Babel

Localization and internationalization

Unicode
\TeX
\pdf\TeX
Lua\TeX
\Xe\TeX
# Source code

5 Identification and loading of required files

6 locale directory

7 Tools

7.1 Multiple languages

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

7.3 base

7.4 key=value options and other general option

7.5 Conditional loading of shorthands

7.6 Interlude for Plain

8 Multiple languages

8.1 Selecting the language

8.2 Errors

8.3 Hooks

8.4 Setting up language files

8.5 Shorthands

8.6 Language attributes

8.7 Support for saving macro definitions

8.8 Short tags

8.9 Hyphens

8.10 Multiencoding strings

8.11 Macros common to a number of languages

8.12 Making glyphs available

8.12.1 Quotation marks

8.12.2 Letters

8.12.3 Shorthands for quotation marks

8.12.4 Umlautsandtremas

8.13 Layout

8.14 Load engine specific macros

8.15 Creating and modifying languages

9 Adjusting the Babel behavior

9.1 Cross referencing macros

9.2 Marks

9.3 Preventing clashes with other packages

9.3.1 \texttt{ifthen}

9.3.2 \texttt{varioref}

9.3.3 \texttt{hhline}

9.4 Encoding and fonts

9.5 Basic bidi support

9.6 Local Language Configuration

9.7 Language options

10 The kernel of Babel (babel.def, common)

11 Loading hyphenation patterns

12 Font handling with fontspec

---

II Source code

5 Identification and loading of required files

6 locale directory

7 Tools

7.1 Multiple languages

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

7.3 base

7.4 key=value options and other general option

7.5 Conditional loading of shorthands

7.6 Interlude for Plain

8 Multiple languages

8.1 Selecting the language

8.2 Errors

8.3 Hooks

8.4 Setting up language files

8.5 Shorthands

8.6 Language attributes

8.7 Support for saving macro definitions

8.8 Short tags

8.9 Hyphens

8.10 Multiencoding strings

8.11 Macros common to a number of languages

8.12 Making glyphs available

8.12.1 Quotation marks

8.12.2 Letters

8.12.3 Shorthands for quotation marks

8.12.4 Umlautsandtremas

8.13 Layout

8.14 Load engine specific macros

8.15 Creating and modifying languages

9 Adjusting the Babel behavior

9.1 Cross referencing macros

9.2 Marks

9.3 Preventing clashes with other packages

9.3.1 \texttt{ifthen}

9.3.2 \texttt{varioref}

9.3.3 \texttt{hhline}

9.4 Encoding and fonts

9.5 Basic bidi support

9.6 Local Language Configuration

9.7 Language options

10 The kernel of Babel (babel.def, common)

11 Loading hyphenation patterns

12 Font handling with fontspec
## 13 Hooks for XeTeX and LuaTeX

- **13.1 XeTeX** 164
- **13.2 Layout** 166
- **13.3 LuaTeX** 168
- **13.4 Southeast Asian scripts** 174
- **13.5 CJK line breaking** 175
- **13.6 Arabic justification** 177
- **13.7 Common stuff** 181
- **13.8 Automatic fonts and ids switching** 182
- **13.9 Bidi** 186
- **13.10 Layout** 188
- **13.11 Lua: transforms** 192
- **13.12 Lua: Auto bidi with basic and basic-r** 200

## 14 Data for CJK

## 15 The 'nil' language

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

- **16.1 Not renaming hyphen.tex** 212
- **16.2 Emulating some \TeX features** 213
- **16.3 General tools** 213
- **16.4 Encoding related macros** 217

## 17 Acknowledgements

### Troubleshooting

- Paragraph ended before `\UTFviii@three@octets` was complete 5
- No hyphenation patterns were preloaded for (babel) the language 'LANG' into the format 6
- You are loading directly a language style 8
- Unknown language 'LANG' 9
- Argument of `\language@active@arg"has an extra }` 12
- Package fontspec Warning: Language 'LANG' not available for font 'FONT' with script 'SCRIPT' 'Default' language used instead 28
- Package babel Info: The following fonts are not babel standard families 28
Part I

User guide

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 New X.XX, and there are some notes for the latest versions in the babel site. The most recent features can be still unstable.

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

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

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

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

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

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 lmroman. Other scripts require loading fontspec. You may want to set the font attributes with fontspec, too.

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

Now consider something like:

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

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

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

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

TROUBLESHOOTING A common source of trouble is a wrong setting of the input encoding. Depending on the \TeX 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.
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.

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
(language `LANG' into the format.
Please, configure your TeX system to add them and rebuild the format. Now I will use the patterns preloaded for \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.

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.

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`).

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

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

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

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

Examples of cases where `main` is useful are the following.

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

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`:

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

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

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

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

```latex
\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.

```latex
\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):

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

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

1.5 Troubleshooting

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

1. No predefined “axis” for modifiers are provided because languages and their scripts have quite different needs.
2. In old versions the error read “You have used an old interface to call babel”, not very helpful.


 Another typical error when using babel is the following:

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

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

```latex
\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** \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.

\foreignlanguage{⟨option-list⟩}{⟨language⟩}{⟨text⟩}

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

**New 3.44** As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

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

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

### 1.8 Auxiliary language selectors

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

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

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

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

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

### 1.9 More on selection

\texttt{\textbackslash babeltags}\{(tag1) = \texttt{\{language1\}}, \{tag2\} = \texttt{\{language2\}},\ldots\}

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

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

**WARNING** There is a clear drawback to this feature, namely, the 'prefix' \texttt{\textbackslash text...} is heavily overloaded in \TeX{} and conflicts with existing macros may arise (\texttt{\textbackslash textlatin}, \texttt{\textbackslash textbar}, \texttt{\textbackslash textit}, \texttt{\textbackslash textcolor} and many others). The same applies to environments, because \texttt{arabic} conflicts with \texttt{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

\begin{Verbatim}
\texttt{\textbackslash babeltags\{de = german\}}
\end{Verbatim}

you can write

\begin{Verbatim}
text \texttt{\textbackslash text\{German text\}} text
\end{Verbatim}

and

\begin{Verbatim}
text
\texttt{\textbackslash begin\{\texttt{de\}\}}
\texttt{\textbackslash German text}
\texttt{\textbackslash end\{\texttt{de\}\}}
text
\end{Verbatim}

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

**NOTE** Actually, there may be another advantage in the 'short' syntax \texttt{\textbackslash text\{tag\}}, namely, it is not affected by \texttt{\textbackslash MakeUppercase} (while \texttt{\textbackslash foreignlanguage} is).
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, \babel ensure redefines the captions for a given language to wrap them with a selector:

\babel ensure{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:

\babel ensure[include=\Today]{spanish}
\babel ensure[fontenc=T5]{vietnamese}

They are activated when the language is selected (at the after extras event), and it makes some assumptions which could not be fulfilled in some languages. Note also you should include only macros defined by the language, not global macros (eg, \TeX or \dag).

With ini files (see below), captions are ensured by default.

### 1.10 Shorthands

A shorthand is a sequence of one or two characters that expands to arbitrary \TeX code. Shorthands can be used for different kinds of things; for example: (1) in some languages shorthands such as ”a“ are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as \ are used to insert the right amount of white space; (3) several kinds of discretionary 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: user, language, system, and language user (by order of precedence). In most cases, you will use only shorthands provided by languages.

**NOTE** Keep in mind the following:

1. Activated chars used for two-char shorthands cannot be followed by a closing brace } and the spaces following are gobbled. With one-char shorthands (eg. \), they are preserved.
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:
It means there is a closing brace just after a shorthand, which is not allowed (e.g., "). Just add {} after (e.g., "{}").

\shorthandoff \{(shorthand-list)\}
\shorthandon \{(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 \{[\langle language\rangle,\langle language\rangle,...\}\{\langle shorthand\}\} {\langle code\rangle}

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

New 3.9a An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add \languageshorthands{\langle lang\rangle} to the corresponding \extras{\langle lang\rangle}, as explained below). By default, user shorthands are (re)defined.

User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.
Let's assume you want a unified set of shorthand for discretionaries (languages do not
define shorthands consistently, and "-, \-, " = have different meanings). You can start with, say:

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

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

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

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

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

\languageshorthands \langle\text{language}\rangle

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).\footnote{Actually, any name not corresponding to a language group does the same as none. However, follow this con-
vention because it might be enforced in future releases of babel to catch possible errors.} Note that for this to work the language should have been specified as
an option when loading the babel package. For example, you can use in english the
shorthands defined by ngerman with

\addto\extrasenglish{\languageshorthands{ngerman}}

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

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

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

\babelshorthand \langle\text{shorthand}\rangle

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

\textbf{EXAMPLE} Since by default shorthands are not activated until \begin{document}, you may use this
macro when defining the \texttt{title} in the preamble:
For your records, here is a list of shorthands, but you must double check them, as they may change:\footnote{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{This declaration serves to nothing, but it is preserved for backward compatibility.}

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

\textbf{New 3.23} Tests if a character has been made a shorthand.

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

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

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

\textbf{EXAMPLE} The following example shows how to replace a shorthand by another

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

\textbf{WARNING} Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand if found, ^ expands to a non-breaking space, because this is the value of ~ (internally, ^ still calls \active@char~ or \normal@char~).

Furthermore, if you change the system value of ^ with \defineshorthand nothing happens.
1.11 Package options

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

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

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

activegrave Same for `. 

shorthands= ⟨char⟩⟨char⟩... | off
The only language shorthands activated are those given, like, eg:

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

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

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

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

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

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

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

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

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

strings= generic | unicode | encoded | \{label\} | \{font encoding\}

Selects the encoding of strings in languages supporting this feature. Predefined labels are generic (for traditional \TeX, LICR and ASCII strings), unicode (for engines like \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 \MakeUppercase and the like (this feature misuses some internal \TeX tools, so use it only as a last resort).

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

New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.\footnote{Turned off in plain.} It can take the following values:

off deactivates this feature and no case mapping is applied;

first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first in heads and foots if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if a single language option has been stated;\footnote{Duplicated options count as several ones.}

select sets it only at \selectlanguage;

other also sets it at other language;

other* also sets it at otherlanguage* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other* for monolingual documents.\footnote{Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either \texttt{xetex} or \texttt{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.}

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

New 3.14 Selects the bidi algorithm to be used in \texttt{luatex} and \texttt{xetex}. See sec. 1.24.

layout= New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.24.
1.12 The base option

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

\AfterBabelLanguage \langle option-name \rangle \{ \langle code \rangle \}

This command is currently the only provided by base. Executes \langle code \rangle when the file loaded by the corresponding package option is finished (at \lf@finish). The setting is global. So \AfterBabelLanguage{french}{...} does ... at the end of french.1df. It can be used in 1df files, too, but in such a case the code is executed only if \langle option-name \rangle is the same as \CurrentOption (which could not be the same as the option name as set in \usepackage!).

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

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

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

1.13 ini files

An alternative approach to define a language (or, more precisely, a locale) is by means of an ini file. Currently babel provides about 200 of these files containing the basic data required for a locale.

ini files are not meant only for babel, and they have been devised as a resource for other packages. To ease 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 \belpprovide. In other words, \belpprovide is mainly meant for auxiliary tasks, and as alternative when the 1df, for some reason, does work as expected.

EXAMPLE Although Georgian has its own 1df file, here is how to declare this language with an ini file in Unicode engines.

\documentclass{book}
\usepackage{belp}
\belpprovide[import, main]{georgian}
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:

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

Or also:

