Babel

Localization and internationalization

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

### 1 User guide

1. The user interface .......................... 4
   1.1 Monolingual documents .................... 4
   1.2 Multilingual documents ................... 6
   1.3 Mostly monolingual documents .......... 8
   1.4 Modifiers ................................ 8
   1.5 Troubleshooting .......................... 8
   1.6 Plain .................................... 9
   1.7 Basic language selectors ................ 9
   1.8 Auxiliary language selectors .......... 10
   1.9 More on selection ....................... 11
   1.10 Shorthands ............................... 12
   1.11 Package options ......................... 16
   1.12 The base option .......................... 18
   1.13 ini files ................................ 18
   1.14 Selecting fonts ........................... 26
   1.15 Modifying a language ..................... 28
   1.16 Creating a language ...................... 29
   1.17 Digits and counters ..................... 33
   1.18 Dates .................................... 35
   1.19 Accessing language info ................ 35
   1.20 Hyphenation and line breaking .......... 36
   1.21 Transforms ................................ 38
   1.22 Selection based on BCP 47 tags .......... 40
   1.23 Selecting scripts ........................ 42
   1.24 Selecting directions ..................... 42
   1.25 Language attributes ...................... 46
   1.26 Hooks .................................... 47
   1.27 Languages supported by babel with ldf files 48
   1.28 Unicode character properties in luatex 49
   1.29 Tweaking some features .................. 49
   1.30 Tips, workarounds, known issues and notes 50
   1.31 Current and future work .................. 51
   1.32 Tentative and experimental code ........ 51

2 Loading languages with language.dat ..... 52
   2.1 Format .................................. 52

3 The interface between the core of babel and the language definition files 53
   3.1 Guidelines for contributed languages .. 54
   3.2 Basic macros ................................ 54
   3.3 Skeleton .................................. 56
   3.4 Support for active characters .......... 57
   3.5 Support for saving macro definitions .. 57
   3.6 Support for extending macros .......... 57
   3.7 Macros common to a number of languages 58
   3.8 Encoding-dependent strings ............... 58

4 Changes .................................... 62
   4.1 Changes in babel version 3.9 ............. 62
II Source code 62
5 Identification and loading of required files 62
6 locale directory 63
7 Tools 63
  7.1 Multiple languages 68
  7.2 The Package File (\LaTeX, babel.sty) 68
  7.3 base 70
  7.4 key=value options and other general options 70
  7.5 Conditional loading of shorthands 72
  7.6 Interlude for Plain 73
8 Multiple languages 74
  8.1 Selecting the language 76
  8.2 Errors 85
  8.3 Hooks 87
  8.4 Setting up language files 89
  8.5 Shorthands 91
  8.6 Language attributes 100
  8.7 Support for saving macro definitions 102
  8.8 Short tags 104
  8.9 Hyphens 104
  8.10 Multiencoding strings 106
  8.11 Macros common to a number of languages 112
  8.12 Making glyphs available 112
    8.12.1 Quotation marks 112
    8.12.2 Letters 114
    8.12.3 Shorthands for quotation marks 115
    8.12.4 Umlauts and tremas 116
  8.13 Layout 117
  8.14 Load engine specific macros 118
  8.15 Creating and modifying languages 118
9 Adjusting the Babel bahavior 139
  9.1 Cross referencing macros 141
  9.2 Marks 144
  9.3 Preventing clashes with other packages 145
    9.3.1 ifthen 145
    9.3.2 varioref 146
    9.3.3 hhline 146
  9.4 Encoding and fonts 147
  9.5 Basic bidi support 148
  9.6 Local Language Configuration 152
  9.7 Language options 152
10 The kernel of Babel (babel.def, common) 156
11 Loading hyphenation patterns 156
12 Font handling with fontspec 161
13 Hooks for XeTeX and LuaTeX 165
  13.1 XeTeX ......................................................... 165
  13.2 Layout .......................................................... 167
  13.3 LuaTeX ............................................................ 169
  13.4 Southeast Asian scripts ........................................ 174
  13.5 CJK line breaking ............................................... 176
  13.6 Arabic justification ............................................ 178
  13.7 Common stuff .................................................. 182
  13.8 Automatic fonts and ids switching .......................... 182
  13.9 Bidi .............................................................. 187
  13.10 Layout .......................................................... 189
  13.11 Lua: transforms ................................................. 193
  13.12 Lua: Auto bidi with basic and basic-r .................... 201
14 Data for CJK ...................................................... 212
15 The ‘nil’ language .................................................. 212
16 Support for Plain TeX (plain.def) 213
  16.1 Not renaming hyphen.tex ...................................... 213
  16.2 Emulating some TeX features ................................. 214
  16.3 General tools .................................................. 214
  16.4 Encoding related macros ...................................... 218
17 Acknowledgements .................................................. 221

Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete ............... 5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the
format .............................................................. 6
You are loading directly a language style ................................ 8
Unknown language ‘LANG’ ............................................ 9
Argument of \language@active@arg”has an extra } ..................... 13
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

What is this document about? This user guide focuses on internationalization and localization with \TeX{} and pdftex, xetex and luatex with the babel package. There are also some notes on its use with e-Plain and pdf-Plain \TeX{}. Part II describes the code, and usually it can be ignored.

What if I’m interested only in the latest changes? Changes and new features with relation to version 3.8 are highlighted with \texttt{New X.XX}, and there are some notes for the latest versions in the \texttt{babel} site. The most recent features can be still unstable.

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current \TeX{} (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 fontspec. You may want to set the font attributes with fontspec, too.

\textbf{EXAMPLE} Here is a simple full example for “traditional” \TeX{} engines (see below for xetex and luatex). The packages fontenc and inputenc do not belong to babel, but they are included in the example because typically you will need them. It assumes UTF-8, the default encoding:
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
\begin{document}
Plus ça change, plus c'est la même chose!
\end{document}

Now consider something like:
\documentclass[french]{article}
\usepackage{babel}
\usepackage{varioref}

With this setting, the package \texttt{varioref} will also see the option \texttt{french} and will be able to use it.

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

\begin{verbatim}
\documentclass[russian]{article}
\usepackage{babel}
\babelfont{rm}{DejaVu Serif}
\begin{document}
Россия, находящаяся на пересечении множества культур, а также с учётом многонационального характера её населения, — отличается высокой степенью этнокультурного многообразия и способностью к межкультурному диалогу.
\end{document}
\end{verbatim}

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

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

Or the more explanatory:

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

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

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

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

The document will be typeset, but very likely the text will not be correctly hyphenated. Some languages 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, Mik\TeX, \TeX\Live, etc.) for further info about how to configure it.

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

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

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

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

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 (e.g., Spanish and French).

EXAMPLE In \LaTeX, the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[english,french]{babel}
\begin{document}
Plus ça change, plus c'est la même chose!
\selectlanguage{english}
And an English paragraph, with a short text in \foreignlanguage{french}{français}.
\end{document}

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

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

NOTE Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.22 for further details.
1.3 Mostly monolingual documents

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 with the default font in English and Spanish, and FreeSerif in Russian is:

```latex
\documentclass[english]{article}
\usepackage{babel}
\babelfont[russian]{rm}{FreeSerif}
\begin{document}

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

\end{document}
```

NOTE Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or tree-letter word is a valid name for a language (eg, yi). See section 1.22 for further details.

1.4 Modifiers

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):

\begin{verbatim}
\usepackage[latin.medieval, spanish.notilde.lcroman, danish]{babel}
\end{verbatim}

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:

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

2In old versions the error read “You have used an old interface to call babel”, not very helpful.
Another typical error when using babel is the following:\(^3\)

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

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

### 1.6 Plain

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

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

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

### 1.7 Basic language selectors

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

The main language is selected automatically when the document environment begins.

```
\selectlanguage{⟨language⟩}
```

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

```
\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.

\(^3\)In old versions the error read “You haven't loaded the language LANG yet”.

9
**WARNING** If used inside braces there might be some non-local changes, as this would be roughly equivalent to:

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

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

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

- `\selectlanguage` should not be used inside some boxed environments (like floats or `minipage`) to switch the language if you need the information written to the aux be correctly synchronized. This rarely happens, but if it were the case, you must use `otherlanguage` instead.
- In addition, this macro inserts a `\write` in vertical mode, which may break the vertical spacing in some cases (for example, between lists). New 3.64 The behavior can be adjusted with `\babeladjust{select.write=⟨mode⟩}`, where `⟨mode⟩` is shift (which shifts the skips down and adds a `\penalty`); keep (the default – with it the `\write` and the skips are kept in the order they are written), and `omit` (which may seem a too drastic solution, because nothing is written, but more often than not this command is applied to more or less short texts with no sectioning or similar commands and therefore no language synchronization is necessary).

\begin{otherlanguage}
\end{otherlanguage}

The environment `\begin{otherlanguage} ... \end{otherlanguage}` does basically the same as `\selectlanguage`, except that language change is (mostly) local to the environment. Actually, there might be some non-local changes, as this environment is roughly equivalent to:

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

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

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

### 1.9 More on selection

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

**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 \begin{\langle tag1 \rangle} is also allowed, but remember to set it locally inside a group.

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

**EXAMPLE** With

\babeltags{de = german}

you can write

\text{\textde{German text} text}

and

\text\begin{de} German text \end{de} text
NOTE Something like \betalgs{finnish = finnish} is legitimate – it defines \textfinnish and \finnish (and, of course, \begin{finnish}).

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

\betalgs{include=⟨commands⟩, exclude=⟨commands⟩, fontenc=⟨encoding⟩}{⟨language⟩}

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

\foreignlanguage{russian}{text} \foreignlanguage{polish}{⟨seename⟩ text}

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

\betalgs{polish}

By default only the basic captions and \today{} are redefined, but you can add further macros with the key include in the optional argument (without commas). Macros not to be modified are listed in exclude. You can also enforce a font encoding with the option fontenc.\footnote{With it, encoded strings may not work as expected.} A couple of examples:

\betalgs{include=\today}{spanish}
\betalgs{fontenc=T5}{vietnamese}

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

1.10 Shorthands

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

NOTE Keep in mind the following:

1. Activated chars used for two-char shorthands cannot be followed by a closing brace } and the spaces following are gobbled. With one-char shorthands (eg, :), they are preserved.
2. If on a certain level (system, language, user, language user) there is a one-char shorthand, two-char ones starting with that char and on the same level are ignored.

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

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

\!

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

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

\shorthandon
\shorthandoff

{⟨shorthand-list⟩}

\*{⟨shorthand-list⟩}

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands \shorthandoff and \shorthandon are provided. They each take a list of characters as their arguments. The command \shorthandoff sets the \catcode for each of the characters in its argument to other (12); the command \shorthandon sets the \catcode to active (13). Both commands only work on ‘known’ shorthand characters.

**New 3.9a** However, \shorthandoff does not behave as you would expect with characters like ~ or ^, because they usually are not “other”. For them \shorthandoff* is provided, so that with

\shorthandoff*{~ ^}

~ is still active, very likely with the meaning of a non-breaking space, and ^ is the superscript character. The catcodes used are those when the shorthands are defined, usually when language files are loaded.

If you do not need shorthands, or prefer an alternative approach of your own, you may want to switch them off with the package option shorthands=off, as described below.

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

\useshorthands

\*{⟨char⟩}

The command \useshorthands initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands.

**New 3.9a** User shorthands are not always alive, as they may be deactivated by languages (for example, if you use " for your user shorthands and switch from german to french, they stop working). Therefore, a starred version \useshorthands*{⟨char⟩} is provided, which makes sure shorthands are always activated.

Currently, if the package option shorthands is used, you must include any character to be activated with \useshorthands. This restriction will be lifted in a future release.

\defineshorthand

[⟨language⟩, ⟨language⟩, …]⟨⟨shorthand⟩⟩{⟨code⟩}

The command \defineshorthand takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to.

**New 3.9a** An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add
\languageshorthands\{lang\} to the corresponding \extras\{lang\}, as explained below. By default, user shorthands are (re)defined. User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

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

\begin{verbatim}
\useshorthands*{"}
\defineshorthand{"*}{\babelhyphen{soft}}
\defineshorthand{"-}{\babelhyphen{hard}}
\end{verbatim}

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

\begin{verbatim}
\defineshorthand[^polish,*portuguese]{-}{\babelhyphen{repeat}}
\end{verbatim}

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.

\begin{verbatim}
\languageshorthands \{language\}
\end{verbatim}

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

\begin{verbatim}
\addto\extrasenglish{\languageshorthands{ngerman}}
\end{verbatim}

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

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

\begin{verbatim}
\newcommand{\myipa}[1]{\languageshorthands{none}\tipaencoding#1}}
\end{verbatim}

\begin{verbatim}
\beshorthand \{shorthand\}
\end{verbatim}

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, \beshorthand{"u} or \beshorthand{:.} (You can conveniently define your own macros, or even your own user shorthands provided they do not overlap.)

\(^5\) Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.
EXAMPLE  Since by default shorthands are not activated until \begin{document}, you may use this macro when defining the \texttt{title} in the preamble:

\begin{verbatim}
\title{Documento científico \babelshorthand{"-} técnico}
\end{verbatim}

For your records, here is a list of shorthands, but you must double check them, as they may change:\footnote{6}{Thanks to Enrico Gregorio}

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

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

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

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.\footnote{7}{This declaration serves to nothing, but it is preserved for backward compatibility.}

\begin{verbatim}
\ifbabelshorthand \langle \texttt{character} \rangle \{ \langle \texttt{true} \rangle \} \{ \langle \texttt{false} \rangle \} \end{verbatim}

\begin{verbatim}
\aliashorthand \langle \texttt{original} \rangle \{ \langle \texttt{alias} \rangle \}
\end{verbatim}

The command \texttt{\aliashorthand} 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 \texttt{\aliashorthand{"} \{/}. 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, \texttt{\aliashorthands} 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 $\texttt{\activechar}\, or $\texttt{\normalchar}$~).

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.

#### KeepShorthandsActive

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

#### activeacute

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

#### activegrave

Same for `.

#### shorthands=

\langle char\rangle\langle char\rangle... | off

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

\begin{verbatim}
\usepackage{esperanto,french,shorthands=:;!?}{babel}
\end{verbatim}

If ’ is included, activeacute is set; if ` is included, activegrave is set. Active characters (like ~) should be preceded by \texttt{\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 \texttt{\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 \\texttt{\nocite}, \texttt{\bibcite} and \texttt{\bibitem} are redefined. With safe=ref only \\texttt{\newlabel}, \texttt{\ref} and \texttt{\pageref} are redefined (as well as a few macros from \texttt{varioref} and \texttt{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 bbl opts .cfg (the file is loaded even with no configs).

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 spoilt by an unexpected .cfg file. However, if the key config is set, this file is loaded.

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

nocase
\textbf{New 3.9l} Language settings for uppercase and lowercase mapping (as set by \texttt{\SetCase}) are ignored. Use only if there are incompatibilities with other packages.

silent
\textbf{New 3.9l} No warnings and no \texttt{infos} are written to the log file.\footnote{You can use alternatively the package silence.}

strings= \texttt{generic} | \texttt{unicode} | \texttt{encoded} | \langle label \rangle | \langle font encoding \rangle
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 \texttt{xetex} and \texttt{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 \texttt{\MakeUppercase} and the like (this feature misuses some internal \LaTeX tools, so use it only as a last resort).

hyphenmap= off | first | select | other | other*
\textbf{New 3.9g} Sets the behavior of case mapping for hyphenation, provided the language defines it.\footnote{Turned off in plain.} It can take the following values:

\texttt{off} deactivates this feature and no case mapping is applied;
\texttt{first} sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \texttt{\begin{document}}, but also the first \texttt{\selectlanguage} in the preamble), and it's the default if a single language option has been stated;\footnote{Duplicated options count as several ones.}
\texttt{select} sets it only at \texttt{\selectlanguage};
\texttt{other} also sets it at other language;
\texttt{other*} also sets it at other language* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \texttt{\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.\footnote{Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn't really useful, but who knows.}

\begin{itemize}
\item[bidi=} Default | basic | basic-r | bidi-l | bidi-r  
\begin{itemize}
\item[New 3.14] Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.24.
\end{itemize}
\item[layout=} New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.24.
\end{itemize}

\subsection{The base option}

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

\AfterBabelLanguage \{(option-name)\}{\{code\}}

This command is currently the only provided by base. Executes \{code\} when the file loaded by the corresponding package option is finished (at \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!)

\textbf{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:

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

\textbf{WARNING} Currently this option is not compatible with languages loaded on the fly.

\subsection{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 Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the \...name strings).
Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them by means of \babelprovide. In other words, \babelprovide is mainly meant for auxiliary tasks, and as alternative when the ldf, for some reason, does 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.

```
\documentclass{book}
\usepackage{babel}
\babelprovide[import, main]{georgian}
\babelfont{rm}[Renderer=Harfbuzz]{DejaVu Sans}
\begin{document}
\tableofcontents
\chapter{სამზარეულო და სუფრის ტრადიციები}
ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.
\end{document}
```

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

- provide=* is the option just explained, for the main language;
- provide+=* is the same for additional languages (the main language is still the ldf file);
- provide==* is the same for all languages, ie, main and additional.

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

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

Or also:

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

**NOTE** The ini files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follow (which could no longer be valid when you read this manual, if the packages involved han been updated). The Harfbuzz renderer has still some issues, so as a rule of thumb prefer the default renderer, and resort to Harfbuzz only if the former does not work for you. Fortunately, fonts can be loaded twice with different renderers; for example:
Arabic  Monolingual documents mostly work in luatex, but it must be fine tuned, particularly graphical elements like picture. In xetex babel resorts to the bidi package, which seems to work.

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

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

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

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

Southeast scripts  Thai works in both luatex and xetex, but line breaking differs (rules 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. Khmer clusters are rendered wrongly with the default renderer. The comment about Indic scripts and lualatex also applies here. Some quick patterns can help, with something similar to:

```
\babelpatterns[\lao]{1 Đ 1 M 1 O 1 Č 1 A} % Random
```

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

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

Latin, Greek, Cyrillic  Combining chars with the default luatex font renderer might be wrong; on then other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenations points are discarded (this bug 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>Script</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>Afrikaans</td>
<td>Afrikaans</td>
</tr>
<tr>
<td>agq</td>
<td>Aghem</td>
<td>Aghem</td>
</tr>
<tr>
<td>ak</td>
<td>Akan</td>
<td>Akan</td>
</tr>
<tr>
<td>am</td>
<td>Amharic</td>
<td>Amharic</td>
</tr>
<tr>
<td>ar</td>
<td>Arabic</td>
<td>Arabic</td>
</tr>
<tr>
<td>ar-DZ</td>
<td>Arabic</td>
<td>Arabic</td>
</tr>
<tr>
<td>ar-MA</td>
<td>Arabic</td>
<td>Arabic</td>
</tr>
<tr>
<td>ar-SY</td>
<td>Arabic</td>
<td>Arabic</td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
<td>Assamese</td>
</tr>
<tr>
<td>asa</td>
<td>Asu</td>
<td>Asu</td>
</tr>
<tr>
<td>ast</td>
<td>Asturian</td>
<td>Asturian</td>
</tr>
<tr>
<td>az</td>
<td>Azerbaijani</td>
<td>Azerbaijani</td>
</tr>
<tr>
<td>az-Cyr</td>
<td>Azerbaijani</td>
<td>Azerbaijani</td>
</tr>
<tr>
<td>az-Latn</td>
<td>Azerbaijani</td>
<td>Azerbaijani</td>
</tr>
<tr>
<td>bas</td>
<td>Basaa</td>
<td>Basaa</td>
</tr>
<tr>
<td>be</td>
<td>Belarusian</td>
<td>Belarusian</td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
<td>Sub-Code</td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>bem</td>
<td>Bemba</td>
<td></td>
</tr>
<tr>
<td>bez</td>
<td>Bena</td>
<td></td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarianul</td>
<td></td>
</tr>
<tr>
<td>bm</td>
<td>Bambara</td>
<td></td>
</tr>
<tr>
<td>bn</td>
<td>Banglaul</td>
<td></td>
</tr>
<tr>
<td>bo</td>
<td>Tibetanu</td>
<td></td>
</tr>
<tr>
<td>brx</td>
<td>Bodo</td>
<td></td>
</tr>
<tr>
<td>bs-Cyrl</td>
<td>Bosnian</td>
<td></td>
</tr>
<tr>
<td>bs-Latn</td>
<td>Bosnianul</td>
<td></td>
</tr>
<tr>
<td>bs</td>
<td>Bosnianul</td>
<td></td>
</tr>
<tr>
<td>ca</td>
<td>Catalanul</td>
<td></td>
</tr>
<tr>
<td>ce</td>
<td>Chechen</td>
<td></td>
</tr>
<tr>
<td>cgg</td>
<td>Chiga</td>
<td></td>
</tr>
<tr>
<td>chr</td>
<td>Cherokee</td>
<td></td>
</tr>
<tr>
<td>ckb</td>
<td>Central Kurdish</td>
<td></td>
</tr>
<tr>
<td>cop</td>
<td>Coptic</td>
<td></td>
</tr>
<tr>
<td>cs</td>
<td>Czechul</td>
<td></td>
</tr>
<tr>
<td>cu</td>
<td>Church Slavic</td>
<td></td>
</tr>
<tr>
<td>cu-Cyrs</td>
<td>Church Slavic</td>
<td></td>
</tr>
<tr>
<td>cu-Glag</td>
<td>Church Slavic</td>
<td></td>
</tr>
<tr>
<td>cy</td>
<td>Welshul</td>
<td></td>
</tr>
<tr>
<td>da</td>
<td>Danishul</td>
<td></td>
</tr>
<tr>
<td>dav</td>
<td>Taita</td>
<td></td>
</tr>
<tr>
<td>de-AT</td>
<td>Germanul</td>
<td></td>
</tr>
<tr>
<td>de-CH</td>
<td>Germanul</td>
<td></td>
</tr>
<tr>
<td>de</td>
<td>Germanul</td>
<td></td>
</tr>
<tr>
<td>dje</td>
<td>Zarma</td>
<td></td>
</tr>
<tr>
<td>dsb</td>
<td>Lower Sorbianul</td>
<td></td>
</tr>
<tr>
<td>dua</td>
<td>Duala</td>
<td></td>
</tr>
<tr>
<td>dyo</td>
<td>Jola-Fonyi</td>
<td></td>
</tr>
<tr>
<td>dz</td>
<td>Dzongkha</td>
<td></td>
</tr>
<tr>
<td>ebu</td>
<td>Embu</td>
<td></td>
</tr>
<tr>
<td>ee</td>
<td>Ewe</td>
<td></td>
</tr>
<tr>
<td>el</td>
<td>Greekul</td>
<td></td>
</tr>
<tr>
<td>el-polyton</td>
<td>Polytonic Greekul</td>
<td></td>
</tr>
<tr>
<td>en-AU</td>
<td>Englishul</td>
<td></td>
</tr>
<tr>
<td>en-CA</td>
<td>Englishul</td>
<td></td>
</tr>
<tr>
<td>en-GB</td>
<td>Englishul</td>
<td></td>
</tr>
<tr>
<td>en-NZ</td>
<td>Englishul</td>
<td></td>
</tr>
<tr>
<td>en-US</td>
<td>Englishul</td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>Englishul</td>
<td></td>
</tr>
<tr>
<td>eo</td>
<td>Esperantoul</td>
<td></td>
</tr>
<tr>
<td>es-MX</td>
<td>Spanishul</td>
<td></td>
</tr>
<tr>
<td>es</td>
<td>Spanishul</td>
<td></td>
</tr>
<tr>
<td>et</td>
<td>Estonianul</td>
<td></td>
</tr>
<tr>
<td>eu</td>
<td>Basqueul</td>
<td></td>
</tr>
<tr>
<td>ewo</td>
<td>Ewondo</td>
<td></td>
</tr>
<tr>
<td>fa</td>
<td>Persianul</td>
<td></td>
</tr>
<tr>
<td>ff</td>
<td>Fulah</td>
<td></td>
</tr>
<tr>
<td>fi</td>
<td>Finnishul</td>
<td></td>
</tr>
<tr>
<td>fil</td>
<td>Filipino</td>
<td></td>
</tr>
<tr>
<td>fo</td>
<td>Faroese</td>
<td></td>
</tr>
<tr>
<td>fr</td>
<td>Frenchul</td>
<td></td>
</tr>
<tr>
<td>fr-BE</td>
<td>Frenchul</td>
<td></td>
</tr>
<tr>
<td>fr-CA</td>
<td>Frenchul</td>
<td></td>
</tr>
<tr>
<td>fr-CH</td>
<td>Frenchul</td>
<td></td>
</tr>
<tr>
<td>fr-LU</td>
<td>Frenchul</td>
<td></td>
</tr>
<tr>
<td>fur</td>
<td>Friulianul</td>
<td></td>
</tr>
<tr>
<td>fy</td>
<td>Western Frisian</td>
<td></td>
</tr>
<tr>
<td>ga</td>
<td>Irishul</td>
<td></td>
</tr>
<tr>
<td>gd</td>
<td>Scottish Gaelicul</td>
<td></td>
</tr>
<tr>
<td>gl</td>
<td>Galicianul</td>
<td></td>
</tr>
<tr>
<td>grc</td>
<td>Ancient Greekul</td>
<td></td>
</tr>
<tr>
<td>gsw</td>
<td>Swiss German</td>
<td></td>
</tr>
<tr>
<td>gu</td>
<td>Gujarati</td>
<td></td>
</tr>
<tr>
<td>guz</td>
<td>Gusi</td>
<td></td>
</tr>
<tr>
<td>gv</td>
<td>Manx</td>
<td></td>
</tr>
<tr>
<td>ha-GH</td>
<td>Hausa</td>
<td></td>
</tr>
<tr>
<td>ha-NE</td>
<td>Hausa</td>
<td></td>
</tr>
<tr>
<td>ha</td>
<td>Hausa</td>
<td></td>
</tr>
<tr>
<td>haw</td>
<td>Hawaiian</td>
<td></td>
</tr>
<tr>
<td>he</td>
<td>Hebrewul</td>
<td></td>
</tr>
<tr>
<td>hi</td>
<td>Hindiul</td>
<td></td>
</tr>
<tr>
<td>hr</td>
<td>Croatianul</td>
<td></td>
</tr>
<tr>
<td>hsb</td>
<td>Upper Sorbianul</td>
<td></td>
</tr>
<tr>
<td>hu</td>
<td>Hungarianul</td>
<td></td>
</tr>
<tr>
<td>hy</td>
<td>Armenianu</td>
<td></td>
</tr>
<tr>
<td>ia</td>
<td>Interlinguaul</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>Indonesianul</td>
<td></td>
</tr>
<tr>
<td>ig</td>
<td>Igbo</td>
<td></td>
</tr>
<tr>
<td>ii</td>
<td>Sichuan Yi</td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>Icelandicul</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>Italianul</td>
<td></td>
</tr>
<tr>
<td>ja</td>
<td>Japanese</td>
<td></td>
</tr>
<tr>
<td>jgo</td>
<td>Ngomba</td>
<td></td>
</tr>
<tr>
<td>jmc</td>
<td>Machame</td>
<td></td>
</tr>
<tr>
<td>ka</td>
<td>Georgianul</td>
<td></td>
</tr>
<tr>
<td>kab</td>
<td>Kabyle</td>
<td></td>
</tr>
<tr>
<td>kam</td>
<td>Kamba</td>
<td></td>
</tr>
<tr>
<td>kde</td>
<td>Makonde</td>
<td></td>
</tr>
<tr>
<td>kea</td>
<td>Kabuverdianu</td>
<td></td>
</tr>
<tr>
<td>khq</td>
<td>Koyra Chini</td>
<td></td>
</tr>
<tr>
<td>ki</td>
<td>Kikuyu</td>
<td></td>
</tr>
<tr>
<td>kk</td>
<td>Kazakh</td>
<td></td>
</tr>
<tr>
<td>kkj</td>
<td>Kako</td>
<td></td>
</tr>
<tr>
<td>kl</td>
<td>Kalaallisut</td>
<td></td>
</tr>
<tr>
<td>kln</td>
<td>Kalenjin</td>
<td></td>
</tr>
<tr>
<td>km</td>
<td>Khmer</td>
<td></td>
</tr>
<tr>
<td>kn</td>
<td>Kannadaul</td>
<td></td>
</tr>
<tr>
<td>ko</td>
<td>Korean</td>
<td></td>
</tr>
<tr>
<td>kok</td>
<td>Konkani</td>
<td></td>
</tr>
<tr>
<td>ks</td>
<td>Kashmiri</td>
<td></td>
</tr>
<tr>
<td>ksb</td>
<td>Shambala</td>
<td></td>
</tr>
<tr>
<td>ksf</td>
<td>Bafia</td>
<td></td>
</tr>
<tr>
<td>ksh</td>
<td>Colognian</td>
<td></td>
</tr>
<tr>
<td>kw</td>
<td>Cornish</td>
<td></td>
</tr>
<tr>
<td>ky</td>
<td>Kyrgyz</td>
<td></td>
</tr>
<tr>
<td>lag</td>
<td>Langi</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
<td>Code</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>------</td>
</tr>
<tr>
<td>lb</td>
<td>Luxembourgish</td>
<td>rof</td>
</tr>
<tr>
<td>lg</td>
<td>Ganda</td>
<td>ru</td>
</tr>
<tr>
<td>lkt</td>
<td>Lakota</td>
<td>rw</td>
</tr>
<tr>
<td>ln</td>
<td>Lingala</td>
<td>rwk</td>
</tr>
<tr>
<td>lo</td>
<td>Lao\ul</td>
<td>sa-a</td>
</tr>
<tr>
<td>lrc</td>
<td>Northern Luri</td>
<td>sa-D</td>
</tr>
<tr>
<td>lt</td>
<td>Lithuanian\ul</td>
<td>sa-G</td>
</tr>
<tr>
<td>lu</td>
<td>Luba-Katanga</td>
<td>sa-K</td>
</tr>
<tr>
<td>luo</td>
<td>Luo</td>
<td>sa-M</td>
</tr>
<tr>
<td>luy</td>
<td>Luyia</td>
<td>sa-T</td>
</tr>
<tr>
<td>lv</td>
<td>Latvian\ul</td>
<td>sa</td>
</tr>
<tr>
<td>mas</td>
<td>Masai</td>
<td>sah</td>
</tr>
<tr>
<td>mer</td>
<td>Meru</td>
<td>saq</td>
</tr>
<tr>
<td>mfe</td>
<td>Morisyen</td>
<td>sbp</td>
</tr>
<tr>
<td>mg</td>
<td>Malagasy</td>
<td>se</td>
</tr>
<tr>
<td>mgh</td>
<td>Makhuwa-Meetto</td>
<td>seh</td>
</tr>
<tr>
<td>mgo</td>
<td>Meta’</td>
<td>ses</td>
</tr>
<tr>
<td>mk</td>
<td>Macedonian\ul</td>
<td>sg</td>
</tr>
<tr>
<td>ml</td>
<td>Malayalam\ul</td>
<td>shi-L</td>
</tr>
<tr>
<td>mn</td>
<td>Mongolian</td>
<td>shi-T</td>
</tr>
<tr>
<td>mr</td>
<td>Marathi\ul</td>
<td>shi</td>
</tr>
<tr>
<td>ms-BN</td>
<td>Malay\ul</td>
<td>si</td>
</tr>
<tr>
<td>ms-SG</td>
<td>Malay\ul</td>
<td>sk</td>
</tr>
<tr>
<td>ms</td>
<td>Malay\ul</td>
<td>sl</td>
</tr>
<tr>
<td>mt</td>
<td>Maltese</td>
<td>smn</td>
</tr>
<tr>
<td>mua</td>
<td>Mundang</td>
<td>sn</td>
</tr>
<tr>
<td>my</td>
<td>Burmese</td>
<td>so</td>
</tr>
<tr>
<td>mzn</td>
<td>Mazanderani</td>
<td>sq</td>
</tr>
<tr>
<td>naq</td>
<td>Nama</td>
<td>sr-C</td>
</tr>
<tr>
<td>nb</td>
<td>Norwegian Bokmål\ul</td>
<td>sr-Cyrl-BA</td>
</tr>
<tr>
<td>nd</td>
<td>North Ndebele</td>
<td>sr-Cyrl-ME</td>
</tr>
<tr>
<td>ne</td>
<td>Nepali</td>
<td>sr-Cyrl-XK</td>
</tr>
<tr>
<td>nl</td>
<td>Dutch\ul</td>
<td>sr-L</td>
</tr>
<tr>
<td>nmg</td>
<td>Kwasio</td>
<td>sr-Latn-BA</td>
</tr>
<tr>
<td>nn</td>
<td>Norwegian Nynorsk\ul</td>
<td>sr-Latn-ME</td>
</tr>
<tr>
<td>nnh</td>
<td>Ngjemboon</td>
<td>sr-Latn-XK</td>
</tr>
<tr>
<td>nus</td>
<td>Nuer</td>
<td>sr</td>
</tr>
<tr>
<td>nyn</td>
<td>Nyankole</td>
<td>sv</td>
</tr>
<tr>
<td>om</td>
<td>Oromo</td>
<td>sw</td>
</tr>
<tr>
<td>or</td>
<td>Odia</td>
<td>ta</td>
</tr>
<tr>
<td>os</td>
<td>Ossetic</td>
<td>te</td>
</tr>
<tr>
<td>pa-Arab</td>
<td>Punjabi</td>
<td>teo</td>
</tr>
<tr>
<td>pa-Guru</td>
<td>Punjabi</td>
<td>th</td>
</tr>
<tr>
<td>pa</td>
<td>Punjabi</td>
<td>ti</td>
</tr>
<tr>
<td>pl</td>
<td>Polish\ul</td>
<td>tk</td>
</tr>
<tr>
<td>pms</td>
<td>Piedmontese\ul</td>
<td>to</td>
</tr>
<tr>
<td>ps</td>
<td>Pashto</td>
<td>tr</td>
</tr>
<tr>
<td>pt-BR</td>
<td>Portuguese\ul</td>
<td>twq</td>
</tr>
<tr>
<td>pt-PT</td>
<td>Portuguese\ul</td>
<td>tzm</td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese\ul</td>
<td>ug</td>
</tr>
<tr>
<td>qu</td>
<td>Quechua</td>
<td>uk</td>
</tr>
<tr>
<td>rm</td>
<td>Romansh\ul</td>
<td>ur</td>
</tr>
<tr>
<td>rn</td>
<td>Rundi</td>
<td>uz-Arab</td>
</tr>
<tr>
<td>ro</td>
<td>Romanian\ul</td>
<td>uz-Cyr</td>
</tr>
<tr>
<td>Language Code</td>
<td>Language Name</td>
<td>Language Code</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>uz-Latn</td>
<td>Uzbek</td>
<td>yue</td>
</tr>
<tr>
<td>uz</td>
<td>Uzbek</td>
<td>zgh</td>
</tr>
<tr>
<td>vai-Latn</td>
<td>Vai</td>
<td>zh-Hans-HK</td>
</tr>
<tr>
<td>vai-Vaii</td>
<td>Vai</td>
<td>zh-Hans-MO</td>
</tr>
<tr>
<td>vi</td>
<td>Vietnamese</td>
<td>zh-Hans-SG</td>
</tr>
<tr>
<td>vun</td>
<td>Vunj</td>
<td>zh-Hant-HK</td>
</tr>
<tr>
<td>wae</td>
<td>Walser</td>
<td>zh-Hant-MO</td>
</tr>
<tr>
<td>xog</td>
<td>Sogo</td>
<td>zh-Hant</td>
</tr>
<tr>
<td>yav</td>
<td>Yangben</td>
<td>zh</td>
</tr>
<tr>
<td>yi</td>
<td>Yiddish</td>
<td>zu</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 \babelprovide with a valueless import.

<table>
<thead>
<tr>
<th>Language Code</th>
<th>Language Name</th>
<th>Language Code</th>
<th>Language Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>aghem</td>
<td>bosnian-cyrillic</td>
<td>akan</td>
<td>bosnian-cyrl</td>
</tr>
<tr>
<td>akan</td>
<td>bosnian-cyrillic</td>
<td>albanian</td>
<td>bosnian-latin</td>
</tr>
<tr>
<td>american</td>
<td>bosnian-cyrillic</td>
<td>amharic</td>
<td>bosnian-latn</td>
</tr>
<tr>
<td>ancientgreek</td>
<td>bosnian-cyrillic</td>
<td>arabic</td>
<td>breton</td>
</tr>
<tr>
<td>arabic</td>
<td>british</td>
<td>arabic-algeria</td>
<td>british</td>
</tr>
<tr>
<td>arabic-DZ</td>
<td>bulgarian</td>
<td>arabic-morocco</td>
<td>bulgarian</td>
</tr>
<tr>
<td>arabic-MA</td>
<td>burmese</td>
<td>arabic-syria</td>
<td>canadian</td>
</tr>
<tr>
<td>arabic-SY</td>
<td>cantonese</td>
<td>armenian</td>
<td>centralatlastamazight</td>
</tr>
<tr>
<td>assamese</td>
<td>centralatlastamazight</td>
<td>asturian</td>
<td>chechen</td>
</tr>
<tr>
<td>asu</td>
<td>chechen</td>
<td>australian</td>
<td>chiga</td>
</tr>
<tr>
<td>austrian</td>
<td>chinese-hans-hk</td>
<td>azerbaijani-cyrillic</td>
<td>chinese-hans-hk</td>
</tr>
<tr>
<td>azerbaijani-cyrillic</td>
<td>chinese-hans-mo</td>
<td>azerbaijani-cyrl</td>
<td>chinese-hans-sg</td>
</tr>
<tr>
<td>azerbaijani-latin</td>
<td>chinese-hans</td>
<td>azeri</td>
<td>chinese-hant-hk</td>
</tr>
<tr>
<td>azerbaijani-latn</td>
<td>chinese-hant-mo</td>
<td>babel</td>
<td>chinese-hant</td>
</tr>
<tr>
<td>babel</td>
<td>chinese-hant</td>
<td>bambara</td>
<td>chinese-simplified-hongkongsarchina</td>
</tr>
<tr>
<td>basaa</td>
<td>chinese-simplified-macausarchina</td>
<td></td>
<td></td>
</tr>
<tr>
<td>basque</td>
<td>chinese-simplified-singapore</td>
<td></td>
<td></td>
</tr>
<tr>
<td>belarusan</td>
<td>chinese-simplified</td>
<td>bemba</td>
<td>chinese-traditional-hongkongsarchina</td>
</tr>
<tr>
<td>benga</td>
<td>chinese-traditional-macausarchina</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bengali</td>
<td>chinese-traditional</td>
<td>bodo</td>
<td>chinese</td>
</tr>
<tr>
<td>Language</td>
<td>Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Church Slavonic</td>
<td>churchslavic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Church Slavonic-Cyrs</td>
<td>churchslavic-cyrs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Church Slavonic-Old Cyrillic</td>
<td>churchslavic-oldcyrillic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Church Slavonic-Glagolitic</td>
<td>churchslavic-glagolitic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colognian</td>
<td>colognian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cornish</td>
<td>cornish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Croatian</td>
<td>croatian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Czech</td>
<td>czech</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Danish</td>
<td>danish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duala</td>
<td>duala</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dutch</td>
<td>dutch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dzongkha</td>
<td>dzongkha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embu</td>
<td>embu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-AU</td>
<td>english-au</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-Australia</td>
<td>english-australia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-CA</td>
<td>english-ca</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-Canada</td>
<td>english-canada</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-GB</td>
<td>english-gb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-New Zealand</td>
<td>english-newzealand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-NZ</td>
<td>english-nz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-UnitedKingdom</td>
<td>english-unitedkingdom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-UnitedStates</td>
<td>english-unitedstates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English-US</td>
<td>english-us</td>
<td></td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>english</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Esperanto</td>
<td>esperanto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estonian</td>
<td>estonian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ewe</td>
<td>ewe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ewondo</td>
<td>ewondo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faroese</td>
<td>faroese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filipino</td>
<td>filipino</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finnish</td>
<td>finnish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-BE</td>
<td>french-be</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-Belgium</td>
<td>french-belgium</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-CA</td>
<td>french-ca</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-Canada</td>
<td>french-canada</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-CH</td>
<td>french-ch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-Lu</td>
<td>french-lu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-Luxembourg</td>
<td>french-luxembourg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French-Switzerland</td>
<td>french-switzerland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>French</td>
<td>french</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friulian</td>
<td>friulian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fulah</td>
<td>fulah</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Galician</td>
<td>galician</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ganda</td>
<td>ganda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Georgian</td>
<td>georgian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>German-AT</td>
<td>german-at</td>
<td></td>
<td></td>
</tr>
<tr>
<td>German-Austria</td>
<td>german-austria</td>
<td></td>
<td></td>
</tr>
<tr>
<td>German-CH</td>
<td>german-ch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>German-Switzerland</td>
<td>german-switzerland</td>
<td></td>
<td></td>
</tr>
<tr>
<td>German</td>
<td>german</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greek</td>
<td>greek</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12 The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.
<table>
<thead>
<tr>
<th>Language</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makhuwameetto</td>
<td>Quechua</td>
</tr>
<tr>
<td>Makonde</td>
<td>Romanian</td>
</tr>
<tr>
<td>Malagasy</td>
<td>Romansh</td>
</tr>
<tr>
<td>Malay-Bn</td>
<td>Rombo</td>
</tr>
<tr>
<td>Malay-Brunei</td>
<td>Rundi</td>
</tr>
<tr>
<td>Malay-Sg</td>
<td>Russian</td>
</tr>
<tr>
<td>Malay-Singapore</td>
<td>Rwa</td>
</tr>
<tr>
<td>Malay</td>
<td>Sakha</td>
</tr>
<tr>
<td>Malayalam</td>
<td>Samburu</td>
</tr>
<tr>
<td>Maltese</td>
<td>Samin</td>
</tr>
<tr>
<td>Manx</td>
<td>Sango</td>
</tr>
<tr>
<td>Marathi</td>
<td>Sangu</td>
</tr>
<tr>
<td>Masai</td>
<td>Sanskrit-Beng</td>
</tr>
<tr>
<td>Mazanderani</td>
<td>Sanskrit-Dev</td>
</tr>
<tr>
<td>Meru</td>
<td>Sanskrit-Devanagari</td>
</tr>
<tr>
<td>Meta</td>
<td>Sanskrit-Gujarati</td>
</tr>
<tr>
<td>Mexican</td>
<td>Sanskrit-Gujr</td>
</tr>
<tr>
<td>Mongolian</td>
<td>Sanskrit-Kannada</td>
</tr>
<tr>
<td>Morisyen</td>
<td>Sanskrit-Knda</td>
</tr>
<tr>
<td>Mundang</td>
<td>Sanskrit-Malayalam</td>
</tr>
<tr>
<td>Nama</td>
<td>Sanskrit-Mlym</td>
</tr>
<tr>
<td>Nepali</td>
<td>Sanskrit-Telu</td>
</tr>
<tr>
<td>Newzealand</td>
<td>Sanskrit-Telugu</td>
</tr>
<tr>
<td>Ngjemboon</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>Ngomba</td>
<td>Scottish Gaelic</td>
</tr>
<tr>
<td>Norsk</td>
<td>Sena</td>
</tr>
<tr>
<td>Northernluri</td>
<td>Serbian-Cyrilic-Bosnia-Herzegovina</td>
</tr>
<tr>
<td>Northernsami</td>
<td>Serbian-Cyrilic-Kosovo</td>
</tr>
<tr>
<td>Northndebele</td>
<td>Serbian-Cyrilic-Montenegro</td>
</tr>
<tr>
<td>Norwegianbokmal</td>
<td>Serbian-Cyrilic</td>
</tr>
<tr>
<td>Norwegiannynorsk</td>
<td>Serbian-Cyril-XK</td>
</tr>
<tr>
<td>Norwegian-German</td>
<td>Serbian-Cyril</td>
</tr>
<tr>
<td>Nuer</td>
<td>Serbian-Latin-Bosnia-Herzegovina</td>
</tr>
<tr>
<td>Nyankole</td>
<td>Serbian-Latin-Kosovo</td>
</tr>
<tr>
<td>Nynorsk</td>
<td>Serbian-Latin-Montenegro</td>
</tr>
<tr>
<td>Occitan</td>
<td>Serbian-Latin</td>
</tr>
<tr>
<td>Oriya</td>
<td>Serbian-Latin-Ba</td>
</tr>
<tr>
<td>Oromo</td>
<td>Serbian-Latin-Me</td>
</tr>
<tr>
<td>Ossetic</td>
<td>Serbian-Latin-XK</td>
</tr>
<tr>
<td>Pashto</td>
<td>Serbian-Latin-Xk</td>
</tr>
<tr>
<td>Persian</td>
<td>Serbian-Latin</td>
</tr>
<tr>
<td>Piedmontese</td>
<td>Shambala</td>
</tr>
<tr>
<td>Polish</td>
<td>Shona</td>
</tr>
<tr>
<td>Polytonic Greek</td>
<td>Sichuan Yi</td>
</tr>
<tr>
<td>Portuguese-BR</td>
<td>Sinhala</td>
</tr>
<tr>
<td>Portuguese-Brazil</td>
<td>Slovak</td>
</tr>
<tr>
<td>Portuguese-Portugal</td>
<td>Slovene</td>
</tr>
<tr>
<td>Portuguese-PT</td>
<td>Slovenian</td>
</tr>
<tr>
<td>Portuguese</td>
<td>Soga</td>
</tr>
<tr>
<td>Punjabi-Arab</td>
<td>Somali</td>
</tr>
<tr>
<td>Punjabi-Arabic</td>
<td>Spanish</td>
</tr>
<tr>
<td>Punjabi-Gurmukhi</td>
<td>Spanish-Mexico</td>
</tr>
<tr>
<td>Punjabi-Guru</td>
<td>Spanish-Mexico</td>
</tr>
<tr>
<td>Punjabi</td>
<td>Spanish-Mexico</td>
</tr>
</tbody>
</table>
Modifying and adding values to ini files

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

1.14 Selecting fonts

New 3.15 Babel provides a high level interface on top of fontspec to select fonts. There is no need to load fontspec explicitly – babel does it for you with the first \babelfont.\footnote{See also the package combofont for a complementary approach.}

\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 \textit{rm}, \textit{sf} or \textit{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.

