Babel

Localization and internationalization

Unicode
\TeX
pdf\TeX
Lua\TeX
Xe\TeX
## Contents

### I User guide

1. The user interface ........................................... 4
   1.1 Monolingual documents .................................. 4
   1.2 Multilingual documents ................................ 6
   1.3 Mostly monolingual documents ......................... 7
   1.4 Modifiers .................................................. 8
   1.5 Troubleshooting ......................................... 8
   1.6 Plain ..................................................... 9
   1.7 Basic language selectors ................................. 10
   1.8 Auxiliary language selectors ......................... 11
   1.9 More on selection ...................................... 12
   1.10 Shorthands .............................................. 15
   1.11 Package options ....................................... 17
   1.12 The base option ........................................ 18
   1.13 ini files ................................................ 26
   1.14 Selecting fonts ......................................... 28
   1.15 Modifying a language ................................. 29
   1.16 Creating a language .................................. 32
   1.17 Digits and counters .................................. 34
   1.18 Dates ..................................................... 35
   1.19 Accessing language info ............................... 38
   1.20 Hyphenation and line breaking ....................... 39
   1.21 Selection based on BCP 47 tags ...................... 43
   1.22 Selecting scripts ...................................... 44
   1.23 Selecting directions .................................. 45
   1.24 Language attributes .................................. 46
   1.25 Hooks .................................................... 47
   1.26 Languages supported by babel with ldf files ....... 48
   1.27 Unicode character properties in luatex ............ 49
   1.28 Tweaking some features ............................... 49
   1.29 Tips, workarounds, known issues and notes ....... 50
   1.30 Current and future work .............................. 51
   1.31 Tentative and experimental code .................... 52

2. Loading languages with language.dat ..................... 49
   2.1 Format ................................................... 49

3. The interface between the core of babel and the language definition files 50
   3.1 Guidelines for contributed languages ............... 51
   3.2 Basic macros ........................................... 52
   3.3 Skeleton ................................................ 53
   3.4 Support for active characters ....................... 54
   3.5 Support for saving macro definitions ............... 55
   3.6 Support for extending macros ....................... 55
   3.7 Macros common to a number of languages .......... 55
   3.8 Encoding-dependent strings .......................... 55

4. Changes ....................................................... 59
   4.1 Changes in babel version 3.9 ......................... 59

II Source code ................................................ 59
### 13 Hooks for XeTeX and LuaTeX

13.1 XeTeX .......................... 158
13.2 Layout .......................... 160
13.3 LuaTeX .......................... 162
13.4 Southeast Asian scripts .... 168
13.5 CJK line breaking ............... 171
13.6 Automatic fonts and ids switching .......................... 172
13.7 Layout .......................... 182
13.8 Auto bidi with basic and basic-r .......................... 185

### 14 Data for CJK

### 15 The ‘nil’ language

### 16 Support for Plain TeX (plain.def)

16.1 Not renaming hyphen.tex .......................... 197
16.2 Emulating some MiTeX features .......................... 198
16.3 General tools: .......................... 198
16.4 Encoding related macros .......................... 202

### 17 Acknowledgements

### Troubleshooting

- Paragraph ended before \UTFviii@three@octets was complete ............... 5
- No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format ............... 5
- You are loading directly a language style ............... 8
- Unknown language ‘LANG’ ............... 8
- Argument of \language@active@arg“has an extra} ............... 12
- Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead ............... 28
- Package babel Info: The following fonts are not babel standard families ............... 28
Part I

User guide

- This user guide focuses on internationalization and localization with \LaTeX. There are also some notes on its use with Plain \TeX.

- Changes and new features with relation to version 3.8 are highlighted with \textcolor{red}{New XXX}, and there are some notes for the latest versions in the \texttt{babel wiki}. The most recent features could be still unstable. Please, report any issues you find in \texttt{GitHub}, which is better than just complaining on an e-mail list or a web forum.

- If you are interested in the \TeX multilingual support, please join the \texttt{kadingira mail list}. You can follow the development of \texttt{babel} in \texttt{GitHub} (which provides many sample files too). If you are the author of a package, feel free to send to me a few test files which I'll add to mine, so that possible issues could be caught in the development phase.

- See section 3.1 for contributing a language.

- The first sections describe the traditional way of loading a language (with \texttt{.ldf} files). The alternative way based on \texttt{.ini} files, which complements the previous one (it does \textit{not} replace it), is described below.

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.

Many languages are compatible with \texttt{xetex} and \texttt{luatex}. With them you can use \texttt{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 \texttt{lmroman}. Other scripts require loading \texttt{fontspec}. You may want to set the font attributes with \texttt{fontspec}, too.

\textbf{EXAMPLE} Here is a simple full example for “traditional” \TeX engines (see below for \texttt{xetex} and \texttt{luatex}). The packages \texttt{fontenc} and \texttt{inputenc} do not belong to \texttt{babel}, but they are included in the example because typically you will need them (however, the package \texttt{inputenc} may be omitted with \LaTeX $\geq$ 2018-04-01 if the encoding is UTF-8):

\begin{verbatim}
\documentclass{article}
\usepackage[T1]{fontenc}
% \usepackage[utf8]{inputenc} % Uncomment if \LaTeX $<$ 2018-04-01
\usepackage[french]{babel}
\begin{document}
Plus ça change, plus c'est la même chose!
\end{document}
\end{verbatim}
**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).

```latex
\documentclass{article}
\usepackage[russian]{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 could 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.

Another approach is making the language (french in the example) a global option in order to let other packages detect and use it:

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

In this last example, the package varioref will also see the option and will be able to use it.

**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:
The document will be typeset, but very likely the text will not be correctly hyphenated. Some languages may be raising this warning wrongly (because they are not hyphenated); it is a bug to be fixed – just ignore it. See the manual of your distribution (Mac\TeX, Mi\TeX, \TeX\Live, etc.) for further info about how to configure it.

\textbf{NOTE} With \texttt{hyperref} you may want to set the document language with something like:

\begin{verbatim}
\usepackage[pdflang=es-MX]{hyperref}
\end{verbatim}

This is not currently done by babel and you must set it by hand.

\section*{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, \texttt{spanish} and \texttt{french}).

\textbf{EXAMPLE} In $\LaTeX$, the preamble of the document:

\begin{verbatim}
\documentclass{article}
\usepackage[dutch,english]{babel}
\end{verbatim}

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 a real reason to do so:

\begin{verbatim}
\documentclass{article}
\usepackage[main=english,dutch]{babel}
\end{verbatim}

Examples of cases where \texttt{main} is useful are the following.

\textbf{NOTE} Some classes load babel with a hardcoded language option. Sometimes, the main language could be overridden with something like that before $\texttt{\documentclass}$:

\begin{verbatim}
\PassOptionsToPackage{main=english}{babel}
\end{verbatim}

\textbf{WARNING} 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 \texttt{main}:

\begin{verbatim}
\documentclass[italian]{book}
\usepackage[ngerman,main=italian]{babel}
\end{verbatim}
\textbf{WARNING} In the preamble the main language has \emph{not} been selected, except hyphenation patterns and the name assigned to `\texttt{\textbackslash 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: `\texttt{selectlanguage}` is used for blocks of text, while `\texttt{foreignlanguage}` is for chunks of text inside paragraphs.

\textbf{EXAMPLE} A full bilingual document follows. The main language is \texttt{french}, which is activated when the document begins. The package \texttt{inputenc} may be omitted with \texttt{\LaTeXX} $\geq$ 2018-04-01 if the encoding is UTF-8.

\begin{Verbatim}
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\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}
\end{Verbatim}

\textbf{EXAMPLE} With \texttt{xetex} and \texttt{luatex}, the following bilingual, single script document in UTF-8 encoding just prints a couple of `\texttt{\textbackslash today}` in Danish and Vietnamese. No additional packages are required.

\begin{Verbatim}
\documentclass{article}
\usepackage[vietnamese,danish]{babel}
\begin{document}
\prefacename{} -- \alsoname{} -- \today
\selectlanguage{vietnamese}
\prefacename{} -- \alsoname{} -- \today
\end{document}
\end{Verbatim}

1.3 Mostly monolingual documents

\textbf{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.

**EXAMPLE**  A trivial document is:

```latex
\documentclass{article}
\usepackage[english]{babel}
\babelfont[russian]{rm}{FreeSerif}
\begin{document}
English. \foreignlanguage{russian}{Русский}.
\foreignlanguage{spanish}{Español}
\end{document}
```

### 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):

```latex
\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. However, modifiers are a more general mechanism.

### 1.5 Troubleshooting

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

```latex
! Package babel Error: You are loading directly a language style. \usepackage[⟨language⟩]{babel}.
```

- Another typical error when using babel is the following:

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

---

1. No predefined “axis” for modifiers are provided because languages and their scripts have quite different needs.
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.”
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 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 Plain.

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:

```
\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.

**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.
The \texttt{\foreignlanguage{\langle language\rangle}{\langle text\rangle}} command 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, \textit{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 \texttt{bidi} option, it also enters in horizontal mode (this is not done always for backwards compatibility).

\textbf{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:

\begin{verbatim}
\foreignlanguage{\langle date\rangle}{\langle polish\rangle}{\\today}
\end{verbatim}

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

### 1.8 Auxiliary language selectors

\begin{verbatim}
\begin{otherlanguage}{\langle language\rangle} ... \end{otherlanguage}
\end{verbatim}

The environment \texttt{\begin{otherlanguage}{\langle language\rangle}} does basically the same as \texttt{\selectlanguage{\langle language\rangle}}, 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:

\begin{verbatim}
\begingroup
\selectlanguage{\langle inner-language\rangle}
...
\endgroup
\selectlanguage{\langle outer-language\rangle}
\end{verbatim}

If you want a change which is really local, you must enclose this environment with an additional grouping, like braces \texttt{\{\}}. Spaces after the environment are ignored.

\begin{verbatim}
\begin{otherlanguage*}{\langle option-list\rangle}{\langle language\rangle} ... \end{otherlanguage*}
\end{verbatim}

Same as \texttt{\foreignlanguage} but as environment. Spaces after the environment are \textit{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 \texttt{\foreignlanguage}, except when the option \texttt{bidi} is set – in this case, \texttt{\foreignlanguage} emits a \texttt{\leavevmode}, while \texttt{\otherlanguage*} does not.

\begin{verbatim}
\begin{hyphenrules}{\langle language\rangle} ... \end{hyphenrules}
\end{verbatim}

The environment \texttt{\hyphenrules} can be used to select \textit{only} the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select `nohyphenation',

---

\textsuperscript{4}Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues have been fixed.
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 discouraged and otherlanguage* (the starred version) is preferred, as the former does not take into account possible changes in encodings of characters like, say, ' done by some languages (eg, italian, french, ukraineb).
To set hyphenation exceptions, use \babelhyphenation (see below).

1.9 More on selection

\babeltags \{\langle tag1\rangle = \langle language1\rangle, \langle tag2\rangle = \langle language2\rangle, \ldots\}\}

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

EXAMPLE With

\text{\textde{German text} text} and

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

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

\NOTE Actually, there may be another advantage in the 'short' syntax \text{\langle tag\rangle}, namely, it is not affected by \MakeUppercase (while \foreignlanguage is).

\babelensure \[include=\langle commands\rangle, exclude=\langle commands\rangle, fontenc=\langle encoding\rangle]\{\langle language\rangle\}

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: 

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 fontenc. A couple of examples:

```
\babelensure{polish}
```

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

They are activated when the language is selected (at the after extras 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 or \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 \l are used to insert the right amount of white space; (3) several kinds of discretionary breaks 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 pdf\TeX provides knbccode, and luatex can manipulate the glyph list. Tools for point 3 can be still very useful in general.

There are three levels of shorthands: user, language, and system (by order of precedence). Version 3.9 introduces the language user level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

**NOTE** Note 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.

2. If on a certain level (system, 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 it is 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, "{}).
\shorthandoff *{\langle\text{shorthand-list}\rangle} 

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.

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.

\useshorthands *{\langle\text{char}\rangle} 

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*{\langle\text{char}\rangle} 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 [\langle\text{language}\rangle, \langle\text{language}\rangle, ...]{\langle\text{shorthand}\rangle}{\langle\text{code}\rangle} 

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 \languageshorthands{\langle lang\rangle} to the corresponding \extras{\langle lang\rangle}, 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 could start with, say:

\useshorthands*{"} 
\defineshorthand{*}{\text{babelhyphen\{soft\}}} 
\defineshorthand{-}{\text{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 could then set:

\footnote{With it, encoded strings may not work as expected.}
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.

\languageshorthand\{\langle language\rangle\}

The command \languageshorthand 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).\(^6\) 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{\languageshorthand{\ngerman}}

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

**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]{\languageshorthand{none} \tipaencoding#1}

\babelshorthand\{\langle shorthand\rangle\}

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.)

**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\(^7\)

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

\(^6\)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.

\(^7\)Thanks to Enrico Gregorio
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\[8\]

\ifbabelshorthand \{\langle character\rangle}\{\langle true\rangle}\{\langle false\rangle\}

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

\aliasshorthand \{\langle original\rangle}\{\langle alias\rangle\}

The command \aliasshorthand can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character / over " in typing Polish texts, this can be achieved by entering \aliasshorthand{"}\{/\}. For the reasons in the warning below, usage of this macro is not recommended.

NOTE The substitute character must not have been declared before as shorthand (in such a case, \aliashorthand is ignored).

EXAMPLE The following example shows how to replace a shorthand by another

\aliasshorthand{-}{^}
\AtBeginDocument{\shorthandoff*{-}}

WARNING Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand if found, ^ expands to a non-breaking space, because this is the value of ~ (internally, ^ still calls \active@char~ or \normal@char~). Furthermore, if you change the system value of ^ with \defineshorthand nothing happens.

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.

\[8\]This declaration serves to nothing, but it is preserved for backward compatibility.
**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 option to set `' as a shorthand in case it is not done by default.

**activegrave**  Same for `.`.

**shorthands=**  `\langle char\rangle\langle char\rangle|off`

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

```
\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 \LaTeX 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=**  `\langle file\rangle`

Load `\langle file\rangle.cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with noconfigs).

**main=**  `\langle language\rangle`

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=**  `\langle language\rangle`

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.
noconfigs  Global and language default config files are not loaded, so you can make sure your document is not spoil 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.

case  New 3.91  Language settings for uppercase and lowercase mapping (as set by \SetCase) are ignored. Use only if there are incompatibilities with other packages.

silent  New 3.91  No warnings and no infos are written to the log file.

strings=  generic | unicode | encoded | ⟨label⟩ | ⟨font encoding⟩

Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional \TeX, LICR and ASCII strings), unicode (for engines like \xetex and \luatex) and encoded (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in \MakeUpperCase and the like (this feature misuses some internal \TeX tools, so use it only as a last resort).

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

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

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;
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.

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. 1.23

layout=

New 3.16  Selects which layout elements are adapted in bidi documents. See sec. 1.23

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

---

9You can use alternatively the package silence.
10Turned off in plain.
11Duplicated options count as several ones.
12Providing 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.
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 \ldef@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}

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 200 of these files containing the basic data
required for a locale. 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 Language 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
currently (by means of \babelprovide), but a higher interface, based on package options,
in under study. In other words, \babelprovide is mainly meant for auxiliary tasks, and as
alternative when the ldf, for some reason, does 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.
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 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, and a recent version of fontspec/loaotfload is required. In xetex babel resorts to the bidi package, which seems to work.

Hebrew  Niqqud marks seem to work in both engines, but cantillation marks are misplaced (xetex or luatex with Harfbuzz seems better, but still problematic).

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 fine tuning the font behavior is not always possible.

Southeast scripts  That works in both luatex and xetex, but line breaking differs (rules can be modified in luatex; they are hard-coded in xetex). 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 luatex also applies here. Some quick patterns could help, with something similar to:

\babelprovide{import,hyphenrules=+}{lao}
\babetapatterns[lao]{1ດ 1ມ 1ອ 1ງ 1ກ 1າ} % 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 short 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 \texttt{ltj}book does with luatex, which can be used in conjunction with the \texttt{ldf} for japanese, because the following piece of code loads luatexja:
**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 seems 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):

<table>
<thead>
<tr>
<th>Code</th>
<th>Language</th>
<th>Code</th>
<th>Language</th>
<th>Code</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td>cs</td>
<td>Czech</td>
<td></td>
<td></td>
</tr>
<tr>
<td>agq</td>
<td>Aghem</td>
<td>cu</td>
<td>Church Slavic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ak</td>
<td>Akan</td>
<td>cu-Cyrs</td>
<td>Church Slavic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>am</td>
<td>Amharic</td>
<td>cu-Glag</td>
<td>Church Slavic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar</td>
<td>Arabic</td>
<td>cy</td>
<td>Welsh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar-DZ</td>
<td>Arabic</td>
<td>da</td>
<td>Danish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar-MA</td>
<td>Arabic</td>
<td>dav</td>
<td>Taita</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ar-SY</td>
<td>Arabic</td>
<td>de-AT</td>
<td>German</td>
<td></td>
<td></td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
<td>de</td>
<td>German</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asa</td>
<td>Asu</td>
<td>de</td>
<td>German</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ast</td>
<td>Asturian</td>
<td>dje</td>
<td>Zarma</td>
<td></td>
<td></td>
</tr>
<tr>
<td>az-Cyr</td>
<td>Azerbaijani</td>
<td>dsb</td>
<td>Lower Sorbian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>az-Latn</td>
<td>Azerbaijani</td>
<td>dua</td>
<td>Duala</td>
<td></td>
<td></td>
</tr>
<tr>
<td>az</td>
<td>Azerbaijani</td>
<td>dyo</td>
<td>Jola-Fonyi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bas</td>
<td>Basaa</td>
<td>dz</td>
<td>Dzongkha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>be</td>
<td>Belarussian</td>
<td>ebu</td>
<td>Embu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bem</td>
<td>Bemba</td>
<td>ee</td>
<td>Ewe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bez</td>
<td>Bena</td>
<td>el</td>
<td>Greek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarian</td>
<td>el-polyton</td>
<td>Polytonic Greek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bm</td>
<td>Bambara</td>
<td>en-AU</td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bn</td>
<td>Bangla</td>
<td>en-CA</td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bo</td>
<td>Tibetan</td>
<td>en-GB</td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brx</td>
<td>Bodo</td>
<td>en-NZ</td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bs-Cyr</td>
<td>Bosnian</td>
<td>en-US</td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bs-Latn</td>
<td>Bosnian</td>
<td>en</td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bs</td>
<td>Bosnian</td>
<td>eo</td>
<td>Esperanto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ca</td>
<td>Catalan</td>
<td>es-MX</td>
<td>Spanish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ce</td>
<td>Chechen</td>
<td>es</td>
<td>Spanish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cgg</td>
<td>Chiga</td>
<td>et</td>
<td>Estonian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>chr</td>
<td>Cherokee</td>
<td>eu</td>
<td>Basque</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ckb</td>
<td>Central Kurdish</td>
<td>ewo</td>
<td>Ewondo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cop</td>
<td>Coptic</td>
<td>fa</td>
<td>Persian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
<td>Code</td>
<td>Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------------------</td>
<td>------</td>
<td>------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ff</td>
<td>Fulah</td>
<td>ksb</td>
<td>Shambala</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fi</td>
<td>Finnish</td>
<td>ksf</td>
<td>Bafia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fil</td>
<td>Filipino</td>
<td>ksh</td>
<td>Colognian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fo</td>
<td>Faroese</td>
<td>kw</td>
<td>Cornish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr</td>
<td>French</td>
<td>ky</td>
<td>Kyrgyz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr-BE</td>
<td>French</td>
<td>lag</td>
<td>Langi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr-CA</td>
<td>French</td>
<td>lb</td>
<td>Luxembourgish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr-CH</td>
<td>French</td>
<td>lg</td>
<td>Ganda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr-LU</td>
<td>French</td>
<td>lkt</td>
<td>Lakota</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fur</td>
<td>Friulian</td>
<td>ln</td>
<td>Lingala</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fy</td>
<td>Western Frisian</td>
<td>lo</td>
<td>Lao</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ga</td>
<td>Irish</td>
<td>lrc</td>
<td>Northern Luri</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gd</td>
<td>Scottish Gaelic</td>
<td>lt</td>
<td>Lithuanian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gl</td>
<td>Galician</td>
<td>lu</td>
<td>Luba-Katanga</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grc</td>
<td>Ancient Greek</td>
<td>luo</td>
<td>Luo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gsw</td>
<td>Swiss German</td>
<td>luy</td>
<td>Luyia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gu</td>
<td>Gujarati</td>
<td>lv</td>
<td>Latvian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>guz</td>
<td>Gusii</td>
<td>mas</td>
<td>Masai</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gv</td>
<td>Manx</td>
<td>mer</td>
<td>Meru</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ha-GH</td>
<td>Hausa</td>
<td>mfe</td>
<td>Morisyen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ha-NE</td>
<td>Hausa</td>
<td>mg</td>
<td>Malagasy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ha</td>
<td>Hausa</td>
<td>mgh</td>
<td>Makuwu-Meetto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>haw</td>
<td>Hawaiian</td>
<td>mgo</td>
<td>Meta</td>
<td></td>
<td></td>
</tr>
<tr>
<td>he</td>
<td>Hebrew</td>
<td>mk</td>
<td>Macedonian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi</td>
<td>Hindi</td>
<td>ml</td>
<td>Malayalam</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hr</td>
<td>Croatian</td>
<td>mn</td>
<td>Mongolian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hsb</td>
<td>Upper Sorbian</td>
<td>mr</td>
<td>Marathi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hu</td>
<td>Hungarian</td>
<td>ms-BN</td>
<td>Malay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hy</td>
<td>Armenian</td>
<td>ms-SG</td>
<td>Malay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ia</td>
<td>Interlingua</td>
<td>ms</td>
<td>Malay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>Indonesian</td>
<td>mt</td>
<td>Maltese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ig</td>
<td>Igbo</td>
<td>mua</td>
<td>Mundang</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ii</td>
<td>Sichuan Yi</td>
<td>my</td>
<td>Burmese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>Icelandic</td>
<td>mzn</td>
<td>Mazanderani</td>
<td></td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>Italian</td>
<td>naq</td>
<td>Nama</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
<td>nb</td>
<td>Norwegian Bokmål</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jgo</td>
<td>Ngomba</td>
<td>nd</td>
<td>North Ndebele</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jmc</td>
<td>Machame</td>
<td>ne</td>
<td>Nepali</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ka</td>
<td>Georgian</td>
<td>nl</td>
<td>Dutch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kab</td>
<td>Kabyle</td>
<td>nmg</td>
<td>Kwasi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kam</td>
<td>Kamba</td>
<td>nn</td>
<td>Norwegian Nynorsk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kde</td>
<td>Makonde</td>
<td>nnh</td>
<td>Ngiemboon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kea</td>
<td>Kabuverdianu</td>
<td>nus</td>
<td>Nuer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>khq</td>
<td>Koyra Chiini</td>
<td>nyn</td>
<td>Nyankole</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ki</td>
<td>Kikuyu</td>
<td>om</td>
<td>Oromo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kk</td>
<td>Kazakh</td>
<td>or</td>
<td>Odia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kkj</td>
<td>Kako</td>
<td>os</td>
<td>Ossetic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kl</td>
<td>Kalaallisut</td>
<td>pa-Arab</td>
<td>Punjabi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kln</td>
<td>Kalenjin</td>
<td>pa-Guru</td>
<td>Punjabi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>km</td>
<td>Khmer</td>
<td>pa</td>
<td>Punjabi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kn</td>
<td>Kannada</td>
<td>pl</td>
<td>Polish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ko</td>
<td>Korean</td>
<td>pms</td>
<td>Piedmontese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kok</td>
<td>Konkani</td>
<td>ps</td>
<td>Pashto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ks</td>
<td>Kashmiri</td>
<td>pt-BR</td>
<td>Portuguese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
<td>Code</td>
<td>Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------------</td>
<td>------</td>
<td>-------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pt-PT</td>
<td>Portuguese</td>
<td>sr</td>
<td>Serbian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
<td>sv</td>
<td>Swedish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qu</td>
<td>Quechua</td>
<td>sw</td>
<td>Swahili</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rm</td>
<td>Romansh</td>
<td>ta</td>
<td>Tamil</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rn</td>
<td>Rundi</td>
<td>te</td>
<td>Telugu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ro</td>
<td>Romanian</td>
<td>teo</td>
<td>Teso</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rof</td>
<td>Rombo</td>
<td>th</td>
<td>Thai</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ru</td>
<td>Russian</td>
<td>ti</td>
<td>Tigrinya</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rw</td>
<td>Kinyarwanda</td>
<td>tk</td>
<td>Turkmen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rwk</td>
<td>Rwa</td>
<td>to</td>
<td>Tongan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sa-Beng</td>
<td>Sanskrit</td>
<td>tr</td>
<td>Turkish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sa-Dev</td>
<td>Sanskrit</td>
<td>twq</td>
<td>Tasawaq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sa-Gujr</td>
<td>Sanskrit</td>
<td>tzm</td>
<td>Central Atlas Tamazight</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sa-Knda</td>
<td>Sanskrit</td>
<td>ug</td>
<td>Uyghur</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sa-Mlym</td>
<td>Sanskrit</td>
<td>uk</td>
<td>Ukrainian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sa-Telu</td>
<td>Sanskrit</td>
<td>ur</td>
<td>Urdu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sa</td>
<td>Sanskrit</td>
<td>uz-Arab</td>
<td>Uzbek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sah</td>
<td>Sakha</td>
<td>uz-Cyr</td>
<td>Uzbek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>saq</td>
<td>Samburu</td>
<td>uz-Latn</td>
<td>Uzbek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sbp</td>
<td>Sangu</td>
<td>uz</td>
<td>Uzbek</td>
<td></td>
<td></td>
</tr>
<tr>
<td>se</td>
<td>Northern Sami</td>
<td>vai-Latn</td>
<td>Vai</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seh</td>
<td>Sena</td>
<td>vai-Vaii</td>
<td>Vai</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ses</td>
<td>Koyraboro Senni</td>
<td>vai</td>
<td>Vai</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sg</td>
<td>Sango</td>
<td>vi</td>
<td>Vietnamese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shi-Latn</td>
<td>Tachelhit</td>
<td>vun</td>
<td>Vunjo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shi-Tfng</td>
<td>Tachelhit</td>
<td>wae</td>
<td>Walser</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shi</td>
<td>Tachelhit</td>
<td>xog</td>
<td>Soga</td>
<td></td>
<td></td>
</tr>
<tr>
<td>si</td>
<td>Sinhala</td>
<td>yav</td>
<td>Yangben</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sk</td>
<td>Slovak</td>
<td>yi</td>
<td>Yiddish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sl</td>
<td>Slovenian</td>
<td>yo</td>
<td>Yoruba</td>
<td></td>
<td></td>
</tr>
<tr>
<td>smn</td>
<td>Inari Sami</td>
<td>yue</td>
<td>Cantonese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn</td>
<td>Shona</td>
<td>zgh</td>
<td>Standard Moroccan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>so</td>
<td>Somali</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sq</td>
<td>Albanian</td>
<td>zh-Hans-HK</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Cyrl-BA</td>
<td>Serbian</td>
<td>zh-Hans-MO</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Cyrl-ME</td>
<td>Serbian</td>
<td>zh-Hans-SG</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Cyrl-XK</td>
<td>Serbian</td>
<td>zh-Hans</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Cyrl</td>
<td>Serbian</td>
<td>zh-Hant-HK</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Latin-BA</td>
<td>Serbian</td>
<td>zh-Hant-MO</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Latin-ME</td>
<td>Serbian</td>
<td>zh-Hant</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Latin-XK</td>
<td>Serbian</td>
<td>zh</td>
<td>Chinese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sr-Latin</td>
<td>Serbian</td>
<td>zu</td>
<td>Zulu</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \`\babelflimport\` with a valueless import.

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>aghem</td>
<td></td>
</tr>
<tr>
<td>akan</td>
<td></td>
</tr>
<tr>
<td>albanian</td>
<td></td>
</tr>
<tr>
<td>ancientgreek</td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Code</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Arabic</td>
<td>arabic</td>
</tr>
<tr>
<td>Arabic-Algeria</td>
<td>arabic-algeria</td>
</tr>
<tr>
<td>Arabic-DZ</td>
<td>arabic-DZ</td>
</tr>
<tr>
<td>Arabic-Morocco</td>
<td>arabic-morocco</td>
</tr>
<tr>
<td>Arabic-Morocco</td>
<td>arabic-morocco</td>
</tr>
<tr>
<td>Arabic-Morocco</td>
<td>arabic-morocco</td>
</tr>
<tr>
<td>Arabic-Syria</td>
<td>arabic-syria</td>
</tr>
<tr>
<td>Arabic-Syria</td>
<td>arabic-syria</td>
</tr>
<tr>
<td>Armenian</td>
<td>armenian</td>
</tr>
<tr>
<td>Assamese</td>
<td>assamese</td>
</tr>
<tr>
<td>Asturian</td>
<td>asturian</td>
</tr>
<tr>
<td>Asu</td>
<td>asu</td>
</tr>
<tr>
<td>Australian</td>
<td>australian</td>
</tr>
<tr>
<td>Austrian</td>
<td>austrian</td>
</tr>
<tr>
<td>Azerbaijani-Cyrillic</td>
<td>azerbajani-cyrillic</td>
</tr>
<tr>
<td>Azerbaijani-Cyril</td>
<td>azerbajani-cyril</td>
</tr>
<tr>
<td>Azerbaijani-Latin</td>
<td>azerbajani-latin</td>
</tr>
<tr>
<td>Azerbaijani-Latin</td>
<td>azerbajani-latin</td>
</tr>
<tr>
<td>Azerbaijani-Latin</td>
<td>azerbajani-latin</td>
</tr>
<tr>
<td>Azerbaijani-Latin</td>
<td>azerbajani-latin</td>
</tr>
<tr>
<td>Azerbaijani</td>
<td>azerbajani</td>
</tr>
<tr>
<td>Bafia</td>
<td>bafia</td>
</tr>
<tr>
<td>Bambara</td>
<td>bambara</td>
</tr>
<tr>
<td>Basaa</td>
<td>basaa</td>
</tr>
<tr>
<td>Basque</td>
<td>basque</td>
</tr>
<tr>
<td>Belarusian</td>
<td>belarusian</td>
</tr>
<tr>
<td>Bemba</td>
<td>bemba</td>
</tr>
<tr>
<td>Bena</td>
<td>bena</td>
</tr>
<tr>
<td>Bengali</td>
<td>bengali</td>
</tr>
<tr>
<td>Bodo</td>
<td>bodo</td>
</tr>
<tr>
<td>Bosnian-Cyrillic</td>
<td>bosnian-cyrillic</td>
</tr>
<tr>
<td>Bosnian-Cyrl</td>
<td>bosnian-cyrl</td>
</tr>
<tr>
<td>Bosnian-Latin</td>
<td>bosnian-latin</td>
</tr>
<tr>
<td>Bosnian</td>
<td>bosnian</td>
</tr>
<tr>
<td>Brazilian</td>
<td>brazilian</td>
</tr>
<tr>
<td>Breton</td>
<td>breton</td>
</tr>
<tr>
<td>British</td>
<td>british</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>bulgarian</td>
</tr>
<tr>
<td>Burmese</td>
<td>burmese</td>
</tr>
<tr>
<td>Canadian</td>
<td>canadian</td>
</tr>
<tr>
<td>Cantonese</td>
<td>cantonese</td>
</tr>
<tr>
<td>Catalan</td>
<td>catalan</td>
</tr>
<tr>
<td>Central Atlas Amazigh</td>
<td>centralatlastamazight</td>
</tr>
<tr>
<td>Central Kurdish</td>
<td>centralkurdish</td>
</tr>
<tr>
<td>Chechen</td>
<td>chechen</td>
</tr>
<tr>
<td>Cherokee</td>
<td>cherokee</td>
</tr>
<tr>
<td>Chiga</td>
<td>chiga</td>
</tr>
<tr>
<td>Chinese-Hans-Mo</td>
<td>chinese-hans-mo</td>
</tr>
<tr>
<td>Chinese-Hans</td>
<td>chinese-hans</td>
</tr>
<tr>
<td>Chinese-Hant-HK</td>
<td>chinese-hant-hk</td>
</tr>
<tr>
<td>Chinese-Hant-MO</td>
<td>chinese-hant-mo</td>
</tr>
<tr>
<td>Chinese-Hant</td>
<td>chinese-hant</td>
</tr>
</tbody>
</table>

13 The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.
ganda
gor
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inari
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowsorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northdebele
norwegianbokmal
norwegianynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
<table>
<thead>
<tr>
<th>Language</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>portuguese</td>
<td>slovak</td>
</tr>
<tr>
<td>punjabi-arab</td>
<td>slovene</td>
</tr>
<tr>
<td>punjabi-arabic</td>
<td>slovenian</td>
</tr>
<tr>
<td>punjabi-gurmukhi</td>
<td>soga</td>
</tr>
<tr>
<td>punjabi-guru</td>
<td>somali</td>
</tr>
<tr>
<td>punjabi</td>
<td>spanish-mexico</td>
</tr>
<tr>
<td>quechua</td>
<td>spanish</td>
</tr>
<tr>
<td>romanian</td>
<td>standardmoroccantamazight</td>
</tr>
<tr>
<td>rombo</td>
<td>swahili</td>
</tr>
<tr>
<td>rundi</td>
<td>swedish</td>
</tr>
<tr>
<td>russian</td>
<td>swissgerman</td>
</tr>
<tr>
<td>rwa</td>
<td>tachelhit-latin</td>
</tr>
<tr>
<td>sakha</td>
<td>tachelhit-latn</td>
</tr>
<tr>
<td>samburu</td>
<td>tachelhit-tifinagh</td>
</tr>
<tr>
<td>samin</td>
<td>tachelhit-tifinagh</td>
</tr>
<tr>
<td>sango</td>
<td>taita</td>
</tr>
<tr>
<td>sangu</td>
<td>tamil</td>
</tr>
<tr>
<td>sanskrit-beng</td>
<td>tasawaq</td>
</tr>
<tr>
<td>sanskrit-bengali</td>
<td>telugu</td>
</tr>
<tr>
<td>sanskrit-deva</td>
<td>teso</td>
</tr>
<tr>
<td>sanskrit-devanagari</td>
<td>thai</td>
</tr>
<tr>
<td>sanskrit-gujarati</td>
<td>tibetan</td>
</tr>
<tr>
<td>sanskrit-gujr</td>
<td>tigrinya</td>
</tr>
<tr>
<td>sanskrit-kannada</td>
<td>tongan</td>
</tr>
<tr>
<td>sanskrit-knda</td>
<td>turkish</td>
</tr>
<tr>
<td>sanskrit-malayalam</td>
<td>turkmen</td>
</tr>
<tr>
<td>sanskrit-mlym</td>
<td>ukenglish</td>
</tr>
<tr>
<td>sanskrit-telu</td>
<td>ukrainian</td>
</tr>
<tr>
<td>sanskrit-telugu</td>
<td>uppersorbian</td>
</tr>
<tr>
<td>sanskrit</td>
<td>urdu</td>
</tr>
<tr>
<td>scottishgaelic</td>
<td>usenglish</td>
</tr>
<tr>
<td>sena</td>
<td>usorbian</td>
</tr>
<tr>
<td>serbian-cyrillic-bosniaherzegovina</td>
<td>uyghur</td>
</tr>
<tr>
<td>serbian-cyrillic-kosovo</td>
<td>uzbek-arab</td>
</tr>
<tr>
<td>serbian-cyrillic-montenegro</td>
<td>uzbek-arabic</td>
</tr>
<tr>
<td>serbian-cyrillic</td>
<td>uzbek-cyrilic</td>
</tr>
<tr>
<td>serbian-cyril-ba</td>
<td>uzbek-cyril</td>
</tr>
<tr>
<td>serbian-cyril-me</td>
<td>uzbek-latin</td>
</tr>
<tr>
<td>serbian-cyril-xk</td>
<td>uzbek-latin</td>
</tr>
<tr>
<td>serbian-cyrl</td>
<td>uzbek-latin</td>
</tr>
<tr>
<td>serbian-latin-bosniaherzegovina</td>
<td>uzbek</td>
</tr>
<tr>
<td>serbian-latin-kosovo</td>
<td>vai-latin</td>
</tr>
<tr>
<td>serbian-latin-montenegro</td>
<td>vai-latin</td>
</tr>
<tr>
<td>serbian-latin</td>
<td>vai-vai</td>
</tr>
<tr>
<td>serbian-ltn-ba</td>
<td>vai-vaii</td>
</tr>
<tr>
<td>serbian-ltn-me</td>
<td>vai</td>
</tr>
<tr>
<td>serbian-ltn-xk</td>
<td>vietnam</td>
</tr>
<tr>
<td>serbian-ltn</td>
<td>vietnamese</td>
</tr>
<tr>
<td>serbian</td>
<td>vnjo</td>
</tr>
<tr>
<td>shambala</td>
<td>walser</td>
</tr>
<tr>
<td>Shona</td>
<td>welsh</td>
</tr>
<tr>
<td>Sichuanyi</td>
<td>westernfrisian</td>
</tr>
<tr>
<td>sinhala</td>
<td>yangben</td>
</tr>
</tbody>
</table>
Modifying and adding values to ini files

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.