```latex
\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 have 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:

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

**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 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:
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:

\begin{verbatim}
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1دعاء 1مرة 1وى 1ى 1ي 1ي} % Random
\end{verbatim}

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

\begin{verbatim}
\documentclass[japanese]{ltjbook}
\usepackage{babel}
\end{verbatim}

**Latin, Greek, Cyrillic** Combining chars with the default luatex font renderer might be wrong; on the 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>af</th>
<th>Afrikaansul</th>
</tr>
</thead>
<tbody>
<tr>
<td>agq</td>
<td>Aghem</td>
</tr>
<tr>
<td>ak</td>
<td>Akan</td>
</tr>
<tr>
<td>am</td>
<td>Amharicul</td>
</tr>
<tr>
<td>ar</td>
<td>Arabicul</td>
</tr>
<tr>
<td>ar-DZ</td>
<td>Arabicul</td>
</tr>
<tr>
<td>ar-MA</td>
<td>Arabicul</td>
</tr>
<tr>
<td>ar-SY</td>
<td>Arabicul</td>
</tr>
<tr>
<td>as</td>
<td>Assamese</td>
</tr>
<tr>
<td>asa</td>
<td>Asu</td>
</tr>
<tr>
<td>ast</td>
<td>Asturianul</td>
</tr>
<tr>
<td>az-Cyr</td>
<td>Azerbaijani</td>
</tr>
<tr>
<td>az-Latn</td>
<td>Azerbaijaniul</td>
</tr>
<tr>
<td>az</td>
<td>Azerbaijaniul</td>
</tr>
<tr>
<td>bas</td>
<td>Basaa</td>
</tr>
<tr>
<td>be</td>
<td>Belarussianul</td>
</tr>
<tr>
<td>bem</td>
<td>Bemba</td>
</tr>
<tr>
<td>bez</td>
<td>Bena</td>
</tr>
<tr>
<td>bg</td>
<td>Bulgarianul</td>
</tr>
<tr>
<td>bm</td>
<td>Bambara</td>
</tr>
<tr>
<td>bn</td>
<td>Banglaul</td>
</tr>
<tr>
<td>bo</td>
<td>Tibetanu</td>
</tr>
<tr>
<td>brx</td>
<td>Bodo</td>
</tr>
<tr>
<td>bs-Cyr</td>
<td>Bosphian</td>
</tr>
<tr>
<td>bs-Latn</td>
<td>Bosphianul</td>
</tr>
<tr>
<td>bs</td>
<td>Bosphianul</td>
</tr>
<tr>
<td>ca</td>
<td>Catalanul</td>
</tr>
<tr>
<td>ce</td>
<td>Chechen</td>
</tr>
<tr>
<td>cgg</td>
<td>Chiga</td>
</tr>
<tr>
<td>chr</td>
<td>Cherokee</td>
</tr>
<tr>
<td>ckb</td>
<td>Central Kurdish</td>
</tr>
<tr>
<td>cop</td>
<td>Coptic</td>
</tr>
<tr>
<td>cs</td>
<td>Czechul</td>
</tr>
<tr>
<td>cu</td>
<td>Church Slavic</td>
</tr>
<tr>
<td>cu-Cyrs</td>
<td>Church Slavic</td>
</tr>
<tr>
<td>cu-Glag</td>
<td>Church Slavic</td>
</tr>
<tr>
<td>Code</td>
<td>Language</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>mr</td>
<td>Marathi</td>
</tr>
<tr>
<td>ms-BN</td>
<td>Malay</td>
</tr>
<tr>
<td>ms-SG</td>
<td>Malay</td>
</tr>
<tr>
<td>ms</td>
<td>Maltese</td>
</tr>
<tr>
<td>mt</td>
<td>Maltese</td>
</tr>
<tr>
<td>mua</td>
<td>Mundang</td>
</tr>
<tr>
<td>my</td>
<td>Burmese</td>
</tr>
<tr>
<td>mzn</td>
<td>Mazanderani</td>
</tr>
<tr>
<td>naq</td>
<td>Nama</td>
</tr>
<tr>
<td>nb</td>
<td>Norwegian Bokmål</td>
</tr>
<tr>
<td>nd</td>
<td>North Ndebele</td>
</tr>
<tr>
<td>ne</td>
<td>Nepali</td>
</tr>
<tr>
<td>nl</td>
<td>Dutch</td>
</tr>
<tr>
<td>nmg</td>
<td>Kwasio</td>
</tr>
<tr>
<td>nn</td>
<td>Norwegian Nynorsk</td>
</tr>
<tr>
<td>nnh</td>
<td>Ngiemboon</td>
</tr>
<tr>
<td>nus</td>
<td>Nuer</td>
</tr>
<tr>
<td>nyn</td>
<td>Nyankole</td>
</tr>
<tr>
<td>om</td>
<td>Oromo</td>
</tr>
<tr>
<td>or</td>
<td>Odia</td>
</tr>
<tr>
<td>os</td>
<td>Ossetic</td>
</tr>
<tr>
<td>pa-Arab</td>
<td>Punjabi</td>
</tr>
<tr>
<td>pa-Guru</td>
<td>Punjabi</td>
</tr>
<tr>
<td>pa</td>
<td>Punjabi</td>
</tr>
<tr>
<td>pl</td>
<td>Polish</td>
</tr>
<tr>
<td>pms</td>
<td>Piedmontese</td>
</tr>
<tr>
<td>ps</td>
<td>Pashto</td>
</tr>
<tr>
<td>pt-BR</td>
<td>Portuguese</td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
</tr>
<tr>
<td>pt</td>
<td>Portuguese</td>
</tr>
<tr>
<td>qu</td>
<td>Quechua</td>
</tr>
<tr>
<td>rm</td>
<td>Romansh</td>
</tr>
<tr>
<td>rn</td>
<td>Rundi</td>
</tr>
<tr>
<td>ro</td>
<td>Romanian</td>
</tr>
<tr>
<td>rof</td>
<td>Rombo</td>
</tr>
<tr>
<td>ru</td>
<td>Russian</td>
</tr>
<tr>
<td>rw</td>
<td>Kinyarwanda</td>
</tr>
<tr>
<td>rwk</td>
<td>Rwa</td>
</tr>
<tr>
<td>sa-Beng</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>sa-Deva</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>sa-Gujr</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>sa-Knda</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>sa-Mlym</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>sa-Telu</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>sa</td>
<td>Sanskrit</td>
</tr>
<tr>
<td>sah</td>
<td>Sakha</td>
</tr>
<tr>
<td>saq</td>
<td>Samburu</td>
</tr>
<tr>
<td>sbp</td>
<td>Sangu</td>
</tr>
<tr>
<td>se</td>
<td>Northern Sami</td>
</tr>
<tr>
<td>seh</td>
<td>Sena</td>
</tr>
<tr>
<td>ses</td>
<td>Koyraboro Senni</td>
</tr>
<tr>
<td>sg</td>
<td>Sango</td>
</tr>
<tr>
<td>shi-Latn</td>
<td>Tachelhit</td>
</tr>
<tr>
<td>shi-Tfng</td>
<td>Tachelhit</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</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>aghem</td>
<td>burmese</td>
</tr>
<tr>
<td>akan</td>
<td>canadian</td>
</tr>
<tr>
<td>albanian</td>
<td>cantonese</td>
</tr>
<tr>
<td>american</td>
<td>catalan</td>
</tr>
<tr>
<td>amharic</td>
<td>centralatlastamazight</td>
</tr>
<tr>
<td>ancientgreek</td>
<td>centralkurdish</td>
</tr>
<tr>
<td>arabic</td>
<td>chechen</td>
</tr>
<tr>
<td>arabic-algeria</td>
<td>cherokee</td>
</tr>
<tr>
<td>arabic-DZ</td>
<td>chiga</td>
</tr>
<tr>
<td>arabic-morocco</td>
<td>chinese-hans-hk</td>
</tr>
<tr>
<td>arabic-MA</td>
<td>chinese-hans-mo</td>
</tr>
<tr>
<td>arabic-syria</td>
<td>chinese-hans-sg</td>
</tr>
<tr>
<td>arabic-SY</td>
<td>chinese-hans</td>
</tr>
<tr>
<td>armenian</td>
<td>chinese-hant-hk</td>
</tr>
<tr>
<td>assamese</td>
<td>chinese-hant-mo</td>
</tr>
<tr>
<td>asturian</td>
<td>chinese-hant</td>
</tr>
<tr>
<td>asu</td>
<td>chinese-simplified-hongkongsarchina</td>
</tr>
<tr>
<td>australian</td>
<td>chinese-simplified-macausarchina</td>
</tr>
<tr>
<td>austrian</td>
<td>chinese-simplified-singapore</td>
</tr>
<tr>
<td>azerbaijani-cyrillic</td>
<td>chinese-simplified</td>
</tr>
<tr>
<td>azerbaijani-cyril</td>
<td>chinese-traditional-hongkongsarchina</td>
</tr>
<tr>
<td>azerbaijani-latin</td>
<td>chinese-traditional-macausarchina</td>
</tr>
<tr>
<td>azerbaijani-latn</td>
<td>chinese-traditional</td>
</tr>
<tr>
<td>azerbaijani</td>
<td>chinese</td>
</tr>
<tr>
<td>bafia</td>
<td>churchslavic</td>
</tr>
<tr>
<td>bambara</td>
<td>churchslavic-cyrs</td>
</tr>
<tr>
<td>basaa</td>
<td>churchslavic-oldcyrillic (^{12})</td>