```latex
\documentclass{article}
\usepackage[swedish, bidi=default]{babel}
\babelprovide[import]{hebrew}
\babelfont{rm}{FreeSerif}
\begin{document}
Svenska \foreignlanguage{hebrew}{תיִרְבִע} svenska.
\end{document}
```

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

```latex
\babelfont{rm}{Iwona}
\babelfont[hebrew]{rm}{FreeSerif}
```

\babelfont can be used to implicitly define a new font family. Just write its name instead of \texttt{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:

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

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

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

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

This makes sure the OpenType script for Devanagari is deva and not dev2, in case it is not detected correctly. You may also pass some options to fontspec: with silent, the warnings about unavailable scripts or languages are not shown (they are only really useful when the document format is being set up).
NOTE Directionality is a property affecting margins, indentation, column order, etc., not just text. Therefore, it is under the direct control of the language, which applies both the script and the direction to the text. As a consequence, there is no need to set Script when declaring a font with \babelfont (nor Language). In fact, it is even discouraged.

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

NOTE The keys Language and Script just pass these values to the font, and do not set the script for the language (and therefore the writing direction). In other words, the ini file or \babel 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.


This is not an error. This warning is shown by fontspec, not by babel. It can 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 \babelfont for a family, very likely you want to define the rest of them. If you don’t, you can find some inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used.

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

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

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

1.15 Modifying a language

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

\setlocalecaption \langle language-name \rangle \langle caption-name \rangle \langle string \rangle

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

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

This works not only with existing caption names, because it also serves to define new ones by setting the caption-name to the name of your choice (name will be postpended). Captions so defined or redefined behave with the ‘new way’ described in the following note.
NOTE There are a few alternative methods:

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

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

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

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

  \texttt{\textbackslash addto\textbackslash captionsenglish{%
    \renewcommand\contentsname{Foo}%
  }\}

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

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

  \texttt{\renewcommand\spanishchaptername{Foo}}

  This redefinition is immediate.

NOTE Do not redefine a caption in the following way:

\texttt{\textbackslash AtBeginDocument{\textbackslash renewcommand\contentsname{Foo}}}

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

Macros to be run when a language is selected can be add to \texttt{\textbackslash extras\langle lang\rangle}:

\texttt{\textbackslash addto\textbackslash extrasrussian\langle mymacro\rangle}

There is a counterpart for code to be run when a language is unselected: \texttt{\textbackslash noextras\langle lang\rangle}.

NOTE These macros (\texttt{\textbackslash captions\langle lang\rangle}, \texttt{\textbackslash extras\langle lang\rangle}) 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 \texttt{\textbackslash babelprovide}, described below in depth. So, something like:

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

first loads danish.1df, and then redefines the captions for danish (as provided by the ini file) and prevents hyphenation. The rest of the language definitions are not touched. Without the optional argument it just loads some additonal tools if provided by the ini file, like extra counters.

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 \{⟨language-name⟩\}

If the language ⟨language-name⟩ has not been loaded as class or package option and there are no ⟨options⟩, it creates an “empty” one with some defaults in its internal structure: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3. In either case, caption, date and language system are not defined.

If no ini file is imported with import, ⟨language-name⟩ is still relevant because in such a case the hyphenation and like breaking rules (including those for South East Asian and CJK) are based on it as provided in the ini file corresponding to that name; the same applies to OpenType language and script.

Conveniently, some options allow to fill the language, and babel warns you about what to do if there is a missing string. Very likely you will find alerts like that in the log file:

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

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

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

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

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

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

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

The main language is not changed (danish in this example). So, you must add \selectlanguage{arhinish} or other selectors where necessary.

If the language has been loaded as an argument in \documentclass or \usepackage, then \babelprovide redefines the requested data.

**import= ⟨language-tag⟩**

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

\babelprovide[import=hu]{hungarian}

Unicode engines load the UTF-8 variants, while 8-bit engines load the LICR (ie, with macros like \’ or \ss) ones.

**New 3.23** It may be used without a value. 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 can be written:
There are about 250 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages may show a warning about the current lack of suitability of some features.

Besides \today, this option defines an additional command for dates: \<language>date, which takes three arguments, namely, year, month and day numbers. In fact, \today calls \<language>date, which in turn calls \<language>date{\the\year}{\the\month}{\the\day}. New 3.44 More convenient is usually \localedate, with prints the date for the current locale.

\texttt{\textbackslash babelprovide[import\{hungarian\}]

\texttt{captions=\langle language-tag \rangle}

Loads only the strings. For example:

\texttt{\textbackslash babelprovide[\texttt{\textbackslash captions=hu}]{hungarian}}

\texttt{\texttt{\textbackslash 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:

\texttt{\textbackslash babelprovide[\texttt{\textbackslash 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:

\texttt{\textbackslash babelprovide[\texttt{\textbackslash hyphenrules=}]{\textbackslash neo} \textbackslash babelpatterns[\textbackslash neo]{a1 e1 i1 o1 u1}}

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

New 3.58 Another special value is unhyphenated, which activates a line breaking mode that allows spaces to be stretched to arbitrary amounts.

\texttt{main}

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

\texttt{\textbackslash usepackage[italian, greek.polutonic]{babel}}

\texttt{\textbackslash usepackage[italian, greek.polutonic]{babel}}

But if, say, accents in Greek are not shown correctly, you can try:
\usepackage{italian}{babel}
\babelprovide[import, main]{polytonicgreek}

Remember there is an alternative syntax for the latter:
\usepackage{italian, polytonicgreek, provide=*}{babel}

\verb|\script-name| \verb|\language-name| \verb|\counter-name|

\verb|\script-name| \verb|\language-name| \verb|\counter-name|
\verb|\alph| \verb|\Alph|

\texttt{\verb|\language-name|} \texttt{\verb|\counter-name|}

\begin{itemize}
\item \texttt{\script-name} \texttt{\language-name} \texttt{\counter-name}
\item \texttt{\alph} \texttt{\Alph}
\item \texttt{\language-name} \texttt{\counter-name}
\item \texttt{\localeid}
\item \texttt{\language}
\item \texttt{\babelcharproperty}
\item \texttt{\babelfont}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\item \texttt{\spaceskip}
\end{itemize}

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

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).

\texttt{justification=} kashida | elongated | unhyphenated

\textbf{New 3.59} There are currently three options, mainly for the Arabic script. It sets the linebreaking and justification method, which can be based on the the \texttt{ARABIC TATWEEL} character or in the 'justification alternatives' OpenType table (jalt). For an explanation see the \textit{babel} site.

\texttt{linebreaking=} \textbf{New 3.59} Just a synonymous for \textit{justification}.

\texttt{mapfont=} direction

Assigns the font for the writing direction of this language (only with \texttt{bidi=basic}). Whenever possible, instead of this option use \texttt{onchar}, based on the script, which usually makes more sense. More precisely, what \texttt{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.

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

1.17 \textbf{Digits and counters}

\textbf{New 3.20} About thirty \texttt{ini} files define a field named \texttt{digits.native}. When it is present, two macros are created: \texttt{\langle\textit{language}\rangle digits} and \texttt{\langle\textit{language}\rangle counter} (only \texttt{xetex} and \texttt{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 \texttt{maparabic} in \texttt{\babelprovide}, \texttt{arabic} is redefined to produce the native digits (this is done \textit{globally}, to avoid inconsistencies in, for example, page numbering, and note as well dates do not rely on \texttt{arabic}.)

For example:

\begin{verbatim}
\ 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}
\end{verbatim}

Languages providing native digits in all or some variants are:

<table>
<thead>
<tr>
<th>Arabic</th>
<th>Central Kurdish</th>
<th>Khmer</th>
<th>Northern Luri</th>
<th>Nepali</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assamese</td>
<td>Dzongkha</td>
<td>Kannada</td>
<td>Malayalam</td>
<td>Odia</td>
</tr>
<tr>
<td>Bangla</td>
<td>Persian</td>
<td>Konkani</td>
<td>Marathi</td>
<td>Punjabi</td>
</tr>
<tr>
<td>Tibetan</td>
<td>Gujarati</td>
<td>Kashmiri</td>
<td>Burmese</td>
<td>Pashto</td>
</tr>
<tr>
<td>Bodo</td>
<td>Hindi</td>
<td>Lao</td>
<td>Mazanderani</td>
<td>Tamil</td>
</tr>
<tr>
<td>Telugu</td>
<td>Uyghur</td>
<td>Uzbek</td>
<td>Cantonese</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>Thai</td>
<td>Urdu</td>
<td>Vai</td>
<td>Chinese</td>
<td></td>
</tr>
</tbody>
</table>

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

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

**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 expendable (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, tigré, 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
In addition, native digits (in languages defining them) may be printed with the numeral style digits.

### Dates

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\}
```

By default the calendar is the Gregorian, but an ini file 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 31ê Çileya Pêşînê 2019.

### Accessing language info

The control sequence \language name 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⟩}
```

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

- name.english as provided by the Unicode CLDR.
- tag.ini is the tag of the ini file (the way this file is identified in its name).
- tag.bcp47 is the full BCP 47 tag (see the warning below).
- language.tag.bcp47 is the BCP 47 language tag.
- tag.openotype is the tag used by OpenType (usually, but not always, the same as BCP 47).
- script.name, as provided by the Unicode CLDR.
- script.tag.bcp47 is the BCP 47 tag of the script used by this locale.
- script.tag.openotype is the tag used by OpenType (usually, but not always, the same as BCP 47).

**WARNING** As of version 3.46 tag.bcp47 returns the full BCP 47 tag. Formerly it returned just the language subtag, which was clearly counterintuitive.
\getlocaleproperty \{macro\}\{locale\}\{property\}

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

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

the macro \hechap will contain the string קרפ.

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

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

**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.
• \texttt{\textbackslash babelhyphen\{soft\}} and \texttt{\textbackslash 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\{\langle text\rangle\}} is a hard “hyphen” using \texttt{\langle text\rangle} 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 \texttt{-}), \texttt{\textbackslash babelhyphen\*\{hard\}}, etc.

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

There are also some differences with \texttt{\LaTeX}: (1) the character used is that set for the current font, while in \texttt{\LaTeX} it is hardwired to \texttt{-} (a typical value); (2) the hyphen to be used in fonts with a negative \texttt{\hyphenchar} is \texttt{-}, like in \texttt{\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,...]{\langle exceptions\rangle}]

\textbf{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 line codes’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:

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

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

\textbf{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{luatex}). Even if there are no patterns for the language, you can add at least some typical cases.

\textbf{NOTE} To set hyphenation exceptions in the preamble before any language is explicitly set with a selector, use \texttt{\textbackslash babelhyphenation} instead of \texttt{\textbackslash hyphenation}. In the preamble the hyphenation rules are not always fully set up and an error can be raised.

\texttt{\textbackslash begin\{hyphenrules\} }{\langle language\rangle} ... \texttt{\textbackslash end\{hyphenrules\}}

The environment \texttt{\textbackslash hyphenrules} can be used to select only the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘\texttt{\textbackslash nohyphenation}’, provided that in \texttt{language.dat} the ‘language’ \texttt{\textbackslash nohyphenation} is defined by loading \texttt{zero\hyph\.tex}. It deactivates language shorthands, too (but not user shorthands). Except for these simple uses, \texttt{\textbackslash hyphenrules} is deprecated and other \texttt{\textbackslash language\*} (the starred version) is preferred, because the former does not take into account possible changes in encodings of characters like, say, ‘ ‘ done by some languages (eg, italian, french, ukraineb).
\babelpatterns \{\langle language\rangle, \langle language\rangle, ...\}\{\langle patterns\rangle\}

**New 3.9m** In *luatex only*,\(^{14}\) adds or replaces patterns for the languages given or, without the optional argument, for all languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \lccodes's done in \extras\langle lang\rangle as well as the language-specific encoding (not set in the preamble by default). Multiple \babelpatterns's are allowed.

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

**New 3.31** (Only *luatex.*) With \babelprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (**New 3.32** it is disabled in verbatim mode, or more precisely when the hyphenrules are set to nohyphenation). It can be activated alternatively by setting explicitly the intraspace.

**New 3.27** Interword spacing for Thai, Lao and 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).

### 1.21 Transforms

Transforms (only *luatex*) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.\(^{15}\)

It currently embraces \babelprehyphenation and \babelposthyphenation.

**New 3.57** Several ini files predefine some transforms. They are activated with the key transforms in \babelprovide, either if the locale is being defined with this macro or the languages has been previously loaded as a class or package option, as the following example illustrates:

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

Here are the transforms currently predefined. (More to follow in future releases.)

<table>
<thead>
<tr>
<th>Language</th>
<th>Transform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>transliteration.dad</td>
<td>Applies the transliteration system devised by Yannis Haralambous for dad (simple and \TeX-friendly). Not yet complete, but sufficient for most texts.</td>
</tr>
<tr>
<td>Croatian</td>
<td>digraphs.ligatures</td>
<td>Ligatures (Dž, ) (Dž, ) (dž, ) (LJ, ) (lj, ) (NJ, ) (nj, ) (Nj, ) (nj). It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.</td>
</tr>
<tr>
<td>Czech, Polish,</td>
<td>hyphen.repeat</td>
<td>Explicit hyphens behave like \babelhyphen {repeat}.</td>
</tr>
<tr>
<td>Portuguese, Slovak,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^{14}\)With *luatex exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and *babel* only provides the most basic tools.

\(^{15}\)They are similar in concept, but not the same, as those in Unicode. The main inspiration for this feature is the Omega transformation processes.
<table>
<thead>
<tr>
<th>Language, Script</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Czech, Polish, Slovak</td>
<td>oneletter.nobreak</td>
<td>Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.</td>
</tr>
<tr>
<td>Greek</td>
<td>diaeresis.hyphen</td>
<td>Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.</td>
</tr>
<tr>
<td>Greek</td>
<td>transliteration.omega</td>
<td>Although he provided combinations are not exactly the same, this transform follows the syntax of Omega: ( = ) for the circumflex, ( v ) for digamma, and so on. For better compatibility with Levy’s system, ( \sim ) (as ‘string’) is an alternative to ( = ).’ is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek.</td>
</tr>
<tr>
<td>Greek</td>
<td>sigma.final</td>
<td>The transliteration system above does not convert the sigma at the end of a word (on purpose). This transforms does it. To prevent the conversion (an abbreviation, for example), write &quot;s.</td>
</tr>
<tr>
<td>Hindi, Sanskrit</td>
<td>transliteration.hk</td>
<td>The Harvard-Kyoto system to romanize Devanagari.</td>
</tr>
<tr>
<td>Hindi, Sanskrit</td>
<td>punctuation.space</td>
<td>Inserts a space before the following four characters: !?:;</td>
</tr>
<tr>
<td>Hungarian</td>
<td>digraphs.hyphen</td>
<td>Hyphenates the long digraphs ( ccs, ddz, ggy, lly, nny, ssz, tty ) and ( zzs ) as ( cs-cs, dz-dz ), etc.</td>
</tr>
<tr>
<td>Indic scripts</td>
<td>danda.nobreak</td>
<td>Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.</td>
</tr>
<tr>
<td>Arabic, Persian</td>
<td>kashida.plain</td>
<td>Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.</td>
</tr>
<tr>
<td>Serbian</td>
<td>transliteration.gajica</td>
<td>(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.</td>
</tr>
</tbody>
</table>

\[ \text{\textbackslash{babelposthyphenation}} \{\text{hyphenrules-name}\} \{\text{lua-pattern}\} \{\text{replacement}\} \]

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

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

39
In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads \( ([ιύ]) \), the replacement could be \{1|ιύ|ιύ\}, which maps \( ί \) to \( ί \), and \( ύ \) to \( ύ \), so that the diaeresis is removed.

This feature is activated with the first \texttt{\textbackslash babelpostphyphenation} or \texttt{\textbackslash babelprehyphenation}. See the babel site for a more detailed description and some examples. It also describes a few additional replacement types (string, penalty).

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

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

\texttt{\textbackslash babelprehyphenation \{\langle locale-name\rangle\}\{\langle lua-pattern\rangle\}\{\langle replacement\rangle\}}

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

This feature is activated with the first \texttt{\textbackslash babelpostphyphenation} or \texttt{\textbackslash babelprehyphenation}.

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

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

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

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

**NOTE** With luatex there is another approach to make text transformations, with the function \texttt{fonts\_handlers\_otf\_addfeature}, which adds new features to an OTF font (substitution and positioning). These features can be made language-dependent, and babel by default recognizes this setting if the font has been declared with \texttt{\babelfont}. The \texttt{transforms} mechanism supplements rather than replaces OTF features.

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

1.22 Selection based on BCP 47 tags

New 3.43 The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in documents (or parts of documents) generated by external sources, and therefore babel will provide a set of tools to select the locales in different situations, adapted to the particular needs of
each case. Currently, babel provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested. It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in babel. Instead the data is taken from the ini files, which means currently about 250 tags are already recognized. Babel performs a simple lookup in the following way: fr-Latn-FR → fr-Latn → fr-FR → fr. Languages with the same resolved name are considered the same. Case is normalized before, so that fr-latn-fr → fr-Latn-FR. If a tag and a name overlap, the tag takes precedence. Here is a minimal example:

\documentclass{article}
\usepackage[danish]{babel}
\babeladjust{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}
\begin{document}
  Chapter in Danish: \chaptername.
  \selectlanguage{de-AT}
  \localedate{2020}{1}{30}
\end{document}

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

- autoload.bcp47 with values on and off.
- autoload.bcp47.options, which are passed to \babelprovide; empty by default, but you may add import (features defined in the corresponding babel-...tex file might not be available).
- autoload.bcp47.prefix. Although the public name used in selectors is the tag, the internal name will be different and generated by prepending a prefix, which by default is bcp47-. You may change it with this key.

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

\babeladjust{ bcp47.toname = on }

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

Currently babel provides no standard interface to select scripts, because they are best selected with either \fontencoding (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.  

Some languages sharing the same script define macros to switch it (eg, \textcyrillic), but be aware they may also set the language to a certain default. Even the babel core defined \textlatin, but 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.

\ensureascii \{⟨text⟩\}

\textbf{New 3.9i} This macro makes sure ⟨text⟩ is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine \TeX{} and \LaTeX{} so that they are correctly typeset even with LGR or X2 (the complete list is stored in \BabelNonASCII, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.

If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also \TeX{} and \LaTeX{} are not redefined); otherwise, \ensureascii switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load LY1, LGR, then it is set to LY1, but if you load LY1, T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for “ordinary” text (they are stored in \BabelNonText, used in some special cases when no Latin encoding is explicitly set).

The foregoing rules (which are applied “at begin document”) cover most of the cases. No assumption is made on characters above 127, which may not follow the LICR conventions – the goal is just to ensure most of the ASCII letters and symbols are the right ones.

1.24 Selecting directions

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

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

\textbf{WARNING} If characters to be mirrored are shown without changes with luatex, try with the following line:

\textbf{16} 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.

\textbf{17} But still defined for backwards compatibility.
There are some package options controlling bidi writing.

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

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

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

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

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

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

```latex
\documentclass{article}
\usepackage[bidi=basic]{babel}
\babelprovide[import, main]{arabic}
\babelfont{rm}{FreeSerif}
\begin{document}
\begin{document}
ٍـﻳ(ﻳﻘﻳﺮﻏﻻا)ﻲﻨﻴﻠﻴﻬﻟاﺮﺼﻌﻟاﺔﻠﻴﻃبﺮﻌﻟاةﺮﻳﺰﺟﻪﺒﺷﺖﻓﺮﻋﺪﻗو

Arabia
وأ
Aravia )

\(ـﺑتﺎﺋدﺎﺑ
Arabia
\) أ
Aravia (باللغة العربية)ً

مودختا،

"ـﺑتﺎﺋدﺎﺑ
Arabia"

أ
Aravia

بادنات يـ "Arabi"اً

قطنيًً، قطنيًً،

غريفة، غريفة،

كانت أكثر مما تعري عليه اليوم.