1.14 Selecting fonts

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.

\documentclass{article}
\usepackage[swedish, bidi=default]{babel}
\babelprovide{import}{hebrew}
\babelfont{rm}{FreeSerif}
\begin{document}
\end{document}

\[14\text{See also the package combofont for a complementary approach.}\]
If on the other hand you have to resort to different fonts, you could replace the red line above with, say:

\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:

\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 could be problematic, and also 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 \babelfont provide 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 \setxxxxfont and \babelfont at the same time is discouraged, but very often works as expected. However, be aware with \setxxxxfont the language system will not be set by babel and should be set with fontspec if necessary.
TROUBLESHOOTING Package fontspec Warning: 'Language 'LANG' not available for font 'FONT' with script 'SCRIPT' 'Default' language used instead'.

This is not an error. This warning is shown by fontspec, not by babel. It could be irrelevant for English, but not for many other languages, including Urdu and Turkish. This is a useful and harmless warning, and if everything is fine with your document the best thing you can do is just to ignore it altogether.

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

This is not an error. babel assumes that if you are using \babel\font 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.

1.15 Modifying a language

Modifying the behavior of a language (say, the chapter “caption”), is sometimes necessary, but not always trivial.

• 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.

• 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}
```

• 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⟩.

• 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.)
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.

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.

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: \mylangchaptername not set. Please, define (babel) it in the preamble with something like: (babel) \renewcommand\mylangchaptername{..} (babel) Reported on input line 18.

In most cases, you will only need to define a few macros.

EXAMPLE If you need a language named arhinish:

\usepackage[danish]{babel}
\babelprovide{arhinish}
\renewcommand\arhinishchaptername{Chapitula}
\renewcommand\arhinishrefname{Refirenke}
\renewcommand\arhinishhyphenmins{22}
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{arinish} 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= \langle language-tag \rangle

New 3.13 Imports data from an ini file, including captions, date, and hyphenmins. 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. 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 could be written:

\babelprovide[import]{hungarian}

There are about 200 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 will show a warning about the current lack of suitability of the date format (french, breton, and occitan).

Besides \today, this option defines an additional command for dates: \\langle language ⟩ date, which takes three arguments, namely, year, month and day numbers. In fact, \today calls \langle language ⟩ today, which in turn calls \langle 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= \langle language-tag \rangle

Loads only the strings. For example:

\babelprovide[captions=hu]{hungarian}

hyphenrules= \langle language-list \rangle

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 – 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 \texttt{\textbackslash babelpatterns}, as for example:

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

In other engines it just suppresses hyphenation (because the pattern list is empty).

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

\textbf{EXAMPLE} Let’s assume your document is mainly in Polytonic Greek, but with some sections in Italian. Then, the first attempt should be:

\begin{verbatim}
\usepackage[italian, greek.polutonic]{babel}
\end{verbatim}

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

\begin{verbatim}
\usepackage[italian]{babel}
\babelprovide[import, main]{polytonicgreek}
\end{verbatim}

\begin{itemize}
\item \textit{script= \texttt{\langle script-name \rangle}}

\textbf{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.

\item \textit{language= \texttt{\langle language-name \rangle}}

\textbf{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.

\item \textit{alph= \texttt{\langle counter-name \rangle}}

Assigns to \texttt{\textbackslash alph} that counter. See the next section.

\item \textit{Alph= \texttt{\langle counter-name \rangle}}

Same for \texttt{\textbackslash Alph}.

\end{itemize}

A few options (only luatex) set some properties of the writing system used by the language. These properties are \textit{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

This option is much like an ‘event’ called when a character belonging to the script of this locale is found. 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). This option is not compatible with mapfont. Characters can be added with \babelcharproperty.

mapfont=direction

Assigns the font for the writing direction of this language (only with bidi=basic). Whenever possible, instead of this option use onchar, based on the script, which usually makes more sense. More precisely, what mapfont=direction means is, ‘when a character has the same direction as the script for the “provided” language, then change its font to that set for this language’. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

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 scripts, like Thai, and CJK.

intrapenalty=⟨penalty⟩

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

NOTE (1) If you need shorthands, you can define them with \usesshorthandselect 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

About thirty ini files define a field named digits.native. When it is present, two macros are created: \languagedigits and \languagecounter (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} % Telugu better with XeTeX
% Or also, if you want:
% \babelprovide[import, maparabic]{telugu}
\babelfont{rm}{Gautami}
\begin{document}
\telugudigits{1234}
\telugucounter{section}
\end{document}

Languages providing native digits in all or some variants are:
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 T\LaTeX code). This means the local digits have the correct bidirectional behavior (unlike `Numbers=Arabic` in fontspec, which is not recommended).

New 4.41 Many `ini` locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with xetex and luatex and are fully exposable (even inside an unprotected `\edef`). Currently, they are limited to numbers below 10000.

There are several ways to use them (for the available 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
- **Belarusan, Bulgarian, Macedonian, Serbian** lower, upper
- **Bengali** alphabetic
- **Coptic** epact, lower.letters
- **Hebrew** letters (neither geresh nor gershayim yet)
- **Hindi** alphabetic
- **Armenian** lower.letter, upper.letter
- **Japanese** hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
- **Georgian** letters
- **Greek** lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)
- **Khmer** consonant
- **Korean** consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, 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
Chinese      cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha,
             fullwidth.upper.alpha

New 3.45 In addition, native digits (in languages defining them) may be printed with the
numerical 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=..⟨year⟩⟨month⟩⟨day⟩\localedate
By default the calendar is the Gregorian, but a ini files may define strings for other
calendars (currently ar, ar-*, he, fa, hi.) 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).
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.

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 macros 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 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 language tag of the script used by this locale.
script.tag.opentype is the tag used by OpenType (usually, but not always, the same as
BCP 47).
\getlocaleproperty \langle \macro \rangle \{ \langle \locale \rangle \} \{ \langle \property \rangle \}

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 קרפ.

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.

NOTE ini files are loaded with \babelprovide and also when languages are selected if there is a \babelfont. To ensure the ini files are loaded (and therefore the corresponding data) even if these two conditions are not met, write \BabelEnsureInfo in the preamble.

\localeid

Each language in the babel sense has its own unique numeric identifier, which can be retrieved with \localeid.

NOTE The \localeid is not the same as the \language identifier, which refers to a set of hyphenation patterns (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 (where it makes sense) as an attribute, too.

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.

\babelhyphen \{ \langle \type \rangle \} \{ \langle \text \rangle \}

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.
• \texttt{\textbackslash 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.

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

• \texttt{\textbackslash babelhyphen\{empty\}} inserts a break opportunity without a hyphen at all.

• \texttt{\textbackslash babelhyphen\{text\}}} is a hard “hyphen” using \texttt{text} instead. A typical case is \texttt{\textbackslash 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: \texttt{\textbackslash babelhyphen\*soft} (which in most cases is equivalent to the original \textbackslash -, \texttt{\textbackslash babelhyphen\*hard}, etc.

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

There are also some differences with \texttt{\textbackslash LaTeX}: (1) the character used is that set for the current font, while in \texttt{\textbackslash LaTeX} it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative \texttt{\textbackslash hyphenchar} is -, like in \texttt{\textbackslash LaTeX}, but it can be changed to another value by redefining \texttt{\textbackslash 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).

\texttt{\textbackslash babelhyphenation\{} \langle language\rangle, \langle language\rangle, \ldots\}\{\langle exceptions\rangle\}\texttt{\textbackslash babelhyphenation}

**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). 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 \texttt{\textbackslash \LCODES\}'s done in \texttt{\textbackslash extras\langle lang\rangle} as well as the language-specific encoding (not set in the preamble by default). Multiple \texttt{\textbackslash babelhyphenation\}'s are allowed. For example:

\begin{quote}
\texttt{\textbackslash babelhyphenation\{Wal-hal-la Dar-bhan-ga\}}
\end{quote}

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

\texttt{\textbackslash babelpatterns\{} \langle language\rangle, \langle language\rangle, \ldots\}\{\langle patterns\rangle\}\texttt{\textbackslash babelpatterns\}

**New 3.9m** In \texttt{\textbackslash LaTeX only\}\textsuperscript{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 \texttt{\textbackslash \LCODES\}'s done in \texttt{\textbackslash extras\langle lang\rangle} as well as the language-specific encoding (not set in the preamble by default). Multiple \texttt{\textbackslash babelpatterns\}'s are allowed.

\textsuperscript{15}With \texttt{\textbackslash LaTeX 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.}
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 Khmer 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).

\babelposthyphenation \langle\langle hyphenrules-name\rangle\rangle\langle\langle lua-pattern\rangle\rangle\langle\langle replacement\rangle\rangle

New 3.37-3.39 With luatex it is now 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. No rules are currently provided by default, 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 (\[\lig\]), the replacement could be \{1|\lig|\lig\}, which maps \lig to \lig, and \lig to \lig, so that the diaeresis is removed.

This feature is activated with the first \babelposthyphenation.

See the babel wiki for a more detailed description and some examples. It also describes an additional replacement type with the key string.

EXAMPLE 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). For example, you can use the string replacement to 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
\babelposthyphenation{russian-latin}{(\[sz\])h} % Create rule
{
  { string = \{1|sz|$\$\} },
  remove
}

In other words, it is a quite general tool. (A counterpart \babelprehyphenation is on the way.)
1.21 Selection based on BCP 47 tags

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: \textit{fr-Latn-FR} $\rightarrow$ \textit{fr-Latn} $\rightarrow$ \textit{fr-FR} $\rightarrow$ \textit{fr}. Languages with the same resolved name are considered the same. Case is normalized before, so that \textit{fr-latn-fr} $\rightarrow$ \textit{fr-Latn-FR}. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

\begin{verbatim}
\documentclass{article}
\usepackage[danish]{babel}
\babeladjust{autoload.bcp47 = on }
\begin{document}
\today
\selectlanguage{fr-CA}
\today
\end{document}
\end{verbatim}

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 \texttt{\babeladjust} with the following parameters:

- \texttt{autoload.bcp47} with values on and off.
- \texttt{autoload.bcp47.options}, which are passed to \texttt{\babelprovide}; empty by default, but you may add \texttt{import} (features defined in the corresponding babel-\ldots.tex file might not be available).
- \texttt{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 \texttt{bcp47-}. You may change it with this key.

1.22 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either \texttt{\fontencoding} (low-level) or a language name (high-level). Even the
Latin script may require different encodings (i.e., sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete. Some languages sharing the same script define macros to switch it (e.g., `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but it 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.

\ensureasciimacrorule

\text

\ensureasciimacro New 3.9i This macro makes sure \texttt{⟨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, 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, `\ensureasciimacro` 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.

1.23 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 could 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 (e.g., 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 could 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 \texttt{graphical} elements, including the \texttt{picture} environment (with \texttt{pict2e} and \texttt{pgf/tikz}. Also, indexes and the like are under study, as well as math (there is progress in the latter, too, but for example cases 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:

\texttt{\begin{verbatim}
\let\ensureasciimacro\textasciimacro
\end{verbatim}}

\footnote{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.}

\footnote{But still defined for backwards compatibility.}
There are some package options controlling bidi writing.

\begin{itemize}
  \item \texttt{default} \hspace{1cm} \texttt{basic} \hspace{1cm} \texttt{basic-r} \hspace{1cm} \texttt{bidi-l} \hspace{1cm} \texttt{bidi-r}
\end{itemize}

\textbf{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 \texttt{xetex} and \texttt{pdftex} this is the only option.

In \texttt{luatex}, \texttt{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. \textbf{New 3.19} Finally, \texttt{basic} supports both L and R text, and it is the preferred method (support for \texttt{basic-r} is currently limited). (They are named \texttt{basic} mainly because they only consider the intrinsic direction of scripts and weak directionality.)

\textbf{New 3.29} In \texttt{xetex}, \texttt{bidi-r} and \texttt{bidi-l} resort to the package \texttt{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 \texttt{lua-bidibasic.tex} and \texttt{lua-secenum.tex}.

\textbf{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 \texttt{luatex} only.

\begin{verbatim}
\documentclass{article}
\usepackage[english, bidi=basic]{babel}
\babelprovide[onchar=ids fonts]{arabic}
\belfont{rm}{FreeSerif}
\begin{document}
ـﺑ(ﻲﻘﻳﺮﻏﻻا)ﻲﻨﻴﻠﻴﻬﻟاﺮﺼﻌﻟاﺔﻠﻴﻃبﺮﻌﻟاةﺮﻳﺰﺟﻪﺒﺷﺖﻓﺮﻋﺪﻗو

Arabia

وأ

Aravia )

ةﻴﻘﻳﺮﻏﻻﺎﺑ

Αραβία (ثﻼﺛنﺎﻣوﺮﻟامﺪﺨﺘﺳا، "ـﺑتﺎﺋدﺎﺑ

Arabia"

ﺎﻬﻧأﻻإ،ﺔﻴﺑﺮﻌﻟاةﺮﻳﺰﺠﻟاﻪﺒﺷﻦﻣﻖﻃﺎﻨﻣثﻼﺛﻰﻠﻋ

مﻮﻴﻟاﻪﻴﻠﻋفﺮﻌﺗﺎﻤﻣﺮﺒﻛأﺖﻧﺎﻛًﺔﻘﻴﻘﺣ

\end{document}
\end{verbatim}

\textbf{EXAMPLE} With \texttt{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 \texttt{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 \texttt{\babelprovide}, as illustrated:

\begin{verbatim}
\documentclass{book}
\usepackage[english, bidi=basic]{babel}
\babelprovide[onchar=ids fonts]{arabic}
\end{document}
\end{verbatim}
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 \textit{fuṣḥā l-ʻaṣr} (MSA) and \textit{fuṣḥā t-turāth} (CA).

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 \textthe must be defined to select the main language):

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

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

\texttt{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 dot-separated list (eg, layout=counters.contents.sectioning). 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, \textit{subsection}.\textit{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 could depend on the counter format.

With counters, \arabic is not only considered \texttt{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 \texttt{c2}.\texttt{c1}. Of course, you may always adjust the order by changing the language, if necessary.\[18\]

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

\[18\] Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.
**WARNING** As of April 2019 there is a bug with \texttt{\parshape} in \LaTeX (a \TeX primitive) which makes lists to be horizontally misplaced if they are inside a \texttt{vbox} (like \texttt{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 \texttt{xetex} and \texttt{pdftex}; in \LaTeX toc entries are \texttt{R} by default if the main language is \texttt{R}.

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

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

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

**tabular** required in \texttt{latex} for \texttt{R} \texttt{tabular} (it has been tested only with simple tables, so expect some readjustments in the future); ignored in \texttt{pdftex} or \texttt{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). \texttt{New 3.18}.

**graphics** modifies the \texttt{picture} environment so that the whole figure is \texttt{L} but the text is \texttt{R}. It \textit{does not} work with the standard \texttt{picture}, and \texttt{pict2e} is required if you want sloped lines. It attempts to do the same for \texttt{pgf/tikz}. Somewhat experimental. \texttt{New 3.32}.

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

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

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

\texttt{\babelsublr} \texttt{\langle\texttt{lr-text}\rangle}

Digits in \texttt{pdftex} must be marked up explicitly (unlike \texttt{latex} with \texttt{bidi=basic} or \texttt{bidi=basic-r} and, usually, \texttt{xetex}). This command is provided to set \texttt{\langle\texttt{lr-text}\rangle} in \texttt{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 \texttt{rl} counterpart. Any \texttt{\babelsublr} in \texttt{explicit} \texttt{L} mode is ignored. However, with \texttt{bidi=basic} and implicit \texttt{L}, it first returns to \texttt{R} and then switches to explicit \texttt{L}. To clarify this point, consider, in an \texttt{R} context:

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

There are three \texttt{R} blocks and two \texttt{L} blocks, and the order is \texttt{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 \texttt{\ref} in an \texttt{L} text inside \texttt{R}, the \texttt{L} text must be marked up explicitly; for example:

```latex
RTL A \foreignlanguage{english}{ltr text \thechapter{}} and still ltr} RTL B
```
Mainly for bidi text, but it could 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

```latex
\{\texttt{cmd}\}\{\texttt{local-language}\}\{\langle before\rangle}\{\langle after\rangle\}
```

#### New 3.17

Something like:

```latex
\BabelFootnote{\parsfootnote}{\languagename}{\{\}}
```

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

```latex
\footnote{\texttt{\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:

```latex
\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:

```latex
\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.

### 1.24 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 (e.g., \ProsodicMarksOn in Latin).

### 1.25 Hooks

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

\AddBabelHook{⟨lang⟩}{⟨name⟩}{⟨event⟩}{⟨code⟩}

The same name can be applied to several events. Hooks 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 \useshorthands* 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:

- **add dialect** (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.
- **default commands** Used (locally) in \StartBabelCommands.
- **encoded commands** (input, font encodings) Used (locally) in \StartBabelCommands. Both xetex and lualatex make sure the encoded text is read correctly.
- **stop commands** Used to reset the above, if necessary.
- **write** This event comes just after the switching commands are written to the aux file.
- **before extras** Just before executing \extras{language}. This event and the next one should not contain language-dependent code (for that, add it to \extras{language}).
- **after extras** Just after executing \extras{language}. For example, the following deactivates shorthands in all languages:

\AddBabelHook{noshort}{afterextras}{\languageshorthandsones(\emptyset)}

- **string process** 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}{string process}{\%\protected@edef\BabelString{\BabelString}}

- **initiate active** (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.
- **after reset** **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}. 
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.

\textbf{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 \texttt{\renewcommand} (it's up to you to make sure no toc type is duplicated).

## 1.26 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.

- **Afrikaans** afrikaans
- **Azerbaijani** azerbaijani
- **Basque** basque
- **Breton** breton
- **Bulgarian** bulgarian
- **Catalan** catalan
- **Croatian** croatian
- **Czech** czech
- **Danish** danish
- **Dutch** dutch
- **English** english, USenglish, american, UKenglish, british, canadian, australian, newzealand
- **Esperanto** esperanto
- **Estonian** estonian
- **Finnish** finnish
- **French** french, francais, canadien, acadian
- **Galician** galician
- **German** australian, german, germanb, ngerman, naustrian
- **Greek** greek, polutanikogreek
- **Hebrew** hebrew
- **Icelandic** icelandic
- **Indonesian** indonesian (bahasa, indon, bahasai)
- **Interlingua** interlingua
- **Irish Gaelic** irish
- **Italian** italian
- **Latin** latin
- **Lower Sorbian** lowersorbian
- **Malay** malay, melayu (bahasam)
- **North Sami** samin
- **Norwegian** norsk, nynorsk
- **Polish** polish
- **Portuguese** portuguese, brazilian (portuges, brazil)\footnote{The two last name comes from the times when they had to be shortened to 8 characters}
Romanian romanian
Russian russian
Scottish Gaelic 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, thai-cjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbanc, frenchle, ethiop and friulan.
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 \TeX.

1.27 Unicode character properties in luatex

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

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

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:
1.28 Tweaking some features

\texttt{\textbackslash babeladjust \{\textbackslash key-value-list\}}

New 3.36 Sometimes you might need to disable some babel features. Currently this macro understands the following keys (and only for luatex), with values on or off: bidi.text, bidi.mirroring, bidi.mapdigits, layout.lists, layout.tabular, linebreak.sea, linebreak.cjk. For example, you can set \texttt{\babeladjust\{bidi.text=off\}} if you are using an alternative algorithm or with large sections not requiring it. With luahbtex you may need bidi.mirroring=off. Use with care, because these options do not deactivate other related options (like paragraph direction with bidi.text).

1.29 Tips, workarounds, known issues and notes

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

- Both ltxdoc and babel use \texttt{\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 \mid it has to be done using the same method at the proper place, with

\texttt{\AtBeginDocument\{\DeleteShortVerb{\mid}\}}

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 \mid 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:

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

(A recent version of inputenc is required.)

- 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\footnote{This explains why \TeX{} assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, \texttt{\savinghyphcodes} is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.}. So, if you write a chunk of French text with \texttt{\foreignlanguage{fr}}, the apostrophes might not be taken into account. This is a limitation of \TeX{}, not of babel. Alternatively, you may use \texttt{\useshorthands} to activate ’ and \texttt{\defineshorthand} (the latter is called by the non-ASCII right quote).

- \texttt{\bibitem} is out of sync with \texttt{\selectlanguage} in the .aux file. The reason is \bibitem uses \texttt{\immediate} (and others, in fact), while \texttt{\selectlanguage} doesn’t. There is no known workaround.
• Babel does not take into account \normalfont codes 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):

\texttt{csquotes} Logical markup for quotes.
\texttt{iflang} Tests correctly the current language.
\texttt{hyphsubst} Selects a different set of patterns for a language.
\texttt{translator} An open platform for packages that need to be localized.
\texttt{siunitx} Typesetting of numbers and physical quantities.
\texttt{biblatex} Programmable bibliographies and citations.
\texttt{bicaption} Bilingual captions.
\texttt{babelbib} Multilingual bibliographies.
\texttt{microtype} Adjusts the typesetting according to some languages (kerning and spacing).
\hspace{1cm} Ligatures can be disabled.
\texttt{substitutefont} Combines fonts in several encodings.
\texttt{mkpattern} Generates hyphenation patterns.
\texttt{tracklang} Tracks which languages have been requested.
\texttt{ucharclasses} (\texttt{xetex}) Switches fonts when you switch from one Unicode block to another.
\texttt{zhspacing} Spacing for CJK documents in \texttt{xetex}.

1.30 Current and future work

The current work is focused on the so-called complex scripts in \texttt{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\footnote{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.} But that is the easy part, because they don’t require modifying the \TeX 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)-ből”, in Spanish an item labelled “3.\textsuperscript{2}” may be referred to as either “item 3." or “3.\textsuperscript{er} item”, and so on.

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

1.31 Tentative and experimental code

See the code section for \texttt{\foreignlanguage*} (a new starred version of \texttt{\foreignlanguage}). For old and deprecated functions, see the wiki.

\texttt{\babelprehyphenation} \texttt{\textbf{New 3.44}} Note it is tentative, but the current behavior for glyphs should be correct.
It is similar to \texttt{\textbackslash babelposthyphenation}, but (as its name implies) applied before hyphenation. There are other differences: (1) the first argument is the locale instead the name of hyphenation patterns; (2) in the search patterns \texttt{=} has no special meaning (\texttt{?} is still reserved, but currently unused); (3) in the replacement, discretionaries are not accepted, only remove, , and string = ... Currently it handles glyphs, not discretionaries or spaces (in particular, it will not catch the hyphen and you can’t insert or remove spaces). Also, you are limited to substitutions as done by \texttt{lpeg}, although a future implementation may alternatively accept \texttt{lpeg}.

Performance is still somewhat poor.

2 Loading languages with \texttt{language.dat}

\LaTeX{} and most engines based on it (\texttt{pdf\LaTeX{}}, \texttt{xetex}, \texttt{\epsilon\LaTeX{}}, the main exception being \texttt{luatex}) require hyphenation patterns to be preloaded when a format is created (eg. \texttt{\textbackslash input\LaTeX{}}, \texttt{Xe\LaTeX{}}, \texttt{pdf\LaTeX{}}). \texttt{babel} provides a tool which has become standard in many distributions and based on a “configuration file” named \texttt{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).

\textbf{New 3.9q} With \texttt{luatex}, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically \texttt{english}, which is preloaded always).

Until 3.9n, this task was delegated to the package \texttt{luatex-hyphen}, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named \texttt{language.dat.lua}, but now a new mechanism has been devised based solely on \texttt{language.dat}. \textbf{You must rebuild the formats} if upgrading from a previous version. You may want to have a local \texttt{language.dat} for a particular project (for example, a book on Chemistry).

2.1 Format

In that file the person who maintains a \LaTeX{} environment has to record for which languages he has hyphenation patterns and in which files these are stored. 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:

\begin{verbatim}
% File : language.dat
% Purpose : tell \LaTeX{} what files with patterns to load.
english english.hyphenations =british

dutch hyphen.dutch exceptions.dutch % Nederlands
german hyphen.ger
\end{verbatim}

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.

\footnote{This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.}

\footnote{The loader for lua(e)tex is slightly different as it’s not based on babel but on \texttt{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 \texttt{language.dat}.}

\footnote{This is because different operating systems sometimes use very different file-naming conventions.}

\footnote{This is not a new feature, but in former versions it didn’t work correctly.}
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 could 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{}.

- 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 \TeX (quotes are entered as ` and `'). Other good choices are characters which are not used in a certain context (e.g., ` 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 `mailto:lang` except for umlauthigh and friends, `mailto:deactivate`, `mailto:(non)`frenchspacing, and language-specific macros. Use always, if possible, `mailto:save` and `mailto: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 \textrm{mailto: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 \texttt{mailto:latintext} is deprecated.

• Please, for “private” internal macros do not use the `mailto: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

Now language files are “outsourced” and are located in a separate directory (/\texttt{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 \texttt{tfm}, \texttt{vf}, \texttt{ps1}, \texttt{otf}, \texttt{mf} files and the like, but also \texttt{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: \url{http://www.texnia.com/incubator.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.

\footnote{\texttt{mailto:But not removed, for backward compatibility.}}
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-outerversion 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.

\langle lang⟩hyphenmins The macro \langle 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@tribute 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 \TeX 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, \TeX 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`.