</tr>
<tr>
<td>basque</td>
<td>churchslavic-glagolitic</td>
</tr>
<tr>
<td>belarusian</td>
<td>colognian</td>
</tr>
<tr>
<td>bema</td>
<td>cornish</td>
</tr>
<tr>
<td>bengali</td>
<td>croatian</td>
</tr>
<tr>
<td>bodo</td>
<td>czech</td>
</tr>
<tr>
<td>bosnian-cyrillic</td>
<td>danish</td>
</tr>
<tr>
<td>bosnian-cyrl</td>
<td>duala</td>
</tr>
<tr>
<td>bosnian-latin</td>
<td>dutch</td>
</tr>
<tr>
<td>bosnian-ltn</td>
<td>dzongkha</td>
</tr>
<tr>
<td>bosnian</td>
<td>embu</td>
</tr>
<tr>
<td>brazilian</td>
<td>english-au</td>
</tr>
<tr>
<td>breton</td>
<td>english-australia</td>
</tr>
<tr>
<td>british</td>
<td>english-ca</td>
</tr>
<tr>
<td>bulgarian</td>
<td>english-canada</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 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Zealand</td>
<td>newzealand</td>
</tr>
<tr>
<td>Ngemboon</td>
<td>ngiemboon</td>
</tr>
<tr>
<td>Ngombe</td>
<td>ngomba</td>
</tr>
<tr>
<td>Norwegian</td>
<td>norsk</td>
</tr>
<tr>
<td>Northern Luri</td>
<td>northernluri</td>
</tr>
<tr>
<td>Northern Sami</td>
<td>northernsami</td>
</tr>
<tr>
<td>Northendebele</td>
<td>northndebele</td>
</tr>
<tr>
<td>Norwegian Bokmål</td>
<td>norwegianbokmal</td>
</tr>
<tr>
<td>Norwegian Nynorsk</td>
<td>norwegiannynorsk</td>
</tr>
<tr>
<td>Nuer</td>
<td>nuer</td>
</tr>
<tr>
<td>Nyankole</td>
<td>nyankole</td>
</tr>
<tr>
<td>Nynorsk</td>
<td>nynorsk</td>
</tr>
<tr>
<td>Occitan</td>
<td>occitan</td>
</tr>
<tr>
<td>Orya</td>
<td>oriya</td>
</tr>
<tr>
<td>Oromo</td>
<td>oromo</td>
</tr>
<tr>
<td>Ossetic</td>
<td>ossetic</td>
</tr>
<tr>
<td>Pashto</td>
<td>pashto</td>
</tr>
<tr>
<td>Persian</td>
<td>persian</td>
</tr>
<tr>
<td>Piedmontese</td>
<td>piedmontese</td>
</tr>
<tr>
<td>Polish</td>
<td>polish</td>
</tr>
<tr>
<td>Polytonic Greek</td>
<td>polytonicgreek</td>
</tr>
<tr>
<td>Portuguese-BR</td>
<td>portuguese-br</td>
</tr>
<tr>
<td>Portuguese-Brazil</td>
<td>portuguese-brazil</td>
</tr>
<tr>
<td>Portuguese-Portugal</td>
<td>portuguese-portugal</td>
</tr>
<tr>
<td>Portuguese-PT</td>
<td>portuguese-pt</td>
</tr>
<tr>
<td>Portuguese</td>
<td>portuguese</td>
</tr>
<tr>
<td>Punjabi-Arab</td>
<td>punjabi-arab</td>
</tr>
<tr>
<td>Punjabi-Arabic</td>
<td>punjabi-arabic</td>
</tr>
<tr>
<td>Punjabi-Gurmukhi</td>
<td>punjabi-gurmukhi</td>
</tr>
<tr>
<td>Punjabi-Guru</td>
<td>punjabi-guru</td>
</tr>
<tr>
<td>Punjabi</td>
<td>punjabi</td>
</tr>
<tr>
<td>Quechua</td>
<td>quechua</td>
</tr>
<tr>
<td>Romanian</td>
<td>romanian</td>
</tr>
<tr>
<td>Romansh</td>
<td>romansh</td>
</tr>
<tr>
<td>Rombo</td>
<td>rombo</td>
</tr>
<tr>
<td>Rundi</td>
<td>rundi</td>
</tr>
<tr>
<td>Russian</td>
<td>russian</td>
</tr>
<tr>
<td>Rwa</td>
<td>rwa</td>
</tr>
<tr>
<td>Sakha</td>
<td>sakha</td>
</tr>
<tr>
<td>Samburu</td>
<td>samburu</td>
</tr>
<tr>
<td>Samin</td>
<td>samin</td>
</tr>
<tr>
<td>Sango</td>
<td>sango</td>
</tr>
<tr>
<td>Sangu</td>
<td>sangu</td>
</tr>
<tr>
<td>Sanskrit-Bengali</td>
<td>sanskrit-bengali</td>
</tr>
<tr>
<td>Sanskrit-Devanagari</td>
<td>sanskrit-devanagari</td>
</tr>
<tr>
<td>Sanskrit-Gujarati</td>
<td>sanskrit-gujarati</td>
</tr>
<tr>
<td>Sanskrit-Gujr</td>
<td>sanskrit-gujr</td>
</tr>
<tr>
<td>Sanskrit-Kannada</td>
<td>sanskrit-kannada</td>
</tr>
<tr>
<td>Sanskrit-Knda</td>
<td>sanskrit-knda</td>
</tr>
<tr>
<td>Sanskrit-Malayalam</td>
<td>sanskrit-malayalam</td>
</tr>
<tr>
<td>Sanskrit-Mlym</td>
<td>sanskrit-mlym</td>
</tr>
<tr>
<td>Sanskrit-Telu</td>
<td>sanskrit-telu</td>
</tr>
<tr>
<td>Sanskrit-Telugu</td>
<td>sanskrit-telugu</td>
</tr>
<tr>
<td>Sanskrit</td>
<td>sanskrit</td>
</tr>
<tr>
<td>Scottish Gaelic</td>
<td>scottishgaelic</td>
</tr>
<tr>
<td>Sena</td>
<td>sena</td>
</tr>
<tr>
<td>Serbian-Cyrilic</td>
<td>serbian-cyrilic-bosniaherzegovina</td>
</tr>
<tr>
<td>Serbian-Cyrilic-Kosovo</td>
<td>serbian-cyrilic-kosovo</td>
</tr>
<tr>
<td>Serbian-Cyrilic-Montenegro</td>
<td>serbian-cyrilic-montenegro</td>
</tr>
<tr>
<td>Serbian-Cyrilic</td>
<td>serbian-cyrilic</td>
</tr>
<tr>
<td>Serbian-Cyril-Ba</td>
<td>serbian-cyril-ba</td>
</tr>
<tr>
<td>Serbian-Cyril-Me</td>
<td>serbian-cyril-me</td>
</tr>
<tr>
<td>Serbian-Cyril-Xk</td>
<td>serbian-cyril-xk</td>
</tr>
<tr>
<td>Serbian-Cyril</td>
<td>serbian-cyril</td>
</tr>
<tr>
<td>Serbian-Latin-Bosniaherzegovina</td>
<td>serbian-latin-bosniaherzegovina</td>
</tr>
<tr>
<td>Serbian-Latin-Kosovo</td>
<td>serbian-latin-kosovo</td>
</tr>
<tr>
<td>Serbian-Latin-Montenegro</td>
<td>serbian-latin-montenegro</td>
</tr>
<tr>
<td>Serbian-Latin</td>
<td>serbian-latin</td>
</tr>
<tr>
<td>Serbian-Latin-Ba</td>
<td>serbian-latin-ba</td>
</tr>
<tr>
<td>Serbian-Latin-Me</td>
<td>serbian-latin-me</td>
</tr>
<tr>
<td>Serbian-Latin-Xk</td>
<td>serbian-latin-xk</td>
</tr>
<tr>
<td>Serbian-Latin</td>
<td>serbian-latin</td>
</tr>
<tr>
<td>Serbian</td>
<td>serbian</td>
</tr>
<tr>
<td>Shambala</td>
<td>shambala</td>
</tr>
<tr>
<td>Shona</td>
<td>shona</td>
</tr>
<tr>
<td>Sichuany</td>
<td>sichuany</td>
</tr>
<tr>
<td>Sinhala</td>
<td>sinhala</td>
</tr>
<tr>
<td>Slovak</td>
<td>slovak</td>
</tr>
<tr>
<td>Slovene</td>
<td>slovene</td>
</tr>
<tr>
<td>Slovenian</td>
<td>slovenian</td>
</tr>
<tr>
<td>Soga</td>
<td>soga</td>
</tr>
<tr>
<td>Somali</td>
<td>somali</td>
</tr>
<tr>
<td>Spanish-Mexico</td>
<td>spanish-mexico</td>
</tr>
<tr>
<td>Spanish-MX</td>
<td>spanish-mx</td>
</tr>
<tr>
<td>Spanish</td>
<td>spanish</td>
</tr>
<tr>
<td>Standard Moroccan-Tifinagh</td>
<td>standardmoroccantamazight</td>
</tr>
<tr>
<td>Swahili</td>
<td>swahili</td>
</tr>
<tr>
<td>Swedish</td>
<td>swedish</td>
</tr>
<tr>
<td>Swiss German</td>
<td>swissgerman</td>
</tr>
<tr>
<td>Tachelhit-Latin</td>
<td>tachelhit-latin</td>
</tr>
<tr>
<td>Tachelhit-Latn</td>
<td>tachelhit-latn</td>
</tr>
<tr>
<td>Tachelhit-Tfng</td>
<td>tachelhit-tfng</td>
</tr>
<tr>
<td>Tachelhit-Tifinagh</td>
<td>tachelhit-tifinagh</td>
</tr>
<tr>
<td>Tachelhit</td>
<td>tachelhit</td>
</tr>
<tr>
<td>Taita</td>
<td>taita</td>
</tr>
<tr>
<td>Tamil</td>
<td>tamil</td>
</tr>
<tr>
<td>Tasawaq</td>
<td>tasawaq</td>
</tr>
<tr>
<td>Telugu</td>
<td>telugu</td>
</tr>
<tr>
<td>Teso</td>
<td>teso</td>
</tr>
<tr>
<td>Thai</td>
<td>thai</td>
</tr>
<tr>
<td>Tibetan</td>
<td>tibetan</td>
</tr>
<tr>
<td>Tigrinya</td>
<td>tigrinya</td>
</tr>
<tr>
<td>Tongan</td>
<td>tongan</td>
</tr>
<tr>
<td>Turkish</td>
<td>turkish</td>
</tr>
<tr>
<td>Turkmen</td>
<td>turkmen</td>
</tr>
</tbody>
</table>
Modifying and adding values to ini files

