) that makes up the shorthand, i.e. ~ or "a; and the code to be executed when the shorthand is encountered. (It does *not* raise an error if the shorthand character has not been "initiated".)

\bbl@add@special  The TeXbook states: "Plain TeX includes a macro called `\dospecials` that is essentially a set
\bbl@remove@special  macro, representing the set of all characters that have a special category code." [4, p. 380] It is used to set text 'verbatim'. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. LaTeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special`⟨*char*⟩ and `\bbl@remove@special`⟨*char*⟩ add and remove the character ⟨*char*⟩ to these two sets.

\@safe@activestrue  Enables and disables the "safe" mode. It is a tool for package and class authors. See the
\@safe@activesfalse  description below.

## 3.5   Support for saving macro definitions

Language definition files may want to *re*define macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this[27].

\babel@save  To save the current meaning of any control sequence, the macro `\babel@save` is provided.

---

[27]This mechanism was introduced by Bernd Raichle.

59

It takes one argument, ⟨*csname*⟩, the control sequence for which the meaning has to be saved.

**\babel@savevariable** A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the \the primitive is considered to be a variable. The macro takes one argument, the ⟨*variable*⟩.

The effect of the preceding macros is to append a piece of code to the current definition of \originalTeX. When \originalTeX is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

## 3.6   Support for extending macros

**\addto** The macro \addto{⟨*control sequence*⟩}{⟨*TEX code*⟩} can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or \relax). This macro can, for instance, be used in adding instructions to a macro like \extrasenglish.

Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using etoolbox, by Philipp Lehman, consider using the tools provided by this package instead of \addto.

## 3.7   Macros common to a number of languages

**\bbl@allowhyphens** In several languages compound words are used. This means that when TEX has to hyphenate such a compound word, it only does so at the '-' that is used in such words. To allow hyphenation in the rest of such a compound word, the macro \bbl@allowhyphens can be used.

**\allowhyphens** Same as \bbl@allowhyphens, but does nothing if the encoding is T1. It is intended mainly for characters provided as real glyphs by this encoding but constructed with \accent in OT1.

Note the previous command (\bbl@allowhyphens) has different applications (hyphens and discretionaries) than this one (composite chars). Note also prior to version 3.7, \allowhyphens had the behavior of \bbl@allowhyphens.

**\set@low@box** For some languages, quotes need to be lowered to the baseline. For this purpose the macro \set@low@box is available. It takes one argument and puts that argument in an \hbox, at the baseline. The result is available in \box0 for further processing.

**\save@sf@q** Sometimes it is necessary to preserve the \spacefactor. For this purpose the macro \save@sf@q is available. It takes one argument, saves the current spacefactor, executes the argument, and restores the spacefactor.

**\bbl@frenchspacing** The commands \bbl@frenchspacing and \bbl@nonfrenchspacing can be used to
**\bbl@nonfrenchspacing** properly switch French spacing on and off.

## 3.8   Encoding-dependent strings

New 3.9a   Babel 3.9 provides a way of defining strings in several encodings, intended mainly for luatex and xetex, although the old way of defining/switching strings still works and it's used by default.

It consist is a series of blocks started with \StartBabelCommands. The last block is closed with \EndBabelCommands. Each block is a single group (ie, local declarations apply until the next \StartBabelCommands or \EndBabelCommands). An ldf may contain several series of this kind.

Thanks to this new feature, string values and string language switching are not mixed any more. Furthermore, strings do no need to be wrapped with formatting commands (eg, to select the writing direction) because babel takes care of it automatically. (See also \setlocalecaption.)

**\StartBabelCommands** {⟨*language-list*⟩}{⟨*category*⟩}[⟨*selector*⟩]

The ⟨*language-list*⟩ specifies which languages the block is intended for. A block is taken into account only if the \CurrentOption is listed here. Alternatively, you can define

60

`\BabelLanguages` to a comma-separated list of languages to be defined (if undefined, `\StartBabelCommands` sets it to `\CurrentOption`). You may write `\CurrentOption` as the language, but this is discouraged – a explicit name (or names) is much better and clearer. A "selector" selects a group of definition are to be used, optionally followed by extra info about the encodings to be used. The name `unicode` must be used for xetex and luatex. Without a selector, the LICR representation (ie, with macros like `\~{n}` instead of ñ) is assumed.

If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like `\providecommand`).

Encoding info is `charset=` followed by a charset, which if given sets how the strings should be translated to the internal representation used by the engine, typically `utf8`, which is the only value supported currently (default is no translations). Note `charset` is applied by luatex and xetex when reading the file, not when the macro or string is used in the document.

A list of font encodings which the strings are expected to work with can be given after `fontenc=` (separated with spaces, if two or more) – recommended, but not mandatory, although blocks without this key are not taken into account if you have requested `strings=encoded`.