(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This `.fd` file will instruct \TeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 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).

```latex
\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@attribute{<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:

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

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 \TeX{} 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 \bbl@deactivate

The command \bbl@activate is used to change the way an active character expands. \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 \bbl@remove@special

The \TeX{}book states: “Plain \TeX{} includes a macro called \dospecials that is essentially a set 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 \dospecials. \TeX{} adds another macro called \@sanitize representing the same character set, but without the curly braces. The macros \bbl@add@special\langle char\rangle and \bbl@remove@special\langle char\rangle add and remove the character \langle char\rangle to these two sets.

3.5 Support for saving macro definitions

Language definition files may want to redefine 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. It takes one argument, \langle csname\rangle, 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, \langle variable\rangle. 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\langle control sequence\rangle\{\langle \TeX{} code\rangle\} can be used to extend the definition of \addto\langle control sequence\rangle\{\langle \TeX{} code\rangle\} can be used to extend the definition of

\footnote{This mechanism was introduced by Bernd Raichle.}
a macro. The macro need not be defined (i.e., 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 could 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 discretionary) 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 \bbl@nonfrenchspacing  
The commands \bbl@frenchspacing and \bbl@nonfrenchspacing can be used to 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. This is the only new feature requiring changes in language files if you want to make use of it.

Furthermore, it must be activated explicitly, with the package option \strings. If there is no \strings, these blocks are ignored, except \SetCases (and except if forced as described below). In other words, 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 (i.e., local declarations apply until the next \StartBabelCommands or \EndBabelCommands). An \lst may contain several series of this kind.

Thanks to this new feature, string values and string language switching are not mixed anymore. No need of \adddo. If the language is french, just redefine \frenchchaptername.

\StartBabelCommands \SetCases \EndBabelCommands  
The \SetCases 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 \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” is a name to be used as value in package option \strings, optionally followed by extra info about the encodings to be used. The name \unicode must be used for \xetex (the key strings has also other two special values: \generic and \encoded).
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 \texttt{charset=} followed by a charset, which if given sets how the strings should be translated to the internal representation used by the engine, typically \texttt{utf8}, which is the only value supported currently (default is no translations). Note \texttt{charset} is applied by \texttt{lualatex} and \texttt{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 \texttt{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 \texttt{strings=encoded}. Blocks without a selector are read always if the key \texttt{strings} has been used. They provide fallback values, and therefore must be the last blocks; 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). Blocks without a selector can be activated explicitly with \texttt{strings=generic} (no block is taken into account except those).

With \texttt{strings=encoded}, strings in those blocks are set as default (internally, \texttt{?}). With \texttt{strings=encoded} strings are protected, but they are correctly expanded in \texttt{\MakeUppercase} and the like. If there is no key \texttt{strings}, string definitions are ignored, but \texttt{\SetCase}s are still honored (in an encoded way).

The \texttt{\langle category \rangle} is either \texttt{captions}, \texttt{date} or \texttt{extras}. You must stick to these three categories, even if no error is raised when using other name\textsuperscript{28} It may be empty, too, but in such a case using \texttt{\SetString} is an error (but not \texttt{\SetCase}).

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

% A real example is:

\texttt{%}
\StartBabelCommands{austrian}{date}
  \[unicode, fontenc=TU EU1 EU2, charset=utf8\]
\SetString{monthiname}{Jänner}
\EndBabelCommands

\texttt{%}
\StartBabelCommands{german,austrian}{date}
  \[unicode, fontenc=TU EU1 EU2, charset=utf8\]
\SetString{monthiiiname}{März}
\EndBabelCommands

\texttt{%}
\StartBabelCommands{austrian}{date}
  \SetString{monthiname}{J"{a}änner}
\EndBabelCommands

\texttt{%}
\StartBabelCommands{german}{date}
  \SetString{monthiname}{Januar}
\EndBabelCommands

\texttt{%}
\StartBabelCommands{german,austrian}{date}
  \SetString{monthiiname}{Februar}
  \SetString{monthiiiname}{M"{a}rz}
  \SetString{monthivname}{April}
\EndBabelCommands

\textsuperscript{28}In future releases further categories may be added.
When used in \texttt{ldf} files, previous values of \texttt{⟨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, i.e., if \texttt{date⟨language⟩} exists).

\begin{verbatim}
\StartBabelCommands
*{⟨language-list⟩}{⟨category⟩}{⟨selector⟩}
\EndBabelCommands
\AfterBabelCommands
\SetString
\SetStringLoop
\end{verbatim}

The starred version just forces strings to take a value – if not set as package option, 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.\footnote{This replaces in 3.9g a short-lived \texttt{UseStrings} which has been removed because it did not work.}
\SetCase \{\textit{map-list}\}\{\texttt{toupper-code}\}\{\texttt{tolower-code}\}
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \texttt{\let\BB\bb} and \texttt{\uccode or \lccode} (although for the reasons explained above, changes in lc/uc codes may not work). A \textit{map-list} is a series of macros using the internal format of \@ucclist (eg, \texttt{\bb\BB\cc\CC}). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in \LaTeX, we could set for Turkish:

\StartBabelCommands{turkish}\{\texttt{ot1enc, fontenc=OT1}\}
\SetCase
  \{\uccode"10=`I\relax\}
  \{\lccode"I="10\relax\}
\EndBabelCommands
\StartBabelCommands{turkish}\{\texttt{unicode, fontenc=TU EU1 EU2, charset=utf8}\}
\SetCase
  \{\uccode`i=`İ\relax
  \uccode`ı=`ı\relax
  \lccode`İ=`i\relax
  \lccode`I=`ı\relax\}
\EndBabelCommands
\StartBabelCommands{turkish}\{}
\SetCase
  \{\uccode`i=9D\relax
  \uccode`I=19\relax
  \lccode`I=9D\relax
  \lccode`I=19\relax\}
\EndBabelCommands

(Note the mapping for OT1 is not complete.)

\SetHyphenMap \{\texttt{to-lower-macros}\}
\textbf{New 3.9g} Case mapping serves in \LaTeX for two unrelated purposes: case transforms (upper/lower) and hyphenation. \SetCase handles the former, while hyphenation is handled by \SetHyphenMap and controlled with the package option \texttt{hyphenmap}. So, even if internally they are based on the same \LaTeX primitive (\texttt{lccode}), babel sets them separately. There are three helper macros to be used inside \SetHyphenMap:

- \texttt{\BabelLower}\{\texttt{uccode}\}\{\texttt{lccode}\} is similar to \texttt{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 \texttt{hyphenmap=first}).

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

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

An example is (which is redundant, because these assignments are done by both luatex and xetex):
This macro is not intended to fix wrong mappings done by Unicode (which are the default in both \texttt{xetex} and \texttt{luatex}) – if an assignment is wrong, fix it directly.

4 Changes

4.1 Changes in \texttt{babel} version 3.9

Most of the changes in version 3.9 were related to bugs, either to fix them (there were lots), or to provide some alternatives. Even new features like \texttt{\babelhyphen} are intended to solve a certain problem (in this case, the lacking of a uniform syntax and behavior for shorthands across languages). These changes are described in this manual in the corresponding place. A selective list follows:

- \texttt{\select@language} did not set \texttt{\languagename}. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was \texttt{german}, a \texttt{\select@language{spanish}} had no effect.

- \texttt{\foreignlanguage} and other \texttt{\textasteriskcentered} messed up \texttt{\textasteriskcentered<language>}. Scripts, encodings and many other things were not switched correctly.

- The \texttt{\textasteriskcentered:ENC} mechanism for hyphenation patterns used the encoding of the \texttt{previous} language, not that of the language being selected.

- \texttt{'} (with \texttt{active\textbackslash acute}) had the original value when writing to an auxiliary file, and things like an infinite loop could happen. It worked incorrectly with \texttt{^} (if activated) and also if deactivated.

- Active chars where not reset at the end of language options, and that lead to incompatibilities between languages.

- \texttt{\textasteriskcentered\textbackslash math} raised and error with a conditional.

- \texttt{\textasteriskcentered\textbackslash alias\textbackslash shorthand} didn’t work (or only in a few and very specific cases).

- \texttt{\textasteriskcentered\textbackslash @english} was defined incorrectly (using \texttt{\let} instead of \texttt{\chardef}).

- \texttt{\textasteriskcentered\textbackslash ldf} files not bundled with \texttt{babel} were not recognized when called as global options.

Part II

Source code

\texttt{babel} is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use \texttt{babel} only as documented (except, of course, if you want to explore and test them – you can post suggestions about multilingual issues to kadingira@tug.org on http://tug.org/mailman/listinfo/kadingira).
5  Identification and loading of required files

*Code documentation is still under revision.*
The following description is no longer valid, because switch and plain have been merged into babel.def.
The babel package after unpacking consists of the following files:

- **switch.def** defines macros to set and switch languages.
- **babel.def** defines the rest of macros. It has two parts: a generic one and a second one only for \LaTeX{}.
- **babel.sty** is the \LaTeX{} package, which set options and load language styles.
- **plain.def** defines some \LaTeX{} macros required by babel.def and provides a few tools for Plain.
- **hyphen.cfg** is the file to be used when generating the formats to load hyphenation patterns.

The babel installer extends docstrip with a few “pseudo-guards” to set “variables” used at installation time. They are used with `<@name@>` at the appropriated places in the source code and shown below with ⟨⟨name⟩⟩. That brings a little bit of literate programming.

6  locale directory

A required component of babel is a set of ini files with basic definitions for about 200 languages. They are distributed as a separate zip file, not packed as dtx. With them, babel will fully support Unicode engines.
Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and polytonic Greek, and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.
This is a preliminary documentation.
ini files contain the actual data; tex files are currently just proxies to the corresponding ini files.
Most keys are self-explanatory.

- **charset** the encoding used in the ini file.
- **version** of the ini file
- **level** “version” of the ini specification, which keys are available (they may grow in a compatible way) and how they should be read.
- **encodings** a descriptive list of font encodings.
- **[captions]** section of captions in the file charset
- **[captions.licr]** same, but in pure ASCII using the LICR
- **date.long** fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (eg, MMMM for the month name) and anything outside is text. In addition, [ ] is a non breakable space and [ . ] is an abbreviation dot.

Keys may be further qualified in a particular language with a suffix starting with a uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter).
*Multi-letter* qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (which start always with a lowercase case). There is an exception, however: the section counters has been devised to have arbitrary keys, so you can add lowercased keys if you want.
7 Tools

Do not use the following macros in .ldf files. They may change in the future. This applies mainly to those recently added for replacing, trimming and looping. The older ones, like \bbl@afterfi, will not change. We define some basic macros which just make the code cleaner. \bbl@add is now used internally instead of \addto because of the unpredictable behavior of the latter. Used in babel.def and in babel.sty, which means in \LaTeX is executed twice, but we need them when defining options and babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

\begin{verbatim}
\longdef\bbl@add@list#1#2{\edef#1{{\bbl@ifunset{\bbl@stripslash#1}{}\ifx#1\@empty\else#1,\fi}\ifx#1\@empty\else#2\fi}}
\end{verbatim}

\bbl@add@list This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated. It presumes expandable character strings.

\begin{verbatim}
\longdef\bbl@afterelse#1\else#2\fi{\fi#1}
\end{verbatim}

\bbl@afterelse Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the \else and \fi parts of an \if\fi-statement\footnote{This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.}. These macros will break if another \if...\fi statement appears in one of the arguments and it is not enclosed in braces.

\begin{verbatim}
\longdef\bbl@exp#1{%
\begingroup
#1
\endgroup
\end{verbatim}

\bbl@exp Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and readable. Here \ stands for \noexpand and \<..> for \noexpand applied to a built macro name (the latter does not define the macro if undefined to \relax, because it is created locally). The result may be followed by extra arguments, if necessary.

\begin{verbatim}
\longdef\bbl@exp#1{%
\begingroup
\end{verbatim}

\bbl@exp
The following piece of code is stolen (with some changes) from keyval, by David Carlisle. It defines two macros: \bbl@trim and \bbl@trim@def. The first one strips the leading and trailing spaces from the second argument and then applies the first argument (a macro, \toks@ and the like). The second one, as its name suggests, defines the first argument as the stripped second argument.

```
\def\bbl@tempa#1{%
  \long\def\bbl@trim##1##2{%
    \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
  \def\bbl@trim@c{%
    \ifx\bbl@trim@a\@sptoken
      \expandafter\bbl@trim@b
    \else
      \expandafter\bbl@trim@b\expandafter#1%
    \fi}%
  \long\def\bbl@trim@b#1##1 \@nil{
    \bbl@trim@i##1}}%
\bbl@tempa{ }
\long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
\long\def\bbl@trim@def#1{\bbl@trim{\def#1}}
```

To check if a macro is defined, we create a new macro, which does the same as \@ifundefined. However, in an \epsilon-tex engine, it is based on \ifcsname, which is more efficient, and do not waste memory.

```
\begingroup
\gdef\bbl@ifunset#1{%
  \expandafter\ifx\csname#1\endcsname\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
\bbl@ifunset{ifcsname}%
\endgroup
```

A tool from url, by Donald Arseneau, which tests if a string is empty or space.

```
\def\bbl@ifblank#1{%
  \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}%
\long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
```

For each element in the comma separated <key>=<value> list, execute <code> with #1 and #2 as the key and the value of current item (trimmed). In addition, the item is passed verbatim as #3. With the <key> alone, it passes \@empty (ie, the macro thus named, not an empty argument, which is what you get with <key>= and no value).
A for loop. Each item (trimmed), is \#1. It cannot be nested (it's doable, but we don't need it).

\bbl@vforeach#1#2{% 
\bbl@forcmd##1{#2} 
\bbl@fornext#1,\@nil,} 
\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

\bbl@replace
\bbl@replace#1#2#3{% in #1 -> repl #2 by #3 
\toks@{}% 
\def\bbl@replace@aux##1#2##2#2{% 
\toks@\expandafter{\the\toks@##1} 
\else 
\toks@\expandafter{\the\toks@##1} 
\bbl@afterfi 
\bbl@replace@aux##2#2% 
\fi} 
\expandafter\bbl@replace@aux#1#2\bbl@nil#2% 
\edef#1{\the\toks@}}

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace elax by ho, then \relax becomes \rhe). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does not work is in \bbl@TG@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I'm not sure checking the replacement is really necessary or just paranoia).

\ifx\detokenize@undefined\else % Unused macros if old Plain TeX 
\bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}\#2->\#3}{\relax} 
\def\bbl@tempa{#1}% 
\def\bbl@tempb{#2}% 
\def\bbl@tempe{#3}% 
\def\bbl@sreplace#1#2#3{% 
\begingroup 
\expandafter\bbl@parsedef\meaning#1\relax 
\def\bbl@tempc{#2}% 
\edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc} 
\def\bbl@tempd{#3}% 
\edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd} 
\bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing 
\ifin@ 
\bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}% Expanded an executed below as 'uplevel'
Two further tools. \texttt{\bbl@samestring} first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \texttt{\bbl@engine} takes the following values: 0 is pdf\TeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

Some files identify themselves with a \LaTeX macro. The following code is placed before them to define (and then undefine) if not in \LaTeX:

Some files identify themselves with a \LaTeX macro. The following code is placed before them to define (and then undefine) if not in \LaTeX.

\texttt{\language} Plain \LaTeX version 3.0 provides the primitive \texttt{\language} that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in switch\_def and hyphen\_cfg; the latter may seem redundant, but remember babel doesn't requires loading switch\_def in the format.
Another counter is used to store the last language defined. For pre-3.0 formats an extra counter has to be allocated.

This macro was introduced for \TeX < 2. Preserved for compatibility.

Now we make sure all required files are loaded. When the command \AtBeginDocument doesn’t exist we assume that we are dealing with a plain-based format or \TeX 2.09. In that case the file plain.def is needed (which also defines \AtBeginDocument, and therefore it is not loaded twice). We need the first part when the format is created, and \orig@dump is used as a flag. Otherwise, we need to use the second part, so \orig@dump is not defined (plain.def undefines it).

Check if the current version of switch.def has been previously loaded (mainly, hyphen.cfg). If not, load it now. We cannot load babel.def here because we first need to declare and process the package options.

7.2 The Package File (\LaTeX, babel.sty)

This file also takes care of a number of compatibility issues with other packages and defines a few additional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

Many of the following options don’t do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user.

The first two options are for debugging.
If the format created a list of loaded languages (in `\bbl@languages`), get the name of the 0-th to show the actual language used. Also available with base, because it just shows info.
The first 'real' option to be processed is base, which set the hyphenation patterns then resets \texttt{ver@babel.sty} so that \LaTeX forgets about the first loading. After a subset of \texttt{babel.def} has been loaded (the old \texttt{switch.def}) and \texttt{\textbackslash{AfterBabelLanguage}} defined, it exits.

Now the base option. With it we can define (and load, with \texttt{luatex}) hyphenation patterns, even if we are not interested in the rest of babel.

\begin{verbatim}
7.3 base

\section{key=value options and other general option}

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to \texttt{\BabelModifiers} at \texttt{\bbl@load@language}; when no modifiers have been given, the former is \texttt{\relax}. How modifiers are handled are left to language styles; they can use \texttt{|in@|}, loop them with \texttt{|@for|} or load \texttt{keyval}, for example.

\begin{verbatim}
\section{key=value and another general options}
\end{verbatim}
\end{verbatim}
The next option tells babel to leave shorthand characters active at the end of processing the package. This is not the default as it can cause problems with other packages, but for those who want to use the shorthand characters in the preamble of their documents this can help.

```
\DeclareOption{KeepShorthandsActive}{}
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
\DeclareOption{debug}{}
\DeclareOption{noconfigs}{}
\DeclareOption{showlanguages}{}
\DeclareOption{silent}{}
\DeclareOption{mono}{}
\DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
% Don’t use. Experimental. TODO.
\newif\ifbbl@single
\DeclareOption{selectors=off}{\bbl@singletrue}
\DeclareOption{selectors=off}{\bbl@singletrue}
```

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax `<key>=<value>`, the second one loads the requested languages, except the main one if set with the key `main`, and the third one loads the latter. First, we “flag” valid keys with a nil value.

```
\let\bbl@opt@shorthands\@nnil
\let\bbl@opt@config\@nnil
\let\bbl@opt@main\@nnil
\let\bbl@opt@headfoot\@nnil
\let\bbl@opt@layout\@nnil
```

The following tool is defined temporarily to store the values of options.

```
\def\bbl@tempa#1=#2\bbl@tempa{\bbl@csarg\ifx{opt@#1}\@nnil
\bbl@csarg\edef{opt@#1}{#2}\else
\bbl@error
{Bad option `#1=#2'. Either you have misspelled the\%
key or there is a previous setting of `#1'. Valid\%
keys are, among others, `shorthands', `main', `bidi',\%
`strings', `config', `headfoot', `safe', `math'.}%
{See the manual for further details.}
\fi}
```

Now the option list is processed, taking into account only currently declared options (including those declared with a =), and `<key>=<value>` options (the former take precedence). Unrecognized options are saved in `\bbl@language@opts`, because they are language options.
Now we finish the first pass (and start over).

7.4 Conditional loading of shorthands

If there is no shorthands=chars, the original babel macros are left untouched, but if there is, these macros are wrapped (in babel.def) to define only those given. A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=....

\bbl@trace{Conditional loading of shorthands}
\def\bbl@sh@string#1{% 
  \ifx#1\@empty\else 
    \ifx#1t\string~% 
    \else\ifx#1c\string,\else\string#1\fi\fi 
  \expandafter\bbl@sh@string
  \fi}
\ifx\bbl@opt@shorthands\@nnil 
  \def\bbl@ifshorthand#1#2#3{#2}%
\else\ifx\bbl@opt@shorthands\@empty 
  \def\bbl@ifshorthand#1#2#3{#3}%
\else 
  The following macro tests if a shorthand is one of the allowed ones.
  \def\bbl@ifshorthand#1{% 
    \bbl@xin\string#1\bbl@opt@shorthands\%
  \ifin@
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
\fi\fi

We make sure all chars in the string are ‘other’, with the help of an auxiliary macro defined above (which also zaps spaces).

\edef\bbl@opt@shorthands{% 
  \expandafter\@firstoftwo
\else
  \expandafter\@secondoftwo
\fi}

The following is ignored with shorthands=off, since it is intended to take some aditional actions for certain chars.

\bbl@ifshorthand{'}% 
{\PassOptionsToPackage{activeacute}{babel}}{}
\bbl@ifshorthand{'}% 
{\PassOptionsToPackage{activegrave}{babel}}{} 
\fi\fi
With \texttt{headfoot=lang} we can set the language used in heads/feet. For example, in \texttt{babel/3796} just adds \texttt{headfoot=english}. It misuses \texttt{@resetactivechars} but seems to work.

```
\ifx\bbl@opt@headfoot\@nnil\else
  \@addto@macro\@resetactivechars{%
    \set@typeset@protect
    \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
    \let\protect\noexpand}
\fi
```

For the option \texttt{safe} we use a different approach – \texttt{\bbl@opt@safe} says which macros are redefined (B for bibs and R for refs). By default, both are set.

```
\ifx\bbl@opt@safe\@undefined
  \def\bbl@opt@safe{BR}
\fi
```

```
\ifx\bbl@opt@main\@nnil\else
  \edef\bbl@language@opts{\
    \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
    \bbl@opt@main}
\fi
```

For layout an auxiliary macro is provided, available for packages and language styles. Optimization: if there is no layout, just do nothing.

```
\bbl@trace{Defining IfBabelLayout}
\ifx\bbl@opt@layout\@nnil
  \newcommand\IfBabelLayout[3]{#3}%
\else
  \newcommand\IfBabelLayout[1]{%
    \@expandtwoargs\in@{.#1}.}{.\bbl@opt@layout.}%
    \ifin@
    \expandafter\@firstoftwo
    \else
    \expandafter\@secondoftwo
  \fi}
\fi
```

**Common definitions. In progress.** Still based on \texttt{babel.def}, but the code should be moved here.

```
\input babel.def
```

### 7.5 Cross referencing macros

The \texttt{\LaTeX} book states:

> The key argument is any sequence of letters, digits, and punctuation symbols; upper- and lowercase letters are regarded as different.

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.

When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category ‘letter’ or ‘other’.

The following package options control which macros are to be redefined.

```
\DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
\DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
\DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
```

70
First we open a new group to keep the changed setting of \protect local and then we set the \@safe@actives switch to true to make sure that any shorthand that appears in any of the arguments immediately expands to its non-active self.

\@testdef An internal \LaTeX macro used to test if the labels that have been written on the .aux file have changed. It is called by the \enddocument macro.

Now that we made sure that \@testdef still has the same definition we can rewrite it. First we make the shorthands ‘safe’. Then we use \@bbi@tempa as an ‘alias’ for the macro that contains the label which is being checked. Then we define \@bbi@tempb just as \@newl@bel does it. When the label is defined we replace the definition of \@bbi@tempa by its meaning. If the label didn’t change, \@bbi@tempa and \@bbi@tempb should be identical macros.

The same holds for the macro \@ref that references a label and \@pageref to reference a page. We make them robust as well (if they weren't already) to prevent problems if they should become expanded at the wrong moment.
\cite  The macro used to cite from a bibliography, \cite, uses an internal macro, \@citex. It is this internal macro that picks up the argument(s), so we redefine this internal macro and leave \cite alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the second argument.

```latex
\bbl@xin@(B)\bbl@opt@safe
\ifin@
  \bbl@redefine\@citex[#1]#2{\@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
  \org@@citex[#1]{\@tempa}}%
\else
  \def\@citex[#1]#2{\@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
  \org@@citex[#1][#2]{\@tempa}}%
\fi
```

Unfortunately, the packages natbib and cite need a different definition of \@citex... To begin with, natbib has a definition for \@citex with three arguments... We only know that a package is loaded when \begin{document} is executed, so we need to postpone the different redefinition.

```latex
\AtBeginDocument{%
  \@ifpackageloaded{natbib}{%
    \bbl@redefine\@citex[#1][#2][#3]{\@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
      \org@@citex[#1][#2][#3]{\@tempa}}%
  }{}
}
```

The package cite has a definition of \@citex where the shorthands need to be turned off in both arguments.

```latex
\AtBeginDocument{%
  \@ifpackageloaded{cite}{%
    \def\@citex[#1]#2{\@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
  }{}}
```

\nocite  The macro \nocite which is used to instruct BiBT\TeX{} to extract uncited references from the database.

```latex
\bbl@redefine\nocite#1{%
  \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
```

\bibcite  The macro that is used in the .aux file to define citation labels. When packages such as natbib or cite are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where \@safe@activestrue is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order to determine during .aux file processing which definition of \bibcite is needed we define \bibcite in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibcite. This new definition is then activated.

```latex
\bbl@redefine\bibcite{%
  \bbl@cite@choice
  \bibcite}
```

\bbl@bibcite  The macro \bbl@bibcite holds the definition of \bibcite needed when neither natbib nor cite is loaded.