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

1.14 Selecting fonts

\babelfont

\babelfont [(language-list)]{(font-family)}[(font-options)]{(font-name)}

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

The main purpose of \babelfont is to define at once in a multilingual document the fonts required by the different languages, with their corresponding language systems (script and language). So, if you load, say, 4 languages, \babelfont {rm}{FreeSerif} defines 4 fonts (with their variants, of course), which are switched with the language by babel. It is a tool to make things easier and transparent to the user.

Here font-family is rm, sf or tt (or newly defined ones, as explained below), and font-name is the same as in fontspec and the like.

If no language is given, then it is considered the default font for the family, activated when a language is selected.

On the other hand, if there is one or more languages in the optional argument, the font will be assigned to them, overriding the default one. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, *devanagari). With this optional argument, the font is not yet defined, but just predeclared. This means you may define as many fonts as you want ‘just in case’, because if the language is never selected, the corresponding \babelfont declaration is just ignored.

Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need font-options, which is the same as in fontspec, but you may add further key/value pairs if necessary.

13See also the package combofont for a complementary approach.
EXAMPLE Usage in most cases is very simple. Let us assume you are setting up a document in Swedish, with some words in Hebrew, with a font suited for both languages.

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

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

\babelfont{rm}{Iwona}
\babelfont{hebrew}{rm}{FreeSerif}

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

EXAMPLE Here is how to do it:

\babelfont{kai}{FandolKai}

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

NOTE You may load fontspec explicitly. For example:

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

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

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

NOTE \texttt{fontspec} is not touched at all, only the preset font families (\texttt{rm}, \texttt{sf}, \texttt{tt}, and the like). If a language is switched when an \texttt{ad hoc} font is active, or you select the font with this command, neither the script nor the language is passed. You must add them by hand. This is by design, for several reasons —for example, each font has its own set of features and a generic setting for several of them can be problematic, and also preserving a “lower-level” font selection is useful.
NOTE The keys Language and Script just pass these values to the font, and do not set the script for the language (and therefore the writing direction). In other words, the ini file or \babelprovide provides default values for \babelfont if omitted, but the opposite is not true. See the note above for the reasons of this behavior.

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

TROUBLESHOOTING Package fontspec Warning: 'Language 'LANG' not available for font 'FONT' with script 'SCRIPT' 'Default' language used instead'.

This is not an error. This warning is shown by fontspec, not by babel. It 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.

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

  \babelfile{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:

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

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

\renewcommand\spanishchaptername{Foo}

This redefinition is immediate.

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

\AtBeginDocument{\renewcommand\contentsname{Foo}}

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

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

\addto\extrarussian{\mymacro}

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

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

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

\usepackage[danish]{babel}
\babelprovide[captions=da, hyphenrules=nomphlenation]{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 aditional 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[(options)]{language-name}

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

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

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

```latex
\langle\text{language-tag}\rangle
```

**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:

\babelprovide[import]{hungarian}

There are about 250 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages may show a warning about the current lack of suitability of some features.

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

**New 3.44** More convenient is usually \localdate, with prints the date for the current locale.
captions= ⟨language-tag⟩
  Loads only the strings. For example:

  \babelprovide[ captions=hu]{hungarian}

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

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

  If none of the listed hyphenrules exist, the default behavior applies. Note in this example we set chavacano as first option – 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 \texttt{luatex}, because you can add some patterns with \texttt{\textbackslash babelpatterns}, as for example:

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

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

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

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

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

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

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

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

  Remember there is an alternative syntax for the latter:

  \begin{verbatim}
  \usepackage[italian, polytonicgreek, provide=*]{babel}
  \end{verbatim}

script= ⟨script-name⟩
  Sets the script name to be used by fontspec (eg, \texttt{Devanagari}). Overrides the value in the ini file. If fontspec does not define it, then babel sets its tag to that provided by the ini file. This value is particularly important because it sets the writing direction, so you must use it if for some reason the default value is wrong.
language= \langle\text{language-name}\rangle

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

alph= \langle\text{counter-name}\rangle

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

Alph= \langle\text{counter-name}\rangle

Same for \texttt{\textbackslash Alph}.