\end{document}
```

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

```latex
\documentclass{book}
\usepackage[english, bidi=basic]{babel}
\babelprovide[onchar=ids fonts]{arabic}
\babelfont{rm}{Crimson}
\babelfont[arabic]{rm}{FreeSerif}
```

43
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 \texttt{arabic}) changes its font to that set for this language (here defined via \texttt{*arabic}, because Crimson does not provide Arabic letters).

**NOTE** Boxes are “black boxes”. Numbers inside an \hbox (for example in a \texttt{\ref}) do not know anything about the surrounding chars. So, \texttt{\ref{A}}-\texttt{\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 \texttt{\ref} ranges, the best option is to define a dedicated macro like this (to avoid explicit direction changes in the body; here \texttt{\texthe} must be defined to select the main language):

\begin{verbatim}
\newcommand\refrange[2]{\babelsublr{\texthe{\ref{#1}}-\texthe{\ref{#2}}}}
\end{verbatim}

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

\begin{verbatim}
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 \texttt{\BabelPatchSection} for further details).

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

With counters, \texttt{\arabic} is not only considered L text always (with \texttt{\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 \texttt{bidi=basic} (as a decimal number), in \texttt{\arabic{c1}.\arabic{c2}} the visual order is \texttt{c2.c1}. Of course, you may always adjust the order by changing the language, if necessary.\footnote{Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.}

lists required in \texttt{\LaTeX} and \texttt{pdftex}, but only in bidirectional (with both R and L paragraphs) documents in \texttt{\LaTeX}.

**WARNING** As of April 2019 there is a bug with \texttt{\parshape} in \texttt{\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.
\end{verbatim}
contents required in xetex and pdftex; in luatex toc entries are R by default if the main language is R.
columns required in xetex and pdftex to reverse the column order (currently only the standard two-column mode); in luatex they are R by default if the main language is R (including multicol).
footnotes not required in monolingual documents, but it may be useful in bidirectional documents (with both R and L paragraphs) in all engines; you may use alternatively \BabelFootnote described below (what this option does exactly is also explained there).
captions is similar to sectioning, but for \caption; not required in monolingual documents with luatex, but may be required in xetex and pdftex in some styles (support for the latter two engines is still experimental) \textbf{New 3.18}.
tabular required in luatex for R tabular, so that the first column is the right one (it has been tested only with simple tables, so expect some readjustments in the future). ignored in pdftex or xetex (which will not support a similar option in the short term). It patches an internal command, so it might be ignored by some packages and classes (or even raise an error). \textbf{New 3.18}.
graphics modifies the picture environment so that the whole figure is L but the text is R. It does not work with the standard picture, and pict2e is required. It attempts to do the same for pgf/tikz. Somewhat experimental. \textbf{New 3.32}.
extras is used for miscellaneous readjustments which do not fit into the previous groups. Currently redefines in luatex \underline and \LaTeX2e \textbf{New 3.19}.

\verb|\BabelPatchSection|\quad\verb|\{section-name\}|
Mainly for bidi text, but it can be useful in other cases. \Babel PatchSection 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

\verb|\babelsublr|\quad\verb|\{lr-text\}|
Digits in pdftex must be marked up explicitly (unlike luatex with bidi=basic or bidi=basic-r and, usually, xetex). This command is provided to set \{lr-text\} in L mode if necessary. it's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no rl counterpart. Any \babelsublr in explicit L mode is ignored. However, with bidi=basic and implicit L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

\texttt{\textbf{RTL A ltr text thechapter{} and still ltr RTL B}}

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

\texttt{\textbf{RTL A foreignlanguage(english){ltr text thechapter{} and still ltr} RTL B}}

\verb|\BabelPatchSection|\quad\verb|\{section-name\}|
Mainly for bidi text, but it can be useful in other cases. \Babel PatchSection 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  \{\langle cmd\rangle\}\{\langle local-language\rangle\}\{\langle before\rangle\}\{\langle after\rangle\}

New 3.17 Something like:

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

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

\footnote{\langle foreignlanguage{\languagename}{note}\rangle}

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

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

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

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

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

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

1.25 Language attributes

\languageattribute

This is a user-level command, to be used in the preamble of a document (after \usepackage[...]{babel}), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language. Very often, using a modifier in a package option is better. Several language definition files use their own methods to set options. For example, french uses \frenchsetup, magyar (1.5) uses \magyarOptions; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, \ProsodicMarksOn in latin).
1.26 Hooks

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

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

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

- **addialect** (language name, dialect name) Used by luababel.def to load the patterns if not preloaded.
- **patterns** (language name, language with encoding) Executed just after the \language has been set. The second argument has the patterns name actually selected (in the form of either lang:ENC or lang).
- **hyphenation** (language name, language with encoding) Executed locally just before exceptions given in \babelhyphenation are actually set.
- **defaultcommands** Used (locally) in \StartBabelCommands.
- **encodedcommands** (input, font encodings) Used (locally) in \StartBabelCommands. Both xetex and luatex make sure the encoded text is read correctly.
- **stopcommands** Used to reset the above, if necessary.
- **write** This event comes just after the switching commands are written to the aux file.
- **beforeextras** Just before executing \extras⟨language⟩. This event and the next one should not contain language-dependent code (for that, add it to \extras⟨language⟩).
- **afterextras** Just after executing \extras⟨language⟩. For example, the following deactivates shorthands in all languages:

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

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

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

- **initiateactive** (char as active, char as other, original char) New 3.9i Executed just after a shorthand has been ‘initiated’. The three parameters are the same character with different catcodes: active, other (\string’ed) and the original one.
- **afterreset** New 3.9i Executed when selecting a language just after \OriginalTeX is run and reset to its base value, before executing \captions⟨language⟩ and \date⟨language⟩.

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.

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

### 1.27 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.

<table>
<thead>
<tr>
<th>Language</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afrikaans</td>
<td>afrikaans</td>
</tr>
<tr>
<td>Azerbaijani</td>
<td>azerbaijani</td>
</tr>
<tr>
<td>Basque</td>
<td>basque</td>
</tr>
<tr>
<td>Breton</td>
<td>breton</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>bulgarian</td>
</tr>
<tr>
<td>Catalan</td>
<td>catalan</td>
</tr>
<tr>
<td>Croatian</td>
<td>croatian</td>
</tr>
<tr>
<td>Czech</td>
<td>czech</td>
</tr>
<tr>
<td>Danish</td>
<td>danish</td>
</tr>
<tr>
<td>Dutch</td>
<td>dutch</td>
</tr>
<tr>
<td>English</td>
<td>english, USenglish, american, UKenglish, british, canadian, australian, newzealand</td>
</tr>
<tr>
<td>Esperanto</td>
<td>esperanto</td>
</tr>
<tr>
<td>Estonian</td>
<td>estonian</td>
</tr>
<tr>
<td>Finnish</td>
<td>finnish</td>
</tr>
<tr>
<td>French</td>
<td>french, francais, canadien, acadian</td>
</tr>
<tr>
<td>Galician</td>
<td>galician</td>
</tr>
<tr>
<td>German</td>
<td>austrian, german, germanb, ngerman, naustrian</td>
</tr>
<tr>
<td>Greek</td>
<td>greek, polutonikgreek</td>
</tr>
<tr>
<td>Hebrew</td>
<td>hebrew</td>
</tr>
<tr>
<td>Icelandic</td>
<td>icelandic</td>
</tr>
<tr>
<td>Indonesian</td>
<td>indonesian (bahasa, indon, bahasai)</td>
</tr>
<tr>
<td>Interlingua</td>
<td>interlingua</td>
</tr>
<tr>
<td>Irish Gaelic</td>
<td>irish</td>
</tr>
<tr>
<td>Italian</td>
<td>italian</td>
</tr>
<tr>
<td>Latin</td>
<td>latin</td>
</tr>
<tr>
<td>Lower Sorbian</td>
<td>lowersorbian</td>
</tr>
<tr>
<td>Malay</td>
<td>malay, melayu (bahasam)</td>
</tr>
<tr>
<td>North Sami</td>
<td>samin</td>
</tr>
<tr>
<td>Norwegian</td>
<td>norsk, nynorsk</td>
</tr>
<tr>
<td>Polish</td>
<td>polish</td>
</tr>
<tr>
<td>Portuguese</td>
<td>portuguese, brazilian (portuges, brazil)</td>
</tr>
<tr>
<td>Romanian</td>
<td>romanian</td>
</tr>
<tr>
<td>Russian</td>
<td>russian</td>
</tr>
<tr>
<td>Scottish Gaelic</td>
<td>scottish</td>
</tr>
<tr>
<td>Spanish</td>
<td>spanish</td>
</tr>
<tr>
<td>Slovakian</td>
<td>slovak</td>
</tr>
<tr>
<td>Slovenian</td>
<td>slovene</td>
</tr>
<tr>
<td>Swedish</td>
<td>swedish</td>
</tr>
</tbody>
</table>

19 The two last names come from the times when they had to be shortened to 8 characters.
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, serbian-c, 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:

```latex
\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.

### 1.28 Unicode character properties in luatex

New 3.32 Part of the babel job is to apply Unicode rules to some script-specific features based on some properties. Currently, they are 3, namely, direction (ie, bidi class), mirroring glyphs, and line breaking for CJK scripts. These properties are stored in lua tables, which you can modify with the following macro (for example, to set them for glyphs in the PUA).

\begin{verbatim}
\babelcharproperty \{char-code\} \{to-char-code\} \{property\} \{value\}
\end{verbatim}

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

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

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:

\begin{verbatim}
\babelcharproperty{`,}{locale}{english}
\end{verbatim}

### 1.29 Tweaking some features
\babeladjust{(key-value-list)}

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

1.30 Tips, workarounds, known issues and notes

- If you use the document class book and you use \ref inside the argument of \chapter (or just use \ref inside \MakeUppercase), \LaTeX{} will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use \lowercase{\ref{foo}} inside the argument of \chapter, or, if you will not use shorthands in labels, set the safe option to none or bib.
- Both \texttt{ltxdoc} and babel use \AtBeginDocument to change some catcodes, and babel reloads hhline to make sure: has the right one, so if you want to change the catcode of | it has to be done using the same method at the proper place, with

\begin{verbatim}
\AtBeginDocument{\DeleteShortVerb{|}}
\end{verbatim}

before loading babel. This way, when the document begins the sequence is (1) make | active (ltxdoc); (2) make it unactive (your settings); (3) make babel shorthands active (babel); (4) reload hhline (babel, now with the correct catcodes for | and :).
- Documents with several input encodings are not frequent, but sometimes are useful. You can set different encodings for different languages as the following example shows:

\begin{verbatim}
\addto\extrasfrench{\inputencoding{latin1}}
\addto\extrarussian{\inputencoding{koi8-r}}
\end{verbatim}

- For the hyphenation to work correctly, lcodes 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 \LaTeX{} assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, \texttt{\textbackslash savinghyphcodes} is not a solution either, because lcodes for hyphenation are frozen in the format and cannot be changed.} So, if you write a chunk of French text with \texttt{\foreignlanguage{french}}, 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}, or redefine \texttt{\textquoteright} (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 \texttt{\bibitem} uses \texttt{\immediate} (and others, in fact), while \texttt{\selectlanguage} doesn't. There is a similar issue with floats, too. There is no known workaround.
- Babel does not take into account \texttt{\normalsf} 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 (i.e., with math code “8000) as a shorthand can make \TeX{} enter in an infinite loop in some rare cases. (Another issue in the ‘to do’ list, although there is a partial solution.)
The following packages can be useful, too (the list is still far from complete):

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

### 1.31 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better). Useful additions would be, for example, time, currency, addresses and personal names. But that is the easy part, because they don’t require modifying the \LaTeX{} internals. Calendars (Arabic, Persian, Indic, etc.) are under study. Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-ből”, in Spanish an item labelled “3.o” may be referred to as either “item 3.0” or “3er item”, and so on. An option to manage bidirectional document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (xe-bidi).

### 1.32 Tentative and experimental code

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

**Options for locales loaded on the fly**

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

**Labels**

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

\footnote{21See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to \LaTeX{} because their aim is just to display information and not fine typesetting.}
2 Loading languages with language.dat

\LaTeX{} and most engines based on it (pdf\LaTeX{}, \texttt{xetex}, \texttt{\epsilon}\LaTeX{}, the main exception being \texttt{luatex}) require hyphenation patterns to be preloaded when a format is created (e\LaTeX{}, \texttt{Xe\LaTeX{}}, pdf\LaTeX{}). \texttt{babel} provides a tool which has become standard in many distributions and based on a “configuration file” named language.dat. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).

\textbf{New 3.9q} With \texttt{luatex}, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typicallyenglish, 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 language.dat.lua, but now a new mechanism has been devised based solely on language.dat. You must rebuild the formats if upgrading from a previous version. You may want to have a local language.dat for a particular project (for example, a book on Chemistry).

\subsection{Format}

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

\begin{verbatim}
german:T1 hyphenT1.ger

german hyphen.ger
\end{verbatim}

With the previous settings, if the encoding when the language is selected is T1 then the patterns in hyphenT1.ger are used, but otherwise use those in hyphen.ger (note the encoding can be set in \texttt{\textbackslash extras\langle lang\rangle}).

A typical error when using \texttt{babel} is the following:

\begin{verbatim}
No hyphenation patterns were preloaded for the language '\langle lang\rangle' into the format.
Please, configure your \TeX{} system to add them and
\end{verbatim}

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

\footnote{The loader for \texttt{lua\LaTeX{}} is slightly different as it’s not based on \texttt{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 \texttt{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.}
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 `\langle\langle lang\rangle\rangle`, `\langle\langle captions\rangle\rangle`, `\langle\langle date\rangle\rangle`, `\langle\langle extras\rangle\rangle` and `\langle\langle noextras\rangle\rangle` (the last two may be left empty); where `\langle lang\rangle` 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, `\langle\langle date\rangle\rangle` but not `\langle\langle captions\rangle\rangle` does not raise an error but can lead to unexpected results.

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

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

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

Some recommendations:

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

- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
• Avoid adding things to `\noextras{lang}` except for `umlauthigh` and friends, `\bbld{deactivate}`, `\bbld{(non)frenchspacing}`, and language-specific macros. Use always, if possible, `\bbld{save}` and `\bbld{savevariable}` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras{lang}`.

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

• Please, for “private” internal macros do not use the `\bbld{prefix}`. It is used by babel and it can lead to incompatibilities.

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

3.1 Guidelines for contributed languages

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

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

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

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

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

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

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

The following page provides a starting point for ldf files:

http://www.texnia.com/incubator.html. See also


If you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

3.2 Basic macros

In the core of the babel system, several macros are defined for use in language definition files. Their purpose is to make a new language known. The first two are related to hyphenation patterns.
\addlanguage  The macro \addlanguage is a non-outer version of the macro \newlanguage, defined in plain.tex version 3.x. Here “language” is used in the \TeX sense of set of hyphenation patterns.

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

\langhyphenmins  The macro \langhyphenmins 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.

\bblDeclareAttribute  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 \ProvidesPackage \TeX command.

\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.

\footnote{But not removed, for backward compatibility.}
\texttt{\textbackslash substitutefontfamily} (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 \texttt{.fd} file will instruct \LaTeX 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 \texttt{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).

\begin{verbatim}
\ProvidesLanguage{<language>}
\ldfInit{<language>}{captions\language}
\ifx\undefined\l@<language>
\@nopatterns{<Language>}
\adddialect\l@<language>0
\fi
\adddialect\l@<dialect>\l@<language>
\bbl@declare@tribute{<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>}
\end{verbatim}

\textbf{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 \texttt{ldf} file, but it can be delayed with \texttt{\AtEndOfPackage}. Macros from external packages can be used \textit{inside} definitions in the \texttt{ldf} itself (for example, \texttt{\extras<language>}), but if executed directly, the code must be placed inside \texttt{\AtEndOfPackage}. A trivial example illustrating these points is:
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 \dospecial. \TeX{} adds another macro called \sanitize representing the same character set, but without the curly braces. The macros \bbl@add@special\{char\} and \bbl@remove@special\{char\} add and remove the character \{char\} to these two sets.

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\(^\text{27}\).

\babel@save To save the current meaning of any control sequence, the macro \babel@save is provided. It takes one argument, \{csname\}, the control sequence for which the meaning has to be saved.

\babel@savevariable A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the \verb\{\ the primitive is considered to be a variable. The macro takes one argument, the \{variable\}. The effect of the preceding macros is to append a piece of code to the current definition of \originalTeX. When \originalTeX is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

3.6 Support for extending macros

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

Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this

\(^\text{27}\)This mechanism was introduced by Bernd Raichle.
behavior is preserved for backward compatibility. If you are using etoolbox, by Philipp Lehman, consider using the tools provided by this package instead of \addto.

### 3.7 Macros common to a number of languages

\bbl@allowhyphens

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

\allowhyphens

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

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

\set@low@box

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

\save@sf@q

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

\bbl@frenchspacing \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

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

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

\StartBabelCommands

{⟨language-list⟩}{⟨category⟩}{⟨selector⟩}

The ⟨language-list⟩ specifies which languages the block is intended for. A block is taken into account only if the \CurrentOption is listed here. Alternatively, you can define \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 and luatex (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 charset= followed by a charset, which if given sets how the strings should be translated to the internal representation used by the engine, typically utf8, which is the
only value supported currently (default is no translations). Note charset is applied by luatex and xetex when reading the file, not when the macro or string is used in the document.

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

Blocks without a selector are read always if the key 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 strings=generic (no block is taken into account except those). With strings=encoded, strings in those blocks are set as default (internally, ?). With strings=encoded strings are protected, but they are correctly expanded in \MakeUppercase and the like. If there is no key strings, string definitions are ignored, but \SetCases are still honored (in a encoded way).

The ⟨category⟩ is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.\footnote{In future releases further categories may be added.} It may be empty, too, but in such a case using \SetString is an error (but not \SetCase).

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

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
  \SetString{\monthiname}{Februar}
  \SetString{\monthiiiname}{M"{a}rz}
  \SetString{\monthivname}{April}
  \SetString{\monthvname}{Mai}
  \SetString{\monthviname}{Juni}
  \SetString{\monthviiname}{Juli}
  \SetString{\monthviiiname}{August}
When used in \texttt{ldf} files, previous values of \verb|⟨category⟩⟨language⟩| are overridden, which means the old way to define strings still works and used by default (to be precise, is first set to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if \verb|\date⟨language⟩| exists).

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

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

\begin{verbatim}
\AfterBabelCommands  {⟨code⟩}
  The code is delayed and executed at the global scope just after \texttt{EndBabelCommands}.
\end{verbatim}

\begin{verbatim}
\SetString  {⟨macro-name⟩}{⟨string⟩}
  Adds \verb|⟨macro-name⟩| to the current category, and defines globally \verb|⟨lang-macro-name⟩| to \verb|⟨code⟩| (after applying the transformation corresponding to the current charset or defined with the hook \texttt{stringprocess}). Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.
\end{verbatim}

\begin{verbatim}
\SetStringLoop  {⟨macro-name⟩}{⟨string-list⟩}
  A convenient way to define several ordered names at once. For example, to define \verb|\abmoniname|, \verb|\abmoniiname|, etc. (and similarly with \verb|\abday|):
\end{verbatim}

\begin{verbatim}
\SetStringLoop{abmon#1name}{en,fb,fr,ru,my,ar,jn,jl,ag,sp,oc,nl,dc}
\SetStringLoop{abday#1name}{lu,ma,mi,ju,vi,sa,do}
\end{verbatim}

\texttt{#1} is replaced by the roman numeral.
Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \let\BB\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A ⟨map-list⟩ is a series of macros using the internal format of \@uclclist (eg, \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 \TeX, we can set for Turkish:

\StartBabelCommands{turkish}{}[ot1enc, fontenc=OT1]
\SetCase
  {\uccode"10=`I\relax}
  {\lccode`I="10\relax}
\EndBabelCommands
\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetCase
  {\uccode`i=`İ\relax
   \uccode`ı=`I\relax
   \lccode`İ=`i\relax
   \lccode`I=`ı\relax
  }
\EndBabelCommands
\StartBabelCommands{turkish}{}
\SetCase
  {\uccode`i=9D\relax
   \uccode"19=`I\relax
   \lccode"9D=`i\relax
   \lccode`I=19\relax
  }
\EndBabelCommands
(Notethe mapping for OT1 is not complete.)

\SetHyphenMap{(to-lower-macros)}

New 3.9g Case mapping serves in \TeX 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 hyphenmap. So, even if internally they are based on the same \TeX primitive (\lccode), babel sets them separately. There are three helper macros to be used inside \SetHyphenMap:

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

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

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

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

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

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

4.1 Changes in 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 \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:

- \select@language did not set \languagename. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was german, a \select@language{spanish} had no effect.
- \foreignlanguage and other language* messed up \extras<language>. Scripts, encodings and many other things were not switched correctly.
- The :ENC mechanism for hyphenation patterns used the encoding of the previous language, not that of the language being selected.
- ' (with activeacute) had the original value when writing to an auxiliary file, and things like an infinite loop can happen. It worked incorrectly with ^ (if activated) and also if deactivated.
- Active chars where not reset at the end of language options, and that lead to incompatibilities between languages.
- \textormath raised an error with a conditional.
- \aliasshorthand didn’t work (or only in a few and very specific cases).
- \l@english was defined incorrectly (using \let instead of \chardef).
- ldf files not bundled with babel were not recognized when called as global options.

Part II

Source code

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 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 tow 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@afterf1, 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 babel1.def and in babel1.sty, which means in \LaTeX is executed twice, but we need them when defining options and babel1.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```latex
\bbl@trace{Basic macros}
\def\bbl@stripslash\expandafter\@gobble\string
\def\bbl@add#1#2{%\bl@ifunset\bbl@stripslash#1{\def#1{#2}}\{\expandafter\def\expandafter#1\expandafter{#1#2}}\def\bbl@xin@\expandafter\in@\def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}\def\bbl@cs\csname bbl@#1\endcsname
```

63
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.

\bbl@add@list
\bbl@afterelse
\bbl@afterfi
\bbl@exp
\bbl@trim

This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.
\bbl@ifunset

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 does not waste memory.

52 \begingroup
53 \def\bbl@ifunset#1{%
54 \expandafter\ifx\csname#1\endcsname\relax
55 \expandafter\@firstoftwo
56 \else
57 \expandafter\@secondoftwo
58 \fi
59 \bbl@ifunset{ifcsname}% TODO. A better test?
60 {}%
61 {%\def\bbl@ifunset#1{%
62 \ifcsname#1\endcsname
63 \expandafter\ifx\csname#1\endcsname\relax
64 \bbl@afterelse\expandafter\@firstoftwo
65 \else
66 \bbl@afterfi\expandafter\@secondoftwo
67 \fi
68 \else
69 \expandafter\@firstoftwo
70 \fi}%}
71 \endgroup

\bbl@ifblank

A tool from url, by Donald Arseneau, which tests if a string is empty or space. The companion macros tests if a macro is defined with some ‘real’ value, ie, not \relax and not empty,

72 \def\bbl@ifblank#1{%
73 \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
74 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil#4{#4}
75 \def\bbl@ifset#1#2#3{%
76 \bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{#1}}{#3}{#2}}

For each element in the comma separated \texttt{<key>=<value>} list, execute \texttt{<code>} with \texttt{#1} and \texttt{#2} as the key and the value of current item (trimmed). In addition, the item is passed verbatim as \texttt{#3}. With the \texttt{<key>} alone, it passes \texttt{\empty} (ie, the macro thus named, not an empty argument, which is what you get with \texttt{<key>=} and no value).

77 \def\bbl@forkv#1#2{%
78 \def\bbl@kvcmd##1##2##3{#2}%
79 \bbl@kvnext#1,\@nil,}
80 \def\bbl@kvnext#1,{%
81 \ifx\@nil#1\relax\else
82 \bbl@ifblank{#1}{}{\bbl@forkv@eq#1=#2=#3\@nil#4{#4}}%
83 \expandafter\bbl@kvnext
84 \fi}
85 \def\bbl@forkv@eq#1=#2=#3=\@nil#4{%
86 \bbl@trim@def\bbl@forkv@a{#1}%
87 \bbl@trim{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}{#2}{#4}}

An \texttt{\textit{for}} loop. Each item (trimmed), is \texttt{{#1}}. It cannot be nested (it's doable, but we don't need it).

88 \def\bbl@vforeach#1#2{%
89 \def\bbl@forcmd#1{#2}%
90 \bbl@fornext#1,\@nil,}
91 \def\bbl@fornext#1,{%
92 \ifx\@nil#1\relax\else
93 \bbl@ifblank{#1}{}{\bbl@vforeach\bbl@forcmd{#1}}%
94 \expandafter\bbl@fornext
95 \fi}
96 \def\bbl@vforeach#1\{\expandafter\bbl@vforeach\expandafter{#1}}
A somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.

Another hackish tool, to apply case changes inside a protected macros. It's based on the internal \let's made by \MakeUppercase and \MakeLowercase between things like \oe and \OE.

An alternative to \IfFormatAtLeastTF for old versions. Temporary.

The following adds some code to \extras... both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with #s. Used to deal with alph, Alph and frenchspacing when there are already changes (with \babel@save).
Some files identify themselves with a \LaTeX macro. The following code is placed before them to define (and then undefine) if not in \LaTeX.

```
\ifx\ProvidesFile\@undefined
\def\ProvidesFile#1[#2 #3 #4]{%
\wlog{File: #1 #4 #3 <#2>}%
\let\ProvidesFile\@undefined}
\fi
```

7.1 Multiple languages

\language

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. The following block is used in switch.def and hyphen.cfg; the latter may seem redundant, but remember babel doesn’t require loading switch.def in the format.

```
\ifx\language\@undefined
\csname newcount\endcsname\language
\fi
```

\last@language

Another counter is used to keep track of the allocated languages. \TeX and \LaTeX reserves for this purpose the count 19.

\addlanguage

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

```
\countdef\last@language=19
\def\addlanguage{\csname newlanguage\endcsname}\language
```

Now we make sure all required files are loaded. When the command \BeginDocument doesn’t exist we assume that we are dealing with a plain-based format. In that case the file plain.def is needed (which also defines \BeginDocument, 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)

```
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{babel}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩]
```

Start with some “private” debugging tool, and then define macros for errors.

```
\ifpackagewith{babel}{debug}
{\providecommand\bbl@trace[1]{\message{\^J[ #1 ]}}%}
\let\bbl@debug@firstofone
\ifx\directlua\@undefined\else
\directlua{ Babel = Babel or {} \directlua{Babel.debug = true}}%}
\input{babel-debug.tex}%
\fi
{\providecommand\bbl@trace{}%}
```

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. But first, include here the Basic macros defined above.

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.
7.3 base

The first ‘real’ option to be processed is base, which set the hyphenation patterns then resets \@babel.sty so that \LaTeX forgets about the first loading. After a subset of babel.def has been loaded (the old switch.def) and \AfterBabelLanguage defined, it exits. Now the base option. With it we can define (and load, with \luatex) hyphenation patterns, even if we are not interested in the rest of babel.

7.4 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 \BabelModifiers at \bbl@load@language; when no modifiers have been given, the former is \relax. How modifiers are handled are left to language styles; they can use \in@, loop them with \@for or load keyval, for example.
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.

Handling of package options is done in three passes. (I 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.

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

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.
7.5 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=....

The following macro tests if a shorthand is one of the allowed ones.

\def\bbl@ifshorthand#1{% 
  \ifx#1\string#1\else 
    \iffalse 
      \else 
        \string#1 
      \fi 
    \fi 
  \fi 
\}
\ifx\bbl@opt@shorthands@nnil
  \def\bbl@ifshorthand#1#2#3{% \else 
    \else
  \fi
\}
\def\bbl@ifshorthand#1#2#3#4{% 
  \def\bbl@ifshorthand#1#2#3{% 
  \else 
    \else
  \fi
\}

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

\def\bbl@opt@shorthands{%
  \expandafter\@firstoftwo 
  \else
    \expandafter\@secondoftwo 
  \fi
}
\def\bbl@opt@shorthands{%
  \expandafter\@firstoftwo 
  \else
    \expandafter\@secondoftwo 
  \fi
}
With headfoot=lang we can set the language used in heads/foots. For example, in babel/3796 just adds headfoot=english. It misuses \resetactivechars but seems to work.

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

Make sure the language set with main is the last one.

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

Interlude for Plain

Because of the way docstrip works, we need to insert some code for Plain here. However, the tools provided by the babel installer for literate programming makes this section a short interlude, because the actual code is below, tagged as \EmulateLaTeX.
That is all for the moment. Now follows some common stuff, for both Plain and \TeX. After it, we will resume the \TeX-only stuff.

441 ⟨/core⟩
442 ⟨+package | core⟩

8 Multiple languages

This is not a separate file (switch.def) anymore.
Plain \TeX 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.

443 \def\breadth{⟨⟨version⟩⟩}
444 \def\bbl@date{⟨⟨date⟩⟩}
445 ⟨⟨Define core switching macros⟩⟩

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

446 \def\addialect\#1\#2{%
447 \global\chardef\#1\#2\relax
448 \bbl@usehooks{addialect}{\#1}{\#2}
449 \begingroup
450 \count\#1\relax
451 \def\bbl@elt##1##2##3##4{%
452 \ifnum\count=##2\relax
453 \edef\bbl@tempa{\expandafter\@gobbletwo\string\#1}%
454 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
set to \expandafter\string\csname l@##1\endcsname\%
\string\language\the\count@}. Reported}%
455 \def\bbl@elt####1####2####3####4{}%
456 \bbl@cs{languages}%
457 \endgroup}
458 \bbl@iflanguage
459 \bbl@iflanguage\#1{%
460 \def\bbl@fixname\#1{%
461 \begingroup
462 \def\bbl@tempe{l@}%
463 \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe\#1}}%
464 \bbl@tempd
465 \lowercase\expandafter{\bbl@tempd}%
466 \@empty
467 \edef\bbl@tempd{\def\noexpand\#1{\#1}}%
468 \uppercase\expandafter{\bbl@tempd}%
469 \@empty
470 \edef\bbl@tempd{\def\noexpand\#1{\#1}}%
471 \lowercase\expandafter{\bbl@tempd}%
472 \@empty
473 \edef\bbl@tempd{\endgroup\def\noexpand\#1{\#1}}%
474 \@empty
475 \bbl@tempd
476 \bbl@exp{\bbl@usehooks{\#1}}
477 \bbl@iflanguage\#1{%
478 \@ifundefined{\#1}{\@nolanerr{\#1}\@gobble}\@firstofone}
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-latn-ba becomes fr-Latn-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}{}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}% 
    \ifx\bbl@bcp\relax 
      \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}% 
    \fi 
  \else 
    \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb 
    \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc 
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}{}{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}% 
    \ifx\bbl@bcp\relax 
      \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}{\let\bbl@bcp{\bbl@tempa-\bbl@tempc}}{}% 
    \fi 
  \fi 
}\def\bbl@fnset{\bbl@bcpmap@\languagename}{% Move uplevel??
  \@nameuse{bbl@bcpmap@\languagename}}
\ifbbl@bcpallowed
  \expandafter\ifx\csname date\languagename\endcsname\relax
    \bbl@bcplookup\languagename-@empty-@empty-@empty\@@
    \ifx\bbl@bcp\relax
      \edef\languagename{\bbl@bcp@prefix\bbl@bcp}\
      \edef\localename{\bbl@bcp@prefix\bbl@bcp}\
      \ifx\csname date\languagename\endcsname\relax
        \let\bbl@initoload\bbl@bcp
        \bbl@exp{\bbl@autoload@bcpoptions}{\languagename}\
        \let\bbl@initoload\relax
      \fi
      \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}\
    \fi
  \fi
\fi
\expandafter\ifx\csname date\languagename\endcsname\relax
  \IfFileExists{babel-\languagename.tex}
  \bbl@exp{\bbl@autoload@options}{\languagename}
  {}\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.

8.1 Selecting the language

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.

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.

The following definition is preserved for backwards compatibility (eg, arabi, koma). It is related to a
trick for 2.09, now discarded.

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 \protect\asftergroup mechanism to help us. The command
\asftergroup stores the token immediately following it to be executed when the current group is
closed. So we define a temporary control sequence \bbl@poplanguage to be executed at the end of
the group. It calls \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 \bbl@language@stack and initially empty.

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@language@stack{}
```

```
\def\bbl@push@language{\ifx\languagename\@undefined\else\ifx\currentgrouplevel\@undefined\xdef\bbl@language@stack{\languagename+\bbl@language@stack}\else\ifnum\currentgrouplevel=\z@\xdef\bbl@language@stack{\languagename+}\else\xdef\bbl@language@stack{\languagename+\bbl@language@stack}\fi\fi\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 \languagename. For this we first define a helper function.

```
\def\bbl@pop@lang#1+#2\@@{\edef\languagename{#1}\xdef\bbl@language@stack{#2}}
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before \bbl@pop@lang is executed \TeXX first expands the stack, stored in \bbl@language@stack. The result of that is that the argument string of \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).

```
\let\bbl@ifrestoring\@secondoftwo
\def\bbl@pop@language{\expandafter\bbl@pop@lang\bbl@language@stack\@@\let\bbl@ifrestoring\@firstoftwo\expandafter\bbl@set@language\expandafter{\languagename}}
```

Once the name of the previous language is retrieved from the stack, it is fed to \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 \localeid. This means \l@... will be reserved for hyphenation patterns (so that two locales can share the same rules).

```
\chardef\localeid\z@
\def\bbl@id@last{0} % No real need for a new counter
\def\bbl@id@assign{\ifunset{bbl@id@@\languagename}{\count@\bbl@id@last\relax\advance\count@\@ne\bbl@csarg\chardef{id@@\languagename}\count@\edef\bbl@id@last{\the\count@}\ifcase\bbl@engine\else
```

77
\directlua{
Babel = Babel or {};
Babel.locale_props = Babel.locale_props or {};
Babel.locale_props[bbl@id@last] = {};
Babel.locale_props[bbl@id@last].name = '');
%
}%
}%
}
\chardef\localeid=bbl@cl{id@}

The unprotected part of selectlanguage.