```latex
\def\bbl@bibcite#1#2{%
  \org@bibcite{#1}{\@safe@activesfalse#2}}
```
The macro \bbl@cite@choice determines which definition of \bibcite is needed. First we give \bibcite its default definition.

470 \def\bbl@cite@choice{%
471  \global\let\bibcite\bbl@bibcite
472  \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
473  \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
474  \global\let\bbl@cite@choice\relax}

When a document is run for the first time, no .aux file is available, and \bibcite will not yet be properly defined. In this case, this has to happen before the document starts.

475 \AtBeginDocument{\bbl@cite@choice}

One of the two internal \LaTeX macros called by \bibitem that write the citation label on the .aux file.

476 \bbl@redefine{\@bibitem}{%\@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
477 else
478 \let\org@nocite\nocite
479 \let\org@@citex\@citex
480 \let\org@bibcite\bibcite
481 \let\org@@bibitem\@bibitem
482 \fi

7.6 Marks

Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \markright and \markboth somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the ‘headfoot’ options is used. We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

484 \bbl@trace{Marks}
485 \IfBabelLayout{sectioning}
486 {\iffx\bbl@opt@headfoot\@nil
487  \@addto@macro\@resetactivechars{%
488    \set@typeset@protect
489    \select@language@x{\bbl@main@language}%
490    \let\protect\noexpand
491    \edef\thepage{% TODO. Only with bidi. See also above
492      \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}}
493 }{\fi}
494 {\iffbbl@single\else
495  \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
496  \markright#1{%
497    \bbl@ifblank{#1}%
498    \{\org@markright{}\}%
499    \{\toks@{}\}%
500    \bbl@exp{%
501      \\org@markright{\protect{\foreignlanguage{\languagename}{\protect{\bbl@restore@actives the\toks@}}}}}}%
502  \fi}

The definition of \markboth is equivalent to that of \markright, except that we need two token registers. The document classes report and book define and set the headings for the page. While doing so they also store a copy of \markboth in \@mkboth. Therefore we need to check whether \@mkboth has already been set. If so we need to do that again with the
new definition of \markboth. (As of Oct 2019, \LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

503 \ifx\@mkboth\markboth
504 \def\bbl@tempc{\let\@mkboth\markboth}
505 \else
506 \def\bbl@tempc{}
507 \fi
508 \bbl@ifunset{\markboth }\bbl@redefine\bbl@redefinerobust
509 \markboth#1#2{%
510 \protected@edef\bbl@tempb##1{%
511 \protect\foreignlanguage
512 {%\languagename}\protect\bbl@restore@actives##1}%
513 \bbl@ifblank{#1}%
514 \{\toks@\}%
515 \bbl@ifblank{#2}%
516 \{\@temptokena\}%
517 \bbl@exp{\org@markboth{\the\toks@}{\the\@temptokena}}%
518 \bbl@tempc
519 \fi} % end ifbbl@single, end \IfBabelLayout

7.7 Preventing clashes with other packages

7.7.1 ifthen
\ifthenelse Sometimes a document writer wants to create a special effect depending on the page a certain fragment of text appears on. This can be achieved by the following piece of code:

\ifthenelse{\isodd{\pageref{some:label}}}{}
\{code for odd pages\}
\{code for even pages\}

In order for this to work the argument of \isodd needs to be fully expandable. With the above redefinition of \pageref it is not in the case of this example. To overcome that, we add some code to the definition of \ifthenelse to make things work. We want to revert the definition of \pageref and \ref to their original definition for the first argument of \ifthenelse, so we first need to store their current meanings. Then we can set the \@safe@actives switch and call the original \ifthenelse. In order to be able to use shorthands in the second and third arguments of \ifthenelse the resetting of the switch and the definition of \pageref happens inside those arguments.

522 \bbl@trace{Preventing clashes with other packages}
523 \bbl@xin{(R)}\bbl@opt@safe
524 \ifin@
525 \AtBeginDocument{%
526 \@ifpackageloaded@ifthen{%2#3{%%
527 \bbl@redefine@long@ifthenelse#1#2#3{%%
528 \let\bbl@temp@pref\pageref
529 \let\pageref\org@pageref
530 \let\bbl@temp@ref\ref
531 \let\ref\org@ref
532 \@safe@activestrue
533 \org@ifthenelse{#1}%
534 \{\let\pageref\bbl@temp@pref
535 \let\ref\bbl@temp@ref
536 \@safe@activesfalse

74
\@vpageref \vrefpagenum \Ref

Whenthe package varioref is in use we need to modify its internal command \@vpageref in order to prevent problems when an active character ends up in the argument of \vref. The same needs to happen for \vrefpagenum.

The package varioref defines \Ref to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of \ref. So we employ a little trick here. We redefine the (internal) command \Ref to call \org@ref instead of \ref. The disadvantage of this solution is that whenever the definition of \Ref changes, this definition needs to be updated as well.

\expandafter\def\csname Ref \endcsname#1{\protected@edef\@tempa{\org@ref{#1}}}\expandafter\MakeUppercase\@tempa}

Delaying the activation of the shorthand characters has introduced a problem with the hhline package. The reason is that it uses the ‘:’ character which is made active by the french support in babel. Therefore we need to reload the package when the ‘:’ is an active character. Note that this happens after the category code of the @-sign has been changed to other, so we need to temporarily change it to letter again.

\AtEndOfPackage{\AtBeginDocument{% \@ifpackageloaded{hhline}{% \@ifpackageloaded{varioref}{% \bbl@redefine@vpageref#1[#2]{% \@safe@activesfalse} \bbl@redefine@vrefpagenum#1#2{% \@safe@activesfalse} \bbl@redefine@vrefpagenum#1{#2}{% \@safe@activesfalse}}} \expandafter\def\csname normal@char\string: endcsname#1{% \protected@edef\@tempa{\org@ref{#1}}}\expandafter\MakeUppercase\@tempa}

A number of interworking problems between babel and hyperref are tackled by hyperref itself. The following code was introduced to prevent some annoying warnings but it broke
bookmarks. This was quickly fixed in hyperref, which essentially made it no-op. However, it will not removed for the moment because hyperref is expecting it. TODO. Still true?

\begin{verbatim}
\AtBeginDocument{%
  \ifx\pdfstringdefDisableCommands\@undefined\else
    \pdfstringdefDisableCommands{\languageshorthands{system}}\fi}
\end{verbatim}

7.7.5 fancyhdr

The package fancyhdr treats the running head and foot lines somewhat differently as the standard classes. A symptom of this is that the command \foreignlanguage which babel adds to the marks can end up inside the argument of \MakeUppercase. To prevent unexpected results we need to define \FOREIGNLANGUAGE here.

\begin{verbatim}
\DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%#1\lowercase{\foreignlanguage{#1}}}\end{verbatim}

substitutefontfamily

The command \substitutefontfamily creates an .fd file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names. This command is deprecated. Use the tools provided by \LaTeXX.

\begin{verbatim}
\def\substitutefontfamily#1#2#3{%
  \lowercase{\immediate\openout15=#1#2.fd\relax}%
  \immediate\write15{%
    \string\ProvidesFile{#1#2.fd}[^{\the\year/\two@digits{\the\month}/\two@digits{\the\day}}]
    \string\DeclareFontFamily{#1}{#2}{}\string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}\string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}\string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}\string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}\string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}\string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}\string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}\string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}}%
  \closeout15}
\end{verbatim}

7.8 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeXX and \LaTeXX always come out in the right encoding. There is a list of non-ASCII encodings. Unfortunately, fontenc deletes its package options, so we must guess which encodings has been loaded by traversing \@filelist to search for ⟨enc⟩.enc.def. If a non-ASCII has been loaded, we define versions of \TeX and \LaTeX for them using \ensurescii. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

\ensurescii

\begin{verbatim}
\bbl@trace{Encoding and fonts}
\newcommand{\BabelNonASCII}{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
\newcommand{\BabelNonText}{TS1,T3,TS3}
\let\org\TeX\TeX
\let\org\LaTeX\LaTeX
\let\ensurescii@firstofone
\end{verbatim}
Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at \begin{document}, which Latin fontencoding to use.

\texttt{\latinencoding}    When text is being typeset in an encoding other than ‘latin’ (OT1 or T1), it would be nice to still have Roman numerals come out in the Latin encoding. So we first assume that the current encoding at the end of processing the package is the Latin encoding.

\texttt{\AtEndOfPackage{\edef{\latinencoding}{\cfencoding}}}

But this might be overruled with a later loading of the package fontenc. Therefore we check at the execution of \begin{document} whether it was loaded with the T1 option. The normal way to do this (using \@ifpackageloaded) is disabled for this package. Now we have to revert to parsing the internal macro \@filelist which contains all the filenames loaded.

\texttt{\IfPackageLoaded{fontspec}{\edef{\latinencoding}{\cfencoding}}}
\latintext Then we can define the command \latintext which is a declarative switch to a Latin font-encoding. Usage of this macro is deprecated.

\textlatin This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

7.9 Basic bidi support

Work in progress. This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.

It is loosely based on rlbabel.def, but most of it has been developed from scratch. This babel module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I’ve also looked at arabi (by Yousef Jabri), which is compatible with babel.

There are two ways of modifying macros to make them “bidi”, namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like rlbabel did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

• pdftex provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.

• xetex is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX grouping.

• luatex can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As Lua\TeX-j a shows, vertical typesetting is possible, too.

As a first step, add a handler for bidi and digits (and potentially other processes) just before luaotfload is applied, which is loaded by default by \Lua\TeX. Just in case, consider the possibility it has not been loaded.
\let\bbl@activate@preotf\relax % only once
\directlua{
  Babel = Babel or {}
  %
  function Babel.pre_otfload_v(head)
    if Babel.numbers and Babel.digits_mapped then
      head = Babel.numbers(head)
    end
    if Babel.bidi_enabled then
      head = Babel.bidi(head, false, dir)
    end
    return head
  end
  %
  function Babel.pre_otfload_h(head, gc, sz, pt, dir)
    if Babel.numbers and Babel.digits_mapped then
      head = Babel.numbers(head)
    end
    if Babel.bidi_enabled then
      head = Babel.bidi(head, false, dir)
    end
    return head
  end
  %
  luatexbase.add_to_callback('pre_linebreak_filter',
    Babel.pre_otfload_v,
    'Babel.pre_otfload_v',
    luatexbase.priority_in_callback('pre_linebreak_filter',
      'luaotfload.node_processor') or nil)
  %
  luatexbase.add_to_callback('hpack_filter',
    Babel.pre_otfload_h,
    'Babel.pre_otfload_h',
    luatexbase.priority_in_callback('hpack_filter',
      'luaotfload.node_processor') or nil)
}
\fi

The basic setup. In luatex, the output is modified at a very low level to set the \bodydir to the \pagedir.
\bbl@trace{Loading basic (internal) bidi support}
\ifodd\bbl@engine
  \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
    \let\bbl@beforeforeign\leavevmode
    \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
    \RequirePackage{luatexbase}
    \bbl@activate@preotf
    \directlua{
      require('babel-data-bidi.lua')
      \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
        require('babel-bidi-basic.lua')
      \or
        require('babel-bidi-basic-r.lua')
      \fi}
    % TODO - to locale_props, not as separate attribute
    \newattribute\bbl@attr@dir
    % TODO. I don't like it, hackish:
    \bbl@exp{\output{\bodydir\pagedir\the\output}}
  \AtEndOfPackage{\EnableBabelHook{babel-bidi}}

79
Now come the macros used to set the direction when a language is switched. First the (mostly) common macros.

\bbl@trace{Macros to switch the text direction}
\edef\bbl@alscripts{,Arabic,Syriac,Thaana,}
\edef\bbl@rscripts{% TODO. Base on codes ??
,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
Old Hungarian,Old Hungarian,Lydia,Mandaean,Manichaean,%
Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
Old South Arabian,}
\def\bbl@provide@dirs#1{%
\bbl@xin{(csname \bbl@xnsname@#1\endcsname){\bbl@alscripts\bbl@rscripts}}%
\ifin@
  \global\bbl@csarg\chardef{wdir@#1}\@ne
\bbl@xin{(csname \bbl@xnsname@#1\endcsname){\bbl@alscripts}}% 
\ifin@
  \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
  \fi
\else
  \global\bbl@csarg\chardef{wdir@#1}\z@
  \fi
\ifodd\bbl@engine
  \bbl@csarg\ifcase{wdir@#1}\fi
  \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }% 
  \or
  \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }% 
  \or
  \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }% 
\fi
\fi}
\def\bbl@switchdir{%
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}
\bbl@exp{\\bbl@setdirs\bbl@cl{wdir}}
}\def\bbl@setdirs#1{% TODO - math
\ifcase\bbl@select@type % TODO - strictly, not the right test
  \bbl@bodydir{#1}%
  \bbl@pardir{#1}%
\fi
\bbl@textdir{#1}}
\AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
\DisableBabelHook{babel-bidi}

Now the engine-dependent macros. TODO. Must be moved to the engine files?
\ifodd\bbl@engine % luatex=1
  \chardef\bbl@thetextdir\z@
  \chardef\bbl@thepardir\z@
  \def\bbl@getluadir#1{%
  \directlua{
    if tex.#1dir == 'TLT' then
      tex.sprint('0')
    elseif tex.#1dir == 'TRT' then
      tex.sprint('1')
    end}}
  \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
    \ifcase#3\relax
      \ifcase\bbl@getluadir{#1}\relax
        #2 TLT\relax
      \else
        #2 TRT\relax
    \fi
\fi
\def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
  \ifcase#3\relax
    \ifcase\bbl@getluadir{#1}\relax
      #2 TLT\relax
    \else
      #2 TRT\relax
  \fi
\fi
\setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
The following command is executed only if there is a right-to-left script (once). It activates the \everypar hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

A tool for weak L (mainly digits). We also disable warnings with hyperref.

7.10 Local Language Configuration

At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension .cfg. For instance the file norsk.cfg will be loaded when the language definition file norsk.lda is loaded. For plain-based formats we don't want to override the definition of \loadlocalcfg from plain.def.
Just to be compatible with \LaTeX 2.09 we add a few more lines of code. TODO. Necessary? Correct place? Used by some ldf file?

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.
Another way to extend the list of 'known' options for babel was to create the file bblopts.cfg in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new .ldf file loading the actual one. You can also set the name of the file with the package option config=<name>, which will load <name>.cfg instead.

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in bbl@language@opts are assumed to be languages (note this list also contains the language given with main). If not declared above, the names of the option and the file are the same.

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an ldf exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.
And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored. The options have to be processed in the order in which the user specified them (except, of course, global options, which \TeX processes before):

\def\AfterBabelLanguage#1{%
  \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}\}
\DeclareOption*{}
\ProcessOptions*

This finished the second pass. Now the third one begins, which loads the main language set with the key main. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. Then execute directly the option (because it could be used only in main). After loading all languages, we deactivate \AfterBabelLanguage.

\bbl@trace{Option 'main'}
\ifsymb\bbl@opt@main\@nnil\else
  \edef\bbl@tempa{@classoptionslist,\bbl@language@opts}
  \let\bbl@tempc@empty
  \bbl@for\bbl@tempb\bbl@tempa{%
    \bbl@xin{@,\bbl@tempb,},{\bbl@loaded,}%
    \ifin@\edef\bbl@tempc{{\bbl@tempb}}\fi}
  \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
  \expandafter\bbl@tempa\bbl@loaded,\@nnil
  \ifx\bbl@tempb\bbl@tempc\else
    \bbl@warning{%
      Last declared language option is '\bbl@tempc',\\%
      but the last processed one was '\bbl@tempb'.\\%
      The main language cannot be set as both a global\\%
      and a package option. Use 'main=\bbl@tempc' as\\%
      option. Reported}\\%
  \fi
\fi
\else
  \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
  \ExecuteOptions{\bbl@opt@main}
  \DeclareOption*{}
  \ProcessOptions*
\fi
\def\AfterBabelLanguage{%
  \bbl@error{Too late for \string\AfterBabelLanguage}%
  {Languages have been loaded, so I can do nothing}}

In order to catch the case where the user forgot to specify a language we check whether \bbl@main@language, has become defined. If not, no language has been loaded and an error message is displayed.

\ifsymb\bbl@main@language\@undefined\else
  \bbl@info{%
    You haven't specified a language. I'll use 'nil'\\%
    as the main language. Reported}\\%
  \bbl@load@language(nil)
\fi

\end{package}
8 The kernel of Babel (babel.def, common)

The kernel of the babel system is currently stored in babel.def. The file babel.def contains most of the code. The file hyphen.cfg is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns. Because plain \TeX users might want to use some of the features of the babel system too, care has to be taken that plain \TeX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain \TeX and \LaTeX, some of it is for the \LaTeX case only.

Plain formats based on etex (etex, xetex, luatex) don’t load hyphen.cfg but etex.src, which follows a different naming convention, so we need to define the babel names. It presumes language.def exists and it is the same file used when formats were created.

8.1 Tools

The file babel.def expects some definitions made in the \LaTeX style file. So, In \LaTeX 2.09 and Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore and alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel.

BabelModifiers can be set too (but not sure it works).

Exit immediately with 2.09. An error is raised by the sty file, but also try to minimize the number of errors.
And continue.

9 Multiple languages

This is not a separate file (switch.def) anymore.
Plain \TeX version 3.0 provides the primitive \language that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

\adddialect The macro \adddialect can be used to add the name of a dialect or variant language, for which an already defined hyphenation table can be used.

\bbl@iflanguage executes code only if the language l@ exists. Otherwise raises and error. The argument of \bbl@fixname has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s intended to fix a long-standing bug when \foreignlanguage and the like appear in a \MakeXXXcase. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note l@ is encapsulated, so that its case does not change.

After a name has been ‘fixed’, the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.
We first need a couple of macros for a simple BCP 47 lookup. It also makes sure, with \bbl@bcpcase, casing is the correct one, so that sr-latin-ba becomes fr-Latin-BA. Note \#4 may contain some \@empty's, but they are eventually removed. \bbl@bcplookup either returns the found ini or it is \relax.

\def\bbl@bcpcase#1#2#3#4\@@#5{% 
  \ifx\@empty#3% 
    \uppercase{\def#5{#1#2}}% 
  \else 
    \uppercase{\def#5{#1}}% 
    \lowercase{\edef#5{#5#2#3#4}}% 
  \fi 
}\def\bbl@bcplookup#1-#2-#3-#4\@@{% 
  \let\bbl@bcp\relax 
  \lowercase{\def\bbl@tempa{#1}}% 
  \ifx\@empty#2% 
    \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}% 
  \else \ifx\@empty#3% 
    \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb 
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}{}% 
    \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}% 
  \else 
    \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb 
    \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc 
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}{}% 
    \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}{}% 
    \fi 
  \fi 
}\let\bbl@autoload@options\@empty 
\let\bbl@initoload\relax 
\def\bbl@provide@locale{% 
  \ifx\babelprovide\@undefined 
    \bbl@error{For a language to be defined on the fly \textquote{base} is not enough, and the whole package must be \textquote{loaded}. Either delete the \textquote{base} option or request the languages explicitly} 
    \{See the manual for further details.\} 
  \else 
    \bbl@provide@locale\% 
    \IfFileExists{babel-\bbl@templatex-\bbl@tempc.ini}{}% 
    \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@templatex-\bbl@tempc.ini}{}% 
    \fi 
  \fi 
}\def\bbl@use{\nameuse\bbl@bcp@map@\languagename} 
}\let\bbl@autoload@options\@empty 
\let\bbl@initoload\relax 
\def\bbl@provide@locale{% 
  \ifx\babelprovide\@undefined 
    \bbl@error{For a language to be defined on the fly \textquote{base} is not enough, and the whole package must be \textquote{loaded}. Either delete the \textquote{base} option or request the languages explicitly} 
    \{See the manual for further details.\} 
  \else 
    \bbl@provide@locale\% 
    \IfFileExists{babel-\bbl@templatex-\bbl@tempc.ini}{}% 
    \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@templatex-\bbl@tempc.ini}{}% 
    \fi 
  \fi 
}% TODO. Option to search if loaded, with \LocaleForEach 
\let\bbl@auxname\languagename 
\bbl@ifunset{bbl@bcp@map@\languagename}{} 
\let\bbl@autoload@options\@empty 
\let\bbl@initoload\relax 
\def\bbl@provide@locale{% 
  \ifx\babelprovide\@undefined 
    \bbl@error{For a language to be defined on the fly \textquote{base} is not enough, and the whole package must be \textquote{loaded}. Either delete the \textquote{base} option or request the languages explicitly} 
    \{See the manual for further details.\} 
  \else 
    \bbl@provide@locale 
    \IfFileExists{babel-\bbl@templatex-\bbl@tempc.ini}{} 
    \ifx\bbl@bcp\relax 
    \IfFileExists{babel-\bbl@templatex-\bbl@tempc.ini}{} 
    \fi 
  \fi 
}\def\bbl@use{\nameuse\bbl@bcp@map@\languagename} 
}
\ifbbl@bcpallowed
\expandafter\ifx\csname date\languagename\endcsname\relax
\expandafter\bbl@bcplookup\languagename-@empty-@empty-@empty@@
\ifx\bbl@bcp\relax% Returned by \bbl@bcplookup
\edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
\edef\localename{\bbl@bcp@prefix\bbl@bcp}%
\expandafter\ifx\csname date\languagename\endcsname\relax
\let\bbl@initoload\bbl@bcp
\bbl@exp{\[\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
\let\bbl@initoload\relax
\fi
\fi
\fi
\fi
\fi
\fi
\expandafter\ifx\csname date\languagename\endcsname\relax
\IfFileExists{babel-\languagename.tex}%
{\bbl@exp{\[\babelprovide[\bbl@autoload@options]{\languagename}}}%
{%
\fi}
\fi

\iflanguage
Users might want to test (in a private package for instance) which language is currently active. For this we provide a test macro, \iflanguage, that has three arguments. It checks whether the first argument is a known language. If so, it compares the first argument with the value of \language. Then, depending on the result of the comparison, it executes either the second or the third argument.

\def\iflanguage#1{\bl@iflanguage{#1}{%\ifnum\csname l@#1\endcsname=\language\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}}

9.1 Selecting the language

\selectlanguage
The macro \selectlanguage checks whether the language is already defined before it performs its actual task, which is to update \language and activate language-specific definitions.

\let\bbl@select@type\z@
\edef\selectlanguage{\noexpand\protect\selectlanguage}

Because the command \selectlanguage could be used in a moving argument it expands to \protect\selectlanguage. Therefore, we have to make sure that a macro \protect exists. If it doesn't it is \let to \relax.

\ifx\@undefined\protect\let\protect\relax\fi

The following definition is preserved for backwards compatibility. It is related to a trick for 2.09.

\let\xstring\string

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.
But when the language change happens inside a group the end of the group doesn’t write anything to the auxiliary files. Therefore we need \texttt{TEx}’s \texttt{aftergroup} mechanism to help us. The command \texttt{aftergroup} stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence \texttt{\bbl@pop@language} to be executed at the end of the group. It calls \texttt{\bbl@set@language} with the name of the current language as its argument.

The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called \texttt{\bbl@language@stack} and initially empty.

```
\def\bbl@language@stack{}
```

When using a stack we need a mechanism to push an element on the stack and to retrieve the information afterwards.

The stack is simply a list of languagenames, separated with a ‘+’ sign; the push function can be simple:

```
\def\bbl@push@language{%
  \ifx\languagename\@undefined\else
    \xdef\bbl@language@stack{\languagename+\bbl@language@stack}\xdef\bbl@ifrestoring\@secondoftwo\fi}
```

Retrieving information from the stack is a little bit less simple, as we need to remove the element from the stack while storing it in the macro \texttt{\languagename}. For this we first define a helper function.

```
\def\bbl@pop@lang#1+%2&3{%
  \edef\languagename{#1}\xdef#3{#2}}
```

There is a somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before \texttt{\bbl@pop@lang} is executed \texttt{TEx} first \texttt{expands} the stack, stored in \texttt{\bbl@language@stack}. The result of that is that the argument string of \texttt{\bbl@pop@lang} contains one or more language names, each followed by a ‘+’-sign (zero language names won’t occur as this macro will only be called after something has been pushed on the stack) followed by the ‘&’-sign and finally the reference to the stack.

```
\let\bbl@ifrestoring\@secondoftwo
\def\bbl@pop@lang#1+%2&3{%
  \edef\languagename{#1}\xdef#3{#2}}
\let\bbl@ifrestoring\@firstoftwo
\expandafter\bbl@set@language\expandafter{\languagename}%
\let\bbl@ifrestoring\@secondoftwo
```

Once the name of the previous language is retrieved from the stack, it is fed to \texttt{\bbl@set@language} to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of \texttt{\localeid}. This means \texttt{\l@...} will be reserved for hyphenation patterns (so that two locales can share the same rules).

```
\chardef\localeid\z@ % No real need for a new counter
\def\bbl@id@last{0} % No real need for a new counter
\def\bbl@id@assign{%
  \bbl@ifunset{bbl@id@@\languagename}%
  \count@\bbl@id@last\relax
```

91
The unprotected part of \selectlanguage.

The macro \bbl@set@language takes care of switching the language environment and of writing entries on the auxiliary files. For historical reasons, language names can be either language of \language. To catch either form a trick is used, but unfortunately as a side effect the catcodes of letters in \languagename are messed up. This is a bug, but preserved for backwards compatibility. The list of auxiliary files can be extended by redefining \BabelContentsFiles, but make sure they are loaded inside a group (as aux, toc, lof, and lot do) or the last language of the document will remain active afterwards.

We also write a command to change the current language in the auxiliary files.

\bbl@set@language

\bbl@set@language{toc,lof,lot}
\def\bbl@set@language##1{% from selectlanguage, pop@
% The old buggy way. Preserved for compatibility.
\edef\languagename{%
\ifnum\escapechar=\string#1\@empty
\else\string#1\@empty\fi%
\ifcat\relax\noexpand#1%
\def\localename{#1}% This one has the correct catcodes
\else
\def\localename{??}%}
\iffalse\expandafter\string\languagename\empty\fi%
\expandafter\string\languagename\empty\fi%
\let\localename\languagename
\else
\bbl@info{Using \string\languagename' instead of 'language' is\%
deprecated. If what you want is to use a\%
macro containing the actual locale, make\%
sure it does not not match any language.\%
Reported}%
\iffalse\expandafter{\expandafter\def\expandafter\localename\expandafter{\languagename}}\fi%
\else
\def\localename{??}%
\fi
\iffalse\expandafter{\expandafter\def\expandafter\localename\expandafter{\languagename}}\fi%
\else
\def\localename{??}% This one has the correct catcodes
\fi
First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring \LaTeX in a certain pre-defined state. The name of the language is stored in the control sequence \language. Then we have to redefine \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the \noextras⟨lang⟩ command at definition time by expanding the \csname primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of \selectlanguage, and calling these macros. The switching of the values of \lefthyphenmin and \righthyphenmin is somewhat different. First we save their current values, then we check if \⟨lang⟩hyphenmins is defined. If it is not, we set default values (2 and 3), otherwise the values in \⟨lang⟩hyphenmins will be used.
\def\bbl@switch#1{% from select@, foreign@
% make sure there is info for the language if so requested
\bbl@ensureinfo{#1}%
% restore
\originalTeX
\expandafter\def\expandafter\originalTeX\expandafter{%
\csname noextras#1\endcsname
\let\originalTeX\@empty
\babel@beginsave}%
\bbl@usehooks{afterreset}{%}
\languageshorthands{none}%
% set the locale id
\bbl@id@assign
% switch captions, date
\ifcase\bbl@select@type
\ifhmode
\hskip\z@skip % trick to ignore spaces
\csname captions#1\endcsname\relax
\csname date#1\endcsname\relax
\loop\ifdim\lastskip>\z@\unskip\repeat\unskip
\else
\csname captions#1\endcsname\relax
\csname date#1\endcsname\relax
\fi
\else
\ifhmode
\hskip\z@skip % trick to ignore spaces
\bbl@xin@{,captions,}{,\bbl@select@opts,}\
\ifin@
\csname captions#1\endcsname\relax
\fi
\bbl@xin@{,date,}{,\bbl@select@opts,}\
\ifin@ % if \foreign... within \langle lang\rangle date
\csname date#1\endcsname\relax
\fi
\loop\ifdim\lastskip>\z@\unskip\repeat\unskip
\else
\bbl@xin@{,captions,}{,\bbl@select@opts,}\
\ifin@
\csname captions#1\endcsname\relax
\fi
\bbl@xin@{,date,}{,\bbl@select@opts,}\
\ifin@
\csname date#1\endcsname\relax
\fi
\fi
% switch extras
\bbl@usehooks{beforeextras}{}%
\csname extras#1\endcsname\relax
\bbl@usehooks{afterextras}{}%
% > babel-ensure
% > babel-sh-<short>
% > babel-bidi
% > babel-fontspec
% hyphenation - case mapping
\ifcase\bbl@opt@hyphenmap\or
\def\BabelLower##1##2{\lccode##1=##2\relax}%
\ifnum\bbl@hymapsel>4\else
94
\fi
%
otherlanguage The otherlanguage environment can be used as an alternative to using the \selectlanguage declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

The \ignorespaces command is necessary to hide the environment when it is entered in horizontal mode.

otherlanguage* The otherlanguage environment is meant to be used when a large part of text from a different language needs to be typeset, but without changing the translation of words such as 'figure'. This environment makes use of \foreignlanguage.

\foreignlanguage The \foreignlanguage command is another substitute for the \selectlanguage command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument. Unlike \selectlanguage this command doesn't switch everything, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a
group and assumes the `\extras{lang}` command doesn't make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

`\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a 'text' command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

(3.11) `\foreignlanguage*` is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around `\par`, things like `\hangindent` are not reset). Do not use it in production, because its semantics and its syntax may change (and very likely will, or even it could be removed altogether). Currently it enters in vmode and then selects the language (which in turn sets the paragraph direction).

(3.11) Also experimental are the hook `foreign` and `foreign*`. With them you can redefine \BabelText which by default does nothing. Its behavior is not well defined yet. So, use it in horizontal mode only if you do not want surprises.

In other words, at the beginning of a paragraph `\foreignlanguage` enters into hmode with the surrounding lang, and with `\foreignlanguage*` with the new lang.

```latex
\providecommand{\bbl@beforeforeign}{}
\edef{\foreignlanguage}{\noexpand\protect\expandafter{\noexpand\csname foreignlanguage\endcsname}}
\expandafter{\def{\csname foreignlanguage\endcsname}}{\@ifstar{\bbl@foreign@s}{\bbl@foreign@x}{}}
\providecommand{\bbl@foreign@x}[3][]{\begingroup\def{\bbl@select@opts}{#1}\let{\BabelText}{\@firstofone}\bbl@beforeforeign\foreignlanguage{#2}\bbl@usehooks{foreign}{}\BabelText{#3}}
\def{\bbl@foreign@s}{\begingroup{\par}\let{\BabelText}{\@firstofone}\foreignlanguage{#1}\bbl@usehooks{foreign*}{}\bbl@dirparastext\BabelText{#2}\par}{\par}\endgroup}
```

`\foreignlanguage` This macro does the work for `\foreignlanguage` and the other `\language*` environment. First we need to store the name of the language and check that it is a known language. Then it just calls `\bbl@switch`.

```latex
\edef{\foreignlanguage}{\% set name\edef{\languagename}{#1}\ifa{\bbl@usedategroup}\bbl@add{\bbl@select@opts}{,date,}\bbl@usedategroupfalse\fi\bbl@fixname\languagename\bbl@iflanguage{\languagename}{\% set name}\bbl@fixname\languagename\% TODO. name@map here?\bbl@provide{locale}\bbl@iflanguage{\languagename}{%}
```

96
This macro selects the hyphenation patterns by changing the \language register. If special hyphenation patterns are available specifically for the current font encoding, use them instead of the default.

It also sets hyphenation exceptions, but only once, because they are global (here language \lccode's has been set, too). \bbl@hyphenation@ is set to relax until the very first \babelhyphenation, so do nothing with this value. If the exceptions for a language (by its number, not its name, so that :ENC is taken into account) has been set, then use \hyphenation with both global and language exceptions and empty the latter to mark they must not be set again.

The environment \hyphenrules can be used to select just the hyphenation rules. This environment does not change \languagename and when the hyphenation rules specified were not loaded it has no effect. Note however, \lccode's and font encodings are not set at all, so in most cases you should use other \language*.
\edef\bbl@tempf{#1}\
\bbl@fixname\bbl@tempf\
\bbl@iflanguage\bbl@tempf{%\
  \expandafter\bbl@patterns\expandafter{\bbl@tempf}\
  \languageshorthands{none}\
  \set@hyphenmins{tw@thr@}@\relax\
  \else\
  \expandafter\expandafter\expandafter\set@hyphenmins\
  \csname\bbl@tempf hyphenmins\endcsname\relax\
  \fi}\
\let\endhyphenrules\@empty\
\providehyphenmins

The macro \providehyphenmins should be used in the language definition files to provide a default setting for the hyphenation parameters \lefthyphenmin and \righthyphenmin. If the macro \langle lang\rangle hyphenmins is already defined this command has no effect.

\def\providehyphenmins#1#2{%\expandafter\ifx\csname #1hyphenmins\endcsname\relax\@namedef{#1hyphenmins}{#2}\fi}

\set@hyphenmins

This macro sets the values of \lefthyphenmin and \righthyphenmin. It expects two values as its argument.

\def\set@hyphenmins#1#2{\lefthyphenmin#1\relax\righthyphenmin#2\relax}

\ProvidesLanguage

The identification code for each file is something that was introduced in \LaTeX2ε. When the command \ProvidesFile does not exist, a dummy definition is provided temporarily. For use in the language definition file the command \ProvidesLanguage is defined by babel. Depending on the format, ie, on if the former is defined, we use a similar definition or not.

\ifx\ProvidesFile\@undefined\def\ProvidesLanguage#1[#2 #3 #4]{\wlog{Language: #1 #4 #3 <#2>}}\else\begingroup\catcode`\ 10 %\@makeother/\@ifnextchar[%]{\@provideslanguage{#1}}{\@provideslanguage{#1}[\]}\endgroup\fi

\originalTeX

The macro \originalTeX should be known to \TeX at this moment. As it has to be expandable we \let it to \@empty instead of \relax.

\ifx\originalTeX\@undefined\let\originalTeX@empty\fi

Because this part of the code can be included in a format, we make sure that the macro which initializes the save mechanism, \begin{save}, is not considered to be undefined.

\ifx\begin{save}\@undefined\let\begin{save}\@empty\fi
A few macro names are reserved for future releases of babel, which will use the concept of ‘locale’:

```latex
\providecommand\setlocale{
\bbl@error
\bbl@warning{
\errhelp{Not yet available}
\errmessage{Find an armchair, sit down and wait}}
\let\uselocale\setlocale
\let\locale\setlocale
\let\selectlocale\setlocale
\let\localename\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale
```

### 9.2 Errors

\@nolanerr The babel package will signal an error when a documents tries to select a language that hasn’t been defined earlier. When a user selects a language for which no hyphenation patterns were loaded into the format he will be given a warning about that fact. We revert to the patterns for `\language=0` in that case. In most formats that will be (US)english, but it might also be empty.

\@nopatterns When the package was loaded without options not everything will work as expected. An error message is issued in that case. When the format knows about `\PackageError` it must be \TeX, so we can safely use its error handling interface. Otherwise we’ll have to ‘keep it simple’.

Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```latex
\edef\bbl@nulllanguage{\string\language=0}
\if\PackageError\@undefined % TODO. Move to Plain
\def\bbl@error#1#2{\begingroup\newlinechar=`\^^J\def\{\^^J(babel) \errhelp{#2}\errmessage{\#1}\endgroup}
\def\bbl@warning#1{\begingroup\newlinechar=`\^^J\def\{\^^J(babel) \message{\#1}\endgroup}
\let\bbl@infowarn\bbl@warning
\def\bbl@info#1{\begingroup\newlinechar=`\^^J\wlog{\#1}\endgroup}
\fi
\def\bbl@nocaption{\protect\bbl@nocaption@i}
\def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
\ifx\PackageError\@undefined \begingroup \newlinechar=`\^^J\def\{\^^J(babel) \errhelp{#1 not set. Please, define \langXname\message{\#1}\endgroup \fi}
\let\bbl@noinfowarn\bbl@warning
\def\bbl@noinfowarn#1{\bbl@warning{#1}}
\def\bbl@noinfo#1{%\bbl@warning{#1}}
\let\bbl@nocaption\bbl@noinfowarn
\let\bbl@nocaption\bbl@noinfo
```

\@backslashchar#2 not set. Please, define `\langXname` in the preamble with something like: `\langXname\message{\#1}`
\begin{verbatim}
\def\bbl@ tentative{\protect\bbl@ tentative@i}
\def\bbl@ tentative@i#1{% 
\bbl@ warning{Some functions for ' #1 ' are tentative.}
They might not work as expected and their behavior could change in the future.
Reported}}
\def\@ nolanerr#1{% \bbl@ error\{You haven't defined the language #1 \space yet.\%
Perhaps you misspelled it or your installation is not complete\%
\{Your command will be ignored, type <return> to proceed\}}
\def\@nopatterns#1{% \bbl@ warning\{No hyphenation patterns were preloaded for the language ' #1 ' into the format.\%
Please, configure your \TeX\ system to add them and rebuild the format. Now I will use the patterns preloaded for \bbl@ nulllanguage \space instead\}
\let\bbl@ usehooks\@ gobbletwo
\ifx\bbl@ onlyswitch\@ empty\endinput\fi
% Here ended switch.def
\ifx\directlua\@ undefined\else
\ifx\bbl@ luapatterns\@ undefined
\input luababel.def
\fi
\fi
\langle\langle Basic macros\rangle\rangle
\bbl@ trace{Compatibility with language.def}
\ifx\bbl@ languages\@ undefined
\ifx\directlua\@ undefined
\openin1 = language.def % TODO. Remove hardcoded number
\ifeof1
\closein1
\message{I couldn't find the file language.def}
\else
\closein1
\begingroup
\def\addlanguage#1#2#3#4#5{%\expandafter\ifx\csname lang@#1\endcsname\relax\else
\global\expandafter\let\csname l@#1\expandafter\endcsname \csname lang@#1\endcsname
\fi\}
\def\uselanguage#1{}%
\input language.def
\endgroup
\fi
\fi
\chardef\l@english\z@
\fi
\addto\Itakestwoarguments,a\langle control sequence\rangle and \TeX-code to be added to the \langle control sequence\rangle.
\end{verbatim}
If the ⟨control sequence⟩ has not been defined before it is defined now. The control sequence could also expand to \relax, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```latex
\def\addto#1#2{%
  \ifx#1\@undefined
    \def#1{#2}%
  \else
    \ifx#1\relax
      \def#1{#2}%
    \else
      {\toks@\expandafter{#1#2}%
        \xdef#1{\the\toks@}}%
    \fi
  \fi
}
```

The macro \initiate@active@char below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. TODO. Always used with additional expansions. Move them here? Move the macro to basic?

```latex
\def\bbl@withactive#1#2{%
  \begingroup
  \lccode`~=`#2\relax
  \lowercase{\endgroup#1~}}
```

\bbl@redefine To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the ‘sanitized’ argument. The reason why we do it this way is that we don’t want to redefine the \TeX macros completely in case their definitions change (they have changed in the past). A macro named \macro will be saved new control sequences named \org\macro.

```latex
\def\bbl@redefine#1{%
  \edef\bbl@tempa{\bbl@stripslash#1}%
  \bbl@ifunset{\bbl@tempa\space}%
    {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
      \bbl@exp{\def\#1{\protect\langle\bbl@tempa\space>}}}%
    {\bbl@exp{\let\org@\bbl@tempa\langle\bbl@tempa\space>}}}%
  \@namedef{\bbl@tempa\space}}
```

\bbl@redefine@long This version of \bbl@redefine can be used to redefine \long commands such as \ifthenelse.

```latex
\def\bbl@redefine@long#1{%
  \edef\bbl@tempa{\bbl@stripslash#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1%
  \expandafter\expandafter\expandafter{\bbl@tempa\space}}
```

\bbl@redefinerobust For commands that are redefined, but which might be robust we need a slightly more intelligent macro. A robust command \foo is defined to expand to \protect\foo. So it is necessary to check whether \foo exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define \foo.

```latex
\def\bbl@redefinerobust#1{%
  \edef\bbl@tempa{\bbl@stripslash#1}%
  \bbl@funset{\bbl@tempa\space}%
  {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
    \bbl@exp{\def\#1{\protect\langle\bbl@tempa\space>}}}%
  {\bbl@exp{\let\org@\bbl@tempa\langle\bbl@tempa\space>}}}%
  \@namedef{\bbl@tempa\space}}
```

\bbl@redefinerobust
9.3 Hooks

Admittedly, the current implementation is somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. \bbl@usehooks is the commands used by babel to execute hooks defined for an event.

\begin{verbatim}
\bbl@trace{Hooks}
\newcommand\AddBabelHook[3][]{\% \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}\expandafter\bbl@tempa\bbl@evargs,#3=,\@empty\bbl@ifunset{bbl@ev@#2@#3@#1}{}{\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}\bbl@csarg\newcommand{ev@#2@#3@#1}{\bbl@tempb}}
\newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
\newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
\def\bbl@usehooks#1#2{\def\bbl@elt##1{\bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}\bbl@cs{ev@#1@}\ifx\languagename\@undefined\else % Test required for Plain (?\}
\bbl@trace{Defining babelensure}
\newcommand\babelensure[2][]{\% TODO - revise test files\AddBabelHook{babel-ensure}{afterextras}{\ifcase\bbl@select@type1\bbl@cl{e}\fi}\begingroup\let\bbl@ens@include\@empty\let\bbl@ens@exclude\@empty\def\bbl@ens@fontenc{\relax}\def\bbl@evargs{},\% <- don't delete this comma\everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,\%adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,\%beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,\%hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,\%beforestart=0,languagename=2}\fontencoding{}\bbl@ens@include\if\bbl@ens@exclude\fi\bbl@ens@fontenc\bbl@ens@exclude\bbl@ens@include\endgroup\bbl@ens@include
\end{verbatim}

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

The user command just parses the optional argument and creates a new macro named \bbl@e@{language}. We register a hook at the afterextras event which just executes this macro in a “complete” selection (which, if undefined, is \relax and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro \bbl@e@{language} contains \bbl@ensure{\langle include\rangle}{\langle exclude\rangle}{\langle fontenc\rangle}, which in in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \vin{}) those in the exclude list. If the fontenc is given (and not \relax), the \fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.
9.4 Setting up language files

\LdfInit The \LdfInit macro takes two arguments. The first argument is the name of the language that will be defined in the language definition file; the second argument is either a control sequence or a string from which a control sequence should be constructed. The existence of the control sequence indicates that the file has been processed before.

At the start of processing a language definition file we always check the category code of the at-sign. We make sure that it is a ‘letter’ during the processing of the file. We also save its name as the last called option, even if not loaded.

Another character that needs to have the correct category code during processing of language definition files is the equals sign, ‘=’, because it is sometimes used in constructions with the \let primitive. Therefore we store its current catcode and restore it later on.

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to \LdfInit is a control sequence. We do that by looking at the first token after passing #2 through \string. When it is equal to \@backslashchar we are dealing with a control sequence which we can compare with \@undefined.

If so, we call \ldf@quit to set the main language, restore the category code of the @-sign and call \endinput.

When #2 was not a control sequence we construct one and compare it with \relax. Finally we check \originalTeX.

\bbl@trace{Macros for setting language files up}
\def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
\let\bbl@screset\@empty
\let\BabelStrings\bbl@opt@string
\let\BabelOptions\@empty
\let\BabelLanguages\relax
\ifx\originalTeX\@undefined
\let\originalTeX\@empty
\else
\originalTeX
\fi}
\def\LdfInit#1#2{%
\chardef\atcatcode=\catcode`@
\catcode`@=11\relax
\chardef\eqcatcode=\catcode`=
\catcode`==12\relax
\expandafter\if\expandafter\@backslashchar
\expandafter\@car\string#2\@nil
\ifx#2\@undefined\else
\ldf@quit{#1}\
\fi
\else
\expandafter\ifx\csname#2\endcsname\relax\else
\ldf@quit{#1}\
\fi
\fi
\else
\expandafter\ifx\csname#2\endcsname\relax\relax\else
\ldf@quit{#1}\
\fi
\fi
\bbl@ldfinit}
\ldf@quit This macro interrupts the processing of a language definition file.
\def\ldf@quit#1{%
\expandafter\main@language\expandafter{#1}%
\catcode`\@=\atcatcode \let\atcatcode\relax
\catcode`\=\eqcatcode \let\eqcatcode\relax
\endinput}
This macro takes one argument. It is the name of the language that was defined in the language definition file.

We load the local configuration file if one is present, we set the main language (taking into account that the argument might be a control sequence that needs to be expanded) and reset the category code of the @-sign.