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

onchar= \texttt{ids}\mid\texttt{fonts}

New 3.38 This option is much like an ‘event’ called when a character belonging to the script of this locale is found (as its name implies, it acts on characters, not on spaces). There are currently two ‘actions’, which can be used at the same time (separated by a space): with \texttt{ids} the \texttt{\textbackslash language} and the \texttt{\textbackslash localeid} are set to the values of this locale; with \texttt{fonts}, the fonts are changed to those of this locale (as set with \texttt{\textbackslash babelfont}). This option is not compatible with \texttt{mapfont}. Characters can be added or modified with \texttt{\textbackslash babelcharproperty}.

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

intraspace= \langle\texttt{base}\rangle\langle\texttt{shrink}\rangle\langle\texttt{stretch}\rangle

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

intrapenalty= \langle\text{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).

justification= \texttt{\textbackslash kashida}\mid\texttt{\textbackslash elongated}\mid\texttt{\textbackslash unhyphenated}

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 ARABIC TATWEEL character or in the ‘justification alternatives’ OpenType table (jalt). For an explanation see the babel site.

linebreaking= \langle\text{penalty}\rangle

Just a synonymous for justification.

mapfont= \langle\text{direction}\rangle

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

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

### 1.17 Digits and counters

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

For example:

```latex
\babelprovide[import]{telugu} % Telugu better with XeTeX
% Or also, if you want:
% \babelprovide[import, maparabic]{telugu}
\babelfont{rm}{Gautami}
\begin{document}
\telugudigits{1234}
\telugucounter{section}
\end{document}
```

Languages providing native digits in all or some variants are:

<table>
<thead>
<tr>
<th>Arabic</th>
<th>Persian</th>
<th>Lao</th>
<th>Odia</th>
<th>Urdu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assamese</td>
<td>Gujarati</td>
<td>Northern Luri</td>
<td>Punjabi</td>
<td>Uzbek</td>
</tr>
<tr>
<td>Bangla</td>
<td>Hindi</td>
<td>Malayalam</td>
<td>Pashto</td>
<td>Vai</td>
</tr>
<tr>
<td>Tibetan</td>
<td>Khmer</td>
<td>Marathi</td>
<td>Tamil</td>
<td>Cantonese</td>
</tr>
<tr>
<td>Bodo</td>
<td>Kannada</td>
<td>Burmese</td>
<td>Telugu</td>
<td>Chinese</td>
</tr>
<tr>
<td>Central Kurdish</td>
<td>Konkani</td>
<td>Mazanderani</td>
<td>Thai</td>
<td></td>
</tr>
<tr>
<td>Dzongkha</td>
<td>Kashmiri</td>
<td>Nepali</td>
<td>Uyghur</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 \b babelprovid e, as an argument to the keys alph and Alph, which redefine what \alph and \Alph print. For example:

\b 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, yemsá
Arabic   abjad, maghrebi.abjad
Belarusian, Bulgarian, Macedonian, Serbian   lower, upper
Bengali   alphabetic
Coptic   epact,lower.letters
Hebrew   letters (neither gersh nor gershayim yet)
Hindi   alphabetic
Armenian lower.letter, upper.letter
Japanese   hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana,
            informal, formal, cjk-earthly-branch, cjk-heavenly-stem,
            fullwidth.lower.alpha, fullwidth.upper.alpha
Georgian   letters
Greek   lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)
Khmer   consonant
Korean   consonant, syllabe, hanja.informal, hanja.formal, hangul.formal,
           cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha,
           fullwidth.upper.alpha
Marathi   alphabetic
Persian   abjad, alphabetic
Russian   lower, lower.full, upper, upper.full
Syriac   letters
Tamil   ancient
Thai   alphabetic
Ukrainian   lower, lower.full, upper, upper.full
Chinese   cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha,
           fullwidth.upper.alpha

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

1.18 Dates

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

\localedate [{\{calendar=..\}, variant=..}]\{\{year\}\{month\}\{day\}

By default the calendar is the Gregorian, but a ini files may define strings for other
calendars (currently ar, ar-*, he, fa, hi.) In the latter case, the three arguments are
the year, the month, and the day in those in the corresponding calendar. They are not the
Gregorian data to be converted (which means, say, 13 is a valid month number with
calendar=hebrew).
Even with a certain calendar there may be variants. In Kurmanji the default variant prints
something like 30. Çîleya Pêşîn 2019, but with variant=izafa it prints 31‘ê Çîleya Pêşînê
2019.
1.19 Accessing language info

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

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

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

`\localeinfo`  
{⟨field⟩}  
*New 3.38*  
If an ini file has been loaded for the current language, you may access the information stored in it. This 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.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).
- `script.name`, as provided by the Unicode CLDR.
- `script.tag.bcp47` is the BCP 47 tag of the script used by this locale.
- `script.tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).

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

`\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 קרו.  
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.
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 stored 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.

\begin{itemize}
  \item `\babelhyphen{soft}` and `\babelhyphen{hard}` are self-explanatory.
  \item `\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.
  \item `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it (even if a space follows).
  \item `\babelhyphen{empty}` inserts a break opportunity without a hyphen at all.
  \item `\babelhyphen{⟨text⟩}` is a hard “hyphen” using ⟨text⟩ instead. A typical case is `\babelhyphen{⟨/⟩}`.
\end{itemize}

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

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

There are also some differences with \TeX: (1) the character used is that set for the current font, while in \TeX it is hardwired to ⟨⟩ (a typical value); (2) the hyphen to be used in fonts with a negative \hyphenchar is ⟨⟩, like in \TeX, but it can be changed to another value by redefining `\bblnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a
\texttt{\verb|\babelhyphenation|} \[\langle \text{language}, \langle \text{language}, \ldots \rangle \rangle \{ \langle \text{exceptions} \rangle \} \]

**New 3.9a** Sets hyphenation exceptions for the languages given or, without the optional argument, for all languages (e.g., proper nouns or common loan words, and of course monolingual documents). Language exceptions take precedence over global ones. It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of \texttt{\lccodes}'s done in \texttt{\extras\langle lang \rangle} as well as the language-specific encoding (not set in the preamble by default). Multiple \texttt{\babelhyphenation}'s are allowed. For example:

\begin{verbatim}
\babelhyphenation{Wal-hal-la Dar-bhan-ga}
\end{verbatim}

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

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

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

\begin{verbatim}
\begin{hyphenrules}
\langle \text{language} \rangle \{ \langle \text{patterns} \rangle \}
\end{hyphenrules}
\end{verbatim}

The environment \texttt{\hyphenrules} can be used to select only the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select ‘nohyphenation’, provided that in \texttt{\language.dat} the ‘language’ nohyphenation is defined by loading \texttt{zerohyph.tex}. It deactivates language shorthands, too (but not user shorthands). Except for these simple uses, \texttt{\hyphenrules} is deprecated and other \texttt{\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 (e.g., Italian, French, Ukrainian).

\begin{verbatim}
\belpatterns \[\langle \text{language}, \langle \text{language}, \ldots \rangle \rangle \{ \langle \text{patterns} \rangle \} \]
\end{verbatim}

**New 3.9m** In \texttt{lualatex only},\footnote[14]{With \texttt{lualatex} exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and \texttt{babel} only provides the most basic tools.} adds or replaces patterns for the languages given or, without the optional argument, for all languages. If a pattern for a certain combination already exists, it gets replaced by the new one. It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \texttt{\lccodes}'s done in \texttt{\extras\langle lang \rangle} as well as the language-specific encoding (not set in the preamble by default). Multiple \texttt{\belpatterns}'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 \texttt{lualatex}.) With \texttt{\belprovide} and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules \footnote[37]{\texttt{New 3.32} it is disabled in verbatim mode, or more precisely when the}
Hyphen rules 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 Unicoide 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.\footnote{They are similar in concept, but not the same, as those in Unicode.}

It currently embraces \babelprehyphenation and \babelposthyphenation.

New 3.57 Several ini files predefines 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, LJ, Lj, NJ, Nj, nj. It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.</td>
</tr>
<tr>
<td>Czech, Polish,</td>
<td>hyphen.repeat</td>
<td>Explicit hyphens behave like \babelhyphen {repeat}.</td>
</tr>
<tr>
<td>Portuguese,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slovak</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Czech, Polish,</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>Slovak</td>
<td></td>
<td></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>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, dz, ggy, lly, nny, ssz, tty and zzs as cs-cs, dz-dz, etc.</td>
</tr>
</tbody>
</table>
Indic scripts  danda.nobreak  Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.

Arabic, Persian  kashida.plain  Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.

Serbian  transliteration.gajica  (Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.