\expandafter\def\csname selectlanguage \endcsname#1{%
\ifnum\bbl@hymapsel=@cclv\let\bbl@hymapsel=tw\fi
\bbl@push@language
\aftergroup\bbl@pop@language
\bbl@set@language{#1}
First, check if the user asks for a known language. If so, update the value of \languages and call \originalTeX to bring \TeX in a certain pre-defined state. The name of the language is stored in the control sequence \languages.

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 \select@language, 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 \langle lang \rangle hyphenmins is defined. If it is not, we set default values (2 and 3), otherwise the values in \langle lang \rangle hyphenmins will be used.
% restore\originalTeX\expandafter\def\expandafter\originalTeX\expandafter{%csname noextras\endcsname\let\originalTeX\empty\bbl@beginsave}%\bbl@usehooks{afterreset}{%\languageshorthands{none}%\% set the locale id\bbl@id@assign%\% switch captions, date\% No text is supposed to be added here, so we remove any\% spurious spaces.\bbl@bsphack%\ifcase\bbl@select@type\csname captions\endcsname\relax\csname date\endcsname\relax\else\bbl@xin@{,captions,}{,\bbl@select@opts,}%\ifin@\csname captions\endcsname\relax\fi\bbl@xin@{,date,}{,\bbl@select@opts,}%\ifin@ % if \foreign... within \lang\csname date\endcsname\relax\fi% switch extras\bbl@usehooks{beforeextras}{}%\csname extras\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\csname\languagename @bbl@hyphenmap\endcsname\fi\chardef\bbl@opt@hyphenmap\z@\else\ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else\csname\languagename @bbl@hyphenmap\endcsname\fi% linebreaking - handle u, e, k (v in the future)\bbl@xin@{/u}{/\bbl@tempa}%\ifin@% elongated forms\bbl@xin@{/e}{/\bbl@tempa}\fi% only kashida\bbl@esphack%\% switch extras\bbl@usehooks{beforeextras}{%\csname extras\endcsname\relax\bbl@usehooks{afterextras}{%% > babel-ensure% > babel-sh-<short>% > babel-bidi% > babel-fontspec% hyphenation - select rules\ifnum\csname l@\languagename\endcsname=\l@unhyphenated\edef\bbl@tempa{u}%\else\edef\bbl@tempa{\bbl@cl{lnbrk}}%\fi% linebreaking - handle u, e, k (v in the future)\bbl@xin@{/u}{/\bbl@tempa}\ifin@% elongated forms\bbl@xin@{/e}{/\bbl@tempa}\fi% only kashida
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.

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.

\begin{verbatim}
\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}% Now in horizontal mode!
  \endgroup}
\def\bbl@foreign@s#1#2{%
  \begingroup
  \let\bbl@select@opts\@empty
  \let\BabelText\@firstofone
  \foreignlanguage{#1}%
  \bbl@usehooks{foreign*}{}%\bbl@dirparastext
  \BabelText{#2}% Still in vertical mode!
  \endgroup}
\end{verbatim}

\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.

\begin{verbatim}
\def\foreignlanguage#1{%
  % set name
  \edef\languagename{#1}%
  \ifbbl@usedategroup
    \bbl@add\bbl@select@opts{,date,}%
    \bbl@usedategroupfalse
  \fi
  \bbl@fixname\languagename
  % TODO - \shapemode, \setpar, ?@@par
  \begingroup
  \let\bbl@select@opts\empty
  \let\BabelText\@firstofone
  \foreignlanguage{#1}%
  \bbl@usehooks{foreign}{}%
  \bbl@dirparastext
  \BabelText{#2}% Still in vertical mode!
  \endgroup
}\end{verbatim}
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. I'll proceed, but expect wrong results.

\bbl@patterns

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 is 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.

\def\hyphenrules#1{\edef\bbl@tempf{#1}\bbl@fixname\bbl@tempf\bbl@iflanguage\bbl@tempf{\expandafter\bbl@patterns\expandafter{\bbl@tempf}\ifx\languageshorthands\@undefined\else\@expandtwoargs\bbl@usehooks{patterns}\{#1}{\bbl@tempa}\%\@undefined\fi\begin{group}
\bbl@xin{\number\language,}{,\bbl@hyphlist}\%\@ifundefined{bbl@hyphenation@}\{\bbl@tempa}\%\@.expandtwoargs\bbl@usehooks{hyphenation}\{#1\}{\bbl@tempa}\%\hyphenation\%\bbl@hyphenation@\@ifundefined{bbl@hyphenation@\#1}\{\space\csname bbl@hyphenation@\#1\endcsname\}\empty\xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}\%\fi\end{group}}

\hyphenrules 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 \otherlanguage*.
\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.

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

\ProvidesLanguage The identification code for each file is something that was introduced in \LaTeX{} 2ε. 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.

\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.

\providecommand\setlocale{%
\bbl@error
\{Not yet available\}
\{Find an armchair, sit down and wait\}
8.2 Errors

`\@nolanerr` The babel package will signal an error when a document 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 \LaTeX, 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.

Here ended the now discarded switch.def. Here also (currently) ends the base option.
\addto It takes two arguments, a \emph{control sequence} and TeX-code to be added to the \emph{control sequence}. If the \emph{control sequence} has not been defined before it is defined now. The control sequence could also expand to \verb|\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.

\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 \verb|\macro| will be saved new control sequences named \verb|\org@macro|.
This version of \bbl@redefine can be used to redefine \long commands such as \ifthenelse.

For commands that are redefined, but which might be robust we need a slightly more intelligent macro. A robust command \texttt{foo} is defined to expand to \protect\texttt{foo}. So it is necessary to check whether \texttt{\textbackslash 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 \texttt{\textbackslash foo}.

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).

\begin{itemize}
\item \texttt{\textbackslash evargs}, \texttt{%} <- don't delete this comma
\item everylanguage=1, loadkernel=1, loadpatterns=1, loadexceptions=1, %
\end{itemize}
\babelensure

The user command just parses the optional argument and creates a new macro \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{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}, which in in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \in@) those in the exclude list. If the \fontencoding 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.

\bbl@trace{Defining babelensure}
\newcommand\babelensure[2][2]{% TODO - revise test files
\AddBabelHook{babel-ensure}{afterextras}{% 
  \ifcase\bbl@select@type 
  1\bbl@cl{}% 
  \fi}%
\begingroup
  \let\bbl@ens@include\@empty
  \let\bbl@ens@exclude\@empty
  \def\bbl@ens@fontenc{\relax}%
  \def\bbl@tempb##1{% \elt for (excluding) \bbl@captionslist list
    \ifx##1\@undefined % 3.32 - Don't assume the macro exists
      \edef##1{\noexpand\bbl@nocaption{\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
    \fi
    \ifx##1\@empty\else
      \in@{##1}{#2}%
      \ifin@% 
        \bbl@ifunset{bbl@ensure@\languagename}{}% 
      \fi
      \bbl@exp{% 
        \\DeclareRobustCommand\bbl@ensure@\languagename[1]{% \foreignlanguage{\languagename}{% 
          \ifx\relax#3\else \fontencoding{#3}\selectfont %
        \fi
        #1}}}%
    \fi
  \fi}
\bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
\def\bbl@tempc{\bbl@ensure}%
\expandafter\bbl@add\expandafter\bbl@tempc\expandafter{\expandafter{\bbl@ens@include}}%
\expandafter\bbl@add\expandafter\bbl@tempc\expandafter{\expandafter{\bbl@ens@exclude}}%
\toks@\expandafter{\bbl@tempc}%
\bbl@exp{% 
  \\begin{the\bbl@e@#2}\{\the\toks@\{\bbl@ens@include\fontenc\}\}\end{the\bbl@e@}
}\def\bbl@ensure@#2#3{% % 1: include 2: exclude 3: fontenc
  \def\bbl@tempb##1{%
    \ifx##1\@undefined % 3.32 - Don't assume the macro exists
      \edef##1{\noexpand\bbl@nocaption{\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
    \fi
    \ifx##1\@empty\else
      \in@{##1}{#2}%
      \ifin@% 
        \bbl@ifunset{bbl@ensure@\languagename}{}% 
      \fi
      \bbl@exp{% 
        \\DeclareRobustCommand\bbl@ensure@\languagename[1]{% \foreignlanguage{\languagename}{% 
          \ifx\relax#3\else \fontencoding{#3}\selectfont %
        \fi
        #1}}%}
8.4 Setting up language files

\LdfInit \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{%
\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`\@=17\relax

This macro interrupts the processing of a language definition file.
\ldef@quit
\ldef@finish
This macro takes one argument. It is the name of the language that was defined in the language definition file.

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

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.
A bit of optimization. Select in heads/foots the language only if necessary.

\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 \TeX is used. It is used only at one place, namely when \initiate@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 \fss@catcodes, added in 3.10.

\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.

8.5 Shorthands

\bbl@add@special

\bbl@remove@special

91
\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' being the character to be made active). Later its definition can be changed to expand to \active@char{char} by calling \bbl@activate{\char{char}}.

For example, to make the double quote character active one could have \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).

\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 
        \fi} 
    \long\@namedef{#3@arg#1}##1{% 
        \expandafter\ifx\csname#2@sh@#1@string##1@endcsname\relax 
        \bbl@afterelse\l@sh@select#2#1{#3@arg#1}{#4#1}% 
        \else 
        \bbl@afterfi\csname#2@sh@#1@string##1@endcsname 
        \fi}%

\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.

\def\bbl@deactivate{\active@prefix \normal@char}
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 \texttt{normal@char\langle\texttt{char}\rangle} to expand to the character in its default state. If the character is mathematically active when \texttt{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 \textit{a posteriori}).

To prevent problems with the loading of other packages after \texttt{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 \texttt{KeepShorthandsActive}). It is re-activate again at \texttt{\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 \texttt{\bibitem} for example. Then we make it active (not strictly necessary, but done for backward compatibility).

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

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.

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (') active we need to change \text@math@prime 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.

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

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.
\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 \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 \bbl@firstcs or \bbl@scndcs. Hence two more arguments need to follow it.

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

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

\begin{verbatim}
\begingroup\if@safe@actives\newif\if@safe@actives\@safe@activesfalse\{\gdef\active@prefix#1{%\if\protect\@typeset@protect\else\if\protect\@unexpandable@protect\noexpand#1%\else\protect#1%\fi\expandafter\@gobble\fi\expandafter\expandafter\expandafter\@gobble\fi\gdef\active@prefix#1{%\ifincsname\string#1\expandafter\@gobble\else\if\protect\@typeset@protect\else\if\protect\@unexpandable@protect\noexpand#1\else\protect#1\fi\expandafter\expandafter\expandafter\@gobble\fi\fi\gdef\active@prefix#1{%\ifincsname\string#1\expandafter\@gobble\else\if\protect\@typeset@protect\else\if\protect\@unexpandable@protect\noexpand#1\else\protect#1\fi\expandafter\expandafter\expandafter\@gobble\fi\fi\endgroup\else\fi}\fi}
\end{verbatim}

\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 @safe@actives is available. The setting of this switch should be checked in the first level expansion of \active@char \langle char \rangle.
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.

\def\bbl@restore@actives{\if\safe@actives\safe@activesfalse\fi}

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.

\chardef\bbl@activated\z@
\def\bbl@activate#1{\chardef\bbl@activated\@ne\bbl@withactive{\expandafter\let\expandafter}#1\csname bbl@active@\string#1\endcsname}
\def\bbl@deactivate#1{\chardef\bbl@activated\tw@\bbl@withactive{\expandafter\let\expandafter}#1\csname bbl@normal@\string#1\endcsname}

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

\def\bbl@firstcs#1#2{\csname#1\endcsname}
\def\bbl@scndcs#1#2{\csname#2\endcsname}

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.

The auxiliary macro \babel@texpdf improves the interoperativity with hyperref and takes 4 arguments: (1) The \TeX code in text mode, (2) the string for hyperref, (3) the \TeX code in math mode, and (4), which is currently ignored, but it's meant for a string in math mode, like a minus sign instead of an hyphen (currently hyperref doesn't discriminate the mode). This macro may be used in \Idf files.

\def\babel@texpdf#1#2#3#4{\ifx\texorpdfstring\@undefined\textormath{#1}{#3}\else\texorpdfstring{\textormath{#1}{#3}}{#2}\fi}
\def\declare@shorthand#1#2#3#4\@nil#4{\def\bbl@tempa{#3}\ifx\bbl@tempa\@empty\expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs\bbl@ifunset{#1@sh@\string#2@}{}\else\def\bbl@tempa{#4}\expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa\else\bbl@info{Redefining #1 shorthand \string#2\% in language \CurrentOption}\fi\@namedef{#1@sh@\string#2@}{#4}\else\expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs\enddef\enddef\enddef\enddef
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.

\textormath
\textormath{\ifmmode\expandafter\@secondoftwo\else\expandafter\@firstoftwo\fi}

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'.

\user@group
\language@group
\system@group

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.

\useshorthands
\useshorthands{\ifstar\bbl@usesh@s{\bbl@usesh@x\{}\}}

Currently we only support two groups of user level shorthands, named internally user and user@<lang> (language-dependent user shorthands). By default, only the first one is taken into account, but if the former is also used (in the optional argument of \defineshorthand) a new level is inserted for it (user@generic, done by \bbl@set@user@generic); we make also sure {} and \protect are taken into account in this new top level.
\languageshorthands 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 to fix it in the same way languages names are fixed. [TODO].

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

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

\shorthandoff 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 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.

```latex
\def\bbl@switch@sh#1#2{\%\n  \ifx#2\@nnil\else\n    \bbl@ifunset{bbl@active@\string#2}\%\n      \{I can’t switch \string#2 on or off--not a shorthand\}%\n      \{This character is not a shorthand. Maybe you made\}\n        a typing mistake? I will ignore your instruction.\}%\n    \ifcase#1\% off, on, off*\n      \catcode\string#2=12\relax\n    \or\n      \catcode\string#2=\active\n    \bbl@ifunset{bbl@shdef@\string#2}\%\n      \bbl@withactive{\expandafter\let\expandafter}#2\%\n      \csname bbl@shdef@\string#2\endcsname\n    \bbl@activate{#2}\%\n    \else\n      \bbl@deactivate{#2}\%\n    \fi\n  \or\n  \bbl@ifunset{bbl@shdef@\string#2}\%\n    \bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2\%\n    \csname bbl@oricat@\string#2\endcsname\n    \csname bbl@oridef@\string#2\endcsname\n  \fi\n}\bbl@afterfi\bbl@switch@sh#1\%\n\}
```

Note the value is that at the expansion time; eg, in the preamble shorthands are usually deactivated.

```latex
\def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}\n\def\bbl@putsh#1{\%\n  \bbl@ifunset{bbl@active@\string#1}\%\n    \{\bbl@withactive{\bbl@csarg\let{shdef@\string#1}}\%\n      \{\csname bbl@active@\string#1\endcsname\}%\n    \def\bbl@putsh@i#1#2\@nnil{\%\n      \csname\language@group @sh@\string#1\endcsname\n      \csname\language@group @sh@\string#1\endcsname\n    \fi\%\n    \bbl@afterfi\bbl@putsh@i\string#1\@empty\@nnil\%\n    \csname\language@group @sh@\string#1\endcsname\n  }\%\n  \let\bbl@s@initiate@active@char\initiate@active@char\n  \def\initiate@active@char#1{%\n    \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}\n  \let\bbl@s@switch@sh\bbl@switch@sh\%\n  \def\bbl@switch@sh#1{#2\%\n    \ifx\empty\@empty\else\string#2@\fi\endcsname\n  }\%\n  \let\bbl@s@activate\bbl@activate\%\n  \def\bbl@activate#1{%\n    \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}\n  }\%\n  \let\bbl@s@initiate@active@char\initiate@active@char\n  \def\initiate@active@char#1{%\n    \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}\n  \let\bbl@s@switch@sh\bbl@switch@sh\%\n  \def\bbl@switch@sh#1{#2\%\n    \ifx\empty\@empty\else\string#2@\fi\endcsname\n  }\%\n  \let\bbl@s@activate\bbl@activate\%\n  \def\bbl@activate#1{%\n    \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}\n  }\%\n```

99
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}}

One of the internal macros that are involved in substituting \textprime for each right quote in mathmode is \texttt{\bbl@prime@}. 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@prime@{\prime\futurelet\@let@token\bbl@prim@s}

\def\bbl@if@primes#1#2{\ifx#1\@let@token\expandafter\@firstoftwo\else\ifx#2\@let@token\bbl@afterelse\expandafter\@firstoftwo\else\bbl@afterfi\expandafter\@secondoftwo\fi\fi}

\begingroup
\catcode`\^=7 \catcode`*=8 \lccode`*=\^ \catcode`\'=12 \catcode`"=12 \lccode`"=\'
\lowercase{\gdef\bbl@prim@s{\bbl@if@primes\''\pr@@s{\bbl@if@primes\^\pr@@t\egroup}}}\endgroup

The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \texttt{\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 \texttt{\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

8.6 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.

\begin{verbatim}
\bbl@trace{Language attributes}
\newcommand\languageattribute[2]{%
def\bbl@tempc{#1}%
\bbl@fixname\bbl@tempc%
\bbl@iflanguage\bbl@tempc{%
\bbl@vforeach{#2}{%
Wewanttomakesurethateachattributeisselectedonlyonce;thereforewestorethealready
selectedattributesin \bbl@known@attribs. Whenthatcontrolsequenceisnotyetdefinedthis
attributeiscertainlynotselectedbefore.

\ifx\bbl@known@attribs\@undefined
\in@false
\else
\bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
\fi
\ifin@
\bbl@warning{ You have more than once selected the attribute '##1' for language #1. Reported}%
\else
Whenweendupheretheattributeisnotselectedbefore. So,weaddittothelistofselected
attributesandexecutetheassociated\TeX-code.
\bbl@exp{%\bbl@add@list\bbl@known@attribs{\bbl@tempc-##1}}%
\edef\bbl@tempa{\bbl@tempc-##1}%
\expandafter\bbl@ifknown@ttribute\expandafter{\bbl@tempa}\bbl@attributes%
{\csname\bbl@tempc @attr@##1\endcsname}%
{\@attrerr{\bbl@tempc}{##1}}%
\fi}}%
\@onlypreamble\languageattribute
\end{verbatim}

The error text to be issued when an unknown attribute is selected.

\begin{verbatim}
\newcommand*{\@attrerr}[2]{%
\bbl@error
{The attribute #2 is unknown for language #1.}%
{Your command will be ignored, type \textless return\textgreater to proceed}}
\end{verbatim}

\bbl@declare@ttribute 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 \extras... for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \begin{document}.

\begin{verbatim}
\def\bbl@declare@ttribute#1#2#3{%
\bbl@xin@{,\bbl@known@attribs{\bbl@tempc-##1},}%
\edef\bbl@tempa{\bbl@tempc-##1}%
\expandafter\bbl@ifknown@ttribute\expandafter{\bbl@tempa}\bbl@attributes%
{\csname\bbl@tempc @attr@#2\endcsname}%
{\@attrerr{\bbl@tempc}{#2}}%
\fi}}%
\@onlypreamble\languageattribute
\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 \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.

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

\bbl@ifknown@trib 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.

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

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

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

\def\bbl@clear@trib#1-#2.{%\expandafter\let\csname#1@attr@#2\endcsname\@undefined}

\AtBeginDocument{\bbl@clear@tribs}

8.7 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 \original\TeX). Note undefined macros are not undefined any more when saved – they are relaxed.

\def\babel@beginsavecnt
\def\babel@beginsave

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

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

\newcounter{\bbl@savecnt}
\begin{document}
\bbl@beginsave

102
The macro \texttt{\babel@save\{csname\}} saves the current meaning of the control sequence \texttt{\{csname\}} to the original \TeX\textsuperscript{31}. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to the original \TeX\ and the counter is incremented. The macro \texttt{\babel@savevariable\{variable\}} saves the value of the variable. \texttt{\{variable\}} can be anything allowed after the \texttt{\the} primitive.

\begin{verbatim}
def\babel@save#1{% 
  \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax 
  \toks0=\originalTeX\let#1={} 
  \bbl@exp{% 
    \def\the\originalTeX{\the\toks0\langle\babel@\number\babel@savecnt\rangle}\relax
  }% 
  \advance\babel@savecnt\@ne
} 
\def\babel@savevariable#1{% 
  \toks0=\originalTeX#1= 
  \bbl@exp{% 
    \def\the\originalTeX{\the\toks0\the#1}\relax
  }
}
def\bbl@frenchspacing{% 
  \ifnum\the\sfcode`\relax=\z@ 
    \let\bbl@nonfrenchspacing\relax 
  \else 
    \frenchspacing 
    \let\bbl@nonfrenchspacing\nonfrenchspacing 
  \fi 
  \bbl@elt\relax
} 
\let\bbl@elt\relax
\edef\bbl@fs@chars{% 
  \bbl@elt{.}\@m{3000}\bbl@elt{?}\@m{3000}\bbl@elt{!}\@m{3000}\bbl@elt{;}\@m{1500}\bbl@elt{,}\@m{1250}}
\def\bbl@pre@fs{% 
  \def\bbl@elt##1##2##3{% 
    \ifnum\sfcode`##1=##2\relax 
      \babel@savevariable{\sfcode`##1}% 
      \sfcode`##1=##3\relax 
    \fi} 
  \bbl@fs@chars
} 
\def\bbl@post@fs{% 
  \bbl@fs@chars 
  \edef\bbl@tempa{\bbl@cl{frspc}}% 
  \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}% 
  \if\bbl@tempa % do nothing 
  \else\if\n\bbl@tempa % non french 
    \def\bbl@elt##1##2##3{% 
      \ifnum\sfcode`##1=##2\relax 
        \babel@savevariable{\sfcode`##1}% 
        \sfcode`##1=##3\relax 
      \fi}% 
    \bbl@fs@chars
  \else\if\y\bbl@tempa % french 
    \def\bbl@elt##1##2##3{% 
      \ifnum\sfcode`##1=\relax 
        \babel@savevariable{\sfcode`##1}% 
        \sfcode`##1=##3\relax 
      \fi}% 
    \bbl@fs@chars
  \fi\fi\fi}
\end{verbatim}

Some languages need to have \texttt{\frenchspacing} in effect. Others don't want that. The command \texttt{\bbl@frenchspacing} switches it on when it's not already in effect and \texttt{\bbl@nonfrenchspacing} switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in \texttt{\babelprovide}. This new method should be ideally the default one.

\begin{verbatim}
\def\bbl@frenchspacing{% 
  \ifnum\the\sfcode`\relax=\z@ 
    \let\bbl@nonfrenchspacing\relax 
  \else 
    \frenchspacing 
    \let\bbl@nonfrenchspacing\nonfrenchspacing 
  \fi 
  \bbl@elt\relax
}
\let\bbl@elt\relax
\edef\bbl@save@sfcodes{% 
  \bbl@elt{.}\@m{3000}\bbl@elt{?}\@m{3000}\bbl@elt{!}\@m{3000}\bbl@elt{;}\@m{1500}\bbl@elt{,}\@m{1250}}
\def\bbl@pre@fs{% 
  \def\bbl@elt##1##2##3{% 
    \ifnum\sfcode`##1=##2\relax 
      \babel@savevariable{\sfcode`##1}% 
      \sfcode`##1=##3\relax 
    \fi} 
  \bbl@fs@chars
} 
\def\bbl@post@fs{% 
  \bbl@fs@chars 
  \edef\bbl@tempa{\bbl@cl{frspc}}% 
  \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}% 
  \if\bbl@tempa % do nothing 
  \else\if\n\bbl@tempa % non french 
    \def\bbl@elt##1##2##3{% 
      \ifnum\sfcode`##1=##2\relax 
        \babel@savevariable{\sfcode`##1}% 
        \sfcode`##1=##3\relax 
      \fi}% 
    \bbl@fs@chars
  \else\if\y\bbl@tempa % french 
    \def\bbl@elt##1##2##3{% 
      \ifnum\sfcode`##1=\relax 
        \babel@savevariable{\sfcode`##1}% 
        \sfcode`##1=##3\relax 
      \fi}% 
    \bbl@fs@chars
  \fi\fi\fi}
\end{verbatim}

\textsuperscript{31}Original \TeX\ has to be expandable, i.e. you shouldn't let it to \texttt{\relax}. 

103
8.8 Short tags

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

\bbl@trace{Short tags}
\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}}}
    \bbl@tempc}%
  \bbl@for\bbl@tempa\bbl@tempa{% 
    \expandafter\bbl@tempb\bbl@tempa\@@}}}%

8.9 Hyphens

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

\bbl@trace{Hyphens}
\@onlypreamble\babelhyphenation
\AtEndOfPackage{% 
  \newcommand\babelhyphenation[2][\@empty]{% 
    \ifx\bbl@hyphenation@\relax 
    \let\bbl@hyphenation@\@empty 
    \fi 
    \ifx\@empty#1% 
    \protected@edef\bbl@hyphenation@{\bbl@hyphenation@#2}% 
    \else 
    \bbl@vforeach{#1}{% 
      \def\bbl@tempa{##1}%
      \bbl@fixname\bbl@tempa 
      \bbl@iflanguage\bbl@tempa{%
        \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{\bbl@ifunset{bbl@hyphenation@\bbl@tempa}{}{\csname bbl@hyphenation@\bbl@tempa\endcsname}#2}}%
    }% 
  }% 
  \bbl@warning{You must not intermingle \string\selectlanguage\space and\% \string\babelhyphenation\space or some exceptions will not\% be taken into account. Reported}%
  \fi 
  \ifx\@empty#1% 
  \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\@empty}% 
  \else 
  \bbl@vforeach{#1}{% 
    \def\bbl@tempa{##1}%
    \bbl@fixname\bbl@tempa 
    \bbl@iflanguage\bbl@tempa{%
      \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{\bbl@ifunset{bbl@hyphenation@\bbl@tempa}{}{\csname bbl@hyphenation@\bbl@tempa\endcsname}#2}}%
  }% 
  \fi}}%

\bbl@allowhyphens  This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hskip Opt plus Opt\footnote{\LaTeX\ begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.}. 

\def\bbl@allowhyphens{\ifmode\else\nobreak\hskip2@skip\fi}
Macros to insert common hyphens. Note the space before @ in \babelhyphen. Instead of protecting it with \DeclareRobustCommand, which could insert a \relax, we use the same procedure as shorthands, with \active@prefix.

The following two commands are used to wrap the “hyphen” and set the behavior of the rest of the word—the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like “-suffix”. \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

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 \bbox@hy@nobreak is redundant.

For some languages the macro \bbox@disc is used to ease the insertion of discretionaries for letters that behave ‘abnormally’ at a breakpoint.
8.10 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.

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

The second one. We need to patch \@ucclist, 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 \@ucclist 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 \lang@bbl@uc1c 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:

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

and starts over (and similarly when lowercasing).

```
@ifpackagewith{babel}{nocase}%
  \let\bbl@patchuc1c@relax%
  \define\bbl@patchuc1c{
    \global\let\bbl@patchuc1c@relax
    \g@addto@macro\@ucclist{\reserved@b{\reserved@b\bbl@uc1c}}%
    \edef\bbl@uc1c@%\let\bbl@encoded\bbl@encoded@uc1c
    \bbl@ifunset{\languagename @bbl@uc1c}% and resumes it
      {##1}%
      {\let\bbl@tempa##1% Used by LANG@bbl@uc1c
        \csname\languagename @bbl@uc1c\endcsname}%
      {\lettobl@uc1c\@empty\bbl@toupper@empty}%
    \edef\bbl@toclient{\csname\languagename @bbl@uc\endcsname}%
    \lettobl@toupper{\csname\languagename @bbl@uc\endcsname}}%
```

The following package options control the behavior of \SetString.

```
\DeclareOption{nocase}{%
  \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}}%
\DeclareOption{nocase}{%
  \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}}%
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.

\onlypreamble\StartBabelCommands
\def\StartBabelCommands{%
\begingroup
\bbl@recatcode{1}%
⟨⟨Macros local to BabelCommands⟩⟩
\def\bbl@provstring##1##2{%
\providecommand##1{##2}%
\bbl@toglobal##1}
\global\let\bbl@scafter\@empty
\let\StartBabelCommands\bbl@startcmds
\ifx\BabelLanguages\relax
\let\BabelLanguages\CurrentOption
\fi
\begingroup
\let\bbl@screset\@nnil % local flag - disable 1st stopcommands
\StartBabelCommands}
\def\bbl@startcmds{%
\ifx\bbl@screset\@nnil\else
\bbl@usehooks{stopcommands}{}%
\fi
\fi
\endgroup
\begingroup
\@ifstar
{\ifx\bbl@opt@strings\@nnil
\let\bbl@opt@strings\BabelStringsDefault
\fi
\bbl@startcmds@i}%
\bbl@startcmds@i}
\def\bbl@startcmds@ii#1#2{%
\edef\bbl@L{\zap@space#1 \@empty}%
\edef\bbl@G{\zap@space#2 \@empty}%
\bbl@startcmds@ii}
\let\bbl@startcommands\StartBabelCommands

Parse the encoding info to get the label, input, and font parts.
Select the behavior of \SetString. There are two main cases, depending on 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.
\newcommand\bbl@startcmds@ii[1][\@empty]%
\let\SetString\gobbletwo
\let\bbl@stringdef\gobbletwo
\let\AfterBabelCommands\gobble
\ifx\empty\@empty%}
\def\bbl@sc@label{generic}%
\def\bbl@encstring##1##2{%
\ProvideTextCommandDefault##1{##2}%
\bbl@toglobal##1%
\expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
\let\bbl@sctest\in@true
\else
\let\bbl@sc@charset\space % <- zapped below

107
There are two versions of \bbl@scswitch. The first version is used when \ldf\texttt{sl} are read, and it makes sure \textbackslash \{\texttt{group}\}/\textbackslash \{\texttt{language}\} 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 \bbl\texttt{sl} and does nothing. The macro \bbl@forlang loops over all languages but its body is executed only if the value is in \texttt{BabelLanguages} (inside babel) or \texttt{\date\{language\}} is defined (after babel has been loaded). There are also two versions of \bbl@forlang. The first one skips the current iteration if the language is not in \texttt{BabelLanguages} (used in \ldf\texttt{sl}), 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 \providescommand). 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{% eg, \prefacename{<string>}
\bbl@forlang\bbl@tempa{% 
\edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}% 
\bbl@ifunset{\bbl@LC}% eg, \germanchaptername
{\bbl@exp{% 
\global\\bbl@add{\bbl@GL\bbl@tempa}{\bbl@scset\#1}{\bbl@LC}{}}}%
}% 
\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.
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.

\begin{verbatim}
\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{\romannumeral\count@}{\the\toks@}% 
\count@=\the\count@\relax}}%}
\end{verbatim}

\begin{verbatim}
\def\bbl@aftercmds#1{% 
\toks@\expandafter{\bbl@scafter#1}% 
\xdef\bbl@scafter{\the\toks@}
\end{verbatim}

Case mapping

The command \SetCase provides a way to change the behavior of \MakeUppercase and \MakeLowercase. \bbl@tempa is set by the patched \@ucclist to the parsing command.

\begin{verbatim}
\newcommand\SetCase[3][]{% 
\bbl@patchuc\lccode\bbl@tempa{% 
\forlang\bbl@tempa{% 
\expandafter\bbl@encstring 
\csname\bbl@tempa@bbl@uc\endcsname{##1}% 
\expandafter\bbl@encstring 
\csname\bbl@tempa@bbl@lc\endcsname{##2}% 
\expandafter\bbl@encstring 
\csname\bbl@tempa@bbl@hyphenmap\endcsname{##3}}}%
\end{verbatim}

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.

\begin{verbatim}
\newcommand\SetHyphenMap[1][]{% 
\forlang\bbl@tempa{% 
\expandafter\bbl@stringdef 
\csname\bbl@tempa@bbl@hyphenmap\endcsname{#1}}}%
\end{verbatim}

There are 3 helper macros which do most of the work for you.

\begin{verbatim}
\def\BabelLower[2]{% one to one. 
\ifnum\lccode#1=#2\else 
\babel@savevariable{\lccode#1}% 
\lccode#1=#2\relax 
\fi}
\end{verbatim}
The following package options control the behavior of hyphenation mapping.

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

This sections ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.
8.11 Macros common to a number of languages

\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.

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

8.12 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.

8.12.1 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{\setlow@box{\textquotedblright}/}
\box\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}{%\UseTextSymbol{OT1}{\quotedblbase}}
\quotesinglbase
We also need the single quote character at the baseline.