Blocks without a selector are read always. They provide fallback values, and therefore they must be the last ones; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks (mainly LGC scripts using the LICR). The ⟨*category*⟩ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other names.[28] It may be empty, too, but in such a case using `\SetString` is an error.

```
\StartBabelCommands{language}{captions}
  [unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetString{\chaptername}{utf8-string}

\StartBabelCommands{language}{captions}
\SetString{\chaptername}{ascii-maybe-LICR-string}

\EndBabelCommands
```

A real example can be:

```
\StartBabelCommands{austrian}{date}
  [unicode, fontenc=TU, charset=utf8]
  \SetString\monthiname{Jänner}

\StartBabelCommands{german,austrian}{date}
  [unicode, fontenc=TU, charset=utf8]
  \SetString\monthiiiname{März}

\StartBabelCommands{austrian}{date}
  \SetString\monthiname{J\"{a}nner}

\StartBabelCommands{german}{date}
  \SetString\monthiname{Januar}

\StartBabelCommands{german,austrian}{date}
  \SetString\monthiiname{Februar}
  \SetString\monthiiiname{M\"{a}rz}
  \SetString\monthivname{April}
  \SetString\monthvname{Mai}
  \SetString\monthviname{Juni}
  \SetString\monthviiname{Juli}
```

---

[28]In future releases further categories may be added.

```
  \SetString\monthviiiname{August}
  \SetString\monthixname{September}
  \SetString\monthxname{Oktober}
  \SetString\monthxiname{November}
  \SetString\monthxiiname{Dezenber}
  \SetString\today{\number\day.~%
    \csname month\romannumeral\month name\endcsname\space
    \number\year}

\StartBabelCommands{german,austrian}{captions}
  \SetString\prefacename{Vorwort}
  [etc.]

\EndBabelCommands
```

When used in ldf files, previous values of \⟨*category*⟩⟨*language*⟩ are overridden, which means the old way to define strings still works and used by default (to be precise, is first set to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if \date⟨*language*⟩ exists).

**NOTE** The package option strings introduced in version 3.9 (around 2013) when Unicode engines were still of marginal use, is now deprecated.

**NOTE** Captions and other strings defined in ini files (in other words, when a locale is loaded with \babelprovide) are internally set with the help of these macros.

\StartBabelCommands ⁕{⟨*language-list*⟩}{⟨*category*⟩}[⟨*selector*⟩]

The starred version just forces strings to take a value – if not set as package option (which is now deprecated), then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.[29]

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands {⟨*code*⟩}

The code is delayed and executed at the global scope just after \EndBabelCommands.

\SetString {⟨*macro-name*⟩}{⟨*string*⟩}

Adds ⟨*macro-name*⟩ to the current category, and defines globally ⟨*lang-macro-name*⟩ to ⟨*code*⟩ (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).
Use this command to define strings, without including any "logic" if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop {⟨*macro-name*⟩}{⟨*string-list*⟩}

A convenient way to define several ordered names at once. For example, to define \abmoniname, \abmoniiname, etc. (and similarly with abday):

```
\SetStringLoop{abmon#1name}{en,fb,mr,ab,my,jn,jl,ag,sp,oc,nv,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
```

#1 is replaced by the roman numeral.

---

[29]This replaces in 3.9g a short-lived \UseStrings which has been removed because it did not work.

\SetCase [⟨*map-list*⟩]{⟨*toupper-code*⟩}{⟨*tolower-code*⟩}     *Deprecated*

> **WARNING**  This feature doesn't work any longer after some changes in the LaTeX kernel. It's now deprecated and an alternative is on the way.

\SetHyphenMap {⟨*to-lower-macros*⟩}

New 3.9g  Case mapping for hyphenation is handled with \SetHyphenMap and controlled with the package option hyphenmap.

There are three helper macros to be used inside \SetHyphenMap:

- \BabelLower{⟨*uccode*⟩}{⟨*lccode*⟩} is similar to \lccode but it's ignored if the char has been set and saves the original lccode to restore it when switching the language (except with hyphenmap=first).

- \BabelLowerMM{⟨*uccode-from*⟩}{⟨*uccode-to*⟩}{⟨*step*⟩}{⟨*lccode-from*⟩} loops though the given uppercase codes, using the step, and assigns them the lccode, which is also increased (MM stands for *many-to-many*).

- \BabelLowerMO{⟨*uccode-from*⟩}{⟨*uccode-to*⟩}{⟨*step*⟩}{⟨*lccode*⟩} loops though the given uppercase codes, using the step, and assigns them the lccode, which is fixed (MO stands for *many-to-one*).

An example is (which is redundant, because these assignments are done by both luatex and xetex):

```
\SetHyphenMap{\BabelLowerMM{"100}{"11F}{2}{"101}}
```

> **NOTE**  This macro is not intended to fix wrong mappings done by Unicode (which are the default in both xetex and luatex) – if an assignment is wrong, fix it directly.

### 3.9   Executing code based on the selector

\IfBabelSelectorTF {⟨*selectors*⟩}{⟨*true*⟩}{⟨*false*⟩}

New 3.67 ⊕  Sometimes a different setup is desired depending on the selector used. Values allowed in ⟨*selectors*⟩ are select, other, foreign, other* (and also foreign* for the tentative starred version), and it can consist of a comma-separated list. For example:

```
\IfBabelSelectorTF{other, other*}{A}{B}
```

is true with any of these two environment selectors.
Its natural place of use is in hooks or in \extras⟨*language*⟩.

## 4   Acknowledgements

# References

[1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.

[2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national LaTeX styles, TUGboat* 10 (1989) #3, p. 401–406.

[3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.

[4] Donald E. Knuth, *The TeXbook*, Addison-Wesley, 1986.

[5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.

[6] Leslie Lamport, *LaTeX, A document preparation System*, Addison-Wesley, 1986.

[7] Leslie Lamport, in: TeXhax Digest, Volume 89, #13, 17 February 1989.

[8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.

[9] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018

[10] Hubert Partl, *German TeX*, *TUGboat* 9 (1988) #1, p. 70–72.

[11] Joachim Schrod, *International LaTeX is ready to use, TUGboat* 11 (1990) #1, p. 87–90.

[12] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using LaTeX*, Springer, 2002, p. 301–373.

[13] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).