```latex
\def\bbl@afterldf#1{% \todo{Merge into the next macro? Unused elsewhere}
  \bbl@afterlang
  \let\bbl@afterlang\relax
  \let\BabelModifiers\relax
  \let\bbl@screset\relax
}\def\ldf@finish#1{% 
  \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
    \loadlocalcfg{#1}\
  \fi
  \bbl@afterldf{#1}\
  \expandafter\main@language\expandafter{#1}\
  \catcode`@=\atcatcode
  \let\atcatcode\relax
}\def\main@language#1{\
  \def\bbl@main@language{#1}\
  \let\languagename\bbl@main@language \todo{Set localename}
  \bbl@id@assign
  \bbl@patterns{\languagename}
}\def\bbl@beforestart{\
  \bbl@usehooks{beforestart}{}\
  \global\let\bbl@beforestart\relax}
\AtBeginDocument{% \@nameuse{bbl@beforestart}\
  \if@filesw
    \providecommand\babel@aux[2]{}\
    \immediate\write\@mainaux{\string\providecommand\string\babel@aux[2]{}\string}
  \fi
  \expandafter\selectlanguage\expandafter{\bbl@main@language}\
  \ifbbl@single % must go after the line above.
    \renewcommand\selectlanguage[1]{}\
    \renewcommand\foreignlanguage[2]{#2}\
    \global\let\babel@aux\@gobbletwo % Also as flag
  \fi
  \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

After the preamble of the document the commands \LdfInit, \ldf@quit and \ldf@finish are no longer needed. Therefore they are turned into warning messages in \LaTeX.

\def\main@language#1{\def\bbl@main@language{#1}\let\languagename\bbl@main@language \todo{Set localename}}

This command should be used in the various language definition files. It stores its argument in \bbl@main@language; to be used to switch to the correct language at the beginning of the document.

We also have to make sure that some code gets executed at the beginning of the document, either when the aux file is read or, if it does not exist, when the \AtBeginDocument is executed. Languages do not set \pagedir, so we set here for the whole document to the main \bodydir.

\def\bbl@beforestart{% 
  \bbl@usehooks{beforestart}{}% 
  \global\let\bbl@beforestart\relax
  \AtBeginDocument{% 
    \@nameuse{bbl@beforestart}\
    \if@filesw
      \providecommand\babel@aux[2]{}\
      \immediate\write\@mainaux{\string\providecommand\string\babel@aux[2]{}\string}
    \fi
    \selectlanguage{\bbl@main@language}\
    \ifbbl@single % must go after the line above.
      \renewcommand\selectlanguage[1]{}\
      \renewcommand\foreignlanguage[2]{#2}\
      \global\let\babel@aux\@gobbletwo % Also as flag
    \fi
    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
A bit of optimization. Select in heads/foots the language only if necessary.

$$\begin{align*}
1826 & \texttt{\def\select@language@x#1{}} \% \\
1827 & \texttt{\ifcase\bbl@select@type} \\
1828 & \texttt{\bbl@ifsamestring\languagename{#1}\{\select@language{#1}\}} \% \\
1829 & \texttt{\else} \\
1830 & \texttt{\select@language{#1}} \% \\
1831 & \texttt{\fi}} \\
\end{align*}$$

9.5 Shorthands

\bbl@add@special The macro \bbl@add@special is used to add a new character (or single character control sequence) to the macro \dospecials (and \@sanitize if \LaTeX{} is used). It is used only at one place, namely when \bbl@activate@active@char is called (which is ignored if the char has been made active before). Because \@sanitize can be undefined, we put the definition inside a conditional.

Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It’s already done with \nfss@catcodes, added in 3.10.

$$\begin{align*}
1832 & \texttt{\bbl@trace{Shorhands}} \\
1833 & \texttt{\def\bbl@add@special#1{}} \% 1: a macro like ", \?, etc. \\
1834 & \texttt{\bbl@add@d reuse@d}{\do#1}\% test \@sanitize = \relax, for back. compat. \\
1835 & \texttt{\bbl@funset{\@sanitize}\{\bbl@add@sanitize\{@makeother\#1}\}} \\
1836 & \texttt{\expandafter\ifx\csname @sanitize\endcsname\relax\else} \\
1837 & \texttt{\begingroup} \\
1838 & \texttt{\catcode`\#1=\active} \\
1839 & \texttt{\nfss@catcodes} \\
1840 & \texttt{\expandafter\ifnum\catcode`\#1=\active} \\
1841 & \texttt{\endgroup} \\
1842 & \texttt{\bbl@add@nfss@catcodes\{@makeother\#1\}} \% \\
1843 & \texttt{\else} \\
1844 & \texttt{\endgroup} \\
1845 & \texttt{\fi} \\
1846 & \texttt{\fi}} \\
\end{align*}$$

\bbl@remove@special The companion of the former macro is \bbl@remove@special. It removes a character from the set macros \dospecials and \@sanitize, but it is not used at all in the babel core.

$$\begin{align*}
1847 & \texttt{\def\bbl@remove@special#1{}} \% \\
1848 & \texttt{\begingroup} \\
1849 & \texttt{\def\#1\#2\{\expandafter\empty} \\
1850 & \texttt{\else\noexpand\#1\noexpand\#2\fi}\% \\
1851 & \texttt{\def\do\{\x\do\}} \\
1852 & \texttt{\def\@makeother\{\x\@makeother\}} \\
1853 & \texttt{\edef\x\{\endgroup} \\
1854 & \texttt{\def\noexpand@d use@d}{\dospecials}\% \\
1855 & \texttt{\@sanitize\endcsname\relax\else} \\
1856 & \texttt{\expandafter\ifx\csname @sanitize\endcsname\relax\else} \\
1857 & \texttt{\fi}\% \\
1858 & \texttt{\fi}} \\
\end{align*}$$

\initiate@active@char A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence \normal@char{char} to expand to the character in its ‘normal state’ and it defines the active character to expand to \normal@char{char} by default (\char{char} being the character to be made active). Later its definition can be changed to expand to \active@char{char} by calling \bbl@activate{char}.

106
For example, to make the double quote character active one could have
\begin{verbatim}
\initiate@active@char{"} in a language definition file. This defines " as \active@prefix "\active@char" (where the first " is the character with its original catcode, when the shorthand is created, and \active@char" is a single token). In protected contexts, it expands to \protect " or \noexpand " (ie, with the original "); otherwise \active@char" is executed. This macro in turn expands to \normal@char" in “safe” contexts (eg, \label), but \user@active" in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, \normal@char" is used. However, a deactivated shorthand (with \bbl@deactivate is defined as \active@prefix "\normal@char".

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string’ed) character, \langle level\rangle @group, <level>@active and <next-level>@active (except in system).

\begin{verbatim}
\def\bbl@active@def#1#2#3#4{%
\@namedef{#3#1}{%
\expandafter\ifx\csname#2@sh@#1@\endcsname\relax
\bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
\else
\bbl@afterfi\csname#2@sh@#1@\endcsname
\}%}
\end{verbatim}

When there is also no current-level shorthand with an argument we will check whether there is a next-level defined shorthand for this active character.

\begin{verbatim}
\def\@initiate@active@char#1{%
\bbl@ifunset{active@char\string#1}%
{\bbl@withactive
{\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
}{}
\end{verbatim}

\initiate@active@char calls \@initiate@active@char with 3 arguments. All of them are the same character with different catcodes: active, other (\string’ed) and the original one. This trick simplifies the code a lot.

\begin{verbatim}
\def\initiate@active@char#1{%
\bbl@ifunset{active@char\string#1}%
{\bbl@withactive
{\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
}{}
\end{verbatim}

The very first thing to do is saving the original catcode and the original definition, even if not active, which is possible (undefined characters require a special treatement to avoid making them \relax).

\begin{verbatim}
\def\@initiate@active@char#1#2#3{%
\bbl@csarg@edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
\ifx#1\undefined
\bbl@csarg@edef{oridef@@#2}{\let\noexpand#1\noexpand@undefined}%
\else
\bbl@csarg@edef{oridef@#2}{\let\noexpand#1\expandafter\csname bbl@oridef@@#2\endcsname}%
\fi%
\end{verbatim}

If the character is already active we provide the default expansion under this shorthand mechanism. Otherwise we write a message in the transcript file, and define \normal@char(\string) to expand to the character in its default state. If the character is mathematically active when babel is loaded (for example ‘) the normal expansion is
somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to "8000 a posteriori").

1887 \ifx#1\#3\relax
1888 \expandafter\let\csname normal@char#2\endcsname#3%
1889 \else
1890 \bbl@info{Making #2 an active character}%
1891 \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1892 \namedef{normal@char#2}{}
1893 \textormath{#3}{\csname bbl@oridef@@#2\endcsname}%
1894 \else
1895 \namedef{normal@char#2}{#3}%
1896 \fi

To prevent problems with the loading of other packages after babel we reset the catcode of the character to the original one at the end of the package and of each language file (except with KeepShorthandsActive). It is re-activate again at \begin{document}. We also need to make sure that the shorthands are active during the processing of the .aux file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of \bibitem for example. Then we make it active (not strictly necessary, but done for backward compatibility).

1897 \bbl@restoreactive{#2}%
1898 \AtBeginDocument{%
1899 \catcode`#2=active
1900 \if@filesw
1901 \immediate\write\@auxout{\catcode`#2=active}%
1902 \fi}%
1903 \expandafter\bbl@add@special\csname#2\endcsname
1904 \catcode`#2=active
1905 \fi

Now we have set \normal@char⟨char⟩, we must define \active@char⟨char⟩, to be executed when the character is activated. We define the first level expansion of \active@char⟨char⟩ to check the status of the @safe@actives flag. If it is set to true we expand to the ‘normal’ version of this character, otherwise we call \user@active{char} to start the search of a definition in the user, language and system levels (or eventually \normal@char⟨char⟩).

1906 \let\bbl@tempa@firstoftwo
1907 \if\string^#2%
1908 \def\bbl@tempa{\noexpand\textormath}%
1909 \else
1910 \if\bbl@mathnormal\@undefined\else
1911 \let\bbl@tempa\bbl@mathnormal
1912 \fi
1913 \fi
1914 \expandafter\edef\csname active@char#2\endcsname{%
1915 \bbl@tempa
1916 \{\noexpand\if@safe@actives
1917 \noexpand\expandafter
1918 \noexpand\expandafter\noexpand\csname normal@char#2\endcsname
1919 \noexpand\else
1920 \noexpand\expandafter
1921 \noexpand\expandafter\noexpand\csname bbl@doactive#2\endcsname
1922 \noexpand\fi%
1923 \{\expandafter\noexpand\csname normal@char#2\endcsname}%
1924 \bbl@csarg\edef\doactive{#2}{}
1925 \expandafter\noexpand\csname user@active#2\endcsname}%

We now define the default values which the shorthand is set to when activated or deactivated. It is set to the deactivated form (globally), so that the character expands to
\active@prefix ⟨char⟩ \normal@char⟨char⟩

(where \active@char⟨char⟩ is one control sequence!).

1926 \bbl@csarg\edef{active@#2}{%
1927 \noexpand\active@prefix
1928 \expandafter\noexpand\csname active@char#2\endcsname}%
1929 \bbl@csarg\edef{normal@#2}{%
1930 \noexpand\active@prefix
1931 \expandafter\noexpand\csname normal@char#2\endcsname}%
1932 \expandafter\let\expandafter#1\csname bbl@normal@#2\endcsname

The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn’t exist we check for a shorthand with an argument.

1933 \bbl@active@def#2\user@group{user@active}{language@active}%
1934 \bbl@active@def#2\language@group{language@active}{system@active}%
1935 \bbl@active@def#2\system@group{system@active}{normal@char}%

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as ’ends up in a heading \TeX{} would see ‘protect’ ‘protect’. To prevent this from happening a couple of shorthand needs to be defined at user level.

1936 \expandafter\edef\csname user@group @sh@#2@@\endcsname
1937 \expandafter\edef\csname user@group @sh@#2@\string\protect@\endcsname

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (’) active we need to change \pr@m@s as well. Also, make sure that a single ’ in math mode ‘does the right thing’. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

1940 \if\string'#2%
1941 \let\prim@s\bbl@prim@s
1942 \let\active@math@prime#1%
1943 \fi
1944 \bbl@usehooks{initiateactive}{(#1){#2}{#3}}

The following package options control the behavior of shorthands in math mode.

1945 \(\langle\langle∗ More package options\rangle⟩ ≡\)
1946 \{\DeclareOption{math=active}{}\}
1947 \{\DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}\}
1948 \{\langle\langle More package options\rangle⟩\}

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package and the end of the .ldf.

1949 \ifpackagewith{babel}{KeepShorthandsActive}%
1950 \{\let\bbl@restoreactive\@gobble}%
1951 \{\def\bbl@restoreactive#1{%
1952 \bbl@exp{%
1953 \AfterBabelLanguage\CurrentOption
1954 {\catcode`#1=\the\catcode`#1\relax}%
1955 \AtEndOfPackage
1956 {\catcode`#1=\the\catcode`#1\relax}}%
1957 \AtEndOfPackage{\let\bbl@restoreactive\@gobble}
\texttt{\bbl@sh@select}  This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of \texttt{\hyphenation}.

This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either \texttt{\bbl@firstcs} or \texttt{\bbl@scndcs}. Hence two more arguments need to follow it.

\begin{verbatim}
1958 \def\bbl@sh@select#1#2{%
1959 \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1960 \bbl@afterelse\bbl@scndcs
1961 \else
1962 \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1963 \fi}
\end{verbatim}

\texttt{\active@prefix}  The command \texttt{\active@prefix} which is used in the expansion of active characters has a function similar to \texttt{\OT1-cmd} in that it \texttt{\protect} the active character whenever \texttt{\protect} is \texttt{not} \texttt{\typeset@protect}. The \texttt{\@gobble} is needed to remove a token such as \texttt{\activechar:} (when the double colon was the active character to be dealt with). There are two definitions, depending of \texttt{\ifincsname} is available. If there is, the expansion will be more robust.

\begin{verbatim}
1964 \begingroup
1965 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1966 {\gdef\active@prefix#1{%
1967 \ifx\protect\@typeset@protect
1968 \else
1969 \ifx\protect\@unexpandable@protect
1970 \noexpand#1%
1971 \else
1972 \protect#1%
1973 \fi
1974 \expandafter\@gobble
1975 \fi}
1976 {\gdef\active@prefix#1{%
1977 \ifincsname
1978 \string#1%
1979 \expandafter\@gobble
1980 \else
1981 \ifx\protect\@typeset@protect
1982 \else
1983 \ifx\protect\@unexpandable@protect
1984 \noexpand#1%
1985 \else
1986 \protect#1%
1987 \fi
1988 \expandafter\expandafter\expandafter\@gobble
1989 \fi
1990 \fi}}
1991 \endgroup
\end{verbatim}

\texttt{\if@safe@actives}  In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch \texttt{@safe@actives} is available. The setting of this switch should be checked in the first level expansion of \texttt{\active@char\langle char\rangle}.

\begin{verbatim}
1992 \newif\if@safe@actives
1993 \@safe@activesfalse
\end{verbatim}

\texttt{\bbl@restore@actives}  When the output routine kicks in while the active characters were made “safe” this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them “unsafe” again.

110
Both macros take one argument, like \initiate@active@char. The macro is used to change the definition of an active character to expand to \active@char (char) in the case of \bbl@activate, or \normal@char (char) in the case of \bbl@deactivate.

\bbl@activate\bbl@deactivate

Both macros take one argument, like \initiate@active@char. The macro is used to change the definition of an active character to expand to \active@char (char) in the case of \bbl@activate, or \normal@char (char) in the case of \bbl@deactivate.

\bbl@firstcs\bbl@scndcs

These macros are used only as a trick when declaring shorthands.

\declare@shorthand

The command \declare@shorthand is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. \~ or "a;
3. the code to be executed when the shorthand is encountered.

\textormath

Some of the shorthands that will be declared by the language definition files have to be usable in both text and math mode. To achieve this the helper macro \textormath is provided.
The current concept of ‘shorthands’ supports three levels or groups of shorthands. For each level the name of the level or group is stored in a macro. The default is to have a user group; use language group ‘english’ and have a system group called ‘system’.

\def\user@group{user}
\def\language@group{english} % TODO. I don’t like defaults
\def\system@group{system}

This is the user level macro. It initializes and activates the character for use as a shorthand character (ie, it’s active in the preamble). Languages can deactivate shorthands, so a starred version is also provided which activates them always after the language has been switched.

\def\useshorthands{%
@ifstar\bbl@usesh@s{%\bbl@usesh@x{}}
\def\bbl@usesh@s#1{%
{\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
#1}
\def\bbl@usesh@sx{%
{\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
#1}
\def\bbl@ifshorthand{#1}{%
\bbl@ifunset{user@generic@active#1}{%
\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
\bbl@active@def#1\user@group{user@generic@active}{language@active}%
\expandafter\edef\csname#2@sh@#1@\string\protect@endcsname{%
\expandafter\noexpand\csname normal@char#1\endcsname}%
\expandafter\edef\csname#2@sh@#1@\string\protect@endcsname{%
\expandafter\noexpand\csname user@active#1\endcsname}}%
@empty}
@empty}
\newcommand\defineshorthand[3][user]{%
\edef\bbl@tempa{\zap@space#1 @empty}%
\bbl@for\bbl@tempb\bbl@tempa{%$
\if*\expandafter\%\car\bbl@tempb\nil
\edef\bbl@tempb{\user@\expandafter\\gobble\bbl@tempb}%
\expandoftwargs
\bbl@set@user@generic{\expandafter\\string@\car#2@\nil}\bbl@tempb$
\fi
\declare@shorthand{\bbl@tempb}{#2}{#3}
\languageshorthand
A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing [TODO. Unclear].

\aliasshorthand
First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with \aliasshorthands{"}{{}} is \active@prefix /\active@char/, so we still need to let the latest to \active@char".

\shorthandon\shorthandoff
The first level definition of these macros just passes the argument on to \bbl@switch@sh, adding \@nil at the end to denote the end of the list of characters.

\bbl@switch@sh
The macro \bbl@switch@sh takes the list of characters apart one by one and subsequently switches the category code of the shorthand character according to the first argument of \bbl@switch@sh.
But before any of this switching takes place we make sure that the character we are dealing with is known as a shorthand character. If it is, a macro such as \active@char" should exist.
Switching off on and on is easy – we just set the category code to ‘other’ (12) and \active. With the starred version, the original catcode and the original definition, saved in @initiate@active@char, are restored.
I cannot switch \string#2' on or off—not a shorthand}%
{This character is not a shorthand. Maybe you made a typing mistake? I will ignore your instruction}\
\ifcase#1\relax
\catcode#2\active\
\catcode#2\active\
\csname bbl@oricat@\string#2\endcsname\csname bbl@oridef@\string#2\endcsname\fi\
\bbl@afterfi\bbl@switch@sh#1%\fi
\bbl@afterfi\bbl@switch@sh#1%
End note the value is that at the expansion time; e.g. in the preamble shorthands are usually deactivated.
\def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
\def\bbl@putsh#1{%\bbl@ifunset{bbl@active@\string#1}{\bbl@putsh@i#1\@empty\@nnil}{\csname bbl@active@\string#1\endcsname}}
\def\bbl@putsh@i#1#2\@nnil{%\csname\languagename @sh@\string#1@%\ifx\@empty#2\else\string#2@\fi\endcsname}
\ifx\bbl@opt@shorthands\@nnil\else
\let\bbl@s@initiate@active@char\initiate@active@char
\def\initiate@active@char#1{%\bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
\let\bbl@s@switch@sh\bbl@switch@sh
\def\bbl@switch@sh#1#2{%\ifx#2\@nnil\else\bbl@afterfi\bbl@s@switch@sh#1{#2}\fi\endcsname}\fi
\let\bbl@s@activate\bbl@activate
\def\bbl@activate#1{%\bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
\let\bbl@s@deactivate\bbl@deactivate
\def\bbl@deactivate#1{%\bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
\fi
You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on or off.
\newcommand\ifbabelshorthand[3]{{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}}
\bbl@prim@s\bbl@pr@@m@s
One of the internal macros that are involved in substituting \prime for each right quote in math mode is \prim@s. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.
\def\bbl@prim@s{%\pr@me\futurelet\@let@token\bbl@pr@m@s}\def\bbl@if@primes#1#2{%\ifx#1\@let@token\expandafter\@firstoftwo\else\ifx#2\@let@token\bbl@afterelse\expandafter\@firstoftwo\else\fi\fi\fi\bbl@afterfi\bbl@if@primes{#1}{#2}\fi\endcsname}
Usually the ~ is active and expands to \penalty\@M\penalty\@M. When it is written to the .aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).

\initiate@active@char{-}
\declare@shorthand{system}{~}{\leavevmode\nobreak\ }
\bbl@activate{~}

\OT1dqpos
\T1dqpos
The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.
\expandafter\def\csname OT1dqpos\endcsname{127}
\expandafter\def\csname T1dqpos\endcsname{4}

When the macro \f@encoding is undefined (as it is in plain TeX) we define it here to expand to OT1
\ifx\f@encoding@undefined
\def\f@encoding{OT1}
\fi

9.6 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute
The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.
\bbl@trace{Language attributes}
\newcommand\languageattribute[2]{%
\def\bbl@tempc{#1}%
\bbl@fixname\bbl@tempc
\bbl@iflanguage\bbl@tempc{%
\bbl@vforeach{#2}{%
We want to make sure that each attribute is selected only once; therefore we store the already selected attributes in \bbl@known@attrs. When that control sequence is not yet defined this attribute is certainly not selected before.
\ifx\bbl@known@attrs\@undefined
\in@false
\else
\bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
\fi

115
When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX-code.

\begin{verbatim}
\bbl@exp(%
  \\\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}%)%  \edef\bbl@tempa{\bbl@tempc-##1}%
\expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
  {\csname\bbl@tempc @attr@##1\endcsname}%
  {\@attrerr{\bbl@tempc}{##1}}%
\fi}}}
\@onlypreamble\languageattribute
\newcommand*{\@attrerr}[2]{%
  \bbl@error
  {The attribute #2 is unknown for language #1.}%
  {Your command will be ignored, type <return> to proceed}}
\end{verbatim}

This command adds the new language/attribute combination to the list of known attributes. Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro \texttt{\textbackslash extras...} for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \texttt{\textbackslash begin\{document\}}.

\begin{verbatim}
\def\bbl@declare@tribute#1#2#3{%
  \bbl@xin{,#1,}{,\BabelModifiers,}%
  \ifin@
  \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
  \fi
  \bbl@add@list\bbl@attributes{#1-#2}
  \expandafter\def\csname#1@attr@#2\endcsname{#3}}
\end{verbatim}

This internal macro has 4 arguments. It can be used to interpret \TeX-code based on whether a certain attribute was set. This command should appear inside the argument to \texttt{\textbackslash AtBeginDocument} because the attributes are set in the document preamble, after babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses. First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far \texttt{\textbackslash ifin@} has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is ‘thrown over the \texttt{\textbackslash fi}'.

\begin{verbatim}
\def\bbl@ifattributeset#1#2#3#4{%
  \ifx\bbl@known@attrs@undefined
  \in@false
  \else
  \bbl@xin{,#1-#2,}{,\bbl@known@attrs,}%
  \fi
  \ifin@
  \bbl@afterelse#3%
  \else
  \bbl@afterfi#4%
\end{verbatim}
\if\fi

\bbl@ifknown@ttrib

An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the \TeX{}-code to be executed when the attribute is known and the \TeX{}-code to be executed otherwise.

We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match. When a match is found the definition of \bbl@tempa is changed. Finally we execute \bbl@tempa.

\def\bbl@ifknown@ttrib#1#2{%
  \let\bbl@tempa@secondoftwo\@secondoftwo
  \bbl@loopx\bbl@tempb{#2}{%}
  \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
  \ifin@
    \let\bbl@tempa@firstoftwo\@firstoftwo
  \else
  \fi}
\bbl@tempa
\}

\bbl@clear@ttribs

This macro removes all the attribute code from \LaTeX{}'s memory at \begin{document} time (if any is present).

\def\bbl@clear@ttribs{%
  \ifx\bbl@attributes\@undefined\else
    \bbl@loopx\bbl@tempa{\bbl@attributes}{{%}
      \expandafter\bbl@clear@ttrib\bbl@tempa.
    }%
  \let\bbl@attributes\@undefined
  \fi}
\bbl@clear@ttribs

9.7 Support for saving macro definitions

To save the meaning of control sequences using \bbl@save, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see \selectlanguage{} and \originalTeX{}). Note undefined macros are not undefined any more when saved – they are \relax'ed.

\bbl@savecnt
\bbl@beginsave

The initialization of a new save cycle: reset the counter to zero.

\bbl@trace{Macros for saving definitions}
\def\bbl@beginsave{\bbl@savecnt\z@}

Before it's forgotten, allocate the counter and initialize all.

\newcount\bbl@savecnt
\bbl@beginsave

\bbl@save
\bbl@savevariable

The macro \bbl@save\langle\csname\rangle\rangle saves the current meaning of the control sequence \langle\csname\rangle\rangle to \originalTeX{}\footnote{\originalTeX{} has to be expandable, i.e. you shouldn't let it to \relax.}. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to \originalTeX{} and the counter is
increased. The macro \texttt{\textbackslash babel@savevariable\{variable\}} saves the value of the variable. \texttt{\{variable\}} can be anything allowed after the \texttt{\textbackslash} the primitive.

\begin{verbatim}
def\babel@save#1{\expandafter\let\csname babel@\number\babel@savecnt\endcsname#1=\relax \toks@\expandafter{\originalTeX \let#1=} \bbl@exp{\def\originalTeX{\the	oks@\langle babel@\number\babel@savecnt\rangle \relax}} \advance\babel@savecnt\@ne}
def\babel@savevariable#1{\toks@\expandafter{\originalTeX #1=} \bbl@exp{\def\originalTeX{\the	oks@\the#1\relax}}}
\end{verbatim}

\texttt{\textbackslash bbl@frenchspacing} Some languages need to have \texttt{\textbackslash frenchspacing} in effect. Others don't want that. The command \texttt{\textbackslash bbl@frenchspacing} switches it on when it isn't already in effect and \texttt{\textbackslash bbl@nonfrenchspacing} switches it off if necessary.

\begin{verbatim}
def\bbl@frenchspacing{% 
  \ifnum\the\sfcode`\textperiodcentered=\@m \let\bbl@nonfrenchspacing\relax 
  \else \frenchspacing \let\bbl@nonfrenchspacing
\nonfrenchspacing \fi}
\let\bbl@nonfrenchspacing\nonfrenchspacing
\end{verbatim}

9.8 Short tags

\texttt{\textbackslash babeltags} This macro is straightforward. After zapping spaces, we loop over the list and define the macros \texttt{\textbackslash text\{tag\}} and \texttt{\{tag\}}. Definitions are first expanded so that they don't contain \texttt{\textbackslash csname} but the actual macro.

\begin{verbatim}
def\babeltags#1{\edef\bbl@tempa{\zap@space#1 \@empty} \def\bbl@tempb##1=##2\@@{\edef\bbl@tempc{\noexpand\newcommand\expandafter\noexpand\csname ##1\endcsname{\noexpand\protect\expandafter\noexpand\csname otherlanguage*\endcsname{##2}}} \noexpand\newcommand\expandafter\noexpand\csname text##1\endcsname{\foreignlanguage{##2}}\bbl@tempc} \bbl@for\bbl@tempa\bbl@tempa{\expandafter\bbl@tempb\bbl@tempa\@@}}
\end{verbatim}

9.9 Hyphens

\texttt{\textbackslash babelhyphenation} This macro saves hyphenation exceptions. Two macros are used to store them: \texttt{\textbackslash babelhyphenation@} for the global ones and \texttt{\textbackslash babelhyphenation<lang>} for language ones. See \texttt{\textbackslash patterns} above for further details. We make sure there is a space between words when multiple commands are used.

\begin{verbatim}
def\babelhyphenation{\@empty} \@onlypreamble\def\babelhyphenation@{% \@onlypreamble\def\babelhyphenation<lang>{% \AtEndOfPackage{% \newcommand\babelhyphenation[2][\@empty]{% \ifx\bbl@hyphenation@\relax
\end{verbatim}
This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hspace 0pt plus 0pt\[32\]
\def\bbl@allowhyphens{%\ifvmode{%\nobreak\hskip\z@skip%}{\fi}
\def\tt@one{T1}
\def\allowhyphens{%\ifx\cf@encoding\tt@one{%\else{%\bbl@allowhyphens%\fi}}
\newcommand\bblnullhyphen{%\char\hyphenchar\font}
\def\bblhyphen{%\active@prefix\bblnullhyphen\bbl@hyphen}
\def\bbl@hyphen{%
@ifstar{%\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
\def\bbl@hyphenchar{%32
TEX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.
Finally, we define the hyphen “types”. Their names will not change, so you may use them in ldf.s. After a space, the \bbox in \bbl@hy@nobreak is redundant.

```latex
\def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}}
\def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}}
\def\bbl@hy@hard{\bbl@usehyphen{\bbl@hyphenchar}}
\def\bbl@hy@@hard{\bbl@@usehyphen{\bbl@hyphenchar}}
\def\bbl@hy@nobreak{\bbl@usehyphen{\bbox{\bbl@hyphenchar}}}
\def\bbl@hy@@nobreak{\bbox{\bbl@hyphenchar}}
\def\bbl@hy@repeat{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
\def\bbl@hy@@repeat{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}

\bbl@disc

For some languages the macro \bbl@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.

\bbl@disc \#1\#2{\bbox{\discretionary{\#2-}{\#1}}}\bbox{\bbl@allowhyphens}
```

### 9.10 Multiencoding strings

The aim following commands is to provide a common interface for strings in several encodings. They also contains several hooks which can be used by luatex and xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

**Tools** But first, a couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

```latex
\def\bbl@trace{Multiencoding strings}
\def\bbl@tobglobal#1{\global\let#1#1}
\def\bbl@recatcode#1{% TODO. Used only once?
\@tempcnta=7F
\ifnum\@tempcnta=FF\else
\catcode\@tempcnta=\relax
\advance\@tempcnta\@ne
\expandafter\bbl@recatcode
\fi%
\def\bbl@tempa{\bbox{\\lang@bbl@uc1c}}

The second one. We need to patch \@uc1list, but it is done once and only if \SetCase is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact \@uc1list is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually \reserved@a), we pass it as argument to \bbl@uc1c. The parser is restarted inside \bbox{\lang} because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:
and starts over (and similarly when lowercasing).

\let\bbl@tolower\@empty\bbl@toupper\@empty

The following package options control the behavior of \SetString.

**Main command** This is the main command. With the first use it is redefined to omit the basic setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not active characters.

\begin{verbatim}
\let\bbl@opt@strings\@nnil % accept strings=value
\DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
\DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
\def\BabelStringsDefault{generic}
\end{verbatim}
Parse the encoding info to get the label, input, and font parts. Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.
We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.
There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes sure \langle group \rangle \langle language \rangle is reset, but only once (\bbl@screset is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing.

The macro \bbl@forlang loops \bbl@L but its body is executed only if the value is in \BabelLanguages (inside babel) or \date\langle language \rangle is defined (after babel has been loaded). There are also two version of \bbl@forlang. The first one skips the current iteration if the language is not in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has been loaded).

Now we define commands to be used inside \StartBabelCommands.
Strings  The following macro is the actual definition of \SetString when it is “active”
First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie,
like \providecommand). With the event stringprocess you can preprocess the string by
manipulating the value of \BabelString. If there are several hooks assigned to this event,
preprocessing is done in the same order as defined. Finally, the string is set.

\def\bbl@setstring#1#2{%
\bbl@forlang\bbl@tempa{%
\edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
\bbl@ifunset{\bbl@LC}{% eg, \germanchaptername
  \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
  \bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}%
}\def\BabelString{#2}%
\bbl@usehooks{stringprocess}{}%
\expandafter\bbl@stringdef\csname\bbl@LC\endcsname{\BabelString}}

Now, some additional stuff to be used when encoded strings are used. Captions then
include \bbl@encoded for string to be expanded in case transformations. It is \relax by
default, but in \MakeUppercase and \MakeLowercase its value is a modified expandable
@changed@cmd.

\ifx\bbl@opt@strings\relax
\def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
\bbl@patchuclc
\let\bbl@encoded\relax
\def\bbl@encoded@uclc#1{\Tilde{\mathwarn}\expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
\expandafter\ifx\csname ?\string#1\endcsname\TextSymbolUnavailable\else
\csname \cf@encoding\string#1\endcsname\fi}
\else
\def\bbl@scset#1#2{\def#1{#2}}
\fi

Define \SetStringLoop, which is actually set inside \StartBabelCommands. The current
definition is somewhat complicated because we need a count, but \count@ is not under
our control (remember \SetString may call hooks). Instead of defining a dedicated count,
we just “pre-expand” its value.

\def\SetStringLoop##1##2{%
\def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
\count@\z@
\bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
\advance\count@\@ne
\toks@\expandafter{\bbl@tempa}%
\bbl@exp{%
\\SetString\bbl@templ{\roman{\numeral\count@}}\the\toks@}%
\count@\relax}{}
\fi
}
Delaying code  Now the definition of \AfterBabelCommands when it is activated.
\def\bbl@aftercmds#1{\toks@\expandafter{\bbl@scafter#1}\xdef\bbl@scafter{\the\toks@}}

Case mapping  The command \SetCase provides a way to change the behavior of \MakeUppercase and \MakeLowercase. \bbl@tempa is set by the patched \@uclclist to the parsing command.
\def\newcommand\SetCase[3][]{\bbl@patchuclc\bbl@forlang\bbl@tempa{\expandafter\bbl@encstring\csname\bbl@tempa@bbl@uc\endcsname{##2}\expandafter\bbl@encstring\csname\bbl@tempa@bbl@lc\endcsname{##3}}}

Macros to deal with case mapping for hyphenation. To decide if the document is monolingual or multilingual, we make a rough guess – just see if there is a comma in the languages list, built in the first pass of the package options.
\newcommand\SetHyphenMap[1]{\bbl@forlang\bbl@tempa{\expandafter\stringdef\csname\bbl@tempa@bbl@hyphenmap\endcsname{##1}}}

There are 3 helper macros which do most of the work for you.
\newcommand\BabelLower[2]{\ifnum\lccode#1=#2\else\babel@savevariable{\lccode#1}\lccode#1=#2\relax\fi}
\newcommand\BabelLowerMM[4]{\ifnum\@tempcnta>#2\else\@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}\advance\@tempcnta#3\expandafter\BabelLower{\the\@tempcnta}{\the\@tempcntb}\fi}
\newcommand\BabelLowerMO[4]{\ifnum\@tempcnta>#2\else\@expandtwoargs\BabelLower{\the\@tempcnta}{#4}\advance\@tempcnta#3\expandafter\BabelLower{\the\@tempcnta}{\the\@tempcntb}\fi}

The following package options control the behavior of hyphenation mapping.