\ProvideTextCommand{\quotesinglbase}{OT1}{%
\save@sf@q{\setlow@box{\textquoteright}/}
\box\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{\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}
\ProvideTextCommand{\guillemotleft}{OT1}{%
\ifmmode
 \ll
\else
 \save@sf@q{\nobreak
 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
 \fi}
\ProvideTextCommand{\guillemotright}{OT1}{%
\ifmmode
 \gg
\else
 \save@sf@q{\nobreak
 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
 \fi}

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

\ProvideTextCommandDefault{\guillemetleft}{%\UseTextSymbol{OT1}{\guillemetleft}}
\ProvideTextCommandDefault{\guillemetright}{%\UseTextSymbol{OT1}{\guillemetright}}
\ProvideTextCommandDefault{\guillemotleft}{%\UseTextSymbol{OT1}{\guillemotleft}}
\ProvideTextCommandDefault{\guillemotright}{%\UseTextSymbol{OT1}{\guillemotright}}
guilsinglleft The single guillemets are not available in OT1 encoding. They are faked.
guilsinglright

2095 \provideTextCommand\{guilsinglleft\}\{OT1\}\{%  
2096 \ifmmode  
2097 \textless \%  
2098 \else  
2099 \save@sf@q\{\nobreak  
2100 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens\}%  
2101 \fi\}  
2102 \provideTextCommand\{guilsinglright\}\{OT1\}\{%  
2103 \ifmmode  
2104 \textgreater \%  
2105 \else  
2106 \save@sf@q\{\nobreak  
2107 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens\}%  
2108 \fi\}  

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

2109 \provideTextCommandDefault\{guilsinglleft\}\{%  
2110 \usetextsymbol\{OT1\}\{\guilsinglleft\}\}  
2111 \provideTextCommandDefault\{guilsinglright\}\{%  
2112 \usetextsymbol\{OT1\}\{\guilsinglright\}\}  

8.12.2 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.

2113 \DeclareTextCommand\{ij\}\{OT1\}\{%  
2114 \kern-0.02em\bbl@allowhyphens j\}  
2115 \DeclareTextCommand\{IJ\}\{OT1\}\{%  
2116 \Kern-0.02em\bbl@allowhyphens J\}  
2117 \DeclareTextCommand\{ij\}\{T1\}\{\char188\}  
2118 \DeclareTextCommand\{IJ\}\{T1\}\{\char156\}  

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

2119 \provideTextCommandDefault\{ij\}\{%  
2120 \usetextsymbol\{OT1\}\{ij\}\}  
2121 \provideTextCommandDefault\{IJ\}\{%  
2122 \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).

2123 \def\crrtic@{\hrule height0.1ex width0.3em}  
2124 \def\crttic@{\hrule height0.1ex width0.33em}  
2125 \def\ddj@{%  
2126 \setbox0\hbox{d}\dimen@=\ht0  
2127 \advance\dimen@1ex  
2128 \dimen@.45\dimen@  
2129 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ii  
2130 \dimen@thr\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@thr  
2131 \leavevmode\rlap{\raise0.5ex\hbox{\kern\dimen@ii\vbox{\crrtic@}}}\}}  
2132 \def\DDJ@{%  
2133 \setbox0\hbox{D}\dimen@=-.55\ht0  
2134 \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@ii  
2135 \advance\dimen@ii.15ex \ correction for the dash position  
2136 \advance\dimen@ii-.15\fontdimen7\font \ correction for cmtt font  
2137 \dimen@thr@\expandafter\rem@pt\the\fontdimen7\font\dimen@thr@
\leavevmode rl{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}}
\ProvideTextCommandDefault{\DJ}{\UseTextSymbol{OT1}{\DJ}}

\SS 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.
\DeclareTextCommand{\SS}{OT1}{SS}
\ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

8.12.3 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
\ProvideTextCommandDefault{\glq}{%}
\textormath{\textquotesinglebase}{\mbox{\textquotesinglebase}}

The definition of \grq depends on the font encoding. With T1 encoding no extra kerning is needed.
\ProvideTextCommand{\grq}{T1}{%}
\textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}
\ProvideTextCommand{TU}{%}
\textormath{\textquoteleft}{\mbox{\textquoteleft}}
\ProvideTextCommand{OT1}{%}
\save@sf@q{\kern-.0125em\textormath{\textquoteleft}{\mbox{\textquoteleft}}\kern.07em\relax}
\ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}{\grq}}

\glqq The ‘german’ double quotes.
\grqq
\ProvideTextCommandDefault{\glqq}{%}
\textormath{\textdoublesinglebase}{\mbox{\textdoublesinglebase}}

The definition of \grqq depends on the font encoding. With T1 encoding no extra kerning is needed.
\ProvideTextCommand{\grqq}{T1}{%}
\textormath{\textquotesingleleft}{\mbox{\textquotesingleleft}}
\ProvideTextCommand{TU}{%}
\textormath{\textquotesingleleft}{\mbox{\textquotesingleleft}}
\ProvideTextCommand{OT1}{%}
\save@sf@q{\kern-.07em\textormath{\textquotesingleleft}{\mbox{\textquotesingleleft}}\kern.07em\relax}
\ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}{\grqq}}

\flq The ‘french’ single guillemets.
\frq
\ProvideTextCommandDefault{\flq}{%}
\textormath{\guilsingleleft}{\mbox{\guilsingleleft}}
\ProvideTextCommandDefault{\frq}{%}
\textormath{\guilsingleright}{\mbox{\guilsingleright}}
The ‘french’ double guillemets.

\providecommanddefault{\flqq}{% \textormath{\guillemetleft}{\mbox{\guillemetleft}}} \providecommanddefault{\frqq}{% \textormath{\guillemetright}{\mbox{\guillemetright}}}

8.12.4 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 \umlautlow

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

\umlauthigh \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 \langle dimen \rangle 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).
Finally, make sure the default hyphenation rules are defined (even if empty). For internal use, another empty \language is defined. Currently 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@unhyphenated\@undefined
\newlanguage\l@unhyphenated
\fi

8.13 Layout

Layout is mainly intended to set bidi documents, but there is at least a tool useful in general.

\bbl@trace{Bidi layout}
\providecommand\IfBabelLayout[3]{#3}
\newcommand\BabelPatchSection[1]{
  \ifundefined{#1}{}{\
    \bbl@exp{\let<\bbl@ss@#1><#1>}{\@namedef{#1}{\@ifstar{\bbl@presec@s{#1}}{\@dblarg{\bbl@presec@x{#1}}}}}}
\def\bbl@presec@x#1[#2]#3{\bbl@exp{\select@language@x{\bbl@main@language}\bbl@cs{ssp@#1}\bbl@cs{ss@#1}}{\foreignlanguage{\languagename}{\unexpanded{#2}}}{\foreignlanguage{\languagename}{\unexpanded{#3}}}}
\def\bbl@presec@s#1#2{\bbl@exp{\select@language@x{\bbl@main@language}\bbl@cs{ssp@#1}^*}{\foreignlanguage{\languagename}{\unexpanded{#2}}}}
\IfBabelLayout{sectioning}{\BabelPatchSection{part}\BabelPatchSection{chapter}\BabelPatchSection{section}\BabelPatchSection{subsection}\BabelPatchSection{subsubsection}\BabelPatchSection{paragraph}\BabelPatchSection{subparagraph}\def\babel@toc#1{\select@language@x{\bbl@main@language}}}{}}
8.14 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

8.15 Creating and modifying languages

\babelprovide is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previously loaded ldf files.

\bbl@trace{Creating languages and reading ini files}
\let\bbl@extend@ini@\gobble
\newcommand\babelprovide[2][]{{%
\let\bbl@savelangname\languagename
\edef\bbl@savelocalc\localeid{%
% Set name and locale id
\edef\languagename{#2}%
\bbl@id@assign
% Initialize keys
\let\bbl@KVP@captions@\nil
\let\bbl@KVP@date@\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
\let\bbl@KVP@linebreaking@\nil
\let\bbl@KVP@justification@\nil
\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@transforms@\nil
\global\let\bbl@release@transforms@\empty
\let\bbl@KVP@alph@\nil
\let\bbl@KVP@Alph@\nil
\let\bbl@KVP@labels@\nil
\bbl@csarg\let\KVP@labels*\nil
\global\let\bbl@inidata@\empty
\global\let\bbl@extend@ini\gobble
\def\bbl@key@list{}{%
\bbl@forkv{#1}{% TODO - error handling
\in@/{{}{#1}%
\ifin@
\\global\let\bbl@extend@ini\bbl@extend@ini@aux
\bbl@renewinikey#1@@{%#2}%
\else
\bbl@csarg\def\KVP@#1{%#2}%
\fi
}
\chardef\bbl@howloaded=\string none; 1:ldf without ini; 2:ini
\bbl@ifunset{date\bbl@llevel@2}\@{\bbl@ifunset{bbl@extracaps@2}\@ne\tw@}\%
% == init ==
\ifx\bbl@screset\@undefined
\bbl@ldfinit
\fi
% == import, captions ==
\ifx\bbl@KVP@import\@nil\else
\bbl@exp{\bbl@KVP@import}\%
{\ifx\bbl@initoload\relax
 \begingroup
 \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}\%
 \bbl@input@texini{#2}\%
 \endgroup
 \else
 \xdef\bbl@KVP@import{\bbl@initoload}\%
 \fi}\%
{\if\bbl@KVP@import\@nil\else
 \let\bbl@KVP@import\bbl@KVP@captions\%
 \fi}\%
\fi
% == Load ini ==
\ifcase\bbl@howloaded
 \bbl@provide@new{#2}\%
\else
 \bbl@ifblank{#1}\@{\bbl@provide@renew{#2}}\%
\fi
% Post tasks
% ----------
% == subsequent calls after the first provide for a locale ==
\ifx\bbl@inidata\@empty\else
\bbl@extend@ini{#2}\%
\fi
% ensure captions ==
\ifx\bbl@KVP@captions\@nil\else
\bbl@ifunset{bbl@extracaps@#2}\%
{\bbl@exp{\bbl@babelensure[exclude=\today]{#2}}}\%
{\bbl@exp{\bbl@babelensure[exclude=\today, include=\bbl@extracaps@2]{#2}}}\%
\fi
% At this point all parameters are defined if 'import'. Now we
% execute some code depending on them. But what about if nothing was
% imported? We just set the basic parameters, but still loading the
% whole ini file.
\bbl@load@basic[#2]%
% == script, language ==
% Override the values from ini or defines them
\l@ifx\bbl@KVP@script\@nil\else
  \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}\fi
\l@ifx\bbl@KVP@language\@nil\else
  \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}\fi
% == onchar ==
\l@ifx\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}%
  \l@fin@
% Needed if no explicit selection
\AddBabelHook{bbl-onchar}{beforestart}{\bbl@starthyphens}%
\l@fin@
\l@ifx\bbl@add\bbl@starthyphens{\bbl@patterns@lua{\languagename}}\@nil\else
\l@fin@
% 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.locale_props[\the/localeid].lc = \the/localeid\space
    Babel.locale_props[\the/localeid].lg = \the/@nameuse{l@\languagename}\space
  end}%
\l@fin@
\l@ifx\bbl@provide@sys{\languagename}\@nil\else
\l@fin@
\l@ifx\bbl@provide@dirs{\languagename}\@nil\else
\l@fin@
\Ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{\{\bbl@mapselect\}}%
    \bbl@selectfont{\}
    \edef\bbl@mapselect{%
      \let\bbl@mapselect\relax
      \edef\bbl@prefontid{\fontid\font}%
    }%
    \def\bbl@mapdir##1{%
      \def\languagename{##1}%
      \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
      \bbl@switchfont
      \directlua{Babel.locale_props{the\csname bbl@id@##1\endcsname}=the\csname bbl@prefontid\endcsname}%
      \fi}%
  }%
  \fi
\bbl@exp{\\bbl@add\\bbl@mapselect{\\bbl@mapdir{\languagename}}}%
\fi
\% TODO - catch non-valid values
\% == mapfont ==
\% For bidi texts, to switch the font based on direction
\if\bbl@KVP@mapfont\nil\else
  \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
  {\bbl@error{Option \bbl@KVP@mapfont unknown for\%}
    mapfont. Use 'direction'.%}
  {See the manual for details.}}%
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
\ift\bbl@mapselect\@undefined % TODO. See onchar.
  \AtBeginDocument{%
    \bbl@patchfont{\{\bbl@mapselect\}}%
    \bbl@selectfont{\}
    \edef\bbl@mapselect{%
      \let\bbl@mapselect\relax
      \edef\bbl@prefontid{\fontid\font}%
    }%
    \def\bbl@mapdir##1{%
      \def\languagename{##1}%
      \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
      \bbl@switchfont
      \directlua{Babel.locale_props{the\csname bbl@wdir@##1\endcsname}=the\csname bbl@prefontid\endcsname}%
      \fi}%
  }%
  \fi
\% == Line breaking: intraspace, intrapenalty ==
\% For CJK, East Asian, Southeast Asian, if interspace in ini
\if\bbl@KVP@intraspacespace\nil\else % We can override the ini or set
  \bbl@csargs{\edef{intsp@#2}{\bbl@KVP@intraspacespace}}%
\fi
\% == Line breaking: CJK quotes ==
\ifcase\bbl@engine\or
  \bbl@xin@{/c}{/bbl@cl{\lnbrk}}%
\fi
\bbl@ifunset{bbl@quote@\languagename}{}%
\bbl@iffalse
```lua
Babel.locale_props[\the\localeid].cjk_quotes = {}
local cs = 'op'
for c in string.utfvalues(%
    [[\csname bbl@quote@\languagename\endcsname]]) do
    if Babel.cjk_characters[c].c == 'qu' then
        Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
    end
    cs = (cs == 'op') and 'cl' or 'op'
end
}}%
\fi
\fi
% == Line breaking: justification ==
\ifx\bbl@KVP@justification@nil\else
    \let\bbl@KVP@linebreaking=\bbl@KVP@justification
\fi
\ifx\bbl@KVP@linebreaking@nil\else
    \bbl@xins,{\bbl@KVP@linebreaking,,}%,elongated,kashida,cjk,unhyphenated,}%
\fin@
\bbl@csarg\xdef{lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking@nil}%
\fi
\fi
\bbl@xins{/e}{/\bbl@cl{lnbrk}}%
\ifin@\bbl@arabicjust\fi
% == Line breaking: hyphenate.other.(locale|script) ==
\ifx\bbl@lbkflag@empty\bbl@ifunset{bbl@hyotl@\languagename}{}% {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
\bbl@startcommands*{\languagename}{}% \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
\ifcase\bbl@engine
    \ifnum##1<257
        \SetHyphenMap{\BabelLower{##1}{##1}}%
    \fi
\else
        \global\lccode##1=##1\relax
    \fi}%
\bbl@endcommands}
\bbl@ifunset{bbl@hyots@\languagename}{}% {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
\bbl@csarg\bbl@foreach{hyots@\languagename}{%
\ifcase\bbl@engine
    \ifnum##1<257
        \global\lccode##1=##1\relax
    \fi
\else
        \global\lccode##1=##1\relax
    \fi}%
\bbl@endcommands}%
\bbl@ifunset{bbl@dyotl@\languagename}{}% {\bbl@csarg\bbl@replace{dyotl@\languagename}{ }{,}%
\bbl@startcommands*{\languagename}{}% \bbl@csarg\bbl@foreach{dyotl@\languagename}{%
\ifcase\bbl@engine
    \ifnum##1<257
        \SetHyphenMap{\BabelLower{##1}{##1}}%
    \fi
\else
        \global\lccode##1=##1\relax
    \fi}%
\bbl@endcommands}
% == Counters: maparabic ==
% Native digits, if provided in ini (TeX level, xe and lua)
\ifcase\bbl@engine\else
    \bbl@ifunset{bbl@dgnat@\languagename}{}% {\bbl@csarg\bbl@replace{dgnat@\languagename}{ }{,}%
\bbl@csarg\bbl@foreach{dgnat@\languagename}{%
\ifcase\bbl@engine
    \ifnum##1<257
        \global\lccode##1=##1\relax
    \fi
\else
        \global\lccode##1=##1\relax
    \fi}%
\bbl@endcommands}
% == Counters: maparabic ==
% Native digits, if provided in ini (TeX level, xe and lua)
\ifcase\bbl@engine\else
    \bbl@ifunset{bbl@dgnat@\languagename}{}% {\bbl@csarg\bbl@replace{dgnat@\languagename}{ }{,}%
\bbl@csarg\bbl@foreach{dgnat@\languagename}{%
\ifcase\bbl@engine
    \ifnum##1<257
        \global\lccode##1=##1\relax
    \fi
\else
        \global\lccode##1=##1\relax
    \fi}%
\bbl@endcommands}
```

122
\ifx\bbl@KVP@maparabic@nil\else
  \ifx\bbl@latin@arabic@undefined
    \expandafter\let\expandafter\@arabic\csname bbl@counter@arabic\endcsname
  \else % ie, if layout=counters, which redefines \@arabic
    \expandafter\let\expandafter\bbl@latinarabic\csname bbl@counter@arabic\endcsname
  \fi
\fi
\fi
\fi\fi
\fi
% == Counters: mapdigits ==
% Native digits (lua level).
\ifodd\bbl@engine
  \ifx\bbl@KVP@mapdigits@nil\else
    \bbl@ifunset{bbl@dgnat@arabic}{}%
    \Directlua{%
      Babel = Babel or {} %%% -> presets in luabel
      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 = Babel.attr_locale
          local GLYPH = node.id'glyph'
          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
      return head
    }%
  \fi
\fi
% == Counters: alph, Alph ==
% What if extras<lang> contains a \bbl@save\@alph? It won't be
% restored correctly when exiting the language, so we ignore
% this change with the \bbl@alph@saved trick.
\ifx\bbl@KVP@alph@nil\else
  \bbl@extras@wrap{\bbl@alph@saved}%
  \let\bbl@alph@saved\@alph%
  \let\@alph\bbl@alph@saved
  \bbl@save@alph%
\fi
\bbl@exp(%
  \bbl@add\<extras\langname>%

Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.
Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values. (TODO. But preserving previous values would be useful.)
The hyphenrules option is handled with an auxiliary macro.

The reader of babel-...tex files. We reset temporarily some catcodes.
The following macros read and store ini files (but don’t process them). For each line, there are 3 possible actions: ignore if starts with ‘;’, switch section if starts with ‘[’, and store otherwise. There are used in the first step of \bbl@read@ini.

2763 \def\bbl@iniline#1\bbl@iniline{%
2764 \@ifnextchar\[\bbl@inisect\bbl@iniskip\bbl@inistore#1\@@}%
2765 \def\bbl@inisect[#1]#2\@@{%\def\bbl@section{#1}}
2766 \def\bbl@iniskip#1\@@{% if starts with ‘;’,
2767 \def\bbl@inistore#1=#2\@@{% full (default)
2768 \bbl@trim\bbl@elt\bbl@section\bbl@tempa\bbl@key@list%}
2769 \ifin\else
2770 \bbl@exp{%
2771 \bbl@inidata\bbl@section\bbl@tempa\bbl@key@list%}
2772 \fi}
2773 \ifx\bbl@readstream\@undefined
2774 \csname newread\endcsname\bbl@readstream
2775 \fi
2776 \def\bbl@read@ini#1#2{%
2777 \global\let\bbl@extend@ini\@gobble
2778 \openin\bbl@readstream=babel-#1.ini
2779 \ifeof\bbl@readstream
2780 \bbl@error
2781 \bbl@section=identification.\bbl@section={\bbl@tempa}\bbl@key@list%
2782 \fi
2783 \fi
2784 Now, the ‘main loop’, which **must be executed inside a group**. At this point, \bbl@inidata may contain data declared in \bbl@provide, with ‘slashed’ keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, ‘export’ some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \bbl@provide it’s either 1 or 2.
2785 \if\bbl@readstream\undefined
2786 \csname newread\endcsname\bbl@readstream
2787 \fi
2788 \global\let\bbl@extend@ini\gobble
2789 \openin\bbl@readstream=babel-#1.ini
2790 \ifeof\bbl@readstream
2791 \bbl@error
2792 {There is no ini file for the requested language\%
2793 (#1). Perhaps you misspelled it or your installation\%
2794 is not complete.}%
2795 {Fix the name or reinstall babel.}%
2796 \else
2797 \global\let\bbl@extend@ini\empty
2798 \global\let\bbl@inidata\empty
2799 \catcode\do12 \catcode\do12 \do12 \catcode\do12 \do12
2800 \bbl@info{Importing
2801 \ifcase#2\font and identification \or basic \fi
2802 data for \languagename\%
2803 from babel-#1.ini. Reported}%
2804 \ifnum#2=a\z@
2805 \global\let\bbl@inidata\empty
2806 \let\bbl@inistore=bbl@inistore\@empty % Remember it’s local
2807 \fi
2808 \def\bbl@section{identification}%
2809 \bbl@exp{{\bbl@inistore tag.ini=#1\bbl@key@list}%
2810 \bbl@inistore load.level=#2\@@
2811 \loop
A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.
A somewhat hackish tool to handle calendar sections. To be improved.

\def\bbl@ini@calendar#1{\lowercase{\def\bbl@tempa{=#1=}}\bbl@replace\bbl@tempa{=date.gregorian}{}\bbl@replace\bbl@tempa{=date.}{}\in@{.licr=}#1=}\ifin@\ifcase\bbl@engine\bbl@replace\bbl@tempa{.licr=}{}\else\let\bbl@tempa\relax\fi\ifx\bbl@tempa\relax\else\bbl@replace\bbl@tempa{=}{}\bbl@exp{\def\bbl@inikv@#1####1####2{\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}\fi}

A key with a slash in babel's provide replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the corresponding key and reset the macro (in \bbl@instore above).

\def\bbl@renewinikey#1/#2\@@#3{\edef\bbl@tempa{\zap@space #1 \@empty}% section\edef\bbl@tempb{\zap@space #2 \@empty}% key\bbl@trim\toks@{#3}% value\bbl@exp{\edef\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}\g@addto@macro\bbl@inidata{\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}}

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

\def\bbl@exportkey#1#2#3{\bbl@ifunset{bbl@@kv@#2}{\bbl@csarg\gdef{#1@languagename}{#3}}{\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty\bbl@csarg\gdef{#1@languagename}{#3}\else\bbl@exp{\global\let\bbl@#1@languagename=\bbl@@kv@#2}\fi}}

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@ini@exports is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

\def\bbl@ini@warn@#1{\bbl@ifunset{bbl@@kv@identification.warning#1}{}{\bbl@warning{From babel-\bbl@cs{l@languagename}.ini:\%\bbl@cs{\@kv@identification.warning#1}\%Reported}}}
\def\bbl@iniexports#1{\% Identification always exported
\bbl@iniwarning{}% 
\ifcase\bbl@engine
\bbl@iniwarning{.pdflatex}%
or
\bbl@iniwarning{.lualatex}%
or
\bbl@iniwarning{.xelatex}%
\fi%
\bbl@exportkey{llevel}{identification.load.level}{}%
\bbl@exportkey{elname}{identification.name.english}{}%
\bbl@exportkey{lname}{identification.name.opentype}{\csname bbl@elname@\languagename\endcsname}%
\bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
\bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
\bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
\bbl@exportkey{esname}{identification.script.name}{}%
\bbl@exportkey{sname}{identification.script.name.opentype}{\csname bbl@esname@\languagename\endcsname}%
\bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
\bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
\ifbbl@bcptoname
\bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
\fi
% Conditional
\ifnum#1>\z@ % 0 = only info, 1, 2 = basic, (re)new
\bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
\bbl@exportkey{hyphr}{typography.hyphenrules}{}%
\bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
\bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
\bbl@exportkey{prehc}{typography.prehyphenchar}{}%
\bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
\bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
\bbl@exportkey{intsp}{typography.intraspace}{}%
\bbl@exportkey{frspc}{typography.frenchspacing}{u}%
\bbl@exportkey{chrng}{characters.ranges}{}%
\bbl@exportkey{quote}{characters.delimiters.quotes}{}%
\bbl@exportkey{dgnat}{numbers.digits.native}{}%
\ifnum#1=\tw@ % only (re)new
\bbl@exportkey{rqtex}{identification.require.babel}{}%
\bbl@toglobal\bbl@savetoday
\bbl@toglobal\bbl@savedate
\bbl@savestrings
\fi
\fi}
A shared handler for key=val lines to be stored in \bbl@@kv@<section>.<key>.
\def\bbl@ini#1{\% key=value
\toks@{#2}% This hides #'s from ini values
\bbl@csarg\edef{@kv@\bbl@section.#1}{\the	oks@}}
By default, the following sections are just read. Actions are taken later.
\let\bbl@iniidentification\bbl@ini
\let\bbl@ini\bbl@ini
\let\bbl@ini\bbl@ini
\let\bbl@ini\bbl@ini

Additive numerals require an additional definition. When \( .1 \) is found, two macros are defined – the basic one, without \( .1 \) called by \texttt{\localenumeral}, and another one preserving the trailing \( .1 \) for the ‘units’.

\begin{verbatim}
def\bbl@inikv@counters#1\#2{% \bbl@ifsamestring\{#1\}\{digits\}% \{\bbl@error\{The counter name 'digits' is reserved for mapping\%\%\%
    decimal digits\}% \{Use another name.\}% \}% \def\bbl@tempc{#1}% \bbl@trim@def{\bbl@tempb*}{#2}% \in@{.1$}{#1$}% \ifin@
    \bbl@replace\bbl@tempc{.1}{}% \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{% \noexpand\bbl@alphnumeral{\bbl@tempc}}% \fi \in@{.F.}{#1}% \ifin@\else\in@{.S.}{#1}\fi \ifin@ \def\bbl@inikv@counters#1#2{% \bbl@ini@counters@aux{#1}{#2}}\else \def\bbl@inikv@counters#1#2{% \bbl@ini@counters@aux{#1}{#2}}\fi
\end{verbatim}

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.

\begin{verbatim}
\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
\end{verbatim}

The auxiliary macro for captions define \langle caption\rangle.name.

\begin{verbatim}
\def\bbl@ini@captions@template#1\#2{% string language tempo=capt-name \bbl@replace\bbl@tempo{.template}{% \bbl@replace\bbl@tempo{[]}\{\nobreakspace{}}{% \bbl@replace\bbl@tempo{[]}\{\csname}{% \bbl@replace\bbl@tempo{[]}\{\csname the}{% \bbl@replace\bbl@tempo{[]}\{\endcsname{}}{% \bbl@replace\bbl@tempo{[]}\{\endcsname{}}{% \bbl@xin@{,\bbl@tempo,}{,chapter,appendix,part,}{% \ifin@
    \nameuse{\bbl@patch\bbl@tempo}\% \global\bbl@csarg\let\bbl@tempo fnt@\#2\bbl@toreplace \fi \bbl@xin@{,\bbl@tempo,}{,figure,table,}{% \ifin@
    \toks@\expandafter{\bbl@toreplace}{% \bbl@exp\{gdef\{fnum@\bbl@tempo\}\{the\toks0\}\}}{% \ifin@
    \def\bbl@ini@captions@aux\#1\#2{% \bbl@to@replace{\bbl@tempo}\bbl@to@replace %}
\end{verbatim}

131
Labels. Captions must contain just strings, no format at all, so there is new group in ini files.

```latex
\def\bbl@list@the{% part, chapter, section, subsection, subsubsection, paragraph, \%
  subparagraph, enumi, enumii, enumiii, enumiv, equation, figure, \%
  table, page, footnote, mpfootnote, mpfn\}
\def\bbl@map@cnt#1{% #1: roman, etc, // #2: enumi, etc
  \bbl@ifunset{bbl@map@#1@languagename}{\@nameuse{#1}}{%\@nameuse{bbl@map@#1@languagename}}
\def\bbl@inikv@labels#1#2{% \in@{.map}{#1}\%
  \ifin@
    \ifx\bbl@KVP@labels@nil\else
      \bbl@xin@{ map }{ \bbl@KVP@labels@space}\%
    \fi
  \fi\%
  \def\bbl@tempc{#1}\%
  \bbl@replace\bbl@tempc{.map}{}\%
  \in@{,#2,}{arabic,roman,roman,alph,Alph,fnsymbol,}\%
  \bbl@exp{%\vdef\bbl@map@\bbl@tempc@languagename{% \{\ifin@{#2}\else{\localecounter{#2}}\fi}}\%
  \bbl@foreach\bbl@list@the{% \bbl@ifunset{the##1}{}\%
    \bbl@exp{\let{\bbl@tempd}{the##1}}\%
    \bbl@exp{%\vdef\bbl@map@\bbl@tempd@languagename{% \{}\ifin@{#2}\else{\localecounter{#2}}\fi}}\%
    \bbl@map@cnt{\bbl@tempd}{##1}\%
  }\%
  \bbl@ifset{\bbl@KVP@labels@nil\fi}\%
}
\def\bbl@tempc{\let{\bbl@tempd}{the\%}\%
  \bbl@exp{%\vdef\bbl@map@\bbl@tempd@languagename{% \{}\ifin@{#2}\else{\localecounter{#2}}\fi}}\%
  \bbl@foreach\bbl@list@the{% \bbl@ifunset{the##1}{}\%
    \bbl@exp{\let{\bbl@tempd}{the##1}}\%
    \bbl@exp{%\vdef\bbl@map@\bbl@tempd@languagename{% \{}\ifin@{#2}\else{\localecounter{#2}}\fi}}\%
    \bbl@map@cnt{\bbl@tempd}{##1}\%
  }\%
  \bbl@ifset{\bbl@KVP@labels@nil\fi}\%
}
```

% The following code is still under study. You can test it and make
% suggestions. Eg, enumerate.2 = (\[enumi\]).(\[enumii\]). It's
% language dependent.
\in@{enumerate.}{#1}%%
\ifin@
  \def\bbl@tempa{#1}%%
  \bbl@replace\bbl@tempa{enumerate.}{}%%
  \def\bbl@toreplace{#2}%%
  \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%%
  \bbl@replace\bbl@toreplace{[}{\csname the}\bbl@tempa{\endcsname}{}}%%
  \toks@\expandafter{\bbl@toreplace}%%
  % TODO. Execute only once:
  \bbl@exp{%\bbl@add\appendix\languagename{}}%%
  \bbl@ifunset{bbl@chaptype fmt@\languagename}{}%%
  \def\bbl@chaptypefmt\romannumeral{\bbl@tempa}{\@chapapp\space\thechapter}{\@nameuse{bbl@chaptype fmt@\languagename}}%%
  \bbl@add\appendix\languagename{}}% Not harmful, I hope
\bbl@sreplace\ps@headings\bbl@tempa{\@chapapp\space\thechapter}{\bbl@chfmt}{%}
\bbl@sreplace\chaptermark\bbl@tempa{\@chapapp\space\thechapter}{\bbl@chfmt}{%}
\bbl@sreplace\@makechapterhead\bbl@tempa{\@chapapp\space\thechapter}{\bbl@chfmt}{%}
\bbl@toglobal\appendix
\bbl@toglobal\ps@headings
\bbl@toglobal\chaptermark
\bbl@toglobal\makechapterhead
\let\bbl@patchchapter\relax
\fi
\fi
}

To show correctly some captions in a few languages, we need to patch some internal macros, because
the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string,
while in Hungarian is placed after. These replacement works in many classes, but not all. Actually,
the following lines are somewhat tentative.
\bbl@toglobal\@part
\fi

**Date. TODO. Document**

% Arguments are _not_ protected.
\let\bbl@calendar\@empty
\DeclareRobustCommand\localedate[1][9]{\bbl@localedate{#1}}
\def\bbl@localedate#1#2#3#4{\begingroup
\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.\bbl@ld@variant\fi}\bbl@replace\bbl@calendar{gregorian}{}\fi\bbl@cased\{\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}\endgroup}

% eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
\def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
\bbl@trim@def\bbl@tempa{#1.#2}\bbl@ifsamestring{\bbl@tempa}{months.wide}\{\bbl@trim@def\bbl@tempa{#3}\bbl@trim\toks@{#5}\@temptokena\expandafter{\bbl@savedate}\bbl@exp{% Reverse order - in ini last wins\def\bbl@savedate{%\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}{\the\@temptokena}}%\\bbl@exp{% to savedate\def\bbl@tempa{\bbl@savedate}\bbl@exp{% Reverse order - in ini last wins\def\bbl@savedate{%\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}{\the\@temptokena}}}%\\bbl@exp{% defined now\def\bbl@tempa{\bbl@toreplace}\bbl@exp{% To date\bbl@exp{% TODO. Move to a better place.\gdef\<language@name>date>{{\protect\<language@name> date >}}{{\protect\<language@name> date >}}\gdef\<language@name> date >##1##2##3\{% \bbl@usedategrouptrue\<bbl@ensure@language>\{% \\localdate{##1}{##2}{##3}\}%\bbl@add\\\bbl@save today\%\\SetString\\today\{%\<language@name> date >\{\\the\year{\\the\month{\\the\day}}}\}%\%\global\bbl@csarg\let{date@\languagename @}\bbl@to replace\ifx\bbl@temp\@empty\else\global\bbl@csarg\let{date@\languagename @\bbl@temp}\bbl@to replace\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 the date or in the month name. Note after \bbl@replace \toks@ contains the resulting string, which is used by \bbl@replace@finish@iii (this implicit behavior doesn't seem a good idea, but it's efficient).

\let\bbl@calendar\empty
\newcommand\BabelDateSpace{\nobreakspace}
\newcommand\BabelDateDot{.\@}
\newcommand\BabelDated[1]{\ifnum#1<10\relax0\fi\number#1}
\newcommand\BabelDatedd[1]{\ifnum#1<9\relax0\fi\number#1}
\newcommand\BabelDateM[1]{\number#1}
\newcommand\BabelDateMM[1]{\ifnum#1<10\relax0\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\relax0\number#1\fi\ifnum#1<100\relax0\number#1\fi\ifnum#1<1000\expandafter\@gobble\number#1\fi\ifnum#1<10000\expandafter\@gobbletwo\number#1\fi}
\newcommand\BabelDateyyyy[1]{\number#1}
\def\bbl@replace@finish@iii#1{\bbl@exp{\def\#1####1####2####3{\the\toks@}}}
\def\bbl@TG@@date{\bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace}{}}
\def\bbl@toreplace#1\{#2\}{#3\}{#4}\relax{#1\{#2\}{#3\}{#4}}
\begingroup % A hack. TODO. Don't require an specific order
\catcode`%=12
\catcode`&=14
\gdef\bbl@transforms#1#2#3{&\ifx\bbl@KVP@transforms\@nil\else\directlua{str = \=[#2]=-}Transforms.
\let\bbl@release@transforms\empty
\@namedef{bbl@inikv@transforms.prehyphenation}{\bbl@transforms}
\@namedef{bbl@inikv@transforms.posthyphenation}{\bbl@transforms}
\def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
\begingroup % A hack. TODO. Don't require an specific order
\catcode`%=12
\catcode`&=14
\gdef\bbl@transforms#1#2#3{&\ifx\bbl@KVP@transforms\@nil\else\directlua{str = \=[#2]=-}
Language and Script values to be used when defining a font or setting the direction are set with the following macros.