\babelposthyphenation

New 3.37-3.39  With lualatex 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):

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

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

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

\babelprehyphenation

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

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

\textbf{EXAMPLE}  You can replace a character (or series of them) by another character (or series of them). Thus, to enter ż as zh and ś as sh in a newly created locale for transliterated Russian:
EXEMPLARY The following rule prevent the word “a” from being at the end of a line:

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

NOTE With \texttt{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{\textbackslash babelfont}. The \texttt{transforms} mechanism supplements rather than replaces OTF features.

With \texttt{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

\textbf{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 auto-loading of locales as described in this section. In these contexts auto-loading is particularly important because we may not know 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: \texttt{fr-Latn-FR \rightarrow fr-Latn \rightarrow fr-FR \rightarrow fr}. Languages with the same resolved name are considered the same. Case is normalized before, so that \texttt{fr-Latn-fr \rightarrow fr-Latn-FR}. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

\documentclass{article}
\usepackage[danish]{babel}
\babeladjjust{
  autoload.bcp47 = on,
  autoload.bcp47.options = import
}
\begin{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.

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

\begin{verbatim}
\babeladjust{ bcp47.toname = on }
\end{verbatim}

(You can deactivate it with off.) So, if dutch is one of the package (or class) options, you can write \selectlanguage{n1}. 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.

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

Some languages sharing the same script define macros to switch it (eg, \textcyrillic), but be aware they may also set the language to a certain default. Even the babel core defined \textlatin, but is was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.\footnote{But still defined for backwards compatibility.}

\begin{verbatim}
\ensureasciic{(text)}
\end{verbatim}

\textbf{New 3.9i} This macro makes sure \textit{(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 T53 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%).

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

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

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

\begin{verbatim}
\babeladjust{bidi.mirroring=off}
\end{verbatim}

There are some package options controlling bidi writing.

- \texttt{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.

42
**EXAMPLE** The following text comes from the Arabic Wikipedia (article about Arabia). Copy-pasting some text from the Wikipedia is a good way to test this feature. Remember basic is available in luatex only.

\documentclass{article}
\usepackage[bidi=basic]{babel}
\ babelprovide[import, main]{arabic}
\babelfont{rm}{FreeSerif}
\begin{document}

ـب(ﻲﻘﻳﺮﻏﻻا)ﻲﻨﻴﻠﻴﻬﻟاﺮﺼﻌﻟاﺔﻠﻴﻃبﺮﻌﻟاةﺮﻳﺰﺟﻪﺒﺷﺖﻓﺮﻋﺪﻗو

Arabia

وأ

Aravia )

ﺔﻴﻘﻳﺮﻏﻻﺎﺑ

Αραβία (ثﻼثنﺎﻣوﺮﻟامﺪﺨﺘﺳا،

”ـبتﺎﺋدﺎﺑ

Arabia”

ﺎﻬﻧأﻻإ،ﺔﻴﺑﺮﻌﻟاةﺮﻳﺰﺠﻟاﻪﺒﺷﻦﻣﻖﻃﺎﻨﻣثﻼﺛﻰﻠﻋ

مﻮﻴﻟاﻪﻴﻠﻋفﺮﻌﺗﺎﻤﻣﺮﺒﻛأﺖﻧﺎﻛًﺔﻘﻴﻘﺣ

\end{document}

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

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

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

\end{document}

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

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

\newcommand\refrange[2]{\babelsublr{\text{the}(\ref{#1})-\text{the}(\ref{#2})}}

In the future a more complete method, reading recursively boxed text, may be added.
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=sectioning.counters.contents.sectioning). This list will be expanded in future releases. Note not all options are required by all engines.

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

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

With counters, \arabic{} is not only considered \texttt{L} text always (with \babelsublr, see below), but also an “isolated” block which does not interact with the surrounding chars. So, while 1.2 in \texttt{R} text is rendered in that order with bidi=basic (as a decimal number), in \arabic{c1}\.\arabic{c2} the visual order is \texttt{c2}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 xetex and pdftex, but only in bidirectional (with both \texttt{R} and \texttt{L} paragraphs) documents in luatex.

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

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

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

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

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

tabular required in luatex for \texttt{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). \footnote{New 3.18}.

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

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

\textbf{EXAMPLE} Typically, in an Arabic document you would need:
\babelsublr \{\langle lr-text \rangle\}

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 \{\langle lr-text \rangle\} 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 \text{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:

\begin{verbatim}
RTL A ltr text \thechapter{} and still ltr RTL B
\end{verbatim}

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:

\begin{verbatim}
RTL A \foreignlanguage{english}\{ltr text \thechapter{} and still ltr\} RTL B
\end{verbatim}

\BabelPatchSection \{\langle section-name \rangle\}

Mainly for bidi text, but it can be useful in other cases. \BabelPatchSection and the corresponding option layout=sectioning takes a more logical approach (at least in many cases) because it applies the global language to the section format (including the \chaptername in \chapter), while the section text is still the current language. The latter is passed to tocs and marks, too, and with sectioning in layout they both reset the “global” language to the main one, while the text uses the “local” language. With layout=sectioning all the standard sectioning commands are redefined (it also “isolates” the page number in heads, for a proper bidi behavior), but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

\BabelFootnote \{\langle cmd \rangle\}\{\langle local-language \rangle\}\{\langle before \rangle\}\{\langle after \rangle\}

New 3.17 Something like:

\begin{verbatim}
\BabelFootnote{\parsfootnote}{\languagename}{()}{()}
\end{verbatim}

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

\begin{verbatim}
\footnote{\langle\foreignlanguage{\languagename}{note}\rangle}
\end{verbatim}

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:

\begin{verbatim}
\BabelFootnote{\footnote}{\languagename}{()}{()}
\BabelFootnote{\localfootnote}{\languagename}{()}{()}
\BabelFootnote{\mainfootnote}{()}{()}
\end{verbatim}
(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 uses \magyarOptions; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, \ProsodicMarksOn in latin).

### 1.26 Hooks

\AddBabelHook

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

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

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

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

- **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}{\% \protect\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.

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

1.27  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.

Afrikaans  afrikaans
Azerbaijani  azerbaijani
Basque  basque
Breton  breton
Bulgarian  bulgarian
Catalan  catalan
Croatian  croatian
Czech  czech
Danish  danish
Dutch  dutch
English  english, USenglish, american, UKenglish, british, canadian, australian, newzealand
 Esperanto  esperanto
 Estonian  estonian
 Finnish  finnish
 French  french, francais, canadien, acadian
 Galician  galician
 German  austrian, german, germanb, ngerman, naustrian
 Greek  greek, polutonikogreek
 Hebrew  hebrew
 Icelandic  icelandic
 Indonesian  indonesian (bahasa, indon, bahasai)
 Interlingua  interlingua
 Irish Gaelic  irish
 Italian  italian
 Latin  latin
 Lower Sorbian  lowersorbian
 Malay  malay, melayu (bahasam)
 North Sami  samin
 Norwegian  norsk, nynorsk
 Polish  polish
 Portuguese  portuguese, brazilian (portuges, brazil)\textsuperscript{19}
 Romanian  romanian
 Russian  russian
 Scottish Gaelic  scottish
 Spanish  spanish
 Slovakian  slovak
 Slovenian  slovene
 Swedish  swedish
 Serbian  serbian
 Turkish  turkish
 Ukrainian  ukrainian
 Upper Sorbian  uppersorbian
 Welsh  welsh

There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchie, ethiop and friulan.
Most of them work out of the box, but some may require extra fonts, encoding files, a preprocessor or even a complete framework (like CJK or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension .dn:

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

Then you preprocess it with devnag (file), which creates (file). tex; you can then typeset the latter with \LaTeX.

\textsuperscript{19}The two last name comes from the times when they had to be shortened to 8 characters
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).

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

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

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

New 3.39 Another property is locale, which adds characters to the list used by onchar in \babelprovide, or, if the last argument is empty, removes them. The last argument is the locale name:

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

1.29 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 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 \texttt{\lowercase{\ref{foo}}} inside the argument of \chapter, or, if you will not use shorthands in labels, set the safe option to none or bib.

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

\texttt{\AtBeginDocument{\DeleteShortVerb{|}}}

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

\addto{\extrasfrench}{\inputencoding{latin1}}
\addto{\extraru}{\inputencoding{koi8-r}}

• For the hyphenation to work correctly, lccodes cannot change, because \TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.\footnote{This explains why \TeX assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, \texttt{savinghyphcodes} is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.} So, if you write a chunk of French text with \texttt{\selectlanguage{fr}}, the apostrophes might not be taken into account. This is a limitation of \TeX, not of \texttt{babel}. Alternatively, you may use \texttt{\useshorthand} to activate ‘ and \texttt{\defineshorthand}, or redefine \texttt{\textquotesingle} (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{\normalsfcode{}} and (non-)French spacing is not always properly (un)set by languages. However, problems are unlikely to happen and therefore this part remains untouched in version 3.9 (but it is in the ‘to do’ list).