```
\DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
\DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\one}
\DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\two}
\DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\three}
\DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap\fourrelax}
\AtEndOfPackage{%
  \ifx\bbl@opt@hyphenmap\@undefined
    \bbl@xin{,}{\bbl@language@opts}{\ifin@4\else\one\fi}
  \fi}
```

Initial setup to provide a default behavior if `hyphenmap` is not set.

```
\set@low@box
```

The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```
\bbl@trace{Macros related to glyphs}
\def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}{\setbox\z@\box\tw@}%
  \ht\z@\ht\tw@ \dp\z@\dp\tw@\kern-.04em\bbl@allowhyphens}
```

```
\save@sf@q
The macro \save@sf@q is used to save and reset the current space factor.
```

```
\def\save@sf@q#1{\leavevmode
  \begingroup
  \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
  \endgroup}
```

9.11 Macros common to a number of languages

9.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be ‘faked’, or that are not accessible through `T1enc.def`.

9.12.1 Quotation marks

```
\quotedblbase In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.
```

```
\ProvideTextCommand{\quotedblbase}{OT1}{% 
  \save@sf@q{\set@low@box{\textquotedblright}}\@SF % 
  \setbox\z@\@box{\textquotedblright} % 
  \setbox\z@\kern-.04em\bbl@allowhyphens}
```

Make sure that when an encoding other than OT1 or T1 is used this glyph can still be typeset.

```
\ProvideTextCommandDefault{\quotedblbase}{% 
  \UsesTextSymbol{OT1}{\quotedblbase}}
```

```
\quotesinglbase We also need the single quote character at the baseline.
```

```
\ProvideTextCommand{\quotesinglbase}{OT1}{% 
  \save@sf@q{\set@low@box{\textquoteright}}% 
  \setbox\z@\kern-.04em\bbl@allowhyphens}
```

126
Make sure that when an encoding other than OT1 or T1 is used this glyph can still be typeset.

\ProvideTextCommandDefault{\quotesinglbase}{%
\UseTextSymbol{OT1}{\quotesinglbase}}
\guillemetleft
\guillemetright

The guillemet characters are not available in OT1 encoding. They are faked. (Wrong names with o preserved for compatibility.)

\ProvideTextCommand{\guillemetleft}{OT1}{%
\ifmmode\ll\else\save@sf@q{\nobreak\raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}\fi}
\ProvideTextCommand{\guillemetright}{OT1}{%
\ifmmode\gg\else\save@sf@q{\nobreak\raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}\fi}
\guillemotleft
\guillemotright

The single guillemets are not available in OT1 encoding. They are faked.

\ProvideTextCommandDefault{\guillemotleft}{%
\UseTextSymbol{OT1}{\guillemotleft}}
\ProvideTextCommandDefault{\guillemotright}{%
\UseTextSymbol{OT1}{\guillemotright}}
\guilsinglleft
\guilsinglright

Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.
Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\ProvideTextCommandDefault{\guilsinglleft}{\UseTextSymbol{OT1}{\guilsinglleft}}
\ProvideTextCommandDefault{\guilsinglright}{\UseTextSymbol{OT1}{\guilsinglright}}

9.12.2 Letters

\ij The dutch language uses the letter ‘ij’. It is available in T1 encoded fonts, but not in the OT1 encoded fonts. Therefore we fake it for the OT1 encoding.

\DeclareTextCommand{\ij}{OT1}{\kern-0.02em\bbl@allowhyphens j}
\DeclareTextCommand{\IJ}{OT1}{I\kern-0.02em\bbl@allowhyphens J}

\DeclareTextCommand{\ij}{T1}{\char188}
\DeclareTextCommand{\IJ}{T1}{\char156}

Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\ProvideTextCommandDefault{\ij}{\UseTextSymbol{OT1}{\ij}}
\ProvideTextCommandDefault{\IJ}{\UseTextSymbol{OT1}{\IJ}}

\dj The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding, but not in the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

\def\crrtic@{\hrule height0.1ex width0.3em}
\def\crttic@{\hrule height0.1ex width0.33em}

\def\ddj@{\setbox0\hbox{d}\dimen@=.55\ht0 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ \dimen@=.45\dimen@ \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ii \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
\def\DDJ@{\setbox0\hbox{D}\dimen@=.55\ht0 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ \dimen@=.45\dimen@ \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ii \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}

\DeclareTextCommand{\dj}{OT1}{\ddj@ d}
\DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

\ProvideTextCommandDefault{\dj}{\UseTextSymbol{OT1}{\dj}}
For the T1 encoding \SS is defined and selects a specific glyph from the font, but for other encodings it is not available. Therefore we make it available here.

\SS

9.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with \ProvideTextCommandDefault, but this is very likely not required because their definitions are based on encoding-dependent macros.

\glq The ‘german’ single quotes.
\grq

\glqq The ‘german’ double quotes.
\grqq

\flq The ‘french’ single guillemets.
\frq

\flqq The ‘french’ double guillemets.
9.12.4 Umlauts and tremas

The command \" needs to have a different effect for different languages. For German for instance, the 'umlaut' should be positioned lower than the default position for placing it over the letters a, o, u, A, O and U. When placed over an e, i, E or I it can retain its normal position. For Dutch the same glyph is always placed in the lower position.

\umlauthigh To be able to provide both positions of \" we provide two commands to switch the positioning, the default will be \umlauthigh (the normal positioning).

\umlautlow

\lower@umlaut The command \lower@umlaut is used to position the \" closer to the letter.
We want the umlaut character lowered, nearer to the letter. To do this we need an extra ⟨dimen⟩ register.

\lower@umlaut The following code fools \TeX's make_accent procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of .45ex depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \accent primitive, reset the old x-height and insert the base character in the argument.

\lower@umlaut For all vowels we declare \" to be a composite command which uses \bbl@umlauta or \bbl@umlaute to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package fontenc with option OT1 is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but babel sets them for all languages – you may want to redefine \bbl@umlauta and/or \bbl@umlaute for a language in the corresponding ldf (using the babel switching mechanism, of course).

\lower@umlaut
Finally, make sure the default hyphen rules are defined (even if empty). For internal use, another empty \language is defined. Currently used in Amharic.

\ifx\l@english\undefined
  \chardef\l@english\z@
\fi
% The following is used to cancel rules in ini files (see Amharic).
\ifx\l@babelnohyphens\undefined
  \newlanguage\l@babelnohyphens
\fi

9.13 Layout

Work in progress.

Layout is mainly intended to set bidi documents, but there is at least a tool useful in general.
9.14 Load engine specific macros

\bbl@trace{Input engine specific macros}
\ifcase\bbl@engine
\input txtbabel.def
\or
\input luababel.def
\or
\input xebabel.def
\fi

9.15 Creating and modifying languages

\bbl@trace{Creating languages and reading ini files}
\newcommand\babelprovide[2][2]{
% Set name and locale id
\edef\languagename{#2}%
%\global@namedef{bbl@lcname@#2}{#2}%
\bbl@id@assign
\let\bbl@KVP@captions\@nil
\let\bbl@KVP@import\@nil
\let\bbl@KVP@main\@nil
\let\bbl@KVP@script\@nil
\let\bbl@KVP@language\@nil
\let\bbl@KVP@hyphenrules\@nil % only for provide@new
\let\bbl@KVP@mapfont\@nil
\let\bbl@KVP@maparabic\@nil
\let\bbl@KVP@mapdigits\@nil
\let\bbl@KVP@intraspace\@nil
\let\bbl@KVP@intrapenalty\@nil
\let\bbl@KVP@onchar\@nil
\let\bbl@KVP@alph\@nil
\let\bbl@KVP@Alph\@nil
\let\bbl@KVP@info\@nil % Ignored with import? Or error/warning?
\bbl@forkv{#1}{% TODO - error handling
\in@{/}{##1}%
\ifin@
\bbl@renewinikey##1@@{##2}%
\else
\bbl@csarg\def{KVP@##1}{##2}%
\fi}%
% == import, captions ==
\fIx\bbl@KVP@import\@nil\else
\bbl@exp{\\bbl@ifblank{\bbl@KVP@import}}%
{\fIx\bbl@initoload\relax
\begingroup
\def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
\InputIfFileExists{babe1-#2.tex}{}{}
\endgroup

\bblproc
\endgroup}
% == script, language ==
\if\bbl@KVP@script@nil\else
\bbl@csarg\edef{sname@#2}{\bbl@KVP@script}\fi
\if\bbl@KVP@language@nil\else
\bbl@csarg\edef{lname@#2}{\bbl@KVP@language}\fi
% == onchar ==
\if\bbl@KVP@onchar@nil\else
\bbl@luahyphenate
\directlua{
if Babel.locale_mapped == nil then
    Babel.locale_mapped = true
    Babel.linebreaking.add_before(Babel.locale_map)
    Babel.loc_to_scr = {}
    Babel.chr_to_loc = Babel.chr_to_loc or {}
end}
\bbl@xin\{ ids \} { \bbl@KVP@onchar space }
\ifin@
\\bbl@starthyphens\undefined % Needed if no explicit selection
\AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}
\fi
\bbl@exp{\bbl@add\bbl@starthyphens
\\bbl@patterns lua{\languagename}}
% TODO - error/warning if no script
\directlua{
if Babel.script_blocks[\bbl@cl{sbcp}] then
    Babel.loc_to_scr[the\localeid] =
    Babel.script_blocks[\bbl@cl{sbcp}]
    Babel.localeProps[the\localeid].lc = the\localeid space
    Babel.localeProps[the\localeid].lg = the\nameuse{l\languagename} space
end
}
\fi
\bbl@xin\{ fonts \} { \bbl@KVP@onchar space }
\ifin@
\bbl@ifunset{bbl@lsys\languagename}{\bbl@provide@lsys\languagename}{}% 
\bbl@ifunset{bbl@wdir\languagename}{\bbl@provide@dirs\languagename}{}%
\directlua{
if Babel.script_blocks[\bbl@cl{sbcp}] then
    Babel.loc_to_scr[the\localeid] =
    Babel.script_blocks[\bbl@cl{sbcp}]
end}
\bbl@mapselect\undefined
\AtBeginDocument{%
\\expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
%{\selectfont}%
\def\bbl@mapselect{%
\let\bbl@mapselect\relax
\edef\bbl@prefontid{\fontid\font}%
\def\bbl@mapdir##1{%
{\def\language##1}%
\let\bbl@ifrestoring\firstoftwo % To avoid font warning
\bbl@switchfont
\directlua{
Babel.localeProps[the\csname bbl@id@##1\endcsname]%
['/bbl@prefontid'] = \fontid\font\space}}% 2968
\fi 2969 \bbl@exp{\\bbl@add\\bbl@mapselect\\\\bbl@mapdir\{\languagename\}}% 2970 \fi 2971 \% TODO - catch non-valid values 2972 \fi 2973 \% == mapfont == 2974 \% For bidi texts, to switch the font based on direction 2975 \ifx\bbl@KVP@mapfont\@nil\else 2976 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}% 2977 \{\bbl@error{Option `\bbl@KVP@mapfont' unknown for\% 2978 \mapfont. Use `direction'.% 2979 \See the manual for details.}}% 2980 \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}% 2981 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}% 2982 \ifx\bbl@mapselect\@undefined 2983 \AtBeginDocument{ 2984 \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}% 2985 \{\selectfont}}% 2986 \def\bbl@mapselect{% 2987 \let\bbl@mapselect\relax 2988 \edef\bbl@prefontid{\fontid\font}}% 2989 \def\bbl@mapdir##1{% 2990 \{\languagename{##1} 2991 \let\bbl@ifrestoring\@firstoftwo % avoid font warning 2992 \bbl@switchfont 2993 \directlua{Babel.fontmap 2994 [the\csname bbl@wdir@##1\endcsname]% 2995 [\bbl@prefontid]\fontid\font}}}% 2996 \fi 2997 \bbl@exp{\\bbl@add\\bbl@mapselect\\\\bbl@mapdir\{\languagename\}}% 2998 \fi 3000 \% == intraspace, intrapenalty == 3001 \% For CJK, East Asian, Southeast Asian, if interspace in ini 3002 \if\bbl@KVP@intraspace\@nil\else \% We can override the ini or set 3003 \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}\% 3004 \fi 3005 \bbl@provide@intraspace 3006 \% == hyphenate.other.locale == 3007 \bbl@ifunset{bbl@hyotl@\languagename}{% 3008 \{\bbl@csarg\bbl@replace{hyotl@\languagename}{}{},% 3009 \bbl@startcommands*{\languagename}{}% 3010 \bbl@csarg\bbl@foreach{hyotl@\languagename}{% 3011 \ifcase\bbl@engine 3012 \ifnum##1<257 3013 \SetHyphenMap{\BabelLower{##1}{##1}}% 3014 \fi 3015 \else 3016 \SetHyphenMap{\BabelLower{##1}{##1}}% 3017 \fi\)% 3018 \bbl@endcommands}% 3019 \% == hyphenate.other.script == 3020 \bbl@ifunset{bbl@hyots@\languagename}{% 3021 \{\bbl@csarg\bbl@replace{hyots@\languagename}{}{},% 3022 \bbl@startcommands*{\languagename}{}% 3023 \ifcase\bbl@engine 3024 \ifnum##1<257 3025 \global\lccode##1=##1\relax 3026 \fi
\else
  \global\lccode##1=##1\relax
\fi}%
% == maparabic ==
% Native digits, if provided in ini (TeX level, xe and lua)
\ifcase\bbl@engine\else
  \bbl@ifunset{bbl@dgnat@languagename}{}%
  {\expandafter\ifx\csname bbl@dgnat@languagename\endcsname\@empty\else
    \expandafter\expandafter\expandafter
    \bbl@setdigits\csname bbl@dgnat@languagename\endcsname
    \ifx\bbl@KVP@maparabic\@nil\else
      \expandafter\expandafter\expandafter
      \bbl@setdigits\csname bbl@dgnat@languagename\endcsname
    \fi
  \fi}%
\fi
% == mapdigits ==
% Native digits (lua level).
\ifodd\bbl@engine
  \ifx\bbl@KVP@mapdigits\@nil\else
    \bbl@ifunset{bbl@dgnat@languagename}{}%
    {\RequirePackage{luatexbase}\
      \bbl@activate@preotf
      \directlua{
        Babel = Babel or {} %%% -> presets in luababel
        Babel.digits_mapped = true
        Babel.digits = Babel.digits or {}
        Babel.digits[\the\localeid] =
          table.pack(string.utfvalue(\bbl@cl{dgnat}'))
        if not Babel.numbers then
          function Babel.numbers(head)
            local LOCALE = luatexbase.registernumber'\bbl@attr@locale'
            local GLYPH = node.id'\bbl@attr@locale'
            local inmath = false
            for item in node.traverse(head) do
              if not inmath and item.id == GLYPH then
                local temp = node.get_attribute(item, LOCALE)
                if Babel.digits[temp] then
                  local chr = item.char
                  if chr > 47 and chr < 58 then
                    item.char = Babel.digits[temp][chr-47]
                  end
                end
              elseif item.id == node.id'math' then
                inmath = (item.subtype == 0)
              end
            end
            return head
          end
        end
        \bbl@activate@preotf
      }%}
\fi
% == alph, Alph ==
A tool to define the macros for native digits from the list provided in the ini file.
Somewhat convoluted because there are 10 digits, but only 9 arguments in TEX. Non-digits
characters are kept. The first macro is the generic “localized” command.

% TODO. Merge with \localenumeral:
% \newcommand{\localeid}{\@nameuse{\languagename digits}}
% \edef{\bbl@setdigits}{\@nameuse{\languagename digits}}
% \bbl@exp{%
%   \def{\languagename digits}{\@nameuse{\languagename digits}}%  ie, \langdigits
%   \bbl@digits{\languagename digits}{\@nil}%
%   \let{\bbl@cntrdigits{\languagename digits}}%\@nil}
%   \def{\languagename counter}{\@nameuse{\languagename digits}}%
%   \edef{\bbl@counter{\languagename digits}}{\@nameuse{\languagename digits}}%
%   \let{\number{\languagename digits}}{\@nil}%
% }%
Depending on whether or not the language exists, we define two macros.

\begin{verbatim}
\def\bbl@provide@new#1{\@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
\@namedef{extras#1}{}% \@namedef{noextras#1}{}% \bbl@startcommands*{#1}{captions}%
\expandafter\ifx\expandafter\today\expandafter\@empty\else\
\SetString\today{\bbl@nocaption\bbl@stripslash\today}{#1\today}\
\expandafter\ifx\expandafter\today\expandafter\@empty\else\
\SetString\today{\bbl@nocaption\bbl@stripslash\today}{#1\today}\
\expandafter\expandafter\expandafter\bbl@tempb\bbl@captionslist\@empty\fi\%\expandafter\bbl@tempb\bbl@captionslist\@empty\else\ifx\bbl@initoload\relax\bbl@read@ini{\bbl@KVP@captions}{data}% Here letters cat = 11\else\bbl@read@ini{\bbl@initoload}{data}% Here all letters cat = 11\fi\%\bbl@savestrings\bbl@endcommands\if\bbl@ifunset{bbl@lname@#1}% TODO. Duplicated\begin{group}
\catcode`\[=12 \catcode`\]=12 \catcode`\|=12 %
\let\bbl@ini@captions@aux\@gobbletwo
\end{group}
\def\BabelBeforeIni##1##2{\begingroup
\catcode`\[=12 \catcode`\]=12 \catcode`\|=12 %
\let\bbl@ini@captions@aux\@gobbletwo
\endgroup}
\end{verbatim}


The hyphenrules option is handled with an auxiliary macro.

\def\bbl@provide@hyphens#1{%
  \let\bbl@tempa\relax
  \ifx\bbl@KVP@hyphenrules@nil\else
    \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
    \bbl@foreach\bbl@KVP@hyphenrules{##1}%
      {\bbl@ifsamestring{##1}{+}{{\bbl@exp{\addlanguage<l@##1>}}}{}%}
      {\bbl@ifunset{l@##1}{}{\bbl@exp{\let\bbl@tempa<l@##1>}}}%
  \fi
\}

\if\bbl@KVP@import@nil\else
  \StartBabelCommands*{#1}{caption}%
  \bbl@readini{\bbl@KVP@caption}{data}% Here all letters cat = 11
  \bbl@afterni
  \bbl@ savetoday
  \bbl@ savedate
  \EndBabelCommands
\fi

% == hyphenrules ==
\bbl@provide@hyphens{#1}
There are three possible cases: a section name (in the form \[...\]), a comment (starting with ;) and a key/value pair.
The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the possibility to take extra actions at the end or at the start (TODO - but note the last section is not ended). By default, key=val pairs are ignored. The secpost “hook” is used only by ‘identification’, while secpre only by date.gregorian.licr.
Reads a key=val line and stores the trimmed val in \bbl@@kv@<section>\.<key>.

The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@secpost@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.
Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

```
\ifcase\bbl@engine
  \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{\bbl@ini@captions@aux{#1}{#2}}
\else
  \def\bbl@inikv@captions#1=#2\@@{\bbl@ini@captions@aux{#1}{#2}}
\fi
```

The auxiliary macro for captions define `<caption>`name.
TODO. Document

% Arguments are _not_ protected.
\let\bbl@calendar\@empty
\DeclareRobustCommand\localedate[1][]\{
\bbl@localedate{#1}
\}
\def\bbl@cased{% TODO. Move
\ifx\oe\OE
\expandafter\in@\expandafter\OE\expandafter{\oe}
\ifin@
\bbl@afterelse\expandafter\MakeUppercase
\else
\bbl@afterfi\expandafter\MakeLowercase
\fi
\else
\expandafter\@firstofone
\fi
\bbl@localedate#1#2#3#4{%
\begin{group}
\ifx\empty#1\empty\else
\let\bbl@ld@calendar\@empty
\let\bbl@ld@variant\@empty
\edef\bbl@tempa{\zap@space#1 \@empty}%
\def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
\bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
\edef\bbl@calendar{\bbl@ld@calendar
\ifx\bbl@ld@variant\@empty\else.bl@ld@variant\fi}%
\bbl@replace\bbl@calendar{gregorian}{}
\fi
\bbl@cased{\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
\end{group}
\def\bbl@localedate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
\bbl@trim@def\bbl@tempa{#1.#2}%
\bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
{\bbl@trim@def\bbl@tempa{#3}%
\bbl@exp{% Reverse order - in ini last wins
\def\\bbl@savedate{\SetString\langle\bbl@tempa name>{\the\toks@}}%
\bbl@exp{%
\bbl@add\bbl@savestrings{%
\SetString\langle\bbl@tempa name>{\the\toks@}}%
\}
\fi
\bbl@replace\bbl@calendar{gregorian}{}%
\bbl@ifunset{\bbl@date\@languagename @}\
{\global{\bbl@csarg\let{date\@languagename @}\bbl@toreplace}
% TODO. Move to a better place.
\bbl@exp{%
  \gdef\<\languagename date>\{\protect\<\languagename date >\}\
  \gdef\<\languagename date >####1####2####3{\
    \bbl@usedategrouptrue\<bbl@ensure\@languagename>{\localedate{####1}{####2}{####3}}\
  }\bbl@savetoday{%
    {\\the\year{\\the\month{\\the\day}}}\
  }%}
}{%}
\ifx\bbl@tempb\@empty\else
\global{\bbl@csarg\let{date\@languagename @\bbl@tempb}\bbl@toreplace}
\fi}\
}

Dates will require some macros for the basic formatting. They may be redefined by
language, so “semi-public” names (camel case) are used. Oddly enough, the CLDR places
particles like “de” inconsistently in either in the date or in the month name.

\let\bbl@calendar\@empty
\newcommand{\BabelDateSpace}{\nobreakspace}
\newcommand{\BabelDateDot}{.\@}
\newcommand{\BabelDated}[1]{{\number#1}}
\newcommand{\BabelDatedd}[1]{{\ifnum#1<10 0\fi\number#1}}
\newcommand{\BabelDateM}[1]{{\number#1}}
\newcommand{\BabelDateMM}[1]{{\ifnum#1<10 0\fi\number#1}}
\newcommand{\BabelDateMMMM}[1]{{\csname month\romannumeral#1\bbl@calendar name\endcsname}}
\newcommand{\BabelDatey}[1]{{\number#1}}
\newcommand{\BabelDateyy}[1]{{%\ifnum#1<10 0\number#1 %\else\ifnum#1<100 \number#1 %\else\ifnum#1<1000 \expandafter\@gobble\number#1 %\else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %\else\bbl@error{Currently two-digit years are restricted to the\range 0-9999.} %\There is little you can do. Sorry.}%\fi\fi\fi\fi}}
\newcommand{\BabelDateyyyy}[1]{{\number#1}} % FIXME - add leading 0
\def{\bbl@replace@finish@iii#1{\bbl@exp{\def\#1####1####2####3{\the\toks@}}}}
\def{\bbl@TG@date%}
\bbl@replace{\bbl@toreplace[ ]}{\BabelDateSpace{}}
\bbl@replace{\bbl@toreplace[.]}{\BabelDateDot{}}
\bbl@replace{\bbl@toreplace[[]}{\BabelDated{}}
\bbl@replace{\bbl@toreplace[[[}{\BabelDatedd{}}
\bbl@replace{\bbl@toreplace[[[}[}{\BabelDateM{}}
\bbl@replace{\bbl@toreplace[[[[}{\BabelDateMM{}}
\bbl@replace{\bbl@toreplace[[[[[}{\BabelDateMMMM{}}
\bbl@replace{\bbl@toreplace[[[[[}{\BabelDatey{}}
\bbl@replace{\bbl@toreplace[[[[][}{\BabelDateyy{}}
\bbl@replace{\bbl@toreplace[[[[][[}{\BabelDateyyyy{}}
\bbl@replace{\bbl@toreplace[[[}{\bbl@datecntr[\@]}%
Language and Script values to be used when defining a font or setting the direction are set
with the following macros.

The following ini reader ignores everything but the identification section. It is called
when a font is defined (i.e., when the language is first selected) to know which
script/language must be enabled. This means we must make sure a few characters are not
active. The ini is not read directly, but with a proxy \text file named as the language (which
means any code in it must be skipped, too).

Alphabetic counters must be converted from a space separated list to an \ifcase structure.

The code for additive counters is somewhat tricky and it's based on the fact the arguments
just before \@@ collects digits which have been left 'unused' in previous arguments, the
first of them being the number of digits in the number to be converted. This explains the
reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the
subkey \F., the number after is treated as a special case, for a fixed form (see
babel-he.ini, for example).
The information in the identification section can be useful, so the following macro just exposes it with a user command. More general, but non-expandable, is \getlocaleproperty. To inspect every possible loaded ini, we define \LocalForEach, where \bbl@ini@loaded is a comma-separated list of locales, built by \bbl@read@ini.
10 Adjusting the Babel behavior

A generic high level interface is provided to adjust some global and general settings.
11 Loading hyphenation patterns

The following code is meant to be read by \texttt{init\TeX} because it should instruct \TeX to read hyphenation patterns. To this end the \texttt{docstrip} option patterns can be used to include this code in the file \texttt{hyphen.cfg}. Code is written with lower level macros.
To make sure that \TeX 2.09 executes the \begindocument hook we would want to alter \begin{document}, but as this done too often already, we add the new code at the front of \preamblecmds. But we can only do that after it has been defined, so we add this piece of code to \dump.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns. Then everything is restored to the old situation and the format is dumped.

\ProvidesFile{hyphen.cfg}[\date\version] Babel hyphens
\xdef\bbl@format{\jobname}
\def\bbl@version{\version}
\def\bbl@date{\date}
\ifx\AtBeginDocument\@undefined
\def\@empty{}
\let\orig@dump\dump
\def\dump{%
\ifx\@ztryfc\@undefined
\else
\toks0=\expandafter{\@preamblecmds}%
\edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
\def\@begindocumenthook{}%
\fi
\let\dump\orig@dump\let\orig@dump\@undefined\dump%
\fi
\Define core switching macros
\process@line Each line in the file language.dat is processed by \process@line after it is read. The first thing this macro does is to check whether the line starts with =. When the first token of a line is an =, the macro \process@synonym is called; otherwise the macro \process@language will continue.
\process@synonym This macro takes care of the lines which start with an =. It needs an empty token register to begin with. \bbl@languages is also set to empty.

When no languages have been loaded yet, the name following the = will be a synonym for hyphenation register 0. So, it is stored in a token register and executed when the first pattern file has been processed. (The \relax just helps to the \if below catching synonmys without a language.) Otherwise the name will be a synonym for the language loaded last. We also need to copy the hyphenmin parameters for the synonym.

\process@line If line begins with =, \process@synonym is called; otherwise \process@language is called.

\process@language This macro processes the line and loads the language.
Themacro \process@languageis used to process a non-empty line from the ‘configuration
file’. Ithasthreearguments, each delimited by white space. The first argument is the
‘name’ of a language; the second is the name of the file that contains the patterns. The
optional third argument is the name of a file containing hyphenation exceptions.
The first thing to do is call \addlanguage to allocate a pattern register and to make that
register ‘active’. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding
selected. This can be specified in the file language.dat by adding for instance ‘:\T1’ to the
name of the language. The macro \bl@get@enc extracts the font encoding from the
language name and stores it in \bl@hyph@enc. The latter can be used in hyphenation files
if you need to set a behavior depending on the given encoding (it is set to empty if no
encoding is given).

Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. \TeX does
not keep track of these assignments. Therefore we try to detect such assignments and store
them in the \(⟨\text{lang}⟩\)hyphenminsmacro. When no assignments were made we provide a
default setting.

Some pattern files contain changes to the \lccode and \uccode arrays. Such changes should
remain local to the language; therefore we process the pattern file in a group; the
\patterns command acts globally so its effect will be remembered.

When the hyphenation patterns have been processed we need to see if a file with
hyphenation exceptions needs to be read. This is the case when the third argument is not
empty and when it does not contain a space token. (Note however there is no need to save
hyphenation exceptions into the format.)

\bl@languages saves a snapshot of the loaded languages in the form
\bl@elt{⟨language-name⟩}{⟨number⟩}{⟨patterns-file⟩}{⟨exceptions-file⟩}. Note the last
2 arguments are empty in ‘dialects’ defined in language.dat with =. Note also the
language name can have encoding info.

Finally, if the counter \language is equal to zero we execute the synonyms stored.
The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. It uses delimited arguments to achieve this.

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way. Besides luatex, format-specific configuration files are taken into account. \loadkernel currently loads nothing, but define some basic macros instead.
\let\locale\setlocale
\let\selectlocale\setlocale
\let\localename\setlocale
\let\textlocale\setlocale
\let\textlanguage\setlocale
\let\languagetext\setlocale

\begingroup
\def\AddBabelHook#1#2{%\expandafter\ifx\csname bbl@hook@#2\endcsname\relax
\def\next{\toks1}%\else
\def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%\fi
\next}
\ifx\directlua\@undefined
\ifx\XeTeXinputencoding\@undefined\else
\input xebabel.def
\fi
\else
\input luababel.def
\fi
\ifx\directlua\@undefined
\input xebabel.def
\else
\input luababel.def
\fi
\openin1 = babel-\bbl@format.cfg
\ifeof1
\else
\input babel-\bbl@format.cfg\relax
\fi
\closein1
\endgroup
\bbl@hook@loadkernel{switch.def}

\readconfigfile The configuration file can now be opened for reading.
\openin1 = language.dat

See if the file exists, if not, use the default hyphenation file hyphen.tex. The user will be informed about this.
\def\languagename{english}%
\ifeof1
\message{I couldn’t find the file language.dat, I will try the file hyphen.tex}
\input hyphen.tex\relax
\chardef\l@english\z@
\else

Pattern registers are allocated using count register \last@language. Its initial value is 0. The definition of the macro \newlanguage is such that it first increments the count register and then defines the language. In order to have the first patterns loaded in pattern register number 0 we initialize \last@language with the value \(-1\).
\last@language\m@ne

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.
\loop
\endlinechar\m@ne
\read1 to \bbl@line
\endlinechar`^^M
If the file has reached its end, exit from the loop here. If not, empty lines are skipped. Add 3 space characters to the end of `\bbl@line`. This is needed to be able to recognize the arguments of `\process@line` later on. The default language should be the very first one.

```
\if \ifeof \fi \relax
\ifx \bbl@line \@empty \else
  \edef \bbl@line { \bbl@line \space \space \space }%
  \expandafter \process@line \bbl@line \relax
\fi
\repeat
```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns, and close the configuration file.

```
\begingroup
  \def \bbl@elt #1#2#3#4 {%
    \global \language=#2 \relax
    \gdef \languagename{#1}%
    \def \bbl@elt ##1##2##3##4{}}%
\bbl@languages
\endgroup
\fi
\closein
```

We add a message about the fact that babel is loaded in the format and with which language patterns to the `\everyjob` register.

```
\if \the \toks@ / \else
  \errhelp {language.dat loads no language, only synonyms}
  \errmessage {Orphan language synonym}
\fi
```

Also remove some macros from memory and raise an error if `\toks@` is not empty. Finally load `switch.def`, but the latter is not required and the line inputting it may be commented out.

```
\let \bbl@line \undefined
\let \process@line \undefined
\let \process@synonym \undefined
\let \process@language \undefined
\let \bbl@get@enc \undefined
\let \bbl@hook@loadkernel \undefined
\let \bbl@hook@everylanguage \undefined
\let \bbl@hook@loadpatterns \undefined
\let \bbl@hook@loadexceptions \undefined
\let \patterns \undefined
```

Here the code for `init\TeX` ends.