\def\bbl@provide@lsys#1{% 
\bbl@ifunset{bbl@lname@#1}{\bbl@load@info{#1}}{} 
\bbl@csarg\let{lsys@#1}@empty 
\bbl@ifunset{bbl@sname@#1}{}% 
\bbl@ifunset{bbl@sotf@#1}{}% 
\bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}{} 
\bbl@ifunset{bbl@lname@#1}{}% 
{\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}{} 
\ifcase\bbl@engine\or\or 
\bbl@ifunset{bbl@prehc@#1}{}% 
{\bbl@exp{\\bbl@ifblank{\bbl@cs{prehc@#1}}}{}% 
{\ifx\bbl@xenohyph\@undefined 
\let\bbl@xenohyph@d\bbl@xenohyph@d 
\ifx\AtBeginDocument\@notprerr 
\expandafter\@secondoftwo \% to execute right now 
\fi 
\AtBeginDocument{% 
\bbl@patchfont{\bbl@xenohyph}% 
\expandafter\selectlanguage\expandafter{\languagename}% 
\fi})% 
\fi 
\bbl@csarg\bbl@toglobal{lsys@#1} 
\def\bbl@xenohyph@d{% 
{\ife\bbl@xenohyph@undefined 
\let\bbl@xenohyph@d\bbl@xenohyph@d 
\ifx\AtBeginDocument\@notprerr 
\expandafter\@secondoftwo \% to execute right now 
\fi 
\AtBeginDocument{% 
\bbl@patchfont{\bbl@xenohyph}% 
\expandafter\selectlanguage\expandafter{\languagename}% 
\fi})% 
\fi 
\bbl@csarg\bbl@toglobal{lsys@#1} 
\def\bbl@xenohyph@d{\% 
{\ife\bbl@xenohyph@undefined 
\let\bbl@xenohyph@d\bbl@xenohyph@d 
\ifx\AtBeginDocument\@notprerr 
\expandafter\@secondoftwo \% to execute right now 
\fi 
\AtBeginDocument{% 
\bbl@patchfont{\bbl@xenohyph}% 
\expandafter\selectlanguage\expandafter{\languagename}% 
\fi})% 
\fi 
\bbl@csarg\bbl@toglobal{lsys@#1} 
\def\bbl@xenohyph@d{\% 
{\ife\bbl@xenohyph@undefined 
\let\bbl@xenohyph@d\bbl@xenohyph@d 
\ifx\AtBeginDocument\@notprerr 
\expandafter\@secondoftwo \% to execute right now 
\fi 
\AtBeginDocument{% 
\bbl@patchfont{\bbl@xenohyph}% 
\expandafter\selectlanguage\expandafter{\languagename}% 
\fi})% 
\fi 
\bbl@warning 
{Neither 0 nor ZERO WIDTH SPACE are available!\% 
in the current font, and therefore the hyphen\% will be printed. Try changing the fontspec's\% 'HyphenChar' to another value, but be aware\% this setting is not safe (see the manual)}% 
\bbl@warning 
\fi
}
The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, 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 tex file named as the language (which means any code in it must be skipped, too).

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.

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 an 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 \localeforeach, where \bbl@ini@loaded is a comma-separated list of locales, built by \bbl@read@ini.

9 Adjusting the Babel behavior

A generic high level interface is provided to adjust some global and general settings.

\newcommand\babeladjust[1]{% TODO. Error handling.
  \bbl@forkv{#1}{
    \bbl@ifunset{bbl@ADJ@##1@##2}{
      \bbl@cs{ADJ@##1@##2}}%
    \bbl@adjust@lua{#1}{#2}}%
  \ifvmode
    \ifnum\currentgrouplevel=\z@
      \directlua{ Babel.#2 }%
      \expandafter\expandafter\expandafter\@gobble
    \fi
  \fi
  \bbl@error % The error is gobbled if everything went ok.
  % (Currently, #1 related features can be adjusted only in
  %  the main vertical list.)%
  % (Maybe things change in the future, but this is what it is.)}}
\@namedef{bbl@ADJ@bidi.mirroring@on}{\bbl@adjust@lua{bidi}{mirroring_enabled=true}}
\@namedef{bbl@ADJ@bidi.mirroring@off}{\bbl@adjust@lua{bidi}{mirroring_enabled=false}}
\@namedef{bbl@ADJ@bidi.text@on}{% 
  \bbl@adjust@lua{bidi}{bidi_enabled=true}}
\@namedef{bbl@ADJ@bidi.text@off}{% 
  \bbl@adjust@lua{bidi}{bidi_enabled=false}}
\@namedef{bbl@ADJ@bidi.mapdigits@on}{% 
  \bbl@adjust@lua{bidi}{digits_mapped=true}}
\@namedef{bbl@ADJ@bidi.mapdigits@off}{% 
  \bbl@adjust@lua{bidi}{digits_mapped=false}}

\@namedef{bbl@ADJ@linebreak.sea@on}{% 
  \bbl@adjust@lua{linebreak}{sea_enabled=true}}
\@namedef{bbl@ADJ@linebreak.sea@off}{% 
  \bbl@adjust@lua{linebreak}{sea_enabled=false}}
\@namedef{bbl@ADJ@linebreak.cjk@on}{% 
  \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
\@namedef{bbl@ADJ@linebreak.cjk@off}{% 
  \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
\@namedef{bbl@ADJ@justify.arabic@on}{% 
  \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
\@namedef{bbl@ADJ@justify.arabic@off}{% 
  \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}

\def\bbl@adjust@layout#1{% 
  \ifvmode
    #1\expandafter\@gobble
  \fi
  \bbl@error % The error is gobbled if everything went ok.
  {Currently, layout related features can be adjusted only\%
    in vertical mode.} %
  {Maybe things change in the future, but this is what it is.}}
\@namedef{bbl@ADJ@layout.tabular@on}{% 
  \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
\@namedef{bbl@ADJ@layout.tabular@off}{% 
  \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
\@namedef{bbl@ADJ@layout.lists@on}{% 
  \bbl@adjust@layout{\let\list\bbl@NL@list}}
\@namedef{bbl@ADJ@layout.lists@off}{% 
  \bbl@adjust@layout{\let\list\bbl@OL@list}}
\@namedef{bbl@ADJ@hyphenation.extra@on}{% 
  \bbl@activateposthyphen}
\@namedef{bbl@ADJ@autoload.bcp47@on}{% 
  \bbl@bcpallowedtrue}
\@namedef{bbl@ADJ@autoload.bcp47@off}{% 
  \bbl@bcpallowedfalse}
\@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{% 
  \def\bbl@bcp@prefix{#1}}
\def\bbl@bcp@prefix{bcp47-}
\@namedef{bbl@ADJ@autoload.options}#1{% 
  \def\bbl@autoload@options{#1}}
\let\bbl@autoload@bcpoptions\@empty
\@namedef{bbl@ADJ@autoload.bcp47.options}#1{% 
  \def\bbl@autoload@bcpoptions{#1}}
\newif\ifbbl@bcptoname
\@namedef{bbl@ADJ@bcp47.toname@on}{% 
  \bbl@bcptonametrue
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.toname@off}{% 
  \bbl@bcptonamefalse
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.toname#1}{% 
  \if\bbl@bcptonametrue
    \bbl@bcptonametrue\let\bbl@bcptonametrue\@empty
    \bbl@bcptonametrue#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalse}{% 
  \if\bbl@bcptonamefalse
    \bbl@bcptonamefalse\let\bbl@bcptonamefalse\@empty
    \bbl@bcptonamefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalse#1}{% 
  \if\bbl@bcptonamefalsefalse
    \bbl@bcptonamefalsefalse\let\bbl@bcptonamefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalsefalse}{% 
  \if\bbl@bcptonamefalsefalsefalse
    \bbl@bcptonamefalsefalsefalse\let\bbl@bcptonamefalsefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalsefalsefalse}{% 
  \if\bbl@bcptonamefalsefalsefalsefalse
    \bbl@bcptonamefalsefalsefalsefalse\let\bbl@bcptonamefalsefalsefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalsefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalsefalsefalsefalse}{% 
  \if\bbl@bcptonamefalsefalsefalsefalsefalse
    \bbl@bcptonamefalsefalsefalsefalsefalse\let\bbl@bcptonamefalsefalsefalsefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalsefalsefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalsefalsefalsefalsefalse}{% 
  \if\bbl@bcptonamefalsefalsefalsefalsefalsefalse
    \bbl@bcptonamefalsefalsefalsefalsefalsefalse\let\bbl@bcptonamefalsefalsefalsefalsefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalsefalsefalsefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalsefalsefalsefalsefalsefalse}{% 
  \if\bbl@bcptonamefalsefalsefalsefalsefalsefalsefalse
    \bbl@bcptonamefalsefalsefalsefalsefalsefalsefalse\let\bbl@bcptonamefalsefalsefalsefalsefalsefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalsefalsefalsefalsefalsefalsefalse}{% 
  \if\bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalse
    \bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalse\let\bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalsefalse#1
  \fi
  \BabelEnsureInfo}
\@namedef{bbl@ADJ@bcp47.tonamefalsefalsefalsefalsefalsefalsefalsefalse}{% 
  \if\bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalsefalse
    \bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalsefalse\let\bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalsefalse\@empty
    \bbl@bcptonamefalsefalsefalsefalsefalsefalsefalsefalsefalsefalse#1
  \fi
  \BabelEnsureInfo
\bbl@bcp\namenamefalse
\namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{
  \directlua{ Babel.ignore_pre_char = function(node)
    return (node.lang == \the\csname l@nohyphenation\endcsname)
  end }}
\namedef{bbl@ADJ@prehyphenation.disable@off}{
  \directlua{ Babel.ignore_pre_char = function(node)
    return false
  end }}
\namedef{bbl@ADJ@select.write@shift}{
  \let\bbl@restorelastskip\relax
  \def\bbl@savelastskip{\let\bbl@restorelastskip\relax
    \ifvmode
      \ifdim\lastskip=\z@
        \let\bbl@restorelastskip\nobreak
      \else
        \bbl@exp{\let\bbl@restorelastskip\relax
          \nobreak \vskip-\skip@ \vskip\skip@}}%
    \fi
  \fi}
\namedef{bbl@ADJ@select.write@keep}{
  \let\bbl@restorelastskip\relax
  \let\bbl@savelastskip\relax}
\namedef{bbl@ADJ@select.write@omit}{
  \let\bbl@restorelastskip\relax
  \def\bbl@savelastskip##1\bbl@restorelastskip{}%}

As the final task, load the code for lua. TODO: use babel name, override
\if\directlua\@undefined\else
  \if\bbl@luapatterns\@undefined
    \input luababel.def
  \fi
\fi

Continue with \LaTeX.
\langle \package | core \rangle
\langle \package \rangle

9.1 Cross referencing macros

The \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.

(\langle \More package options \rangle) \equiv
\DeclareOption{safe=none}{\let\bbl@opt@safe@empty}
\DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
\DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
(\langle \More package options \rangle)
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.

\@newl@bel \trace{Cross referencing macros}
\ifx\bbl@opt@safe@empty\else
\let\@newl@bel#1#2#3=%
\relax
\ifdef\@multiplelabels{%
\@latex@warning@no@line{There were multiply-defined labels}}%
\@latex@warning@no@line{Label `#2' multiply defined}}%
\global\@namedef{#1@#2}{#3}}

\@testdef An internal \TeX macro used to test if the labels that have been written on the .aux file have changed. It is called by the \enddocument macro.
\def\@testdef#1#2#3{% TODO. With @samestring?
\@safe@actives true
\expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
\def\bbl@tempb{#3}\
\@safe@actives false
\ifx\bbl@tempa\relax
\else
\edef\bbl@tempa{\strip@prefix\meaning\bbl@tempa}\fi
\edef\bbl@tempb{\strip@prefix\meaning\bbl@tempb}\ifx\bbl@tempa\bbl@tempb
\else\fi
\fi
\fi
\ref 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.
\def\ref#1{% TODO. With @samestring?
\@safe@actives true
\expandafter\let\expandafter\bbl@tempa\csname #1\endcsname
\def\bbl@tempb{#1}\
\@safe@actives false
\ifx\bbl@tempa\relax
\else
\edef\bbl@tempa{\strip@prefix\meaning\bbl@tempa}\fi
\edef\bbl@tempb{\strip@prefix\meaning\bbl@tempb}\ifx\bbl@tempa\bbl@tempb
\else\fi
\let\org@ref\ref
\let\org@pageref\pageref
\fi

The macro \citex 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.

\begin{verbatim}
\edef\@tempa{#3}\@safe@activesfalse\org@@citex[#1]{\@tempa}
\end{verbatim}

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.

\begin{verbatim}
\def\@citex[#1]#2{\@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse\org@@citex[#1]{#2}}\end{verbatim}

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

\begin{verbatim}
\ifpackageloaded{cite}{\def\@citex[#1]{\@safe@activestrue\org@@citex[#1]{#1}\@safe@activesfalse}}{}
\end{verbatim}

The macro \nocite which is used to instruct BiBTeX to extract uncited references from the database.

\begin{verbatim}
\def\nocite#1{\@safe@activestrue\org\nocite{#1}\@safe@activesfalse}
\end{verbatim}

The macro \bibcite 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.

\begin{verbatim}
\def\bibcite{\bbl@bibcite\bbl@cite@choice\bibcite}
\end{verbatim}

\begin{verbatim}
\def\bibcite{\bbl@bibcite\bbl@cite@choice\bibcite}
\end{verbatim}

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

\begin{verbatim}
\edef\@tempa{#3}\@safe@activesfalse\org@bibcite{#1}{\@safe@activesfalse#2}
\end{verbatim}

The macro \bbl@cite@choice determines which definition of \bibcite is needed. First we give \bibcite its default definition.

\begin{verbatim}
\def\bbl@cite@choice{}
\end{verbatim}
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.

\AtBeginDocument{\bbl@cite@choice}

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.)

\Ifx@\@mkboth\markboth
\def\bbl@tempc{\let\@mkboth\markboth}
\else
\def\bbl@tempc{}
\fi

\markboth

\IfBabelLayout{sectioning}{
\ifx\bbl@opt@headfoot\@nnil
\g@addto@macro\@resetactivechars{%
\set@typeset@protect
\select@language@x\select@language{\bbl@main@language}%
\let\protect\noexpand
\ifcase\bbl@bidimode\else % Only with bidi. See also above
\edef\thepage{%
\noexpand\babelsublr{\unexpanded{\thepage}}}%
\fi}%
\fi}\fi}

\IfBabelLayout{sectioning}{
\if\bbl@single\else
\bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
\markright#1{%
\bbl@ifblank{#1}{\org@markright{}}{
\toks@{#1}\bbl@exp{%
\bbl@exp}%
\bbl@exp%}
\bbl@exp%
\bbl@exp%
\bbl@exp%
}
\fi}%

\markboth

\IfBabelLayout{sectioning}{
\if\bbl@single\else
\bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
\markright#1{%
\bbl@ifblank{#1}{\org@markright{}}{
\toks@{#1}\bbl@exp{%
\bbl@exp%}
\bbl@exp%
\bbl@exp%
\bbl@exp%
}
\fi}%

\markright
9.3 Preventing clashes with other packages

9.3.1 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 \isodd 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.

```
\bbl@trace{Preventing clashes with other packages}
\bbl@xin\{R}\bbl@opt@safe
\ifin@ \AtBeginDocument{%}
  \@ifpackage{ifthen}{%
    \bbl@redefine@long\ifthenelse#1#2#3{%}
    \let\bbl@temp@pref\pageref
    \let\pageref\org@pageref
    \let\bbl@temp@ref\ref
    \let\ref\org@ref
    \@safe@activestrue
    \org@ifthenelse{#1}%
    {\let\pageref\bbl@temp@pref
     \let\ref\bbl@temp@ref
     \@safe@activesfalse
     #2}%
    {\let\pageref\bbl@temp@pref
     \let\ref\bbl@temp@ref
     \@safe@activesfalse
     #3}%
  }%}
}```
9.3.2 varioref

When the 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.

\AtBeginDocument{%
  \ifpackageloaded{varioref}{%
    \@safe@activetrue
    \org@@vpageref{#1}{#2}{#3}{%}
    \@safe@activefalse%}
  \bbl@redefine\@@vpageref#1[#2]{%\@safe@activetrue
    \org@@vpageref{#1}{#2}{#3}{%}
    \@safe@activefalse}%
  \bbl@redefine\vrefpagenum#1{#2}{%\@safe@activetrue
    \org@vrefpagenum{#1}{#2}{%}
    \@safe@activefalse}%
}\fi

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{\protect\edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}{}
\fi

9.3.3 hhline

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}\
    {\expandafter\ifx\csname normal@char\string:\endcsname\relax
      \else
      \makeatletter
      \def\@currname{hhline}\input{hhline.sty}\makeatother
      \fi}
  \fi\}
}\fi

\substitutefontfamily

Deprecated. Use the tools provided by \LaTeX. 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.

\def\substitutefontfamily#1#2#3{\lowercase{\immediate\openout15=#1#2.fd}\relax%
\immediate\write15{\string\ProvidesFile{#1#2.fd}[	he\year/\two@digits{\the\month}/\two@digits{\the\day}]
\space generated font description file]^^J
\string\DeclareFontFamily{#1}{#2}{}^^J
\string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
\string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
\string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
\string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J

146
9.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of \TeX and \LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Requested encodings are currently stored in \@fontenc@load@list. If a non-ASCII has been loaded, we define versions of \TeX and \LaTeX for them using \ensureascii. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

\ensureascii

\latinencoding

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 encoding to use.
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.

\begin{verbatim}
\AtBeginDocument{% 
@ifpackageloaded{fontspec}% 
{\xdef\latinencoding{% 
  \ifdef\UTFencname\undefined
  EU\ifcase\bbl@engine\or2\or1\fi
  \else
    \UTFencname
  \fi}
  OT1}% 
\ifdef\cf@encoding\bbl@t@one
  \xdef\latinencoding{\bbl@t@one}% 
\else
  \ifdef\elt{,T1,}.bbl@tempa
    \let\elt\relax
    \bbl@xin{,T1,}.bbl@tempa
    \ifin@
      \xdef\latinencoding{\bbl@t@one}%
    \fi
  \fi%
}\latinencoding{OT1}%
\ifdef\cf@encoding\bbl@t@one
  \xdef\latinencoding{\bbl@t@one}%
\else
  \ifdef\elt{,T1,}.bbl@tempa
    \let\elt\relax
    \bbl@xin{,T1,}.bbl@tempa
    \ifin@
      \xdef\latinencoding{\bbl@t@one}%
    \fi
  \fi%
\latinencoding{OT1}%

\fontencoding{\latinencoding}\selectfont
\def\encodingdefault{\latinencoding}
\end{verbatim}

\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.

\\begin{verbatim}
\ifdef\undefined\DeclareTextFontCommand
  \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
\else
  \DeclareTextFontCommand{\textlatin}{\latintext}
\fi
\end{verbatim}

For several functions, we need to execute some code with \selectfont. With \LaTeXX 2021-06-01, there
is a hook for this purpose, but in older versions the \LaTeX command is patched (the latter solution will
be eventually removed).

9.5 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 Youssef
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 r1babel 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{}-ja shows, vertical typesetting is possible, too.

3792 \bbl@trace{Loading basic (internal) bidi support}
3793 \ifodd\bbl@engine
3794 \else % TODO. Move to txtbabel
3795 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3796 \bbl@error
3797 \{The bidi method 'basic' is available only in\%
3798 luatex. I'll continue with 'bidi=default', so\%
3799 expect wrong results}\
3800 \{See the manual for further details.\%
3801 \let\bbl@beforeforeign\leavevmode
3802 \AtEndOfPackage{%
3803 \EnableBabelHook{babel-bidi}%
3804 \bbl@xebidipar}
3805 \fi\fi
3806 \define\bbl@loadxebidi#1{%
3807 \ifx\RTLfootnotetext\@undefined
3808 \AtEndOfPackage{%
3809 \EnableBabelHook{babel-bidi}%
3810 \ifx\fontspec\@undefined
3811 \bbl@loadfontspec % bidi needs fontspec
3812 \fi
3813 \usepackage#1{bidi}\
3814 \fi}
3815 \ifnum\bbl@bidimode=200
3816 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode or
3817 \bbl@tentative{bidi=bidi}\
3818 \bbl@loadxebidi{}\
3819 \or
3820 \bbl@loadxebidi[{rldocument}]
3821 \or
3822 \bbl@loadxebidi{}
3823 \fi
3824 \fi
3825 \fi
3826 % TODO? Separate:
3827 \ifnum\bbl@bidimode=\@ne
3828 \let\bbl@beforeforeign\leavevmode
3829 \ifodd\bbl@engine
3830 \newattribute\bbl@attr@dir
3831 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir'}
3832 \bbl@exp{\output{\bodydir\pagedir\the\output}}
3833 \fi
3834 \AtEndOfPackage{%
3835 \EnableBabelHook{babel-bidi}%
3836 \ifodd\bbl@engine\else
Now come the macros used to set the direction when a language is switched. First the (mostly) common macros.

```latex
\bbl@trace{Macros to switch the text direction}
def\bbl@alscripts{,Arabic,Syriac,Thai,}
def\bbl@rscripts{\% TODO. Base on codes ??
  ,Imperial Aramaic,Avestan,Cypriotic,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@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}\%
  \ifin@\global\bbl@csarg\chardef{wdir@#1}\@ne
  \bbl@xin{\csname bbl@sname@#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 % luatex=1
    \bbl@dirlevel
    \chardef\bbl@thetextdir\z@
    \chardef\bbl@thepardir\z@
    \def\bbl@textdir#1{\ifcase#1\relax\chardef\bbl@thetextdir\z@\bbl@textdir@i\beginL\endL\%}%
  \or\def\bbl@textdir#1{\ifcase#1\relax\chardef\bbl@thetextdir\z@\bbl@textdir@i\beginL\endL\%}%
  \or\def\bbl@textdir#1{\ifcase#1\relax\chardef\bbl@thetextdir\z@\bbl@textdir@i\beginL\endL\%}%
  \fi\}
  \def\bbl@switchdir{%}
    \bbl@funset{\bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}{}%}
    \bbl@funset{\bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}{}%}
    \bbl@exp{\\bbl@setdirs\bbl@cl{wdir}}\%}
  \def\bbl@setdirs#1{% TODO - math
    \iffalse\bbl@select@type \% TODO - strictly, not the right test
      \bbl@bodydir{/1}%
    \else \bbl@pardir{#1}%
    \fi\}
  \bbl@textdir{#1} \% TODO - only if \bbl@bidimode > 0?
  \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
  \DisableBabelHook{babel-bidi}
}
```

Now the engine-dependent macros. TODO. Must be moved to the engine files.

```latex
\bbl@xelatex\% luatex=1
\else \pdftex=0, xetex=2
\newcount\bbl@dirlevel
\chardef\bbl@thetextdir\z@
\chardef\bbl@thepardir\z@
\def\bbl@textdir\beginL\endL\z@\%}
\def\bbl@switchdir{%}
  \ifodd\bbl@engine \% luatex=1
    \bbl@dirlevel
    \chardef\bbl@thetextdir\z@
    \chardef\bbl@thepardir\z@
    \def\bbl@textdir\beginL\endL\z@\%}
  \else \pdftex=0, xetex=2
    \newcount\bbl@dirlevel
    \chardef\bbl@thetextdir\z@
    \chardef\bbl@thepardir\z@
    \def\bbl@textdir\beginL\endL\z@\%}
  \fi\}
  \bbl@textdir{#1} \% TODO. Only if \bbl@bidimode > 0?
  \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
  \DisableBabelHook{babel-bidi}
```

Now the engine-dependent macros. TODO. Must be moved to the engine files.
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).

\def\bbl@xebidipar{%
  \let\bbl@xebidipar\relax
  \TeXXeTstate\one
  \def\bbl@xeeverypar{%
    \ifcase\bbl@theparidir
      \ifcase\bbl@thetextdir\else\beginR\fi
    \else
      \{\setbox\z@\lastbox\beginR\box\z@}\%
    \fi\%
    \let\bbl@severypar\everypar
    \newtoks\everypar
    \everypar=\bbl@severypar
  \bbl@severypar\bbl@xeeverypar\{\bbl@xeeverypar\the\everypar}\%
  \ifnum\bbl@bidimode>200%
    \let\bbl@textdir@i@\empty
  \AddBabelHook{bidi}{foreign}{%
    \def\BabelText####1}{%
    \bbl@textdir@i\beginR\endR
  \bbl@textdir@i\bbl@textdir@i\beginR\endR
  \fi}
  \fi
}\bbl@dirlevel\currentgrouplevel
  \fi
  \bbl@dirlevel\currentgrouplevel
\fi
\def\bbl@pardir#1{%
  \chardef\bbl@thepardir#1\relax
  \let\bbl@bodydir\@gobble
  \let\bbl@pagedir\@gobble
  \def\bbl@dirparastext{%
    \chardef\bbl@thetextdir\bbl@textdir@i\bbl@textdir@i\beginR\endR
  \fi
}\bbl@pardir@i{I'll insert a new group, but expect wrong results.}%
\bgroup\aftergroup\egroup
\else
  \ifcase\currentgrouptype\or % 0 bottom
    \aftergroup\bgroup\aftergroup\egroup % 2 hbox
  \or
    \bgroup\aftergroup\egroup % 3 adj hbox
  \or
    \bgroup\aftergroup\egroup % 7 noalign
  \or
    \bgroup\aftergroup\egroup % output math disc insert vcent mathchoice
  \or
    \aftergroup\bgroup \begingroup % 14 \begingroup
  \else
    \bgroup\aftergroup\egroup % 15 adj
  \fi
  \bbl@dirparastext
\fi
\bgroup\aftergroup\egroup % %
\makeatother
\overfullrule=1pt
\hbox{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).}

\def\bbl@xebidipar{%
  \let\bbl@xebidipar\relax
  \TeXXeTstate\one
  \def\bbl@xeeverypar{%
    \ifcase\bbl@theparidir
      \ifcase\bbl@thetextdir\else\beginR\fi
    \else
      \{\setbox\z@\lastbox\beginR\box\z@}\%
    \fi\%
    \let\bbl@severypar\everypar
    \newtoks\everypar
    \everypar=\bbl@severypar
  \bbl@severypar\bbl@xeeverypar\{\bbl@xeeverypar\the\everypar}\%
  \ifnum\bbl@bidimode>200%
    \let\bbl@textdir@i@\empty
  \AddBabelHook{bidi}{foreign}{%
    \def\BabelText####1}{%
    \bbl@textdir@i\beginR\endR
  \bbl@textdir@i\bbl@textdir@i\beginR\endR
  \fi}
  \fi
}\bbl@pardir@i{I'll insert a new group, but expect wrong results.}%
\bgroup\aftergroup\egroup
\else
  \ifcase\currentgrouptype\or % 0 bottom
    \aftergroup\bgroup\aftergroup\egroup % 2 hbox
  \or
    \bgroup\aftergroup\egroup % 3 adj hbox
  \or
    \bgroup\aftergroup\egroup % 7 noalign
  \or
    \bgroup\aftergroup\egroup % output math disc insert vcent mathchoice
  \or
    \aftergroup\bgroup \begingroup % 14 \begingroup
  \else
    \bgroup\aftergroup\egroup % 15 adj
  \fi
  \bbl@dirparastext
\fi
\bgroup\aftergroup\egroup % %
\makeatother
\overfullrule=1pt
\hbox{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).}
9.6 Local Language Configuration

\loadlocalcfg

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.ldf is loaded.

For plain-based formats we don’t want to override the definition of \loadlocalcfg from plain.def.

9.7 Language options

Languages are loaded when processing the corresponding option except if a main language has been set. In such a case, it is not loaded until all options has been processed. The following macro inputs the ldf file and does some additional checks (\input works, too, but possible errors are not caught).
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.

```latex
\def\bbl@try@load@lang#1#2#3{%
\IfFileExists{\CurrentOption.ldf}%%
{\bbl@load@language{\CurrentOption}}%
{#1\bbl@load@language(#2)#3}%%
\}

\DeclareOption{hebrew}{%
\input{rlbabel.def}%
\bbl@load@language{hebrew}}%
\DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
\DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
\DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
\DeclareOption{polutonikogreek}{%
\bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}%
\DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
\DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
\DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}

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.

\ifx\bbl@opt@config\@nnil
@ifpackagewith{babel}{noconfigs}{}%
{\InputIfFileExists{bblopts.cfg}%
{\typeout{*************************************\%*
Local config file bblopts.cfg used\%^J\%
*})}%
{}}%
\else
\InputIfFileExists{\bbl@opt@config.cfg}%
{\typeout{*************************************\%
Local config file \bbl@opt@config.cfg used\%^J\%
*})}%
{\bbl@error{%
Local config file \`\bbl@opt@config.cfg' not found}{%
Perhaps you misspelled it.}}%
\fi
```

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 filesystem just to see if the option could be a language.

If a main language has been set, store it for the third pass.

{%}
\let\bbl@opt@main\@nnil
\let\bbl@clsoptlst\@classoptionslist
\bbl@foreach\@classoptionslist{%}
\let\bbl@iniflag\@nnil
{%\IfFileExists{#1.1df}%
{\def\bbl@tempc{\@nnil}%
\DeclareOption{#1}{%
  \ifnum\bbl@iniflag>\@ne
  \bbl@ldfinit
  \bbl@loadlanguage{#1}%
  \fi}%
  }
}{%\IfFileExists{babel-#1.tex}%
{\def\bbl@tempc{\@nnil}%
\ifnum\bbl@iniflag>\z@
{\def\bbl@tempb{#1}%
\DeclareOption{#1}{%
  \ifnum\bbl@iniflag>\@ne
  \bbl@ldfinit
  \bbl@loadlanguage{#1}%
  \fi}%
}{%}
}{%}
}%

{}}%
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 \LaTeX processes before):

\AfterBabelLanguage{#1}%
\if\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'}
\if\bbl@opt@main\@nil
\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
\if\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 can't be set as both a global\%
and a package option. Use 'main=\bbl@tempc' as\%
option. Reported}%
\fi
\else
\ifodd\bbl@iniflag % case 1,3
\bbl@ldfinit
\let\CurrentOption\bbl@opt@main
\ifs\bbl@opt@provide\@nil
\bbl@exp{\bbl@babelprovide[import,main]{\bbl@opt@main}}%
\else
\bbl@exp{\bbl@forkv{@nameuse{@raw@opt@babel.sty}}}{%\bbl@xin@{,provide,}{,#1,}%
\ifi
\def\bbl@opt@provide{#2}%
\bbl@replace\bbl@opt@provide{;}{,}%
\fi}
\bbl@exp{\bbl@babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
\fi
\bbl@afterldf{}%
\else % case 0,2
\chardef\bbl@iniflag\z@ % Force ld
\expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain

155
\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.
\ifx\bbl@main@language\@undefined
\bbl@info{%
You haven't specified a language. I'll use 'nil'\%
as the main language. Reported}
\bbl@load@language{nil}
\fi

\section{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.
A proxy file for switch.def
\let\bbl@onlyswitch\@empty
\input babel.def
\let\bbl@onlyswitch\@undefined

\section{Loading hyphenation patterns}

The following code is meant to be read by \ini\TeX because it should instruct \TeX to read hyphenation
patterns. To this end the docstrip option patterns is used to include this code in the file
hyphen.cfg. Code is written with lower level macros.
\ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
xdef\bbl@format{\jobname}
xdef\bbl@version{⟨⟨version⟩⟩}
xdef\bbl@date{⟨⟨date⟩⟩}
\ifx\AtBeginDocument\@undefined
\def\@empty{}
\fi

⟨⟨Define coreswitching macros⟩⟩
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.

```latex
\def\process@line#1#2 #3 #4 {%
  \ifx=#1%
    \process@synonym{#2}%
  \else
    \process@language(#1#2){#3}{#4}%
  \fi
  \ignorespaces}
```

\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.

```latex
\def\toks@{}
\def\bbl@languages{}

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 synonyms 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.

```latex
\def\process@synonym#1{%
  \ifnum\last@language=\m@ne
    \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
  \else
    \expandafter\chardef\csname l@#1\endcsname\last@language
    \wlog{\string\l@#1=\string\language\the\last@language}%
    \expandafter\let\csname #1hyphenmins\expandafter\endcsname
      \csname\languagename hyphenmins\endcsname
    \let\bbl@elt\relax
    \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
  \fi
}
```

\process@language The macro \process@language is used to process a non-empty line from the ‘configuration file’. It has three arguments, 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 ‘\texttt{T1}’ to the name of the language.