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

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

\begin{itemize}
  \item \texttt{csquotes} Logical markup for quotes.
  \item \texttt{iflang} Tests correctly the current language.
  \item \texttt{hyphsubst} Selects a different set of patterns for a language.
  \item \texttt{translator} An open platform for packages that need to be localized.
  \item \texttt{siunitx} Typesetting of numbers and physical quantities.
  \item \texttt{biblatex} Programmable bibliographies and citations.
  \item \texttt{bicaption} Bilingual captions.
  \item \texttt{babelbib} Multilingual bibliographies.
  \item \texttt{microtype} Adjusts the typesetting according to some languages (kerning and spacing).
\end{itemize}

\begin{itemize}
  \item Ligatures can be disabled.
  \item \texttt{substitutefont} Combines fonts in several encodings.
  \item \texttt{mkpattern} Generates hyphenation patterns.
  \item \texttt{tracklang} Tracks which languages have been requested.
  \item \texttt{ucharclasses} (xetex) Switches fonts when you switch from one Unicode block to another.
  \item \texttt{zhspacing} Spacing for CJK documents in xetex.
\end{itemize}

\section{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.\footnote{See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to \TeX because their aim is just to display information and not fine typesetting.} But that is the easy part, because they don’t require modifying the \TeX internals.

Calendars (Arabic, Persian, Indic, etc.) are under study.
Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ból”, but “from (3)” is “(3)-ból”, in Spanish an item labelled “3.” may be referred to as either “item 3.” or “3rd 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 an deprecated functions, see the wiki.

Options for locales loaded on the fly

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

Labels

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

2 Loading languages with language.dat

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

New 3.9q With luatex, however, patterns are loaded on the fly when requested by the language (except the “0th” language, typically english, which is preloaded always).\footnote{This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.} Until 3.9n, this task was delegated to the package luatex-hyphen, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named language.dat.lua, but now a new mechanism has been devised based solely on language.dat. You must 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).\footnote{The loader for luatex is slightly different as it’s not based on babel but on etex. 5rc. Until 3.9p it just didn’t work, but thanks to the new code it works by reloading the data in the babel way, i.e., with language.dat.}

2.1 Format

In that file the person who maintains a \TeX{} environment has to record for which languages he has hyphenation patterns and in which files these are stored\footnote{This is because different operating systems sometimes use very different file-naming conventions.}. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file after the file with the hyphenation patterns.

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

\hspace{1cm}

\hspace{1cm}
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:

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

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

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

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

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

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are \⟨lang⟩\hyphenmins, \Captions⟨lang⟩, \date⟨lang⟩, \extras⟨lang⟩ and \noextras ⟨lang⟩ (the last two may be left empty); where ⟨lang⟩ is either the name of the language definition file or the name of the \LaTeX option that is to be used. These macros and their functions are

---

25 This is not a new feature, but in former versions it didn't work correctly.
discussed below. You must define all or none for a language (or a dialect); defining, say, \date{lang} but not \captions{lang} does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define \l@{lang} to be a dialect of \language when \l@{lang} is undefined.

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

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

Some recommendations:

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

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

- Avoid adding things to \noextras{lang} except for umlauthigh and friends, \bbl@deactivate, \bbl@frenchspacing, and language-specific macros. Use always, if possible, \bbl@save and \bbl@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{But not removed, for backward compatibility.}

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

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

### 3.1 Guidelines for contributed languages

Currently, the easiest way to contribute a new language is by taking one the the 500 or so ini templates available on GitHub as a basis. Just make a pull request to download it and then, after filling the fields, send 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.

\footnote{But not removed, for backward compatibility.}
• 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.

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

\renewcommand\spanishhyphenmins(34)

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

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

\captions<lang>
The macro \captions<lang> defines the macros that hold the texts to replace the original hard-wired texts.

\date<lang>
The macro \date<lang> defines \today.

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

\noextras<lang>
Because we want to let the user switch between languages, but we do not know what state TeX might be in after the execution of \extras<lang>, a macro that brings TeX into a predefined state is needed. It will be no surprise that the name of this macro is \noextras<lang>.
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.

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.

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

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.

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.

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.

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

(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct \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 ldf file, with a language, a dialect and an attribute. Strings are best defined using the method explained in sec. 3.8 (babel 3.9 and later).

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

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

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

\StartBabelCommands*{<language>}{captions}
\SetString\chaptername{<chapter name>}
% More strings
\StartBabelCommands*{<language>}{date}
3.4 Support for active characters

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

\initiate@active@char

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

\bbl@activate
\bbl@deactivate

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

\declare@shorthand

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

\bbl@add@special
\bbl@remove@special

The \TeX{}book states: “Plain \TeX{} includes a macro called \dospecials that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]

It is used to set text ‘verbatim’. To make this work if more characters get a special category code, you have to add this character to the macro \dospecials. \TeX{} adds another macro called \@sanitize representing the same character set, but without the curly braces. The macros \bbl@add@special{char} and \bbl@remove@special{char} add and remove the character \texttt{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\footnote{This mechanism was introduced by Bernd Raichle.}.

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

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

\texttt{\texttt{\addto}} The macro \texttt{\addto\langle control sequence\rangle\{\langle \TeX code \rangle\}} can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or \texttt{\relax}). This macro can, for instance, be used in adding instructions to a macro like \texttt{\textbackslash extrasenglish}. Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using etoolbox, by Philipp Lehman, consider using the tools provided by this package instead of \texttt{\addto}.

3.7 Macros common to a number of languages

\texttt{\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 \texttt{\bbl@allowhyphens} can be used.

\texttt{\allowhyphens} Same as \texttt{\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 \texttt{\accent} in OT1. Note the previous command (\texttt{\bbl@allowhyphens}) has different applications (hyphens and discretionary) than this one (composite chars). Note also prior to version 3.7, \texttt{\allowhyphens} had the behavior of \texttt{\bbl@allowhyphens}.

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

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

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

3.8 Encoding-dependent strings

\texttt{New 3.9a} Babel 3.9 provides a way of defining strings in several encodings, intended mainly for luatex and xetex. This is the only new feature requiring changes in language files if you want to make use of it. Furthermore, it must be activated explicitly, with the package option strings. If there is no strings, these blocks are ignored, except \texttt{\SetCase} (and except if forced as described...
below). In other words, the old way of defining/switching strings still works and it's used by default. It consist is a series of blocks started with \StartBabelCommands. The last block is closed with \EndBabelCommands. Each block is a single group (ie, local declarations apply until the next \StartBabelCommands or \EndBabelCommands). An ldf may contain several series of this kind. Thanks to this new feature, string values and string language switching are not mixed any more. No need of \addto. If the language is french, just redefine \frenchchaptername.

\StartBabelCommands{\langle language-list\rangle}{\langle category\rangle}{\langle selector\rangle}

The \langle language-list\rangle 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 \langle category\rangle is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.\footnote{\ref{future_categories}} 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}

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

\EndBabelCommands

A real example is:

\footnote{\ref{future_categories} In future releases further categories may be added.}
When used in ldf files, previous values of \langle category\rangle\langle language\rangle are overridden, which means the old way to define strings still works and used by default (to be precise, is first set to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if \date\langle language\rangle exists).

\Star{\langle language-list\rangle\langle category\rangle\langle selector\rangle}

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 \UseStrings which has been removed because it did not work.}

\End

\After{\langle code\rangle}

The code is delayed and executed at the global scope just after \EndBabelCommands.
\SetString \{\macro-name\}\{\string\}

Adds \macro-name to the current category, and defines globally \lang-macro-name to \code (after applying the transformation corresponding to the current charset or defined with the hook stringprocess).

Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

\SetStringLoop \{\macro-name\}\{\string-list\}

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

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

#1 is replaced by the roman numeral.

\SetCase \{\map-list\}\{\toupper-code\}\{\tolower-code\}

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 @uc1clist (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 T\textsuperscript{1} is the default case mapping in \LaTeX, we can set for Turkish:

\begin{verbatim}
\StartBabelCommands{turkish}{}[ot1enc, fontenc=OT1]
    \SetCase
        \uccode"10=`I\relax
        \lccode`I="10\relax

\StartBabelCommands{turkish}{}[unicode, fontenc=TUEU1 EU2, charset=utf8]
    \SetCase
        \uccode`i=`İ\relax
        \uccode`ı=`I\relax
        \lccode`İ=`i\relax
        \lccode`I=`ı\relax

\StartBabelCommands{turkish}{}
    \SetCase
        \uccode`i="9D\relax
        \uccode"19=`I\relax
        \lccode"9D=`i\relax
        \lccode"19=`I\relax

\EndBabelCommands
\end{verbatim}

(Note the mapping for OT\textsuperscript{1} 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 otherlanguage* 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 (e.g., 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 encondings.
- **[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 (e.g., 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 (e.g., babel.name.A, babel.name.B) or a name (e.g., 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 `.ltx` files. They may change in the future. This applies mainly to those recently added for replacing, trimming and looping. The older ones, like \bbl@afterfi, will not change.