## 12 Font handling with fontspec

Add the bidi handler just before `luatexload`, which is loaded by default by LaTeX. Just in case, consider the possibility it has not been loaded. First, a couple of definitions related to bidi [misplaced].

```
\DeclOption { bidi=\chardef \bbl@bidimode \@one }
\DeclOption { bidi=\chardef \bbl@bidimode=\@one }
\DeclOption { bidi=\chardef \bbl@bidimode=\@one }
\DeclOption { bidi=basic-r } { \chardef \bbl@bidimode=\@two }
```

154
With explicit languages, we could define the font at once, but we don’t. Just wait and see if the language is actually activated. bbl@font replaces hardcoded font names inside \..family by the corresponding macro \..default.

If the family in the previous command does not exist, it must be defined. Here is how:

The following macro is activated when the hook babel-fontspec is enabled. But before we define a macro for a warning, which sets a flag to avoid duplicating them.
There is nothing intrinsically wrong with this warning, and you can ignore it altogether if you do not need these families. But if they are used in the document, you should be aware ‘babel’ will no set Script and Language for them, so you may consider defining a new family with \string\babelfont. See the manual for further details about \string\babelfont.
4068 settings for all or some languages:\%
4069 \bbl@tempa
4070 There is nothing intrinsically wrong with it, but\%
4071 'babel' will no set Script and Language, which could\%
4072 be relevant in some languages. If your document uses\%
4073 these families, consider redefining them with \string\babelfont.\%
4074 Reported}\%
4075 \fi
4076 \endgroup}
4077 \fi
4078 \fi

Now the macros defining the font with fontspec.
When there are repeated keys in fontspec, the last value wins. So, we just place the ini
settings at the beginning, and user settings will take precedence. We must deactivate
temporarily \bbl@mapselect because \selectfont is called internally when a font is
defined.

4079 \def\bbl@font@set#1#2#3{\% eg \bbl@rmdefault \rmfamily
4080 \bbl@xin@<>(#1)}\fi
4081 \ifin@
4082 \bbl@exp{\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\#3}\fi
4083 \fi
4084 \bbl@exp{\% TODO - next should be global?, but even local does its job. I'm
4085 \let\bbl@fontspec@set\#2(\% eg, \rmdefault{\bbl@rmdefault}\lang
4086 \bbl@if@string{\lang\#2}{\{f@family}\{\#3\let\\bbl@tempa\relax\}}}\}
4087 \% still not sure -- must investigate:
4088 \def\bbl@fontspec@set#1#2#3#4{\% eg \bbl@rmdefault\lang fnt-opt fnt-nme \xxfamily
4089 \let\bbl@tempfam\bbl@mapselect
4090 \let\bbl@mapselect@relax
4091 \let\bbl@mapselect@relax
4092 \let\bbl@tempfam\fam\fi
4093 \let\#4\#4\empty  \% Make sure \renewfontfamily is valid
4094 \bbl@exp{\% eg, ', \{f@family\}' to be restored below
4095 \let\#4\empty \% Make sure \renewfontfamily is valid
4096 \bbl@exp{\% eg, ', \{f@family\}'
4097 \{\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}\%
4098 \{\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}\%
4099 \{\newfontfamily\{\#4\%
4100 \{[\bbl@cs{languagename},#2]}\#3\} ie \bbl@exp{..}{}\}
4101 \begin{group}
4102 \begin{group}
4103 \end{group}
4104 \end{group}
4105 \end{group}
4106 \texttt{\let\#4\bbl@tempfam
4107 \bbl@exp{\let\\bbl@strip\space}\bbl@tempfam
4108 \let\bbl@mapselect\bbl@tempfam\%

font@rst and famrst are only used when there is no global settings, to save and restore de
previous families. Not really necessary, but done for optimization.

4109 \def\bbl@font@fams\{#1\#2\#3\#4\%
4110 \bbl@csarg\def\famrst\{#4\}\{\bbl@font@fams\{#1\#2\#3\})

The default font families. They are eurocentric, but the list can be expanded easily with
\babelfont.

4111 \def\bbl@font@fams\{rm,sf,tt\}

The old tentative way. Short and preserved for compatibility, but deprecated. Note there is
no direct alternative for \babelfont\{features\}. The reason in explained in the user guide, but
essentially – that was not the way to go :-).

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8, which seems a sensible default.
Now, the code.

\def\BabelStringsDefault{\text{}%
13.2 Layout

In progress.

Note elements like headings and margins can be modified easily with packages like fancyhdr, typearea or titelles, and geometry. \bbl@startskip and \bbl@endskip are available to package authors. Thanks to the \TeX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for tex–xet babel, which is the hidi model in both pdftex
and \textxet.

\providecommand{\provide@intraspaces{}%}
\bbl@trace{Redefinitions for bidi layout}
\def\bbl@spre@caption{\
\bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
\ifx\bbl@opt@layout\@nnil\endinput % No layout
\def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
\def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
\ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
\def\@hangfrom#1{\
\setbox\@tempboxa\hbox{{#1}}% \
\hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi 
\noindent\box\@tempboxa}
\def\raggedright{% 
\let\@centercr\@centercr 
\bbl@startskip\z@skip 
\@rightskip\@flushglue 
\bbl@endskip\z@skip 
\parindent\z@ 
\parfillskip\bbl@startskip}
\def\raggedleft{% 
\let\@centercr\@centercr 
\bbl@startskip\@flushglue 
\bbl@endskip\z@skip 
\parindent\z@ 
\parfillskip\bbl@endskip}
\fi
\IfBabelLayout{lists}{\bbl@sreplace\list 
{\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@list@leftmargin}% 
\def\bbl@list@leftmargin{% 
\ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}% 
\ifcase\bbl@engine 
\def\labellenumii\theenumii\% pdftex doesn't reverse () 
\def\p@enumii\p@enumii\theenumii\% 
\fi
\bbl@replace@verbatim 
{\leftskip\@totalleftmargin}
\bbl@replace@verbatim 
{\rightskip\z@skip} 
\bbl@sreplace\verbatim 
{\bbl@startskip\textwidth 
\advance\bbl@startskip-\linewidth}
\bbl@replace\verbatim 
{\rightskip\z@skip}
\bbl@replace@verbatim 
{\bbl@endskip\z@skip} 
{}}
\IfBabelLayout{contents}{\bbl@sreplace\dottedtocline{\leftskip}{\bbl@startskip}{\bbl@endskip}}
{}}
\IfBabelLayout{columns}{\bbl@sreplace\outputdbcol{\hb@xt@\textwidth}{\bbl@outputbox}% 
\def\bbl@outputbox@#1{% 
\hb@xt@\textwidth 
\hskip\columnwidth 
\hfil 
{\normalcolor\vrule \@width\columnseprule}% 
\hfil 
\hb@xt@\columnwidth{\box@\leftcolumn \hss}
Implicitly reverses sectioning labels in bidi=basic, because the full stop is not in contact with \l@ numbers any more. I think there must be a better way.

13.3 LuaTeX

The loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn't be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).

The names \l@<language> are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, \bbl@hyphendata@<num> exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for ‘english’, so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the “0th” language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, the are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the “0th” language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.
This file is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

\begin{verbatim}
4342 \langle\texttt{luatex}\rangle
4343 \% When plain.def, babel.sty starts
4344 \bbl@trace{Read language.dat}
4345 \% hyphen.cfg, to modify some macros
4346 \csname newread\endcsname\bbl@readstream
4347 \fi
4348 \begingroup
4349 \toks@{}
4350 \count@\z@ % 0=start, 1=0th, 2=normal
4351 \def\bbl@process@line#1#2 #3 #4 {%
4352 \ifx=#1%
4353 \bbl@process@synonym(#2)%
4354 \else
4355 \bbl@process@language{#1#2}{#3}{#4}%
4356 \fi
4357 \ignorespaces
4358 \def\bbl@manylang{%
4359 \ifnum\bbl@last>\@ne
4360 \bbl@info{Non-standard hyphenation setup}%
4361 \fi
4362 \let\bbl@manylang\relax
4363 \def\bbl@process@language#1#2#3{%
4364 \ifcase\count@
4365 \ifundefined{zth@#1}\count@\tw@\count@\@ne%}
4366 \or
4367 \count@\tw@
4368 \fi
4369 \ifnum\count@=\tw@
4370 \expandafter\addlanguage\csname l@#1\endcsname
4371 \language\allocationnumber
4372 \chardef\bbl@last\allocationnumber
4373 \bbl@manylang
4374 \let\bbl@elt\relax
4375 \edef\bbl@languages{%
4376 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4377 \fi
4378 \the\toks@
4379 \toks@{}}
4380 \def\bbl@process@synonym@aux#1#2{%
4381 \global\expandafter\addlanguage\csname l@#1\endcsname#2\relax
4382 \let\bbl@elt\relax
4383 \edef\bbl@languages{%
4384 \bbl@languages\bbl@elt{#1}{#2}{}}{}
4385 \def\bbl@process@synonym#1{%
4386 \ifcase\count@
4387 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4388 \or
4389 \ifundefined{zth@#1}\bbl@process@synonym@aux{#1}{0}{}}%
4390 \else
4391 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4392 \fi
4393 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4394 \chardef\l@english\z@
4395 \chardef\l@USenglish\z@
\end{verbatim}
\chardef\bbl@last\z@
\global\namedef{bbl@hyphendata@0}{{hyphen.tex}{}%}
\def\bbl@languages{%
  \bbl@elt{english}{0}{hyphen.tex}{}%
  \bbl@elt{USenglish}{0}{}{}
}\else
\global\let\bbl@languages@format\bbl@languages
\def\bbl@elt#1#2#3#4{% Remove all except language 0
  \ifnum#2>\z@\else
    \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
  \fi
}\xdef\bbl@languages{\bbl@languages}%
\def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
\bbl@languages
\openin\bbl@readstream=language.dat
\ifeof\bbl@readstream
  \bbl@warning{I couldn't find language.dat. No additional patterns loaded. Reported}%
\else
\loop\endlinechar\m@ne
  \read\bbl@readstream to \bbl@line\endlinechar`\^^M
  \if T\ifeof\bbl@readstream F\fi T\relax
  \ifx\bbl@line\@empty\else
    \edef\bbl@line{\bbl@line\space\space\space}%
    \expandafter\bbl@process@line\bbl@line\relax
  \fi
\repeat
\fi
\endgroup
\bbl@trace{Macros for reading patterns files}
\def\bbl@get@enc#1:#2:#3@@@{\def\bbl@hyph@enc{#2}}
\ifx\babelcatcodetablenum\@undefined
  \ifx\newcatcodetable\@undefined
    \def\babelcatcodetablenum{5211}
  \else\newcatcodetable\babelcatcodetablenum\fi\newcatcodetable\bbl@pattcodes\relax
\else\def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}\fi
\def\bbl@luapatterns#1#2{\bbl@get@enc#1::@@@\setbox\z@\hbox{\begingroup}
  \savecatcodetable\babelcatcodetablenum\relax
  \initcatcodetable\bbl@pattcodes\relax
  \catcode`#=6 \catcode`$=3 \catcode`&=4 \catcode`^=7
  \catcode`_=8 \catcode`\%=2 \catcode`\-=13
  \catcode`@=11 \catcode`\^^I=10 \catcode`\^^J=12
  \catcode`<=12 \chardef\bbl@luapatterns\chardef\bbl@luapatterns
  \catcode`\*=12 \catcode`\={12 \catcode`\=12
  \input \#\relax
}
\catcodetable\babelcatcodetablenum\relax
\endgroup
\def\bbl@tempa{#2}\
\ifx\bbl@tempa\@empty\else
\input #2\relax
\fi
\egroup}\
\def\bbl@patterns@lua#1{\
\language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
\csname l@#1\endcsname\
\edef\bbl@tempa{#1}\
\else
\csname l@#1:\f@encoding\endcsname\
\edef\bbl@tempa{#1:\f@encoding}\
\fi\relax
@namedef{lu@texhyphen@loaded@	he\language}{}% Temp
\@ifundefined{bbl@hyphendata@	he\language}\{
\def\bbl@elt##1##2##3##4{\
\ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...\
\def\bbl@tempb{##3}\
\ifx\bbl@tempb\@empty\else % if not a synonymous\
\def\bbl@tempc{{##3}{##4}}\
\fi\
\bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}\
\fi}\
\bbl@languages
\@ifundefined{bbl@hyphendata@	he\language}\{
\bbl@info{No hyphenation patterns were set for\
language `\bbl@tempa'. Reported}}\
\else
\expandafter\expandafter\expandafter\bbl@luapatterns
\csname bbl@hyphendata@	he\language\endcsname\}}}}
\endinput\fi
% Here ends \ifx\AddBabelHook\@undefined
% A few lines are only read by hyphen.cfg
\ifx\DisableBabelHook\@undefined
\AddBabelHook{luatex}{everylanguage}{%
  \def\process@language##1##2##3#1}{}}
\AddBabelHook{luatex}{loadpatterns}{%
  \input #1\relax
  \expandafter\gdef\csname bbl@hyphendata@	he\language\endcsname
  {{#1}}}{}
\AddBabelHook{luatex}{loadexceptions}{%
  \input #1\relax
  \expandafter\edef\csname bbl@hyphendata@	he\language\endcsname
  {\expandafter\expandafter\expandafter\bbl@tempb
  \csname bbl@hyphendata@	he\language\endcsname}}
\endinput\fi
% Here stops reading code for hyphen.cfg
% The following is read the 2nd time it's loaded
\begin{group}
\catcode`\%=12
\catcode`\=12
\catcode`\%=12
\directlua{
Babel = Babel or {}
function Babel.bytes(line)
return line.gsub("(.\)",
  function (chr) return unicode.utf8.char(string.byte(chr)) end)
end

function Babel.begin_process_input()
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.add_to_callback('process_input_buffer',
      Babel.bytes, 'Babel.bytes')
  else
    Babel.callback = callback.find('process_input_buffer')
    callback.register('process_input_buffer', Babel.bytes)
  end
end

function Babel.end_process_input()
  if luatexbase and luatexbase.remove_from_callback then
    luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
  else
    callback.register('process_input_buffer', Babel.callback)
  end
end

function Babel.addpatterns(pp, lg)
  local lg = lang.new(lg)
  local pats = lang.patterns(lg) or ''
  lang.clear_patterns(lg)
  for p in pp:gmatch('[^%s\+]+') do
    ss = ''
    for i in string.utfcharacters(p:gsub('%d', '')) do
      ss = ss .. '%d?' .. i
    end
    ss = ss:gsub('^%d%.', '%%.') .. '%d?'
    ss = ss:gsub('%.%d$.', '%%.')
    pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
    if n == 0 then
      tex.sprint("
        [[\string\csname space bbl@info\endcsname{New pattern: ]]
        .. p .. [[]]])
      pats = pats .. ' ' .. p
    else
      tex.sprint("
        [[\string\csname space bbl@info\endcsname{Renew pattern: ]]
        .. p .. [[]]])
    end
    end
  lang.patterns(lg, pats)
end

{group}
\def\BabelStringsDefault{unicode}
\let\luabbl@stop\relax
\AddBabelHook{luatex}{encodedcommands}{
  \def\bbl@tempa{utf8}\def\bbl@tempb{#1}\ifx\bbl@tempa\bbl@tempb\else
    \directlua{Babel.begin_process_input()}%}
\directlua{Babel.end_process_input()}</group}
This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the global ones and \bbl@patterns@<lang> for language ones. We make sure there is a space between words when multiple commands are used.
13.4 Southeast Asian scripts

First, some general code for line breaking, used by \babelpostphyphenation. In progress. Replace regular (i.e., implicit) discretionary by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous character always together). Other discretionary are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

\directlua{
Babel = Babel or {}
Babel.linebreaking = Babel.linebreaking or {}
Babel.linebreaking.before = {}
Babel.linebreaking.after = {}
Babel.locale = {} % Free to use, indexed with \localeid
function Babel.linebreaking.add_before(func)
tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
table.insert(Babel.linebreaking.before, func)
end

function Babel.linebreaking.add_after(func)
tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
table.insert(Babel.linebreaking.after, func)
end
}
\def\bbl@intraspace#1 #2 #3\@@{\directlua{
Babel = Babel or {}
Babel.intraspaces = Babel.intraspaces or {}
Babel.intraspaces[[\csname bbl@sbcp@\languagename\endcsname']] = %
{b = #1, p = #2, m = #3}
Babel.locale_props[\the\localeid].intraspace = %
{b = #1, p = #2, m = #3}
}}
\def\bbl@intrapenalty#1\@@{\directlua{
Babel = Babel or {}
Babel.intrapenalties = Babel.intrapenalties or {}
Babel.intrapenalties[[\csname bbl@sbcp@\languagename\endcsname']] = #1
Babel.locale_props[\the\localeid].intrapenalty = #1
}}
\begingroup
\catcode`%=12
\catcode`^=14
\catcode`\'=12
\catcode`~=12
\gdef\bbl@seaintraspace{^}
\let\bbl@seaintraspace\relax

168
```lua
Babel = Babel or {}
Babel.sea_enabled = true
Babel.sea_ranges = Babel.sea_ranges or {}
function Babel.set_chranges (script, chrng)
  local c = 0
  for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
    Babel.sea_ranges[script..c] = {tonumber(s,16), tonumber(e,16)}
    c = c + 1
  end
end

function Babel.sea_disc_to_space (head)
  local sea_ranges = Babel.sea_ranges
  local last_char = nil
  local quad = 655360 * 10  \^ 10 pt = 655360 = 10 * 65536
  for item in node.traverse(head) do
    local i = item.id
    if i == node.id'glyph' then
      last_char = item
    elseif i == 7 and item.subtype == 3 and last_char
      and last_char.char > 0x0C99 then
      quad = font.getfont(last_char.font).size
      for lg, rg in pairs(sea_ranges) do
        if last_char.char > rg[1] and last_char.char < rg[2] then
          lg = lg:sub(1, 4)  \^ Remove trailing number of, eg, Cyril
          local intraspace = Babel.intraspaces[lg]
          local intrapenalty = Babel.intrapenalties[lg]
          local n
          if intrapenalty ~= 0 then
            n = node.new(14, 0)  \^ penalty
            n.penalty = intrapenalty
            node.insert_before(head, item, n)
          end
          n = node.new(12, 13)  \^ (glue, spaceskip)
          node.setglue(n, intraspace.b * quad,
                        intraspace.p * quad,
                        intraspace.m * quad)
          node.insert_before(head, item, n)
          node.remove(head, item)
        end
      end
    end
  end
end

function Babel.cjk_linebreak (head)
  local GLYPH = node.id'glyph'
  local last_char = nil
  local quad = 655360  \^ 10 pt = 655360 = 10 * 65536
  for item in node.traverse(head) do
    local i = item.id
    if i == node.id'glyph' then
      last_char = item
    elseif i == 7 and item.subtype == 3 and last_char
      and last_char.char > 0x0C99 then
      quad = font.getfont(last_char.font).size
      for lg, rg in pairs(Babel.cjk_intraspaces) do
        if last_char.char > rg[1] and last_char.char < rg[2] then
          lg = lg:sub(1, 4)  \^ Remove trailing number of, eg, Cyril
          local intraspace = Babel.cjk_intraspaces[lg]
          local intrapenalty = Babel.cjk_intrapenalties[lg]
          local n
          if intrapenalty ~= 0 then
            n = node.new(14, 0)  \^ penalty
            n.penalty = intrapenalty
            node.insert_before(head, item, n)
          end
          n = node.new(12, 13)  \^ (glue, spaceskip)
          node.setglue(n, intraspace.b * quad,
                        intraspace.p * quad,
                        intraspace.m * quad)
          node.insert_before(head, item, n)
          node.remove(head, item)
        end
      end
    end
  end
end
```

169
for item in node.traverse(head) do
  if item.id == GLYPH then
    local lang = item.lang
    local LOCALE = node.get_attribute(item,
      \luatexbase.registernumber\texttt{bbl@attr\texttt{locale'}})
    local props = Babel.locale_props[LOCALE]
    local class = Babel.cjk_class[item.char].c
    if class == 'cp' then class = 'cl' end % )
    if class == 'id' then class = 'I' end
    local br = 0
    if class and last_class and Babel.cjk_breaks[last_class][class] then
      br = Babel.cjk_breaks[last_class][class]
    end
    if br == 1 and props.linebreak == 'c' and
      lang ~= \the\l@nohyphenation\space and
      last_lang ~= \the\l@nohyphenation then
      local intrapenalty = props.intrapenalty
      if intrapenalty ~= 0 then
        local n = node.new(14, 0) % penalty
        n.penalty = intrapenalty
        node.insert_before(head, item, n)
      end
      local intraspace = props.intraspace
      local n = node.new(12, 13) % (glue, spaceskip)
      node.setglue(n, intraspace.b * quad,
        intraspace.p * quad,
        intraspace.m * quad)
      node.insert_before(head, item, n)
      end
    end
    quad = font.getfont(item.font).size
    last_class = class
    last_lang = lang
    else % if penalty, glue or anything else
      last_class = nil
    end
  lang.hyphenate(head)
end
\let\bbl@luahyphenate\relax
\directlua{
  \luatexbase.add_to_callback('hyphenate',
    \function (head, tail)
      \if Babel.linebreaking.before then
        for k, func in ipairs(Babel.linebreaking.before) do
          func(head)
        end
      end
      \if Babel.cjk_enabled then
13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth vs. halfwidth), not yet used. There is a separate file, defined below.
13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverses the node list to carry out the replacements. The table `loc_to_scr` gets the locale form as script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```latex
\directlua{
Babel.script_blocks = {
  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
  ['Armn'] = {{0x0530, 0x058F}},
  ['Beng'] = {{0x0980, 0x09FF}},
  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
  ['Cyril'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
  ['Deva'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
             {0x0AB00, 0x0AB2F}},
  ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF}},
  ['Hebr'] = {{0x0590, 0x05FF}},
  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
  ['Khm'] = {{0x01780, 0x017FF}, {0x19E0, 0x19FF}},
  ['Knda'] = {{0x0C80, 0x0CFF}},
  ['Kore'] = {{0x11100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
              {0x4E00, 0x9FAF}, {0xA900, 0xA97F}, {0xAC00, 0xD7AF}},
  ['Laoo'] = {{0x0E80, 0x0EFF}},
  ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
  ['Majh'] = {{0x11150, 0x1117F}},
  ['Mlym'] = {{0x0D00, 0x0D7F}},
  ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
  ['Orya'] = {{0x0B00, 0x0B7F}},
  ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
}
```
['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
['Taml'] = {{0x0B80, 0x0BFF}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x2D30, 0x2D7F}},
['Thai'] = {{0x0E00, 0x0EFF}},
['Tibt'] = {{0x0F00, 0x0FFF}},
['Vaii'] = {{0xA500, 0xA63F}},
['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
}

Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl

function Babel.locale_map(head)
if not Babel.locale_mapped then return head end
local LOCALE = luatexbase.registernumber'bbl@attr@locale'
local GLYPH = node.id('glyph')
local inmath = false
local toloc_save
for item in node.traverse(head) do
  local toloc
  if not inmath and item.id == GLYPH then
    % Optimization: build a table with the chars found
    if Babel.chr_to_loc[item.char] then
      toloc = Babel.chr_to_loc[item.char]
    else
      for lc, maps in pairs(Babel.loc_to_scr) do
        for _, rg in pairs(maps) do
          if item.char >= rg[1] and item.char <= rg[2] then
            Babel.chr_to_loc[item.char] = lc
            toloc = lc
            break
          end
        end
      end
    end
    if toloc and toloc > -1 then
      if Babel.locale_props[toloc].lg then
        item.lang = Babel.locale_props[toloc].lg
        node.set_attribute(item, LOCALE, toloc)
      end
      if Babel.locale_props[toloc]['/..item.font'] then
        item.font = Babel.locale_props[toloc]['/..item.font']
      end
      toloc_save = toloc
    end
  else if not inmath and item.id == 7 then
    item.replace = item.replace and Babel.locale_map(item.replace)
item.pre = item.pre and Babel.locale_map(item.pre)
item.post = item.post and Babel.locale_map(item.post)
elseif item.id == node.id'math' then
  inmath = (item.subtype == 0)
end
end
return head
end
}
The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.
\newcommand\babelcharproperty[1]{%\count@=#1\relax\ifvmode\expandafter\bbl@chprop\else\bbl@error{\string\babelcharproperty\space can be used only in vertical mode (preamble or between paragraphs)}%{See the manual for further info}%{See the manual for further info}}%
\newcommand\bbl@chprop[3][\the\count@]{%\@tempcnta=#1\relax\bbl@ifunset{bbl@chprop@#2}{\bbl@error{No property named \string'#2'. Allowed values are direction (bc), mirror (bmg), and linebreak (lb)}%{See the manual for further info}}%{}%\loop\bbl@cs{chprop@#2}{#3}\ifnum\count@<\@tempcnta\advance\count@\@ne\repeat}
\def\bbl@chprop@direction#1{%\directlua{Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}Babel.characters[\the\count@]['d'] = '#1'}}\let\bbl@chprop@bc=\bbl@chprop@direction\def\bbl@chprop@mirror#1{%\directlua{Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}Babel.characters[\the\count@]['m'] = '\number#1'}}\let\bbl@chprop@bmg=\bbl@chprop@mirror\def\bbl@chprop@linebreak#1{%\directlua{Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}Babel.cjk_characters[\the\count@]['c'] = '#1'}}\let\bbl@chprop@lb=\bbl@chprop@linebreak\def\bbl@chprop@locale#1{%\directlua{Babel.chr_to_loc = Babel.chr_to_loc or {}Babel.chr_to_loc[\the\count@] = \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space}}%

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still
some issues with speed (not very slow, but still slow).
After declaring the table containing the patterns with their replacements, we define some
auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list,
taking the node at base as a model (font, language, etc.); `fetch_word` fetches a series of
glyphs and discretionaries, which pattern is matched against (if there is a match, it is
called again before trying other patterns, and this is very likely the main bottleneck).
'post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the
automatic hyphenation points are known. As empty captures return a byte position (as
explained in the `luatex` manual), we must convert it to a utf8 position. With `first`, the last
byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the
resulting length. With `last` we must take into account the capture position points to the
next character. Here `word_head` points to the starting node of the text to be matched.

```lua
\begin{verbatim}
\catcode`\#=12 \catcode`%=12 \catcode`&=14
\directlua{
Babel.linebreaking.post_replacements = {}
Babel.linebreaking.pre_replacements = {}

function Babel.str_to_nodes(fn, matches, base)
  local n, head, last
  if fn == nil then return nil end
  for s in string.utfvalues(fn(matches)) do
    if base.id == 7 then
      base = base.replace
    end
    n = node.copy(base)
    n.char = s
    if not head then
      head = n
    else
      last.next = n
    end
    last = n
  end
  return head
end