The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@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 \langle lang\rangle hyphenmins macro. 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.

Then we globally store the settings of \lefthyphenmin and \righthyphenmin and close the group. 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.)

\bbl@languages saves a snapshot of the loaded languages in the form `\bbl@elt\langle language-name\rangle\langle number\rangle \langle patterns-file\rangle\langle exceptions-file\rangle`. 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.

\def\process@language#1#2#3{%
  \expandafter\addlanguage\csname l@#1\endcsname
  \expandafter\language\csname l@#1\endcsname
  \edef\languagename{#1}\
  \bbl@hook@everylanguage{#1}%
  \begingroup
    \lefthyphenmin\m@ne
    \bbl@hook@loadpatterns{#2}%
    % > luatex
    \bbl@get@enc#1::\@@@@
    \ifnum\lefthyphenmin=\m@ne
      \else
        \expandafter\expandafter\expandafter\set@hyphenmins
        \csname #1hyphenmins\endcsname
    \fi
    \endgroup
    \def\bbl@tempa{#3}%
    \ifx\bbl@tempa\@empty
      \else
        \bbl@hook@loadexceptions{#3}%
    \fi
    \let\bbl@elt\relax
    \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
    \ifnum\the\language=\z@
      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
        \set@hyphenmins\tw@\thr@@\relax
      \else
        \expandafter\expandafter\expandafter\set@hyphenmins
        \csname #1hyphenmins\endcsname
      \fi
      \the\toks@
      \toks@{}%
    \fi
  }

\bbl@get@enc
\bbl@hyph@enc

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.

4221 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}

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.

4222 \def\bbl@hook@everylanguage{}%
4223 \def\bbl@hook@loadpatterns{\input \#1\relax}
4224 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4225 \def\bbl@hook@loadkernel{}%
4226 \def\addlanguage{\csname newlanguage\endcsname}%
4227 \def\adddialect##1##2{%
  \global\chardef##1##2\relax
  \wlog{\string##1 = a dialect from \string\language##2}%
4228 \def\ifianguage##1{%
  \@nolanerr{##1}%
  \else
    \expandafter\expandafter\expandafter\@firstoftwo
    \ifnum\the\language=\z@
      \else
        \expandafter\expandafter\expandafter\@firstoftwo
        \ifnum\csname #1hyphenmins\endcsname=\m@ne
          \else
            \expandafter\expandafter\expandafter\set@hyphenmins
            \csname #1hyphenmins\endcsname
          \fi
          \the\toks@
          \toks@{}%
        \fi
      \fi
    \fi
  \fi}
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

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 \texttt{-1}.

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.

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.

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.

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

12 Font handling with fontspec

Add the bidi handler just before luatolatex, 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].

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. \bblfont replaces hardcoded font names inside \..family by the corresponding macro \..default. At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading message, which is replaced by a more explanatory one.
\ifx\fontspec\@undefined
  \bbl@loadfontspec
\fi
\EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
\bbl@bblfont}
\newcommand\bbl@bblfont[2]{% 1=features 2=fontname, @font=rm|sf|tt
  \bbl@ifunset{\bbl@tempb family}\
  {\bbl@providefam{\bbl@tempb family}}%
\}
% For the default font, just in case:
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\expandafter\bbl@ifblank\expandafter{\bbl@tempa}\
  \def{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
  \bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
    \bbl@csarg\edef{\bbl@tempb dflt@##1}{<>{#1}{#2}}}

If the family in the previous command does not exist, it must be defined. Here is how:
\def\bbl@providefam#1{%
  \bbl@exp{%\def#1default{}% Just define it
    \bbl@add@list\bbl@font@fams{#1}%
    \DeclareRobustCommand{#1family}{%\not@math@alphabet{#1default}% TODO. Fails
      \fontfamily{#1default}%
      \ifx>\UseHooks\@undefined<else>\UseHook{#1family}<fi>%
      \selectfont}
    \DeclareTextFontCommand{<text#1>}{#1family}}}

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 duplicate them.
\def\bbl@nostdfont#1{%
  \bbl@addlist{\bbl@font@fams}{#1}%
  \DeclareRobustCommand{#1family}{%\not@math@alphabet{#1family}\relax
    % % %\prepare@family@series@update{#1}{#1default}% TODO. Fails
    % \fontfamily{#1family}{#1default}%
    %\ifx>\UseHooks\@undefined<else>\UseHook{#1family}<fi>%
    \selectfont}
  \DeclTextFontCommand{<text#1>}{#1family}{}
}

% The current font is not a babel standard family:\%
#1%
\fontfamily{#1}\%
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 \protect\babelfont.\%
See the manual for further details about \protect\babelfont.\%
\Reported}
%
}%
gdef\bbl@switchfont{%
  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
  \bbl@exp{% eg Arabic -> arabic
    \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}%\bbl@cmd{#1default}{#1}1
  }
  \bbl@foreach\bbl@font@fams{%
    \bbl@ifunset{bbl@##1dflt@}{\bbl@cmd{##1dflt@*}{#1}2}
    \bbl@ifunset{bbl@##1dflt@}{\bbl@cmd{##1dflt@}{#1}3}
      {\bbl@cmd{##1dflt@}{#1}4}
  }
  \bbl@exp{% 1=features 2=fontname, @font=rm|sf|tt
    \bbl@ifunset{\bbl@tempb family}\
    {\bbl@providefam{\bbl@tempb family}}%
\}
% For the default font, just in case:
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\expandafter\bbl@ifblank\expandafter{\bbl@tempa}\
  \def{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
  \bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
    \bbl@csarg\edef{\bbl@tempb dflt@##1}{<>{#1}{#2}}}

If the family in the previous command does not exist, it must be defined. Here is how:
\def\bbl@providefam#1{%
  \bbl@exp{%\def#1default{}% Just define it
    \bbl@add@list\bbl@font@fams{#1}%
    \DeclareRobustCommand{#1family}{%\not@math@alphabet{#1default}% TODO. Fails
      \fontfamily{#1default}%
      \ifx>\UseHooks\@undefined<else>\UseHook{#1family}<fi>%
      \selectfont}
    \DeclareTextFontCommand{<text#1>}{#1family}}}

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 duplicate them.
\def\bbl@nostdfont#1{%
  \bbl@addlist{\bbl@font@fams}{#1}%
  \DeclareRobustCommand{#1family}{%\not@math@alphabet{#1family}\relax
    % % %\prepare@family@series@update{#1}{#1default}% TODO. Fails
    % \fontfamily{#1family}{#1default}%
    %\ifx>\UseHooks\@undefined<else>\UseHook{#1family}<fi>%
    \selectfont}
  \DeclTextFontCommand{<text#1>}{#1family}{}
}

% The current font is not a babel standard family:\%
#1%
\fontfamily{#1}\%
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 \protect\babelfont.\%
See the manual for further details about \protect\babelfont.\%
\Reported}
%
}%
\global\let\bbl@##1dflt@\languagename\%
\global\let\bbl@##1dflt@\{\bbl@exp\%
\global\let\bbl@##1dflt@{\bbl@exp{2=T - from script
\global\let\bbl@##1dflt@{\bbl@exp\%
\global\let\bbl@##1dflt@{\bbl@exp{1=T - language, already defined\}
\def\bbl@tempa\{\bbl@nostdfont\{\%
\bbl@foreach\bbl@font@fams{% don't gather with prev for
\bbl@ifunset\{bbl@##1dflt@\languagename\%
\bbl@cs\{famrst@##1\}%
\global\bbl@csarg\let\{famrst@##1\}\relax%
{\bbl@exp{% order is relevant. TODO: but sometimes wrong!
\bbl@font@rst\{bbl@cl{##1dflt}@\%
\bbl@font@set\{bbl@##1dflt@\languagename\%
\bbl@ifrestoring{}{\bbl@tempa}}%
\bbl@ifunset\{bbl@##1dflt@\languagename\%
\bbl@cs\{famrst@##1\}%
\global\bbl@csarg\let\{famrst@##1\}\relax%
\bbl@foreach\bbl@font@fams{% the main part!
\bbl@font@rst\{bbl@cl{##1dflt}@\%
\bbl@font@set\{bbl@##1dflt@\languagename\%
\bbl@ifrestoring{}{\bbl@tempa}}%
The following is executed at the beginning of the aux file or the document to warn about fonts not
defined with \babelfont.
\ifx\@family\@undefined\else % if latex
  \ifcase\bbl@engine % if pdftex
    \let\bbl@ckeckstdfonts\relax
  \else
    \def\bbl@ckeckstdfonts\{\begingroup
      \global\let\bbl@ckeckstdfonts\relax
      \let\bbl@tempa\@empty
      \bbl@foreach\bbl@font@fams{% Flag
        \bbl@ifunset\{bbl@##1{\family}\%
        \bbl@csarg\gdef\{WFF@\f@family\}{}%
        \bbl@exp{\bbl@add\{\bbl@tempa{* \f@family=\f@family\\%\space\space\fontname\font\\}}%
        \bbl@csarg\xdef\{##1dflt@\}{\f@family}%
        \expandafter\xdef\csname ##1default\endcsname\{\f@family\}%
      }}%
      \ifx\bbl@tempa\@empty\else
        \bbl@infowarn\{The following font families will use the default\%
        settings for all or some languages:\%
        \bbl@tempa
        There is nothing intrinsically wrong with it, but\%
        'babel' will no set Script and Language, which could\%
        be relevant in some languages. If your document uses\%
        these families, consider redefining them with \string\babelfont.\%
        Reported\%
      \fi
    \endgroup}
  \fi
\fi
\bbl@font@set{% eg \bbl@rmdefault \rmfamily
  \bbl@xin\{<>\}##1\%
  \ifin@
\bbl@exp{\bbl@fontspec@set\#1\expandafter\@gobbletwo#1\#3}%
\fi
\bbl@exp{% 'Unprotected' macros return prev values
\def\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
\bbl@ifsamestring{#2}{\f@family}%
{\#3%
\bbl@ifsamestring{{\f@series}}{\bfdefault}{}%\fseries
\let\bbl@tempa\relax}}%
}
% TODO - next should be global?, but even local does its job. I'm
% still not sure -- must investigate:
\def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
\let\bbl@tempe\bbl@mapselect
\let\bbl@mapselect\relax
\let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
\let\#4\@empty % Make sure \renewfontfamily is valid
\bbl@exp{%
\let\bbl@temp@pfam<\bbl@stripslash#4\space>% eg, '\rmfamily '
<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%\
ewfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}%<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%\
ewfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}%\enewfontfamily\#4%
[\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{\#3}
\begingroup
#4%
\xdef#1{\f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
\endgroup
\let\#4\bbl@temp@fam
\bbl@exp{\let<\bbl@stripslash#4\space>\bbl@temp@pfam}
\let\bbl@mapselect\bbl@tempe}%

\def\bbl@font@rst#1#2#3#4{%
\bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}%

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

\begin{verbatim}
\def\BabelFootnote#1#2#3#4{% 
  \ifx\bbl@fn@footnote\@undefined
    \let\bbl@fn@footnote\footnote
  \fi
  \ifx\bbl@fn@footnotetext\@undefined
    \let\bbl@fn@footnotetext\footnotetext
  \fi
  \bbl@ifblank{#2}%
    \def#1{\bbl@footnote{\@firstofone}{#3}{#4}}%
    \@namedef{\bbl@stripslash#1text}\
    \bbl@footnotetext{\@firstofone}{#3}{#4}}%
  \else
    \def#1{\bbl@exp{\bl@footnote{\oreignlanguage{#2}}}{#3}{#4}}%
    \@namedef{\bbl@stripslash#1text}\
    \bbl@exp{\bl@footnotetext{\oreignlanguage{#2}}}{#3}{#4}}
\fi
\end{verbatim}
\ifx\bbl@KVP@intrapenalty\@nil\else
\expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@\fi
\bbl@exp{%
% TODO. Execute only once (but redundant):
\\bbl@add{\languagename}{%}
\XeTeXlinebreaklocale "\bbl@cl{tbcp}%
\bbl@add{\languagename}{%}
\XeTeXlinebreaklocale "en}%
\bbl@add{\languagename}{%}
\bbl@add{\noextras\languagename}{%}
\XeTeXlinebreaklocale "en"}%
\bbl@add{\noextras\languagename}{%}
\\bbl@add{\languagename}{%}
\bbl@add{\noextras\languagename}{%}
\\bbl@add{\languagename}{%}
\\bbl@add{\noextras\languagename}{%}
\\gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
\ifx\AtBeginDocument\@notprerr
\expandafter\@secondoftwo % to execute right now
\fi
\AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
\fi
\fi}
\ifx\DisableBabelHook\@undefined\endinput\fi
\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
\DisableBabelHook{babel-fontspec}
⟨⟨Fontselection⟩⟩
\input\txtbabel.def
⟨/xetex⟩

13.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, 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 \texttt{tex–xetbabel}, which is the bidi model in both pdftex and xetex.
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 tablestack). 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 babel1.def, with the commands and other definitions for luatex (eg, \bapelstpatterns).

4739 \langle\texttt{luatex}\rangle
4740 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4741 \bbl@trace{Read language.dat}
4742 \ifx\bbl@readstream\@undefined
4743 \csname newread\endcsname\bbl@readstream
4744 \fi
4745 \begingroup
4746 \toks@{}
4747 \count@0 % 0=start, 1=0th, 2=normal
4748 \def\bbl@process@line#1#2 #3 #4 {%\texttt{\%}
4749 \ifa@1%
4750 \bbl@process@synonym{#2}%
4751 \else
4752 \bbl@process@language{#1#2}{#3}{#4}%
4753 \fi
4754 \ignorespaces
4755 \def\bbl@manylang{%
4756 \ifnum\bbl@last>\@ne one
4757 \bbl@info{Non-standard hyphenation setup}%
4758 \fi
4759 \let\bbl@manylang\relax
4760 \def\bbl@process@language{#1#2#3}{%
4761 \ifcase\count@
4762 \ifnum\bbl@language{zth}@1\{"\count@<\tw@}{\count@<\@ne}%
4763 \or

\count@\tw@
\fi
\ifnum\count@=\tw@
\expandafter\addlanguage\csname l@#1\endcsname
\language\allocationnumber
\chardef\bbl@last\allocationnumber
\bbl@manylang
\let\bbl@elt\relax
\xdef\bbl@languages{%
\bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
\fi
\the\toks@
\toks@{}}
\def\bbl@process@synonym@aux#1#2{%
\global\expandafter\chardef\csname l@#1\endcsname#2\relax
\let\bbl@elt\relax
\xdef\bbl@languages{%
\bbl@languages\bbl@elt{#1}{#2}{}{}}}%
\def\bbl@process@synonym#1{%
\ifcase\count@
\toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
\or
\ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}
\else
\bbl@process@synonym@aux{#1}{\the\bbl@last}%
\fi
\fi%
\def\bbl@languages\undefined % Just a (sensible?) guess
\chardef\l@english\z@
\chardef\l@USenglish\z@
\chardef\bbl@last\z@
\global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}%}
\global\gdef\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%
\fi
\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\space}\%
\expandafter\bbl@process@line\bbl@line\relax
\fi
\repeat
\end
\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
\newcatcodetable\bbl@pattcodes
\fi
\else
\def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
\fi
\def\bbl@luapatterns#1#2{%
def\bbl@get@enc#1::@@\setbox\z@\hbox\bgroup\begingroup\savecatcodetable\babelcatcodetablenum\relax
\initcatcodetable\bbl@pattcodes\relax
\catcodetable\bbl@pattcodes\relax
\catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
\catcode`\_=8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
\catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
\catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
\catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
\input #1\relax
\catcodetable\babelcatcodetablenum\relax
\def\bbl@tempa{#2}\
\ifx\bbl@tempa\@empty
\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
\savecatcodetable\babelcatcodetablenum\relax
\catcodetable\bbl@pattcodes\relax
\def\bbl@tempa{#2}\
\ifx\bbl@tempa\@empty\else
\input #2\relax
\fi
\csname bbl@hyphendata@\the\language\endcsname}}{{}}
\endinput\fi
% Here ends \if\AddBabelHook@undefined
% A few lines are only read by hyphen.cfg
\AddBabelHook{luatex}{everylanguage}{%
\def\process@language##1##2##3{%
\def\process@line####1####2 ####3 ####4 {}}}
\AddBabelHook{luatex}{loadpatterns}{%
\input #1\relax
\expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
{{#1}{}}}
\AddBabelHook{luatex}{loadexceptions}{%
\input #1\relax
\def\bbl@tempb##1##2{{##1}{#1}}%
\expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
{%
\csname bbl@hyphendata@\the\language\endcsname}}}
\endinput\fi
% Here stops reading code for hyphen.cfg
% The following is read the 2nd time it's loaded
\begingroup % TODO - to a lua file
\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.callback)
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 .. '.'
end
ss = ss:gsub('^%d%?%.', '%%.').%d?'
ss = ss:gsub('%.%d%?\$', '%%.')
end
172
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)
endgroup
if\newattribute\undefined\else
\newattribute\bbl@attr@locale
\directlua{ Babel.attr_locale = luatexbase.registernumber'\bbl@attr@locale' }\AddBabelHook{luatex}{beforeextras}{% \setattribute\bbl@attr@locale{localeid} \fi\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()}% \def\luabbl@stop{\directlua{Babel.end_process_input()}}% \fi}%\AddBabelHook{luatex}{stopcommands}{\luabbl@stop\let\luabbl@stop\relax}\AddBabelHook{luatex}{patterns}{% \@ifundefined{bbl@hyphendata@\the\language}{\def\bbl@elt##1##2##3##4{% \ifnum##2=\csname l@#2\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@\the\language}{\def\bbl@info{No hyphenation patterns were set for\% language '#2'. Reported}}% \expandafter\expandafter\expandafter\bbl@luapatterns \csname bbl@hyphendata@\the\language\endcsname}}{}% \@ifundefined{bbl@patterns@}{}{\begingroup \bbl@xin@{,\number\language,}{,\bbl@pttnlist}% \ifin@ else \def\bbl@patterns@{}\directlua{ Babel.addpatterns( \[[\bbl@patterns@], \number\language\] )} \fi \@ifundefined{bbl@patterns@#1}%
\belpatterns

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 \belposthyphenation. Replace regular (ie, implicit) discretionary spaces, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionary spaces are not touched. See Unicode UAX 14.
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@@{% 
  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\@@{% 
  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
  \directlua{
    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 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, Cyril1
                  local intraspaces = Babel.intraspaces[lg]
                  local intrapenalty = Babel.intrapenalties[lg]
                  local n
                  if intrapenalty ~= 0 then
                    n = node.new(14, 0) ^% penalty
                    n.penalty = intrapenalty
                  end
                  if ..
                end
              end
            end
          end
        end
      end
    end
end
\begingroup
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
}}^^
\bbl@luahyphenate

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.

\catcode`\%=14
\gdef\bbl@cjkintraspace{%
\let\bbl@cjkintraspace\relax
\directlua{
Babel = Babel or {}
require('babel-data-cjk.lua')
Babel.cjk_enabled = true
function Babel.cjk_linebreak(head)
  local GLYPH = node.id'glyph'
  local last_char = nil
  local quad = 655360 % 10 pt = 655360 = 10 * 65536
  local last_class = nil
  local last_lang = nil
  for item in node.traverse(head) do
    if item.id == GLYPH then
      local lang = item.lang
      local LOCALE = node.get_attribute(item, Babel.attr_locale)
      local props = Babel.locale_props[LOCALE]
      local class = Babel.cjk_class[item.char].c
      if props.cjk_quotes and props.cjk_quotes[item.char] then
        class = props.cjk_quotes[item.char]
      end
      if class == 'cp' then class = 'cl' end % )\] as CL
      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
if font.getfont(item.font) then
  quad = font.getfont(item.font).size
end
last_class = class
last_lang = lang
else % if penalty, glue or anything else
  last_class = nil
end
lang.hyphenate(head)
end
}\bbl@luahyphenate
\gdef\bbl@luahyphenate{%\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
    Babel.cjk_linebreak(head)
  end
  lang.hyphenate(head)
  if Babel.linebreaking.after then
    for k, func in ipairs(Babel.linebreaking.after) do
      func(head)
    end
  end
  if Babel.sea_enabled then
    Babel.sea_disc_to_space(head)
  end,
  'Babel.hyphenate')
}
13.6 Arabic justification

\begingroup
\catcode`_=11 \catcode`:=11
\gdef\bblar@nofswarn\{\gdef\msg_warning:nnx##1##2##3{}\}
\endgroup
\gdef\bbl@arabicjust\{\let\bbl@arabicjust\relax
\newattribute\bblar@kashida\directlua{ Babel.attr_kashida = luatexbase.registernumber'\bblar@kashida' }
\bblar@kashida=\z@
\bbl@patchfont{{\bbl@parsejalt}}\directlua{ Babel.arabic.elong_map = Babel.arabic.elong_map or {} Babel.arabic.elong_map[{\the\localeid}] = {} luatexbase.add_to_callback('post_linebreak_filter',

\endgroup
5270 \texttt{Babel.arabic.justify, 'Babel.arabic.justify')}  
5271 \texttt{luatexbase.add_to_callback('hpack_filter,}  
5272 \texttt{Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')}  
5273 \texttt{)}}%  
5274 \texttt{% Save both node lists to make replacement. TODO. Save also widths to}  
5275 \texttt{% make computations}  
5276 \texttt{\def\bblar@fetchjalt#1#2#3#4{%}  
5277 \texttt{\bbl@exp{\{\bbl@foreach{#1}}{%}  
5278 \texttt{\bbl@ifunset{bblar@JE@##1}%  
5279 \texttt{\setbox\z@\hbox{\char'##1#2}}%  
5280 \texttt{\setbox\z@\hbox{\char'@nameuse{bblar@JE@##1}#2}}%  
5281 \texttt{\directlua{%}  
5282 \texttt{local last = nil}  
5283 \texttt{for item in node.traverse(tex.box[0].head) do}  
5284 \texttt{if item.id == node.id'glyph' and item.char > 0x600 and}  
5285 \texttt{not (item.char == 0x200D) then}  
5286 \texttt{last = item}  
5287 \texttt{end}  
5288 \texttt{end}  
5289 \texttt{Babel.arabic.#3['##1#4'] = last.char}  
5290 \texttt{}}}%  
5291 \texttt{% Brute force. No rules at all, yet. The ideal: look at jalt table. And}  
5292 \texttt{% perhaps other tables (falt?, cswh?). What about kaf? And diacritic}  
5293 \texttt{% positioning?}  
5294 \texttt{\gdef\bblar@parsejalt{%}  
5295 \texttt{\if\addfontfeature\@undefined\else}  
5296 \texttt{\bbl@xin@{/e}{/\bblar@cl{lnbrk}}}%  
5297 \texttt{\fi}  
5298 \texttt{\directlua{%}  
5299 \texttt{if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then}  
5300 \texttt{Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}}  
5301 \texttt{tex.print([[\string\csname\space bblar@parsejalt\endcsname]]}}  
5302 \texttt{end}  
5303 \texttt{}}%  
5304 \texttt{\fi}  
5305 \texttt{\gdef\bblar@parsejalti{%}  
5306 \texttt{\begingroup}  
5307 \texttt{\let\bblar@parsejalt\relax}  
5308 \texttt{\edef\bblar@tempb{\fontid\font}\%}  
5309 \texttt{\bblar@nofswarn}  
5310 \texttt{\bblar@fetchjalt\bblar@elongated}{\from}{\}}%  
5311 \texttt{\bblar@fetchjalt\bblar@chars{\char'064a}{\from}}{\a}% Alef maksura  
5312 \texttt{\bblar@fetchjalt\bblar@chars{\char'0649}{\from}}{\y}% Yeh  
5313 \texttt{\addfontfeature{RawFeature=+jalt}\%}  
5314 \texttt{\namedef{bblar@JE@0643}{06AA}.todo: catch medial kaf}  
5315 \texttt{\bblar@fetchjalt\bblar@elongated}{\dest}{\}}%  
5316 \texttt{\bblar@fetchjalt\bblar@chars{\char'064a}{\dest}}{\a}%  
5317 \texttt{\bblar@fetchjalt\bblar@chars{\char'0649}{\dest}}{\y}%  
5318 \texttt{\directlua{%}  
5319 \texttt{for k, v in pairs(Babel.arabic.from) do}  
5320 \texttt{if Babel.arabic.dest[k] and}  
5321 \texttt{not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then}  
5322 \texttt{Babel.arabic.elong_map[\the\localeid][\bblar@tempb]}  
5323 \texttt{[Babel.arabic.from[k]] = Babel.arabic.dest[k]}  
5324 \texttt{end}  
5325 \texttt{end}  
5326 \texttt{}}%  
5327 \texttt{\endgroup)
\begingroup
\catcode`#=11
\catcode`~=11
\directlua{
Babel.arabic = Babel.arabic or {}
Babel.arabic.from = {}
Babel.arabic.dest = {}
Babel.arabic.justify_factor = 0.95
Babel.arabic.justify_enabled = true

function Babel.arabic.justify(head)
if not Babel.arabic.justify_enabled then return head end
for line in node.traverse_id(node.id'hlist', head) do
    Babel.arabic.justify_hlist(head, line)
end
return head
end

function Babel.arabic.justify_hbox(head, gc, size, pack)
local has_inf = false
if Babel.arabic.justify_enabled and pack == 'exactly' then
    for n in node.traverse_id(12, head) do
        if n.stretch_order > 0 then has_inf = true end
    end
    if not has_inf then
        Babel.arabic.justify_hlist(head, nil, gc, size, pack)
    end
end
return head
end

function Babel.arabic.justify_hlist(head, line, gc, size, pack)
local d, new
local k_list, k_item, pos_inline
local width, width_new, full, k_curr, wt_pos, goal, shift
local subst_done = false
local elong_map = Babel.arabic.elong_map
local last_line
local GLYPH = node.id'glyph'
local KASHIDA = Babel.attr_kashida
local LOCALE = Babel.attr_locale

if line == nil then
    line = {}
    line.glue_sign = 1
    line.glue_order = 0
    line.head = head
    line.shift = 0
    line.width = size
end

% Exclude last line. todo. But-- it discards one-word lines, too!
% ? Look for glue = 12:15
if (line.glue_sign == 1 and line.glue_order == 0) then
    elongs = {}
    k_list = {}
    pos_inline = 0 % Not yet used
for n in node.traverse_id(GLYPH, line.head) do
    pos_inline = pos_inline + 1 % To find where it is. Not used.
    % Elongated glyphs
    if elong_map then
        local locale = node.get_attribute(n, LOCALE)
        if elong_map[locale] and elong_map[locale][n.font] and
            elong_map[locale][n.font][n.char] then
            table.insert(elongs, {node = n, locale = locale} )
            node.set_attribute(n.prev, KASHIDA, 0)
        end
    end
    % Tatwil
    if Babel.kashida_wts then
        local k_wt = node.get_attribute(n, KASHIDA)
        if k_wt > 0 then % todo. parameter for multi inserts
            table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
        end
    end
end % of node.traverse_id

if #elongs == 0 and #k_list == 0 then goto next_line end
full = line.width
shift = line.shift
goal = full * Babel.arabic.justify_factor % A bit crude
width = node.dimensions(line.head) % The 'natural' width

% == Elongated ==
% Original idea taken from 'chikenize'
while (#elongs > 0 and width < goal) do
    subst_done = true
    local x = #elongs
    local curr = elongs[x].node
    local oldchar = curr.char
    curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
    width = node.dimensions(line.head) % Check if the line is too wide
    if width > goal then
        curr.char = oldchar
        break
    end
end % If continue, pop the just substituted node from the list:
table.remove(elongs, x)

% == Tatwil ==
if #k_list == 0 then goto next_line end
width = node.dimensions(line.head) % The 'natural' width
kCurr = #k_list
wt_pos = 1

while width < goal do
    subst_done = true
    k_item = k_list[kCurr].node
    if k_list[kCurr].weight == Babel.kashida_wts[wt_pos] then
\begin{verbatim}
5447   d = node.copy(k_item)
5448   d.char = 0x0640
5449   line.head, new = node.insert_after(line.head, k_item, d)
5450   width_new = node.dimensions(line.head)
5451   if width > goal or width == width_new then
5452     node.remove(line.head, new) % Better compute before
5453     break
5454   end
5455   width = width_new
5456 end
5457 if k_curr == 1 then
5458   k_curr = #k_list
5459   wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5460 else
5461   k_curr = k_curr - 1
5462 end
5463 end
5464 ::next_line::
5465 % Must take into account marks and ins, see luatex manual.
5466 % Have to be executed only if there are changes. Investigate
5467 % what's going on exactly.
5468 if subst_done and not gc then
5469   d = node.hpack(line.head, full, 'exactly')
5470   d.shift = shift
5471   node.insert_before(head, line, d)
5472   node.remove(head, line)
5473 end
5474 end % if process line
5475 end
5476 \end{verbatim}

13.7 Common stuff

13.8 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 traverse 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 \texttt{language} and the \texttt{localeid} as stored in \texttt{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 discretionary are handled in a special way.

% TODO - to a lua file
\directlua{
Babel.script_blocks = {
  ['dflt'] = {},
  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EFFF}},
  ['Armn'] = {{0x0530, 0x058F}},
  ['Beng'] = {{0x0980, 0x09FF}},
  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABB0}},
}
```
% Don't follow strictly Unicode, which places some Coptic letters in
% the 'Greek and Coptic' block
['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
{0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
{0xAB00, 0xAB2F}},
['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
['Hans'] = {{0x0E00, 0x0E7F}},
['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
{0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
['Knda'] = {{0x0C80, 0x0CFF}},
['Kore'] = {{0x1100, 0x11FF}, {0x3130, 0x318F}, {0xA960, 0xA97F},
{0xAC00, 0xD7AF}, {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
['Laoo'] = {{0x0E80, 0x0EFF}},
['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
{0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
{0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
['Hebr'] = {{0x0590, 0x05FF}},
['Mlym'] = {{0x0D00, 0x0D7F}},
['Mymr'] = {{0x0D00, 0x0D7F}},
['Orya'] = {{0x0B00, 0x0B7F}},
['Sinh'] = {{0x0D80, 0x0DFF}, {0x0E00, 0x0E7F}},
['Thai'] = {{0x0E00, 0x0E7F}},
['Telu'] = {{0x0C00, 0x0C7F}},
['Tfng'] = {{0x0D30, 0x0D7F}},
['Tibet'] = {{0x0E00, 0x0E7F}},
['Vaii'] = {{0x0A50, 0x0A63F}},
['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
```
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

% Now, take action, but treat composite chars in a different
% fashion, because they 'inherit' the previous locale. Not yet
% optimized.
if not toloc and
    (item.char >= 0x0300 and item.char <= 0x036F) or
    (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
    (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
    toloc = toloc_save
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
elseif 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
return head
end
}

The code for \babelcharproperty is straightforward. Just note the modified lua table can be
different.

\newcommand{\bbl@chprop}[1][]{\count@=#1\relax
\ifvmode
\bbl@chprop
\else
\bbl@error{\string\bbl@chprop\space can be used only in\% vertical mode (preamble or between paragraphs)\% See the manual for further info\%}
\fi
\newcommand{\bbl@chprop}[3][\the\count@]{\@tempcnta=#1\relax
\bbl@ifunset{\bbl@chprop@#2}%
{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@][\textquoteleft d\textquoteright] = \textquoteleft #1\textquoteright}
\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@][\textquoteleft m\textquoteright] = \textquoteleft \number\textquoteleft #1\textquoteright\textquoteright}
\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@][\textquoteleft c\textquoteright] = \textquoteleft #1\textquoteright}
\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). The Lua code is below.

\directlua{Babel.nohyphenation = \the\l@nohyphenation}

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 \texttt{lua load} – 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).