function Babel.fetch_word(head, funct)
  local word_string = ''
  local word_nodes = {}
  local lang
  local item = head
  local inmath = false
  while item do
    if item.id == 29
      and not(item.char == 124) \&% ie, not | 
      and not(item.char == 61) \&% ie, not = 
      and not inmath 
      and (item.lang == lang or lang == nil) then
      lang = lang or item.lang
      word_string = word_string .. unicode.utf8.char(item.char)
      word_nodes[#word_nodes+1] = item
    end
end
\end{verbatim}
```
elseif item.id == 7 and item.subtype == 2 and not inmath then
    word_string = word_string .. '='
    word_nodes[#word_nodes+1] = item
elseif item.id == 7 and item.subtype == 3 and not inmath then
    word_string = word_string .. '|' 
    word_nodes[#word_nodes+1] = item
elseif item.id == 11 and item.subtype == 0 then
    inmath = true
elseif word_string == '' then
    &% pass
else
    return word_string, word_nodes, item, lang
end
item = item.next

function Babel.post_hyphenate_replace(head)
    local u = unicode.utf8
    local lbkr = Babel.linebreaking.post_replacements
    local word_head = head
    while true do
        local w, wn, nw, lang = Babel.fetch_word(word_head)
        if not lang then return head end
        if not lbkr[lang] then break end
        for k=1, #lbkr[lang] do
            local p = lbkr[lang][k].pattern
            local r = lbkr[lang][k].replace
            while true do
                local matches = { u.match(w, p) }
                if #matches < 2 then break end
                local first = table.remove(matches, 1)
                local last = table.remove(matches, #matches)
                &% Fix offsets, from bytes to unicode.
                first = u.len(w:sub(1, first-1)) + 1
                last = u.len(w:sub(1, last-1))
                local new &% used when inserting and removing nodes
                local changed = 0
                &% This loop traverses the replace list and takes the
                &% corresponding actions
                for q = first, last do
                    local crep = r[q-first+1]
                    local char_node = wn[q]
                    local char_base = char_node
if crep and crep.data then
    char_base = wn[crep.data+first-1]
end

if crep == {} then
    break
else if crep == nil then
    changed = changed + 1
    node.remove(head, char_node)
else if crep and (crep.pre or crep.no or crep.post) then
    changed = changed + 1
    d = node.new(7, 0) &% (disc, discretionary)
    d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
    d.post = Babel.str_to_nodes(crep.post, matches, char_base)
    d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
    d.attr = char_base.attr
    if crep.pre == nil then &% TeXbook p96
        d.penalty = crep.penalty or tex.hyphenpenalty
    else
        d.penalty = crep.penalty or tex.exhyphenpenalty
    end
    head, new = node.insert_before(head, char_node, d)
    node.remove(head, char_node)
    if q == 1 then
        word_head = new
    end
else if crep and crep.string then
    changed = changed + 1
    local str = crep.string(matches)
    if str == '' then
        if q == 1 then
            word_head = char_node.next
        end
        head, new = node.remove(head, char_node)
    elseif char_node.id == 29 and u.len(str) == 1 then
        char_node.char = string.utfvalue(str)
    else
        local n
        for s in string.utfvalues(str) do
            if char_node.id == 7 then
                log('Automatic hyphens cannot be replaced, just removed.')
            else
                n = node.copy(char_base)
            end
            n.char = s
            if q == 1 then
                head, new = node.insert_before(head, char_node, n)
                word_head = new
            else
                node.insert_before(head, char_node, n)
            end
        end
        node.remove(head, char_node)
    end
end &% for char in match
if changed > 20 then
texio.write('Too many changes. Ignoring the rest.')
elseif changed > 0 then
    w, wn, nw = Babel.fetch_word(word_head)
end
end &% for match
end &% for patterns
word_head = nw
end &% for words
return head
end &% for words

&%%
&% Preliminary code for \
babelprehyphenation
&% TODO. Copypaste pattern. Merge with fetch_word
function Babel.fetch_subtext(head, funct)
    local word_string = ''
    local word_nodes = {}
    local lang
    local item = head
    local inmath = false
    while item do
        if item.id == 29 then
            local locale = node.get_attribute(item, Babel.attr_locale)
            if not(item.char == 124) &% ie, not | = space
                and not inmath
                and (locale == lang or lang == nil) then
                lang = lang or locale
                word_string = word_string .. unicode.utf8.char(item.char)
                word_nodes[#word_nodes+1] = item
            end
        end
        if item == node.tail(head) then
            item = nil
            return word_string, word_nodes, item, lang
        end
        elseif item.id == 12 and item.subtype == 13 and not inmath then
            word_string = word_string .. '|' 
            word_nodes[#word_nodes+1] = item
        elseif item.id == 11 and item.subtype == 0 then
            inmath = true
        elseif word_string == '' then
            &% pass
        else
            return word_string, word_nodes, item, lang
        end
    end

178
item = item.next
end
end

&% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
function Babel.pre_hyphenate_replace(head)
local u = unicode.utf8
local lbkr = Babel.linebreaking.pre_replacements
local word_head = head

while true do
local w, wn, nw, lang = Babel.fetch_subtext(word_head)
if not lang then return head end
if not lbkr[lang] then
break
end

for k=1, #lbkr[lang] do
local p = lbkr[lang][k].pattern
local r = lbkr[lang][k].replace

while true do
local matches = { u.match(w, p) }
if #matches < 2 then break end
local first = table.remove(matches, 1)
local last = table.remove(matches, #matches)
&% Fix offsets, from bytes to unicode.
first = u.len(w:sub(1, first-1)) + 1
last = u.len(w:sub(1, last-1))
local new &% used when inserting and removing nodes
local changed = 0

&% This loop traverses the replace list and takes the
&% corresponding actions
for q = first, last do
local crep = r[q-first+1]
local char_node = wn[q]
local char_base = char_node

if crep and crep.data then
char_base = wn[crep.data+first-1]
end
if crep == {} then
break
elseif crep == nil then
changed = changed + 1
node.remove(head, char_node)
elseif crep and crep.string then
changed = changed + 1
local str = crep.string(matches)
if str == '' then
if q == 1 then
word_head = char_node.next
end
end
end
end
if crep == {} then
break
elseif crep == nil then
changed = changed + 1
node.remove(head, char_node)
elseif crep and crep.string then
changed = changed + 1
local str = crep.string(matches)
if str == '' then
if q == 1 then
word_head = char_node.next
end
end
end
end
head, new = node.remove(head, char_node)

elseif char_node.id == 29 and u.len(str) == 1 then
    char_node.char = string.utfvalue(str)
else
    local n
    for s in string.utfvalues(str) do
        if char_node.id == 7 then
            log('Automatic hyphens cannot be replaced, just removed.')
        else
            n = node.copy(char_base)
            n.char = s
        if q == 1 then
            head, new = node.insert_before(head, char_node, n)
            word_head = new
        else
            node.insert_before(head, char_node, n)
        end
        end
        n = node.remove(head, char_node)
    end &% string length
end &% if char and char.string
end &% for char in match
if changed > 20 then
    texio.write('Too many changes. Ignoring the rest.')
elseif changed > 0 then
    w, wn, nw = Babel.fetch_subtext(word_head)
end
end &% for match
end &% for patterns
word_head = nw
end &% for words
return head
end &%%% end of preliminary code for \babelprehyphenation

&% The following functions belong to the next macro
&% This table stores capture maps, numbered consecutively
Babel.capture_maps = {}

function Babel.capture_func(key, cap)
    local ret = "[[" .. cap.gsub('{{([0-9])}}, "]..m[%]\..[" .. "]"]
    ret = ret.gsub('{{([0-9])}}{{([\^])}}\..\..\..', Babel.capture_func_map)
    ret = ret.gsub('%\%\%.%\%.\%', ')
    return key .. [[=function(m) return ]] .. ret .. [[ end]]
end

function Babel.capt_map(from, mapno)
    return Babel.capture_maps[mapno][from] or from
end

&% Handle the {n|abc|ABC} syntax in captures
function Babel.capture_func_map(capno, from, to)
Now the \TeX{} high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the \{n\} syntax. For example, pre={1}{1}- becomes function(m) return m[1]..m[1]..'-' end, where m are the matches returned after applying the pattern. With a mapped capture the functions are similar to function(m) return Babel.capt_map(m[1],1) end, where the last argument identifies the mapping to be applied to m[1]. The way it is carried out is somewhat tricky, but the effect in not dissimilar to lua – save the code as string in a \TeX{} macro, and expand this macro at the appropriate place. As \directlua does not take into account the current catcode of @, we just avoid this character in macro names (which explains the internal group, too).

\catcode`#=6
\gdef\babelposthyphenation#1#2#3{%
\bbl@activateposthyphen
\begingroup
\def\babeltempa{\bbl@add@list\babeltempb}&%
\let\babeltempb\@empty
\bbl@foreach{#3}{&%
\bbl@ifsamestring{##1}{\texttt{remove}}&%
{\bbl@add@list\babeltempb{nil}}&%
{\directlua{
local rep = \texttt{[[##1]}
rep = rep:gsub( '\texttt{(no)\%s*=\%s*([^%s,]*)}', Babel.capture_func)
rep = rep:gsub( '\texttt{(pre)\%s*=\%s*([^%s,]*)}', Babel.capture_func)
rep = rep:gsub( '\texttt{(post)\%s*=\%s*([^%s,]*)}', Babel.capture_func)
rep = rep:gsub( '\texttt{(string)\%s*=\%s*([^%s,]*)}', Babel.capture_func)
tex.print(\texttt{[[\texttt{\string\babeltempa{\texttt{[\texttt{\texttt{%s}}} \texttt{.. replace = \texttt{[\texttt{\texttt{]]}}}}}]]}}
}}&%}
\directlua{
local lbkr = Babel.linebreaking.post_replacements
local u = unicode.utf8
\% Convert pattern:
local patt = string.gsub([==[#2]==], '\texttt{(%s}', '')
if not u.find(patt, '\texttt{'}(), nil, true) then
  patt = '\texttt{'}() .. patt .. '\texttt{'}()
end
patt = u.gsub(patt, '\texttt{(..)}',
  function (n)
    return '\' .. (tonumber(n) and (tonumber(n)+1) or n)
  end)
lbkr[[\the\csname l@#1\endcsname]] = lbkr[[\the\csname l@#1\endcsname]] or {}
table.insert(lbkr[[\the\csname l@#1\endcsname]]
  { pattern = patt, replace = { \babeltempb }})
}
13.7 Layout

Unlike \texttt{xetex}, \texttt{luatex} requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with \texttt{bidi=basic}, without having to patch almost any macro where text direction is relevant. \texttt{@\textbackslash hangfrom} is useful in many contexts and it is redefined always with the \texttt{layout} option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by \texttt{\textbackslash bodydir}), and when \texttt{\textbackslash parbox} and \texttt{\textbackslash hangindent} are involved. Fortunately, latest releases of \texttt{luatex} simplify a lot the solution with \texttt{\textbackslash shapemode}.

With the issue \#15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, tabular seems
to work (at least in simple cases) with array, tabularx, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.
Implicitly reverses sectioning labels in \texttt{bidi=r}, because the full stop is not in contact with 1 numbers any more. I think there must be a better way. Assumes \texttt{bidi= basic}, but there are some additional readjustments for \texttt{bidi=default}.

Some \LaTeX{} macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.
13.8 Auto bidi with basic and basic-r

The file babel-data-bidi.lua currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```lua
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

For the meaning of these codes, see the Unicode standard.

Now the basic-r bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs bidi.c (which also attempts to implement the bidi algorithm with a single loop):

```
Arrrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!
```

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, what they do and why, and not only how), but I think (or I hope) I've managed to understand them.

In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually two R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.
local characters = Babel.characters
local ranges = Babel.ranges

local DIR = node.id("dir")

local function dir_mark(head, from, to, outer)
  dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
  d = node.new(DIR)
  d.dir = '+' .. dir
  node.insert_before(head, from, d)
  d = node.new(DIR)
  d.dir = '-' .. dir
  node.insert_after(head, to, d)
end

function Babel.bidi(head, ispar)
  local first_n, last_n -- first and last char with nums
  local last_es -- an auxiliary 'last' used with nums
  local first_d, last_d -- first and last char in L/R block
  local dir, dir_real

  Next also depends on script/lang (<al>/<r>). To be set by babel. tex.pardir is dangerous,
  could be (re)set but it should be changed only in vmode. There are two strong's – strong =
  l/al/r and strong_lr = l/r (there must be a better way):

  local strong = ('TRT' == tex.pardir) and 'r' or 'l'
  local strong_lr = (strong == 'l') and 'l' or 'r'
  local outer = strong

  local new_dir = false
  local first_dir = false
  local inmath = false

  local last_lr

  local type_n = ''

  for item in node.traverse(head) do
    -- three cases: glyph, dir, otherwise
    if item.id == node.id'glyph'
      local itemchar
      if item.id == 7 and item.subtype == 2 then
        itemchar = item.replace.char
      else
        itemchar = item.char
      end
      local chardata = characters[itemchar]
      dir = chardata and chardata.d or nil
      if not dir then
        for nn, et in ipairs(ranges) do
          if itemchar < et[1] then
            break
          elseif itemchar <= et[2] then
            dir = et[3]
            break
          end
        end
      end
    end
    if item.id == node.id'glyph'
      itemchar = item.replace.char
    end
  end
end
Next is based on the assumption babel sets the language AND switches the script with its dir. We treat a language block as a separate Unicode sequence. The following piece of code is executed at the first glyph after a ‘dir’ node. We don’t know the current language until then. This is not exactly true, as the math mode may insert explicit dirs in the node list, so, for the moment there is a hack by brute force (just above).

If new_dir then
attr_dir = 0
for at in node.traverse(item.attr) do
  if at.number == luatexbase.registernumber'bbl@attr@dir' then
    attr_dir = at.value % 3
  end
end
if attr_dir == 1 then
  strong = 'r'
elseif attr_dir == 2 then
  strong = 'al'
else
  strong = 'l'
end
strong_lr = (strong == 'l') and 'l' or 'r'
outer = strong_lr
new_dir = false
end
if dir == 'nsm' then dir = strong end -- W1

Numbers. The dual <al>/<r> system for R is somewhat cumbersome.

By W2, there are no <en> <et> <es> if strong == <al>, only <an>. Therefore, there are not <et en> nor <en et>, W5 can be ignored, and W6 applied:

Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

elseif item.id == node.id'dir' and not inmath then
  new_dir = true
  dir = nil
elseif item.id == node.id'math' then
  inmath = (item.subtype == 0)
else
  dir = nil -- Not a char
end

Numbers in R mode. A sequence of <en>, <et>, <an>, <es> and <cs> is typeset (with some rules) in L mode. We store the starting and ending points, and only when anything different is found (including nil, ie, a non-char), the textdir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some
Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```python
if dir == 'en' or dir == 'an' or dir == 'et' then
  if dir ~= 'et' then
    type_n = dir
  end
  first_n = first_n or item
  last_n = last_es or item
  last_es = nil
elseif dir == 'es' and last_n then -- W3+W6
  last_es = item
elseif dir == 'cs' then -- it's right - do nothing
elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
  if strong_lr == 'r' and type_n ~= '' then
    dir_mark(head, first_n, last_n, 'r')
  elseif strong_lr == 'l' and first_d and type_n == 'an' then
    dir_mark(head, first_d, last_d, outer)
    first_d, last_d = nil, nil
  elseif strong_lr == 'l' and type_n ~= '' then
    last_d = last_n
  end
  type_n = ''
  first_n, last_n = nil, nil
end
```

R text in L, or L text in R. Order of dir mark's are relevant: d goes outside n, and therefore it's emitted after. See dir mark to understand why (but is the nesting actually necessary or is a flat dir structure enough?). Only L, R (and AL) chars are taken into account – everything else, including spaces, whatsits, etc., are ignored:

```python
if dir == 'l' or dir == 'r' then
  if dir ~= outer then
    first_d = first_d or item
    last_d = item
  elseif first_d and dir ~= strong_lr then
    dir_mark(head, first_d, last_d, outer)
    first_d, last_d = nil, nil
  end
end
```

Mirroring. Each chunk of text in a certain language is considered a "closed" sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, respictly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```python
if dir and not last_lr and dir ~= 'l' and outer == 'r' then
  item.char = characters[item.char] and
  characters[item.char].m or item.char
elseif (dir or new_dir) and last_lr ~= item then
  local mir = outer .. strong_lr .. (dir or outer)
  if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
    for ch in node.traverse(node.next(last_lr)) do
      if ch == item then break end
      if ch.id == node.id'glyph' and characters[ch.char] then
        ch.char = characters[ch.char].m or ch.char
      end
    end
  end
end
```
Save some values for the next iteration. If the current node is ‘dir’, open a new sequence. Since dir could be changed, strong is set with its real value (dir_real).

if dir == 'l' or dir == 'r' then
    last_lr = item
    strong = dir_real -- Don't search back - best save now
    strong_lr = (strong == 'l') and 'l' or 'r'
elseif new_dir then
    last_lr = nil
end

Mirror the last chars if they are no directed. And make sure any open block is closed, too.

if last_lr and outer == 'r' then
    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
        if characters[ch.char] then
            ch.char = characters[ch.char].m or ch.char
        end
    end
end

if first_n then
    dir_mark(head, first_n, last_n, outer)
end
if first_d then
    dir_mark(head, first_d, last_d, outer)
end

In boxes, the dir node could be added before the original head, so the actual head is the previous node.

return node.prev(head) or head
end

And here the Lua code for bidi=basic:

Babel = Babel or {}
-- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
Babel.fontmap = Babel.fontmap or {}
Babel.fontmap[0] = {} -- l
Babel.fontmap[1] = {} -- r
Babel.bidi_enabled = true
Babel.mirroring_enabled = true
require('babel-data-bidi.lua')
local characters = Babel.characters
local ranges = Babel.ranges
local DIR = node.id('dir')
local GLYPH = node.id('glyph')
local function insert_implicit(head, state, outer)
dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
local d = node.new(DIR)
d.dir = '+' .. dir
node.insert_before(head, state.sim, d)
local d = node.new(DIR)
d.dir = '-' .. dir
node.insert_after(head, state.eim, d)
end
new_state.sim, new_state.eim = nil, nil
return head, new_state
end

local function insert_numeric(head, state)
local new_state = state
if state.san and state.ean and state.san ~= state.ean then
local d = node.new(DIR)
d.dir = '+TLT'
_, new = node.insert_before(head, state.san, d)
if state.san == state.sim then state.sim = new end
local d = node.new(DIR)
d.dir = '-TLT'
_, new = node.insert_after(head, state.ean, d)
if state.ean == state.eim then state.eim = new end
end
new_state.san, new_state.ean = nil, nil
return head, new_state
end
end

-- TODO - \hbox with an explicit dir can lead to wrong results
-- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
-- was s made to improve the situation, but the problem is the 3-dir
-- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
-- well.

function Babel.bidi(head, ispar, hdir)
local d -- d is used mainly for computations in a loop
local prev_d = ''
local new_d = false
local nodes = {}
local outer_first = nil
local inmath = false
local glue_d = nil
local glue_i = nil
local has_en = false
local first_et = nil
local ATDIR = luatexbase.registernumber'bbl@attr@dir'
local save_outer
local temp = node.get_attribute(head, ATDIR)
if temp then
temp = temp % 3
save_outer = (temp == 0 and 'l') or
(temp == 1 and 'r') or
(temp == 2 and 'al')
elseif ispar then  -- Or error? Shouldn't happen
  save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
else  -- Or error? Shouldn't happen
  save_outer = ('TRT' == hdir) and 'r' or 'l'
end

-- when the callback is called, we are just _after_ the box,
-- and the textdir is that of the surrounding text
-- if not ispar and hdir ~= tex.textdir then
-- save_outer = ('TRT' == hdir) and 'r' or 'l'
-- end
local outer = save_outer
local last = outer
-- 'al' is only taken into account in the first, current loop
if save_outer == 'al' then save_outer = 'r' end

local fontmap = Babel.fontmap
for item in node.traverse(head) do
  -- In what follows, #node is the last (previous) node, because the
  -- current one is not added until we start processing the neutrals.
  -- three cases: glyph, dir, otherwise
  if item.id == GLYPH
    or (item.id == 7 and item.subtype == 2) then
    local d_font = nil
    local item_r
    if item.id == 7 and item.subtype == 2 then
      item_r = item.replace  -- automatic discs have just 1 glyph
    else
      item_r = item
    end
    local chardata = characters[item_r.char]
    d = chardata and chardata.d or nil
    if not d or d == 'nsm' then
      for nn, et in ipairs(ranges) do
        if item_r.char < et[1] then
          break
        elseif item_r.char <= et[2] then
          if not d then d = et[3] elseif d == 'nsm' then d_font = et[3]
        end
        break
      end
    end
    d = d or 'l'
    d_font = d_font or d
    d_font = (d_font == 'l' and 0) or
    (d_font == 'nsm' and 0) or
    (d_font == 'r' and 1) or
    (d_font == 'al' and 2) or
    (d_font == 'an' and 2) or nil
    if d_font and fontmap and fontmap[d_font][item_r.font] then
      item_r.font = fontmap[d_font][item_r.font]
    end
  end
  end
  d = d or 'l'
  d_font = d_font or d
  d_font = (d_font == 'l' and 0) or
  (d_font == 'nsm' and 0) or
  (d_font == 'r' and 1) or
  (d_font == 'al' and 2) or
  (d_font == 'an' and 2) or nil
  if d_font and fontmap and fontmap[d_font][item_r.font] then
    item_r.font = fontmap[d_font][item_r.font]
  end
end
if new_d then
  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
if inmath then
  attr_d = 0
else
  attr_d = node.get_attribute(item, ATDIR)
  attr_d = attr_d % 3
end
if attr_d == 1 then
  outer_first = 'r'
  last = 'r'
elseif attr_d == 2 then
  outer_first = 'r'
  last = 'al'
else
  outer_first = 'l'
  last = 'l'
end
outer = last
has_en = false
first_et = nil
new_d = false
end
if glue_d then
  if (d == 'l' and 'l' or 'r') ~= glue_d then
    table.insert(nodes, {glue_i, 'on', nil})
  end
  glue_d = nil
  glue_i = nil
end
elseif item.id == DIR then
  d = nil
  new_d = true
elseif item.id == node.id'glue' and item.subtype == 13 then
  glue_d = d
  glue_i = item
  d = nil
elseif item.id == node.id'math' then
  inmath = (item.subtype == 0)
else
  d = nil
end
-- AL <= EN/ET/ES -- W2 + W3 + W6
if last == 'al' and d == 'en' then
  d = 'an' -- W3
elseif last == 'al' and (d == 'et' or d == 'es') then
  d = 'on' -- W6
end
-- EN + CS/ES + EN -- W4
if d == 'en' and #nodes >= 2 then
  if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
and nodes[#nodes-1][2] == 'en' then
    nodes[#nodes][2] = 'en'
end
end
end

-- AN + CS + AN -- W4 too, because uax9 mixes both cases
if d == 'an' and #nodes >= 2 then
    if (nodes[#nodes][2] == 'cs')
        and nodes[#nodes-1][2] == 'an' then
            nodes[#nodes][2] = 'an'
    end
end

-- ET/EN -- W5 + W7->l / W6->on
if d == 'et' then
    first_et = first_et or (#nodes + 1)
elseif d == 'en' then
    has_en = true
    first_et = first_et or (#nodes + 1)
elseif first_et then
    -- d may be nil here!
    if has_en then
        if last == 'l' then
            temp = 'l' -- W7
        else
            temp = 'en' -- W5
        end
    else
        temp = 'on' -- W6
    end
    for e = first_et, #nodes do
        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
    end
    first_et = nil
    has_en = false
end

if d then
    if d == 'al' then
        d = 'r'
        last = 'al'
    elseif d == 'l' or d == 'r' then
        last = d
    end
    prev_d = d
    table.insert(nodes, {item, d, outer_first})
end

outer_first = nil

-- TODO -- repeated here in case EN/ET is the last node. Find a
-- better way of doing things:
if first_et then
    -- d may be nil here!
    if has_en then
        if last == 'l' then
            temp = 'l' -- W7
        else
            temp = 'en' -- W5
        end
end
else
  temp = 'on' -- W6
end
for e = first_et, #nodes do
  if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
end

-- dummy node, to close things
table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})

---------------- NEUTRAL ----------------

outer = save_outer
last = outer
local first_on = nil
for q = 1, #nodes do
  local item
  local outer_first = nodes[q][3]
  outer = outer_first or outer
  last = outer_first or last
  local d = nodes[q][2]
  if d == 'an' or d == 'en' then d = 'r' end
  if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
  if d == 'on' then
    first_on = first_on or q
  elseif first_on then
    if last == d then
      temp = d
    else
      temp = outer
    end
    for r = first_on, q - 1 do
      nodes[r][2] = temp
      item = nodes[r][1] -- MIRRORING
      if Babel.mirroring_enabled and item.id == GLYPH
        and temp == 'r' and characters[item.char] then
        local font_mode = font.fonts[item.font].properties.mode
        if font_mode ~= 'harf' and font_mode ~= 'plug' then
          item.char = characters[item.char].m or item.char
        end
      end
    end
  end
  if d == 'r' or d == 'l' then last = d end
end

---------------- IMPLICIT, REORDER ----------------

outer = save_outer
last = outer
local state = {}
state.has_r = false

for q = 1, #nodes do
    local item = nodes[q][1]
    outer = nodes[q][3] or outer
    local d = nodes[q][2]
    if d == 'nsm' then d = last end -- W1
    if d == 'en' then d = 'an' end
    local isdir = (d == 'r' or d == 'l')
    if outer == 'l' and d == 'an' then
        state.san = state.san or item
        state.ean = item
        if state.sim and state.has_r then
            head, state = insert_numeric(head, state)
        end
    elseif isdir then
        last = d -- Don't search back - best save now
    elseif d == 'l' and state.san then
        state.san = state.san or item
        state.ean = item
    else
        if d == 'an' or d == 'l' then
            if nodes[q][3] then -- nil except after an explicit dir
                state.sim = item -- so we move sim 'inside' the group
            else
                state.sim = state.sim or item
            end
            state.eim = item
        elseif d == 'r' and state.sim then
            head, state = insert_implicit(head, state, outer)
        elseif d == 'r' then
            state.sim, state.eim = nil, nil
        end
    end
end

if isdir then
    last = d -- Don't search back - best save now
else if d == 'on' and state.san then
    state.san = state.san or item
    state.ean = item
end

return node.prev(head) or head


14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},
```

For the meaning of these codes, see the Unicode standard.

15 The ‘nil’ language

This ‘language’ does nothing, except setting the hyphenation patterns to nohyphenation. For this language currently no special definitions are needed or available. The macro \LdfInit takes care of preventing that this file is loaded more than once, checking the category code of the @ sign, etc.

```
\ProvidesLanguage{nil}{\langle\langle date\rangle\rangle}{\langle\langle version\rangle\rangle} Nil language
\LdfInit{nil}{\date\nil}
```

When this file is read as an option, i.e. by the \usepackage command, nil could be an ‘unknown’ language in which case we have to make it known.

```
\ifx\l@nil\@undefined
   \newlanguage\l@nil
   \@namedef{bbl@hyphendata@\the\l@nil}{{}\{}% Remove warning
   \let\bbl@elt\relax
   \edef\bbl@languages{% Add it to the list of languages
      \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
\fi
```

This macro is used to store the values of the hyphenation parameters \lefthyphenmin and \righthyphenmin.

```
\providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
```

```
\let\captionsnil\@empty
\let\datenil\@empty
```

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

```
\Ldf@finish(nil)
```
16  Support for Plain \TeX{} (plain.def)

16.1  Not renaming \hyphen\.tex

As Don Knuth has declared that the filename \hyphen\.tex may only be used to designate his version of the American English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based \TeX{}-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file localhyphen\.tex or whatever they like, but they mustn’t diddle with \hyphen\.tex (or plain\.tex except to preload additional fonts).

The files bplain\.tex and blplain\.tex can be used as replacement wrappers around plain\.tex and lplain\.tex to achieve the desired effect, based on the babel package. If you load each of them with \texttt{\input}, you will get a file called either bplain\.fmt or blplain\.fmt, which you can use as replacements for plain\.fmt and lplain\.fmt. As these files are going to be read as the first thing \texttt{\input} sees, we need to set some category codes just to be able to change the definition of \texttt{\input}.

\begin{verbatim}
\catcode`{=1 % left brace is begin-group character
\catcode`}=2 % right brace is end-group character
\catcode`#=6 % hash mark is macro parameter character
\end{verbatim}

If a file called \texttt{hyphen\.cfg} can be found, we make sure that it will be read instead of the file \texttt{hyphen\.tex}. We do this by first saving the original meaning of \texttt{\input} (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

\begin{verbatim}
\openin 0 hyphen\.cfg
\ifeof0
\else
 \let\a\input
Then \texttt{\input} is defined to forget about its argument and load \texttt{hyphen\.cfg} instead. Once that’s done the original meaning of \texttt{\input} can be restored and the definition of \texttt{\a} can be forgotten.
\def\input #1 {%
 \let\input\a
 \a hyphen\.cfg
 \let\a\undefined
} \fi
\end{verbatim}

\begin{verbatim}
(/bplain | blplain)
\end{verbatim}

Now that we have made sure that \texttt{hyphen\.cfg} will be loaded at the right moment it is time to load \texttt{plain\.tex}.

\begin{verbatim}
(bplain)\a plain\.tex
(bplain)\a lplain\.tex
\end{verbatim}

Finally we change the contents of \texttt{\fmtname} to indicate that this is not the plain format, but a format based on plain with the babel package preloaded.

\begin{verbatim}
(bplain)\def\fmtname{babel-plain}
(bplain)\def\fmtname{babel-lplain}
\end{verbatim}

When you are using a different format, based on plain\.tex you can make a copy of blplain\.tex, rename it and replace plain\.tex with the name of your format file.
16.2 Emulating some \LaTeX features

The following code duplicates or emulates parts of \LaTeX\ 2e that are needed for babel.

\begin{verbatim}
\% == Code for plain ==
\def\@empty{}
\def\loadlocalcfg#1{\
  \openin0#1.cfg
  \ifeof0
    \closein0
    \else
      \closein0
      \immediate\write16{*************************************}\
      \immediate\write16{* Local config file #1.cfg used}\
      \immediate\write16{*}
    \}
  \input #1.cfg\relax
  \fi
\@endofldf
\end{verbatim}

16.3 General tools

A number of \LaTeX\ macro's that are needed later on.

\begin{verbatim}
\long\def\@firstofone#1{#1}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\def\@nnil{\@nil}
\def\@gobbletwo#1#2{\relax}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\def\@star@or@long#1{\@ifstar {\let\l@ngrel@x\relax#1} {\let\l@ngrel@x\long#1}}
\let\l@ngrel@x\relax
\def\@car#1#2\@nil{#1}
\def\@cdr#1#2\@nil{#2}
\let\@typeset@protect\relax
\let\protected@edef\edef
\long\def\@gobble#1{\relax}
\edef\@backslashchar{\expandafter\@gobble\string\\}
\def\strip@prefix#1>{\relax}
\def\g@addto@macro#1#2{{\toks@\expandafter{#1#2}\
  \xdef#1{\the\toks@}}}
\def\@namedef#1\@nameuse#1{\csname #1\endcsname}
\def\@ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax
  \expandafter\@firstoftwo
  \else \expandafter\@secondoftwo \fi}
\def\@expandtwoargs#1#2#3{\edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
\def\zap@space#1 #2{#1\ifx#2\@empty\else\expandafter\zap@space\fi #2}
\end{verbatim}

\set\bbl@trace@gobble

\LaTeX{}\n\times has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.

\ifx\@preamblecmds\@undefined
\let\@preamblecmds\empty
\fi
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble

Mimick \LaTeX{}'s \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

\def\begindocument{\@begindocumenthook
\global\let\@begindocumenthook\@undefined
\def\do##1{\global\let##1\@undefined}\
\@preamblecmds
\global\let\do\noexpand}
\ifx\@begindocumenthook\@undefined
\def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

We also have to mimick \LaTeX{}'s \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\def\@endofldf{}
\@onlypreamble\@endofldf
\let\bbl@afterlang\@empty
\chardef\bbl@opt@hyphenmap\z@

\LaTeX{} needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer \ifx. The same trick is applied below.

\catcode\&=\z@
\ifx&if@filesw\@undefined
\expandafter\let\csname if@filesw\expandafter\endcsname\csname iffalse\endcsname
\fi
\catcode\&=4

\LaTeX{} needs to switch on writing to its auxiliary files when defining a control sequence.

\def\newcommand{\@star@or@long\new@command}
\def\new@command#1{\@testopt{\@newcommand#1}0}
\def\@newcommand#1[#2]#3{\@ifnextchar{\@xargdef#1[#2]#3}{}{\@argdef#1[#2]#3}}
\long\def\@argdef#1[#2]#3{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter\csname#1\endcsname{#3}}}
\long\def\@xargdef#1[#2][#3]#4{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter\csname#1\endcsname{#3}}}
\long\def\@argdef#1[#2][#3]#4{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter\csname#1\endcsname{#3}}}

\LaTeX{} needs to switch on writing to its auxiliary files when defining a control sequence.

\def\newcommand{\@star@or@long\new@command}
\def\new@command#1{\@testopt{\@newcommand#1}0}
\def\@newcommand#1[#2]#3{\@ifnextchar{\@xargdef#1[#2]#3}{}{\@argdef#1[#2]#3}}
\long\def\@argdef#1[#2]#3{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter\csname#1\endcsname{#3}}}
\long\def\@xargdef#1[#2][#3]#4{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter\csname#1\endcsname{#3}}}
\long\def\@argdef#1[#2][#3]#4{\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter\csname#1\endcsname{#3}}}

\LaTeX{} needs to switch on writing to its auxiliary files when defining a control sequence.
The following little macro \in@ is taken from \texttt{latex.ltx}; it checks whether its first argument is part of its second argument. It uses the boolean \texttt{\in@}; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of \texttt{\bbl@tempa}.

\begin{verbatim}
\def\bbl@tempa{\csname newif\endcsname&ifin@}
\end{verbatim}

\input{fix}\input{fixmath}\input{fix} has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain \LaTeX{} we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

\def\@ifpackagewith#1#2#3#4{#3}

The \LaTeX{} macro \@ifl@aded checks whether a file was loaded. This functionality is not needed for plain \TeX{} but we need the macro to be defined as a no-op.

\def\@ifl@aded#1#2#3#4{}

For the following code we need to make sure that the commands \newcommand and \providecommand exist with some sensible definition. They are not fully equivalent to their \LaTeX{} versions; just enough to make things work in plain \TeX{} environments.

\ifx\@tempcnta\@undefined \csname newcount\endcsname\@tempcnta\relax \fi
\ifx\@tempcntb\@undefined \csname newcount\endcsname\@tempcntb\relax \fi

To prevent wasting two counters in \LaTeX{} 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

\ifx\bye\@undefined \else\fi
\ifx\@ifnextchar\@undefined \else\fi
\def\@ifnextchar#1#2#3{\let\reserved@d=#1\def\reserved@a{#2}\def\reserved@b{#3}\futurelet\@let@token\@ifnch}
\def\@ifnch{\ifx\@let@token\@sptoken \let\reserved@c=\@xifnch \else \ifx\@let@token\reserved@d \let\reserved@c=\reserved@a \else \let\reserved@c=\reserved@b \fi \fi \reserved@c}
\def\:{\let\@sptoken=} \expandafter\def\::{\futurelet\@let@token\@ifnch}
\def\@testopt#1#2{\@ifnextchar\[{#1}{#1[#2]}}
\def\@protected@testopt#1{\ifx\protect\@typeset@protect \expandafter\@testopt \fi}
\def\expandafter\@testopt\futurelet\@let@token\@ifnch}
\def\expandafter\@testopt\fi}\expandafter\def\::{\futurelet\@let@token\@ifnch}
16.4 Encoding related macros

Code from ltoutenc.dtx, adapted for use in the plain \TeX\ environment.

\def\DeclareTextCommand{% 
  \@dec@text@cmd\providecommand
}
\def\ProvideTextCommand{% 
  \@dec@text@cmd\providecommand
}
\def\DeclareTextSymbol#1#2#3{% 
  \@dec@text@cmd\chardef#1{#2}#3\relax
}
\def\@dec@text@cmd#1#2#3{% 
  \expandafter\def\expandafter#2\expandafter{\csname#3-cmd\expandafter\endcsname#2\csname#3\string#2\endcsname\expandafter\@ix@protect#1}%
  \let\@ifdefinable\@rc@ifdefinable
  \expandafter\@ifdefinable\csname#1\expandafter\endcsname\expandafter\@ifdefinable\csname#3\string#2\endcsname
  \else\noexpand#1\expandafter\@gobble\fi
}
\def\@current@cmd#1{% 
  \ifx\protect\@typeset@protect\else
    \noexpand#1\expandafter\@gobble\fi
}
\def\@changed@cmd#1#2{% 
  \ifx\protect\@typeset@protect
    \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
      \expandafter\ifx\csname ?\string#1\endcsname\relax
        \@changed@x@err{#1}\
      \fi
    \global\expandafter\let\csname\cf@encoding\string#1\endcsname\expandafter\@ifdefinable\csname#3\string#2\endcsname
  \fi
  \csname\cf@encoding\string#1\endcsname
  \else
    \noexpand#1
  \fi
}
\def\@changed@x@err#1{\
  \errhelp{Your command will be ignored, type <return> to proceed}\
  \errmessage{Command \protect#1 undefined in encoding \cf@encoding}\
}
\def\DeclareTextCommandDefault#1{% 
  \DeclareTextCommand#1%
Currently we only use the \LaTeX\_2\& method for accents for those that are known to be made active in some language definition file.

The following control sequences are used in babel.def but are not defined for plain \TeX. For a couple of languages we need the \LaTeX-control sequence \textsize to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \LaTeX\ has, we just let it to \sevenrm.

A proxy file:
17 Acknowledgements

I would like to thank all who volunteered as $\beta$-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References