\begingroup\catcode\%=12\catcode\&=14
\gdef\babelposthyphenation#1#2#3{&%\bbl@activateposthyphen
\begingroup\def\babeltempa{\bbl@add@list\babeltempb}&%\let\babeltempb\@empty\def\bbl@tempa{#3}&%\bbl@replace\bbl@tempa{,}{ ,}&%\expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%} \endgroup}

\begin{verbatim}
\catcode\`\-=12 \catcode\`\%=12 \catcode\`\&=14
\gdef\babelposthyphenation#1#2#3{&% \bbl@activateposthyphen
\begin{verbatim}
\def\babeltempa{\bbl@add@list\babeltempb}&%
\let\babeltempb\@empty
\def\bbl@tempa{#3}&%
\bbl@replace\bbl@tempa{,}{ ,}&%
\expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%}
\end{verbatim}
\end{verbatim}
\end{verbatim}
\end{verbatim}
\end{verbatim}
\bbl@ifsamestring{##1}{remove}&\
{\bbl@add@list!bbl@temp{nil}}&\
{\directlua{
  local rep = ['=#1]=
  rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
  rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
  rep = rep:gsub(' (no)%s*=%s*([^%s,.]+)', Babel.capture_func)
  rep = rep:gsub(' (pre)%s*=%s*([^%s,.]+)', Babel.capture_func)
  rep = rep:gsub(' (post)%s*=%s*([^%s,.]+)', Babel.capture_func)
  rep = rep:gsub('string)%s*=%s*([^%s,.]+)', Babel.capture_func)
  rep = rep:gsub('space)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
    'space = (' .. '%2, %3, %4' .. ')
  rep = rep:gsub('spacefactor)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
    'spacefactor = { (' .. '%2, %3, %4' .. ')
  rep = rep:gsub('kashida)%s*=%s*([^%s,.]+)', Babel.capture_kashida)
  tex.print([[\string\bbl@temp{[]} .. rep .. []]])
}}}&\
\directlua{
  local lbkr = Babel.linebreaking.replacements[1]
  local u = unicode.utf8
  local id = \the\csname l@#1\endcsname
  \% Convert pattern:
  local patt = string.gsub([==[=#2]==], '%s', '')
  if not u.find(patt, '()', nil, true) then
    patt = '(' .. patt .. '()'
  end
  patt = string.gsub(patt, '%(%)%^', '^()')
  patt = string.gsub(patt, '%$%(%)', '()$')
  patt = u.gsub(patt, '{(.)}',
    function (n)
      return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
    end)
  patt = u.gsub(patt, '{(%x%x%x%x+)}',
    function (n)
      return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
    end)
  lbkr[id] = lbkr[id] or {}
  table.insert(lbkr[id], { pattern = patt, replace = { \bbl@tempb } })
}}&\
\% TODO. Copy paste pattern.
gdef\bbl@prehyphenation#1#2#3{&\
  \bbl@activateprehyphen
  \begingroup
  \let\bbl@temp\@empty
  \def\bbl@tempa{\bbl@add@list\bbl@tempb}&\
  \let\bbl@tempb\@empty
  \def\bbl@tempa{#3}&\
  \bbl@ifsamestring{##1}{remove}&\
  {\bbl@add@list!bbl@temp{nil}}&\
  \directlua{
    local rep = ['=#1]=
    rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
    rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
    rep = rep:gsub(' (no)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub(' (pre)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub(' (post)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub('string)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub('space)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
      'space = (' .. '%2, %3, %4' .. ')
    rep = rep:gsub('spacefactor)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
      'spacefactor = { (' .. '%2, %3, %4' .. ')
    rep = rep:gsub('kashida)%s*=%s*([^%s,.]+)', Babel.capture_kashida)
    tex.print([[\string\bbl@temp{[]} .. rep .. []]])
  }}&\
\directlua{
  local lbkr = Babel.linebreaking.replacements[1]
  local u = unicode.utf8
  local id = \the\csname l@#1\endcsname
  \% Convert pattern:
  local patt = string.gsub([==[=#2]==], '%s', '')
  if not u.find(patt, '()', nil, true) then
    patt = '(' .. patt .. '()'
  end
  patt = string.gsub(patt, '%(%)%^', '^()')
  patt = string.gsub(patt, '%$%(%)', '()$')
  patt = u.gsub(patt, '{(.)}',
    function (n)
      return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
    end)
  patt = u.gsub(patt, '{(%x%x%x%x+)}',
    function (n)
      return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
    end)
  lbkr[id] = lbkr[id] or {}
  table.insert(lbkr[id], { pattern = patt, replace = { \bbl@tempb } })
  \endgroup
}}&\
% TODO. Copy paste pattern.
\gdef\bbl@prehyphenation#1#2#3{&\
  \bbl@activateprehyphen
  \begingroup
  \let\bbl@temp\@empty
  \def\bbl@tempa{\bbl@add@list\bbl@tempb}&\
  \let\bbl@tempb\@empty
  \def\bbl@tempa{#3}&\
  \bbl@ifsamestring{##1}{remove}&\
  {\bbl@add@list!bbl@temp{nil}}&\
  \directlua{
    local rep = ['=#1]=
    rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
    rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
    rep = rep:gsub(' (no)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub(' (pre)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub(' (post)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub('string)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub('space)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
      'space = (' .. '%2, %3, %4' .. ')
    rep = rep:gsub('spacefactor)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
      'spacefactor = { (' .. '%2, %3, %4' .. ')
    rep = rep:gsub('kashida)%s*=%s*([^%s,.]+)', Babel.capture_kashida)
    tex.print([[\string\bbl@temp{[]} .. rep .. []]])
  }}&\endgroup
}}&\endgroup
% TODO. Copy paste pattern.
\gdef\bbl@prehyphenation#1#2#3{&\
  \bbl@activateprehyphen
  \begingroup
  \let\bbl@temp\@empty
  \def\bbl@tempa{\bbl@add@list\bbl@tempb}&\
  \let\bbl@tempb\@empty
  \def\bbl@tempa{#3}&\
  \bbl@ifsamestring{##1}{remove}&\
  {\bbl@add@list!bbl@temp{nil}}&\
  \directlua{
    local rep = ['=#1]=
    rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
    rep = rep:gsub('^%s*(insert)%s*$', 'insert = true, ')
    rep = rep:gsub(' (no)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub(' (pre)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub(' (post)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub('string)%s*=%s*([^%s,.]+)', Babel.capture_func)
    rep = rep:gsub('space)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
      'space = (' .. '%2, %3, %4' .. ')
    rep = rep:gsub('spacefactor)%s*=%s*([^%s,.]+)%s+(%d%+.+)%s+(%d%+.+)%s+(%d%+.+)',
      'spacefactor = { (' .. '%2, %3, %4' .. ')
    rep = rep:gsub('kashida)%s*=%s*([^%s,.]+)', Babel.capture_kashida)
    tex.print([[\string\bbl@temp{[]} .. rep .. []]])
  }}&\endgroup
}}% TODO. Copy paste pattern.
local lbkr = Babel.linebreaking.replacements[0]
local u = unicode.utf8
local id = \the\csname bbl@id@@#1\endcsname

&% Convert pattern:
local patt = string.gsub([==[#2]==], '%s', '')
local patt = string.gsub(patt, '|', ' ')
if not u.find(patt, '()', nil, true) then
  patt = '(' .. patt .. ')'
end
&% patt = string.gsub(patt, '%(%)%^', '^()')
&% patt = string.gsub(patt, '([^%%])%$%(%)', '%1()$')
patt = u.gsub(patt, '{(.)}',
  function (n)
    return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
  end)
patt = u.gsub(patt, '{(%x%x%x%x+)}',
  function (n)
    return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
  end)
lbkr[id] = lbkr[id] or {}
table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })

13.9 Bidi

As a first step, add a handler for bidi and digits (and potentially other processes) just before \luaofloatload is applied, which is loaded by default by \LaTeX. Just in case, consider the possibility it has not been loaded.

\def\bbl@activateposthyphen{%
  \let\bbl@activateposthyphen\relax
  \directlua{
    require('babel-transforms.lua')
    Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
  }
}%
\def\bbl@activateprehyphen{%
  \let\bbl@activateprehyphen\relax
  \directlua{
    require('babel-transforms.lua')
    Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
  }
}
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_ofload_v,
    'Babel.pre_ofload_v',
    luatexbase.priority_in_callback('pre_linebreak_filter',
        'luaotfload.node_processor') or nil)
%
luatexbase.add_to_callback('hpack_filter',
    Babel.pre_ofload_h,
    'Babel.pre_ofload_h',
    luatexbase.priority_in_callback('hpack_filter',
        'luaotfload.node_processor') or nil)
%

The basic setup. The output is modified at a very low level to set the \bodydir to the \pagedir. Sadly, we have to deal with boxes in math with basic, so the \bbl@mathboxdir hack is activated every math with the package option bidi=.

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
\directlua{ Babel.attr_dir = luatexbase.registernumber'\bbl@attr@dir' }
% TODO. I don't like it, hackish:
\bbl@exp{'\output{\bodydir\pagedir\the\output}}
\AtEndOfPackage{\EnableBabelHook{babel-bidi}}
\fi
\chardef\bbl@thetextdir\z@
\chardef\bbl@thepardir\z@
\def\bbl@getluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
\ifcase#3\relax
    \ifcase\bbl@getluadir#1#20\relax
        \textdir TLT
    \else
        \textdir TRT
    \fi
\else
    \ifcase\bbl@getluadir#1#21\relax
        \textdir 0
    \else
        \textdir 1
    \fi
\fi
end}
\def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
\ifcase#3\relax
    \ifcase\bbl@getluadir#1#22\relax
        \textdir TLT\relax
    \else
        \textdir TRT\relax
    \fi
end}
\else
13.10 Layout

Unlike xetex, 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 \textit{bidi=basic}, without having to patch almost any macro where text direction is relevant. \texttt{@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{\bodydir}), and when \texttt{\parbox} and \texttt{\hangindent} are involved. Fortunately, latest releases of luatex simplify a lot the solution with \texttt{\shapemode}.

With the issue \#151 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, \texttt{\tabular} seems to work (at least in simple cases) with \texttt{\array}, \texttt{\tabularx}, \texttt{\hhline}, \texttt{\colortbl}, \texttt{\longtable}, \texttt{\booktabs}, etc. However, \texttt{\dcolumn} still fails.
Babel.picture_has_bidi = 0

function Babel.picture_dir(head)
  if not Babel.get_picture_dir then return head end
  for item in node.traverse(head) do
    if item.id == node.id'glyph' then
      local itemchar = item.char
      % TODO. Copypaste pattern from Babel.bidi (-r)
      local chardata = Babel.characters[itemchar]
      local dir = chardata and chardata.d or nil
      if not dir then
        for nn, et in ipairs(Babel.ranges) do
          if itemchar < et[1] then
            break
          elseif itemchar <= et[2] then
            dir = et[3]
            break
          end
        end
        if dir and (dir == 'al' or dir == 'r') then
          Babel.picture_has_bidi = 1
        end
      end
    end
  end
  return head
end

luatexbase.add_to_callback("hpacak_filter", Babel.picture_dir, "Babel.picture_dir")

\AtBeginDocument{%
\longdef\put(#1,#2)#3{%\@killglue
% Try:
  \ifx\bbl@pictresetdir\relax
    \def\bbl@tempc{0}%
  \else
    \directlua{
      Babel.get_picture_dir = true
      Babel.picture_has_bidi = 0
    }
    \setbox2@\hb@xt@\z@{%
    \@defaultunitsset\@tempdimc{#1}\unitlength
    \kern\@tempdimc
    #3\hss}%
    \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
  \fi
% Do:
  \@defaultunitsset\@tempdimc{#2}\unitlength
  \raise\@tempdimc\@h@xt\@z@{%
  \@defaultunitsset\@tempdimc{#1}\unitlength
  \kern\@tempdimc
  {\ifnum\bbl@tempc<\z@\bbl@pictresetdir\fi#3}\hss}%
  \ignorespaces%
  \MakeRobust\put}%
\fi\AtBeginDocument
% TODO. Allow deactivate?
  \ifx\AddToHook\undefined
    \bbl@sreplace\pgfpicture\@undefined\undefined
  \else
    \pgfpicture@undefined
  \fi

Implicitly reverses sectioning labels in bi=−basic−r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bi=−basic, but there are some additional readjustments for bi=−default.

Some \TeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.
13.11 Lua: transforms

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 discretionary patterns 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
Babel.linebreaking.replacements = {}
Babel.linebreaking.replacements[0] = {} -- pre
Babel.linebreaking.replacements[1] = {} -- post

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

Babel.fetch_subtext = {}

Babel.ignore_pre_char = function(node)
    return (node.lang == Babel.nohyphenation)
end

function Babel.fetch_subtext[0] = function(head)
    local word_string = ''
```
local word_nodes = {}
local lang
local item = head
local inmath = false
while item do
  if item.id == 11 then
    inmath = (item.subtype == 0)
  end
  if inmath then
    -- pass
  elseif item.id == 29 then
    local locale = node.get_attribute(item, Babel.attr_locale)
    if lang == locale or lang == nil then
      lang = lang or locale
      if Babel.ignore_pre_char(item) then
        word_string = word_string .. Babel.us_char
      else
        word_string = word_string .. unicode.utf8.char(item.char)
      end
      word_nodes[#word_nodes+1] = item
    else
      break
    end
  elseif item.id == 12 and item.subtype == 13 then
    word_string = word_string .. ' '
    word_nodes[#word_nodes+1] = item
  -- Ignore leading unrecognized nodes, too.
  elseif word_string ~= '' then
    word_string = word_string .. Babel.us_char
    word_nodes[#word_nodes+1] = item -- Will be ignored
  end
  item = item.next
end
-- Here and above we remove some trailing chars but not the
-- corresponding nodes. But they aren't accessed.
if word_string:sub(-1) == '' then
  word_string = word_string:sub(1,-2)
end
word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
return word_string, word_nodes, item, lang
end
Babel.fetch_subtext[1] = function(head)
  local word_string = ''
  local word_nodes = {}
  local lang
  local item = head
  local inmath = false
  while item do

if item.id == 11 then
    inmath = (item.subtype == 0)
end

if inmath then
    -- pass
elseif item.id == 29 then
    if item.lang == lang or lang == nil then
        if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
            lang = lang or item.lang
            word_string = word_string .. unicode.utf8.char(item.char)
            word_nodes[#word_nodes+1] = item
        end
    else
        break
    end
elseif item.id == 7 and item.subtype == 2 then
    word_string = word_string .. '='
    word_nodes[#word_nodes+1] = item
elseif item.id == 7 and item.subtype == 3 then
    word_string = word_string .. '|' 
    word_nodes[#word_nodes+1] = item
else
    break
end

-- (1) Go to next word if nothing was found, and (2) implicitly
-- remove leading USs.
else if word_string == '' then
    -- pass
    -- This is the responsible for splitting by words.
    elseif (item.id == 12 and item.subtype == 13) then
        break
else
    word_string = word_string .. Babel.us_char
    word_nodes[#word_nodes+1] = item -- Will be ignored
end
item = item.next

word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
return word_string, word_nodes, item, lang

function Babel.pre_hyphenate_replace(head)
    Babel.hyphenate_replace(head, 0)
end

function Babel.post_hyphenate_replace(head)
    Babel.hyphenate_replace(head, 1)
end

Babel.us_char = string.char(31)
local u = unicode.utf8
local lbkr = Babel.linebreaking.replacements[mode]
local word_head = head
while true do -- for each subtext block
    local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
    if Babel.debug then
        print()
        print((mode == 0) and '@@@@<' or '@@@@>', w)
    end
    if nw == nil and w == '' then break end
    if not lang then goto next end
    if not lbkr[lang] then goto next end
    -- For each saved (pre|post)hyphenation. TODO. Reconsider how
    -- loops are nested.
    for k=1, #lbkr[lang] do
        local p = lbkr[lang][k].pattern
        local r = lbkr[lang][k].replace
        if Babel.debug then
            print('*****', p, mode)
        end
        local last_match = 0
        local step = 0
        -- For every match.
        while true do
            if Babel.debug then
                print('=====')
            end
            local new -- used when inserting and removing nodes
            local matches = { u.match(w, p, last_match) }
            if #matches < 2 then break end
            local first = table.remove(matches, 1)
            local last = table.remove(matches, #matches)
            local save_last = last -- with A()BC()D, points to D
            local matches = { u.match(w, p, last_match) }
            if #matches < 2 then break end
            -- Get and remove empty captures (with ()'s, which return a
            -- number with the position), and keep actual captures
            -- (from (...)), if any, in matches.
            local first = table.remove(matches, 1)
            local last = table.remove(matches, #matches)
            -- Non re-fetched substrings may contain \31, which separates
            -- subsubstrings.
            if string.find(w:sub(first, last-1), Babel.us_char) then break end
            -- Fix offsets, from bytes to unicode. Explained above.
            first = u.len(w:sub(1, first-1)) + 1

last = u.len(w:sub(1, last-1)) -- now last points to C

-- This loop stores in a small table the nodes
-- corresponding to the pattern. Used by 'data' to provide a
-- predictable behavior with 'insert' (w_nodes is modified on
-- the fly), and also access to 'remove'd nodes.
local sc = first-1 -- Used below, too
local data_nodes = {}

for q = 1, last-first+1 do
    data_nodes[q] = w_nodes[sc+q]
end

-- This loop traverses the matched substring and takes the
-- corresponding action stored in the replacement list.
-- sc = the position in substr nodes / string
-- rc = the replacement table index
local rc = 0

while rc < last-first+1 do -- for each replacement
    if Babel.debug then
        print('.....', rc + 1)
    end
    sc = sc + 1
    rc = rc + 1

    if Babel.debug then
        Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
        local ss = ''
        for itt in node.traverse(head) do
            if itt.id == 29 then
                ss = ss .. unicode.utf8.char(itt.char)
            else
                ss = ss .. '(' .. itt.id .. ')'
            end
        end
        print('***************', ss)
    end

    local crep = r[rc]
    local item = w_nodes[sc]
    local item_base = item
    local placeholder = Babel.us_char
    local d

    if crep and crep.data then
        item_base = data_nodes[crep.data]
    end

    if crep then
        step = crep.step or 0
    end

    if crep and next(crep) == nil then -- = {}
        last_match = save_last -- Optimization
        goto next
    elseif crep == nil or crep.remove then

197
node.remove(hand, item)
node.remove(w_nodes, sc)
w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
sc = sc - 1 -- Nothing has been inserted.
last_match = utf8.offset(w, sc+1+step)
goto next

e elseif crep and crep.kashida then -- Experimental
node.set_attribute(item,
    Babel.attr_kashida,
    crep.kashida)
last_match = utf8.offset(w, sc+1+step)
goto next

e elseif crep and crep.string then
local str = crep.string(matches)
if str == ' ' then -- Gather with nil
    node.remove(hand, item)
    table.remove(w_nodes, sc)
w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
s = sc - 1 -- Nothing has been inserted.
else
    local loop_first = true
    for s in string.utfvalues(str) do
        d = node.copy(item_base)
d.char = s
        if loop_first then
            loop_first = false
            head, new = node.insert_before(hand, item, d)
        if sc == 1 then
            word_head = head
        end
        w_nodes[sc] = d
        w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
    else
        sc = sc + 1
        head, new = node.insert_before(hand, item, d)
        table.insert(w_nodes, sc, new)
w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
end
if Babel.debug then
    print('.....', 'str')
    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
end
end -- for
node.remove(hand, item)
end -- if ''
last_match = utf8.offset(w, sc+1+step)
goto next

e elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
    d = node.new(7, 0) -- (disc, discretionary)
d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
d.post = Babel.str_to_nodes(crep.post, matches, item_base)
d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
d.attr = item_base.attr
if crep.pre == nil then -- TeXbook p96
    d.penalty = crep.penalty or tex.hyphenpenalty
else

198
d.penalty = crep.penalty or tex.hyphenpenalty
end
placeholder = '|' head, new = node.insert_before(head, item, d)
elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
  -- ERROR
elseif crep and crep.penalty then
d = node.new(14, 0) -- (penalty, userpenalty)
d.attr = item_base.attr
d.penalty = crep.penalty
head, new = node.insert_before(head, item, d)
elseif crep and crep.space then
  -- 655360 = 10 pt = 10 * 65536 sp
if mode == 0 then
  placeholder = ' '
end
head, new = node.insert_before(head, item, d)
elseif crep and crep.spacefactor then
if mode == 0 then
  placeholder = ' '
end
head, new = node.insert_before(head, item, d)
else if mode == 0 and crep and crep.space then
  -- ERROR
end -- ie replacement cases
-- Shared by disc, space and penalty.
if sc == 1 then word_head = head end
if crep.insert then
  w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc) table.insert(w_nodes, sc, new) last = last + 1
else
  w_nodes[sc] = d node.remove(head, item) w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1) end
last_match = utf8.offset(w, sc+1+step)
end -- for each replacement

if Babel.debug then
  print('.....', '/')
  Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
end

end -- for match

end -- for patterns

::next::

word_head = nw

end -- for substring

return head

-- This table stores capture maps, numbered consecutively
Babel.capture_maps = {}

-- The following functions belong to the next macro

function Babel.capture_func(key, cap)
  local ret = "[" .. cap.gsub('([0-9])', ']..m[%1]..[" .. "]"
  local cnt
  local u = unicode.utf8
  ret, cnt = ret:gsub('{([0-9])|([[^|]+)|(.-)\)', Babel.capture_func_map)
  if cnt == 0 then
    ret = u.gsub(ret, '{(%x%x%x%x+)}', function (n)
      return u.char(tonumber(n, 16))
    end)
  end
  ret = ret:gsub('%[[]%]' .. '.', '')
  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)
  local u = unicode.utf8
  from = u.gsub(from, '{(\%x\%x\%x\%x+)\}', function (n)
    return u.char(tonumber(n, 16))
  end)
  to = u.gsub(to, '(\%x\%x\%x\%x+)\', function (n)
    return u.char(tonumber(n, 16))
  end)
  local froms = {}
  for s in string.utfcharacters(from) do
    table.insert(froms, s)
  end
  local cnt = 1
  for s in string.utfcharacters(to) do
    table.insert(froms, s)
  end
  return from, to, froms, cnt
end
table.insert(Babel.capture_maps, {})
local mlen = table.getn(Babel.capture_maps)
for s in string.utfcharacters(to) do
  Babel.capture_maps[mlen][froms[cnt]] = s
  cnt = cnt + 1
end
return "]]..Babel.capt_map(m[" .. capno .. "]," ..
  (mlen) .. ") .. "][.."
end

-- Create/Extend reversed sorted list of kashida weights:
function Babel.capture_kashida(key, wt)
wt = tonumber(wt)
if Babel.kashida_wts then
  for p, q in ipairs(Babel.kashida_wts) do
    if wt == q then
      break
    elseif wt > q then
      table.insert(Babel.kashida_wts, p, wt)
      break
    elseif table.getn(Babel.kashida_wts) == p then
      table.insert(Babel.kashida_wts, wt)
    end
  end
else
  Babel.kashida_wts = { wt }
end
return 'kashida = ' .. wt
end

⟨/transforms⟩

13.12 Lua: 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):

Arrrgh!! 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 latex excels, because everything related to bidi writing is under our control.

```latex
6521 ⟨basic-r⟩
6522 Babel = Babel or {}
6523 Babel.bidi_enabled = true
6524 require('babel-data-bidi.lua')
6525 local characters = Babel.characters
6526 local ranges = Babel.ranges
6527 local DIR = node.id("dir")
6528 function dir_mark(head, from, to, outer)
6529   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6530   local d = node.new(DIR)
6531   d.dir = '+' .. dir
6532   node.insert_before(head, from, d)
6533   d = node.new(DIR)
6534   d.dir = '-' .. dir
6535   node.insert_after(head, to, d)
6536 end
6537 function Babel.bidi(head, ispar)
6538   local first_n, last_n -- first and last char with nums
6539   local last_es -- an auxiliary 'last' used with nums
6540   local first_d, last_d -- first and last char in L/R block
6541   local dir, dir_real
6542   next also depends on script/lang (<al>/<r>). To be set by babel. tex.pardir is dangerous, could be (reset) but it should be changed only in vmode. There are two strong's – strong = 1/al/r and strong_lr = l/r (there must be a better way):
6543   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6544   local strong_lr = (strong == 'l') and 'l' or 'r'
6545   local outer = strong
6546   local new_dir = false
6547   local first_dir = false
6548   local inmath = false
6549   local last_lr
6550   local type_n = ''
6551   for item in node.traverse(head) do
6552     -- three cases: glyph, dir, otherwise
6553     if item.id == node.id'glyph'
6554       or (item.id == 7 and item.subtype == 2) then
```

202
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
dir = dir or 'l'
if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' 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 == Babel.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.

if dir_real = dir -- We need dir_real to set strong below
if dir == 'al' then dir = 'r' end -- W3

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:

if strong == 'al' then
  if dir == 'en' then dir = 'an' end -- W2
  if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
  strong_lr = 'r' -- W3
end
Once finished the basic setup for glyphs, consider the two other cases: dir node and the rest.

```plaintext
6613 elseif item.id == node.id'dir' and not inmath then
6614     new_dir = true
6615     dir = nil
6616 elseif item.id == node.id'math' then
6617     inmath = (item.subtype == 0)
6618 else
6619     dir = nil -- Not a char
6620 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 text dir is set. This means you cannot insert, say, a whatsit, but this is what I would expect (with luacolor you may colorize some digits). Anyway, this behavior could be changed with a switch in the future. Note in the first branch only <an> is relevant if <al>.

```plaintext
6621 if dir == 'en' or dir == 'an' or dir == 'et' then
6622     if dir ~= 'et' then
6623         type_n = dir
6624     end
6625     first_n = first_n or item
6626     last_n = last_es or item
6627     last_es = nil
6628 elseif dir == 'es' and last_n then -- W3+W6
6629     last_es = item
6630 elseif dir == 'cs' then -- it's right - do nothing
6631 elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6632     if strong_lr == 'r' and type_n ~= '' then
6633         dir_mark(head, first_n, last_n, 'r')
6634     elseif strong_lr == 'l' and first_d and type_n == 'an' then
6635         dir_mark(head, first_n, last_n, 'r')
6636     elseif dir == 'l' and first_d and last_d, outer)
6637     first_d, last_d = nil, nil
6638     elseif strong_lr == 'l' and type_n == '' then
6639     last_d = last_n
6640     end
6641     type_n = ''
6642     first_n, last_n = nil, nil
6643 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 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:

```plaintext
6644 if dir == 'l' or dir == 'r' then
6645     if dir ~= outer then
6646         first_d = first_d or item
6647     last_d = item
6648 elseif first_d and dir ~= strong_lr then
6649     dir_mark(head, first_d, last_d, outer)
6650     first_d, last_d = nil, nil
6651 end
6652 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>, resp, 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.

```plaintext
6653 if dir and not last_lr and dir ~= 'l' and outer == 'r' then
```

204
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
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
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

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
end (/basic-r)

And here the Lua code for bidi=basic:

{+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)
  local new_state = state
  if state.sim and state.eim and state.sim ~= state.eim then
    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

-- TODO - \hbox with an explicit dir can lead to wrong results
-- \hspace{<R \hbox dir TLT{<R>}} and \hspace{<L \hbox dir TRT{<L>}}. A small attempt
-- was 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

  -- implementation continues here
local has_en = false
local first_et = nil
local ATDIR = Babel.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
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] end
          elseif d == 'nsm' then d_font = et[3] end
        end
      break
    end
    end
  end
end
d = d or 'l'

-- A short 'pause' in bidi for mapfont

if d_font and fontmap and fontmap[d_font][item_r.font] then
    item_r.font = fontmap[d_font][item_r.font]
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 = 'al'
    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
-- 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

-- 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)
else 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

-- Force mathdir in math if ON (currently works as expected only
-- with 'l')
if inmath and d == 'on' then
  d = ('TRT' == tex.mathdir) and 'r' or 'l'
end
if d then
  if d == 'al' then
    d = 'r'
    last = 'al'
  elseif d == 'l' or d == 'r' then

last = d

prev_d = d

table.insert(nodes, {item, d, outer_first})

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    -- dir 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
end

-- dummy node, to close things

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]
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
first_on = nil
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
        elseif state.san then
            head, state = insert_numeric(head, state)
        end
    if outer == 'l' then
        if d == 'an' or d == 'r' then -- im -> implicit
            if d == 'r' then state.has_r = true end
            state.sim = state.sim or item
            state.eim = item
            elseif d == 'l' and state.sim and state.has_r then
                head, state = insert_implicit(head, state, outer)
            elseif d == 'l' then
                state.sim, state.eim, state.has_r = nil, nil, false
            end
        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
            end
            state.eim = item
end
```latex
7057    \textbf{elseif d == 'r' and state.sim then}
7058    \quad head, state = insert_implicit(head, state, outer)
7059    \textbf{elseif d == 'r' then}
7060    \quad state.sim, state.eim = nil, nil
7061    \textbf{end}
7062    \textbf{end}
7063
7064    \textbf{if isdir then}
7065    \quad last = d \quad \text{-- Don't search back - best save now}
7066    \textbf{elseif d == 'on' and state.san then}
7067    \quad state.san = state.san or item
7068    \quad state.ean = item
7069    \textbf{end}
7070    \textbf{end}
7071    \textbf{return node.prev(head) or head}
7072\textbf{end}
7073\textbf{end}
7074\textbf{/basic}
```

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.

```latex
\textbf{\LdfInit{nil}{datenil}}
```

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.

```latex
\textbf{providehyphenmins{\CurrentOption}{\m@ne\m@ne}}
```
The next step consists of defining commands to switch to (and from) the ‘nil’ language.

\caption{nil}
\datenil
7087 \let\captionnil\@empty
7088 \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.

7089 \ldf@finish(nil)}
7090 (/nil)

16 Support for Plain \TeX \texttt{(plain.def)}

16.1 Not renaming \texttt{hyphen.tex}

As Don Knuth has declared that the filename \texttt{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-\texttt{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 locally\texttt{hyphen.tex} or whatever they like, but they mustn't diddle with \texttt{hyphen.tex} (or plain.tex except to preload additional fonts).

The files \texttt{bplain.tex} and \texttt{blplain.tex} can be used as replacement wrappers around \texttt{plain.tex} and \texttt{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 \texttt{bplain.fmt} or \texttt{blplain.fmt}, which you can use as replacements for \texttt{plain.fmt} and \texttt{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}.

7091 ⟨\texttt{bplain} \textbar \texttt{blplain}⟩
7092 \catcode`\{=1 % left brace is begin-group character
7093 \catcode`\}=2 % right brace is end-group character
7094 \catcode`\#=6 % hash mark is macro parameter character

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).

7095 \openin 0 hyphen.cfg
7096 \ifeof0
7097 \else
7098 \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.

7099 \def\input #1 {%
7100 \let\input\a
7101 \a hyphen.cfg
7102 \let\a\undefined
7103 }
7104 \fi
7105 ⟨\texttt{bplain} \textbar \texttt{blplain}⟩

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}.

7106 (\texttt{bplain})\a plain.tex
7107 (\texttt{blplain})\a lplain.tex
Finally we change the contents of \fmtname to indicate that this is not the plain format, but a format based on plain with the babel package preloaded.

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 file babel.def expects some definitions made in the \LaTeX2ε style file. So, in 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 \belfeaturestrings and \beloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

16.3 General tools

A number of \LaTeX macro’s that are needed later on.
\LaTeX has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.

\begin{verbatim}
\ifx\@preamblecmds\@undefined
\def\@preamblecmds{}
\fi
\def\@onlypreamble#1{%
\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}%
\@preamblecmds
\global\let\do\noexpand
\ifx\@begindocumenthook\@undefined
\def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
\end{verbatim}

We also have to mimic \LaTeX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

\begin{verbatim}
\ifx\@preamblecmds\@undefined
\def\@preamblecmds{}
\fi
\def\@onlypreamble#1{%
\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}%
\@preamblecmds
\global\let\do\noexpand
\ifx\@begindocumenthook\@undefined
\def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
\end{verbatim}

Mimick \LaTeX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

\begin{verbatim}
\ifx\@begindocumenthook\@undefined
\def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\end{verbatim}
\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

Mimick \LaTeX's commands to define control sequences.
\def\newcommand\@star@or@long\new@command
\def\new@command#1{% 
\@testopt\@newcommand#10
\def\@newcommand#1[#2]{% 
\@ifnextchar [{{{\@xargdef#1[#2]}}}% 
{\@argdef#1[#2]}}
\long\def\@argdef#1[#2]#3{% 
\@yargdef#1@ne{#2}{#3}}
\long\def\@xargdef#1[#2][#3]#4{% 
\expandafter\def\expandafter#1\expandafter{\expandafter\@protected@testopt\expandafter #1% 
\csname\string#1\endcsname{#3}}% 
\expandafter\@yargdef \csname\string#1\endcsname\tw@{#2}{#4}}
\def\providecommand\@star@or@long\provide@command
\def\provide@command#1{% 
\begingroup\escapechar\m@ne\xdef\@gtempa{{\string#1}}% 
\endgroup\expandafter\@ifundefined\@gtempa {\def\reserved@a{\new@command#1}}% 
{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}% 
\reserved@a}%
\def\DeclareRobustCommand\@star@or@long\declare@robustcommand
\def\declare@robustcommand#1{% 
\edef\reserved@a{\string#1}% 
\def\reserved@b{#1}% 
\edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}% 
\edef#1{% 
\ifx\reserved@a\reserved@b 
\else 
\expandafter\@edef\reserved@a{\reserved@a}% 
\fi 
\edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}% 
\edef@reserved@b}
The following little macro \in@ is taken from latex.ltx; it checks whether its first argument is part of its second argument. It uses the boolean \in@, allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of \bbl@tempa.

\def\bbl@tempa{\csname newif\endcsname\in@}
\catcode\&=4
\ifx\in@\@undefined
\def\in@#1#2{\def\in@@##1#1##2##3\in@@{\ifx\in@##2\in@false\else\in@true\fi}\in@@#2#1\in@\in@@}
\else
\let\bbl@tempa\@empty
\fi
\bbl@tempa

\LaTeX 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 \TeX 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\_2\_e versions; just enough to make things work in plain \TeX environments.

\if\tempcnta\@undefined
\csname newcount\endcsname\tempcnta\relax
\fi
\if\tempcntb\@undefined
\csname newcount\endcsname\tempcntb\relax
\fi

To prevent wasting two counters in \LaTeX (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

\if\bye\@undefined
\advance\count10 by -2\relax
\fi
\if\ifnextchar\@undefined

217
16.4 Encoding related macros

Code from \texttt{ltoutenc.dtx}, adapted for use in the plain \TeX environment.
Currently we only use the \LaTeX{} 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 \texttt{\scriptsize} to be available. Because plain \TeX{} doesn't have such a sophisticated font mechanism as \LaTeX{} has, we just \texttt{\let} it to \texttt{\sevenrm}.

And a few more “dummy” definitions.
A proxy file:

\input babel.def

\section{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.

\section*{References}