We define some basic macros which just make the code cleaner. \bbl@add is now used internally instead of \addto because of the unpredictable behavior of the latter. Used in babel.def and in babel.sty, which means in \TeX{} is executed twice, but we need them when defining options and babel.def cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

\begin{verbatim}
\bbl@addlist \This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated. It presumes expandable character strings.
\begin{verbatim}
\def\bbl@addlist#1#2{\edef#1{\bbl@ifunset{\bbl@stripslash#1}{}{\ifx#1\@empty\else#1,\fi}#2}}
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\bbl@afterelse \bbl@afterfi \This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.
\begin{verbatim}
\def\bbl@afterelse#1\else#2\fi{\fi#1}
\def\bbl@afterfi#1\fi{\fi#1}
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\bbl@exp \This is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.
\begin{verbatim}
\def\bbl@exp#1{\begingroup\let\\noexpand\bbl@exp@en\let\[\bbl@exp@ue\edef\bbl@exp@aux{\endgroup#1}}
\end{verbatim}
\end{verbatim}
The following piece of code is stolen (with some changes) from \keyval, by David Carlisle. It defines two macros: \bbl@trim and \bbl@trim@def. The first one strips the leading and trailing spaces from the second argument and then applies the first argument (a macro, \toks@ and the like). The second one, as its name suggests, defines the first argument as the stripped second argument.

\begin{verbatim}
\bbl@tempa{ }
\end{verbatim}

\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.

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

\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,

\begin{verbatim}
\end{verbatim}

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

\bbl@replace Returns implicitly \toks@ with the modified string.

An extension to the previous macro. It takes into account the parameters, and it is string based (i.e., if you replace \texttt{relax} by \texttt{ho}, then \texttt{relax} becomes \texttt{rho}). No checking is done at all, because it is not a general-purpose macro, and it is used by \texttt{babel} only when it works (an example where it does not work is in \texttt{bbl@TG@date}, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \texttt{bbl@replace}; I'm not sure checking the replacement is really necessary or just paranoia).
\makeatletter % "internal" macros with @ are assumed
\scantokens{%
\bbl@tempa\@namedef{\bbl@stripslash#1}\bbl@temp\bbl@tempe}%
\catcode64=\the\catcode64\relax% Restore @
\else\let\bbl@tempc\@empty % Not \relax\fi
\bbl@exp{% For the 'uplevel' assignments
\endgroup
\bbl@tempc}% empty or expand to set #1 with changes
\fi

Two further tools. \bbl@samestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bbl@engine takes the following values: 0 is pdf\TeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.
\def\bbl@ifsamestring#1#2{%
\begingroup
\protected@edef\bbl@tempb{#1}\
\edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}\
\protected@edef\bbl@tempc{#2}\
\edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}\
\ifx\bbl@tempb\bbl@tempc
\aftergroup\@firstoftwo\
\else\aftergroup\@secondoftwo\fi
\endgroup}
\chardef\bbl@engine=%
\ifx\directlua\@undefined\ifx\XeTeXinputencoding\@undefined\z@
\else\tw@\fi\else\@ne\fi

An somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.
\def\bbl@bsphack{%
\ifhmode\hskip\z@skip
\def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}\
\else\let\bbl@esphack\@empty
\fi}

Another hackish tool, to apply case changes inside a protected macros. It’s based on the internal \let’s made by \MakeUppercas and \MakeLowercas between things like \oe and \OE.
\def\bbl@cased{%
\ifx\oe\OE
\expandafter\in\expandafter{
\{\expandafter\OE\expandafter}\expandafter\}\expandafter\oe\expandafter}
\ifin@\bbl@afterelse\expandafter\MakeUppercas
\else\bbl@afterfi\expandafter\MakeLowercas
\fi
\else
\fi}
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).

\def\bbl@extras@wrap#1#2#3{% in-test, 2:before, 3:after
\toks@\expandafter\expandafter\expandafter{\csname extras\languagename\endcsname}\
\bbl@exp{\in@{#1}{\the	oks@}}\
\ifin@\else
\@temptokena{#2}\
\edef\bbl@tempc{\the\@temptokena\the	oks@}\
\toks@\expandafter{\bbl@tempc#3}\
\expandafter\edef\csname extras\languagename\endcsname{\the	oks@}\
\fi}
\langle⟨/Basic macros⟩⟩

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

\def\bbl@extras@wrap#1#2#3{% in-test, 2:before, 3:after
\toks@\expandafter\expandafter\expandafter{\csname extras\languagename\endcsname}\
\bbl@exp{\in@{#1}{\the	oks@}}\
\ifin@\else
\@temptokena{#2}\
\edef\bbl@tempc{\the\@temptokena\the	oks@}\
\toks@\expandafter{\bbl@tempc#3}\
\expandafter\edef\csname extras\languagename\endcsname{\the	oks@}\
\fi}
\langle⟨/Basic macros⟩⟩

\langle⟨∗ Define core switching macros ∗⟩⟩ ≡
\ifx\language\@undefined
\csname newcount\endcsname\language
\fi
\langle⟨/Define core switching macros⟩⟩

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.

\langle⟨∗ Define core switching macros ∗⟩⟩ ≡
\countdef\last@language=19
\edef\addlanguage{\csname newlanguage\endcsname}\language
\langle⟨/Define core switching macros⟩⟩

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

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

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.

\if\bbl@languages@undefined\else
\begingroup
\catcode`\^^I=12
\@ifpackagewith{babel}{showlanguages}{% 
\begingroup
\def\bbl@elt#1#2#3#4{%wlog{#2\^^I#1\^^I#3\^^I#4}%
\wlog{<*languages>}%
\bbl@languages
\wlog{</languages>}%
\endgroup}
\endgroup
\def\bbl@elt#1#2#3#4{% 
\ifnum#2=\z@ 
\gdef\bbl@nulllanguage{#1} 
\def\bbl@elt##1##2##3##4{}% 
\fi}
\bbl@languages
\fi%
\endinput
\fi%
\fi%

7.3 base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets ver@babel.sty so that \H{}X 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.

\@iftrace{Defining option 'base'}
\@ifpackagewith{babel}{base}{% 
\let\bbl@onlyswitch\@empty
\let\bbl@provide@locale\relax
\input babel.def
\let\bbl@onlyswitch@undefined
\if\directlua\@undefined
\DeclareOption*{\bbl@patterns{\CurrentOption}}%
\else
\input luababel.def
\DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
\fi
\fi

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 \lin@, loop them with \@for or load keyval, for example.

295 \bbl@trace{key=value and another general options}
296 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
297 \edef\bbl@tempb#1.#2{% Remove trailing dot
298   #1if\@empty#2\else,\bbl@after\bbl@tempb2\fi%
299   \edef\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
300      \if\@empty#2%
301        \edef\bbl@tempc{%
302          \if\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
303      \else
304        \edef\bbl@tempc{%
305          \if\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
306        \in@{,provide=}{,#1}%
307        \if\in@
308          \edef\bbl@tempc{%
309            \if\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
310          \else
311          \edef\bbl@tempc{%
312            \if\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
313          \in@{=}{#1}%
314          \if\in@
315            \edef\bbl@tempc{%
316              \if\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
317            \else
318              \edef\bbl@tempc{%
319                \if\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
320              \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
321            \fi
322        \fi
323      \fi
324    \let\bbl@tempc\@empty
325    \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
326    \expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc
The next option tells babel to leave shorthand characters active at the end of processing the package. This is not 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.

327 \DeclareOption{KeepShorthandsActive}{}
328 \DeclareOption{activeacute}{}
329 \DeclareOption{activegrave}{}
330 \DeclareOption{debug}{}
331 \DeclareOption{nocconfigs}{}
332 \DeclareOption{showlanguages}{}
333 \DeclareOption{silent}{}
334 \% \DeclareOption{mono}{}
335 \ DeclareOption{shorthands=off}{\bbl@tempa shorthands=\\bbl@tempa}
336 \chardef\bbl@iniflag\z@
337 \ DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
338 \ DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
339 \ DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
340 \% A separate option
341 \DeclareOption{auto-load@options}@empty
342 \ DeclareOption{provide=!*}{\def\bbl@auto-load@options{import}}
343 \% Don’t use. Experimental. TODO.
344 \newif\ifbbl@single
345 \ DeclareOption{selectors=off}{\bbl@singletrue}
346 \% (More package options)
Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax <key>=<value>, the second one loads the requested languages, except the main one if set with the key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

347 \let\bbl@opt@shorthands\@nil
348 \let\bbl@opt@config\@nil
349 \let\bbl@opt@main\@nil
70
The following tool is defined temporarily to store the values of options.

\def\bbl@tempa#1=#2\bbl@tempa{%
  \bbl@csarg\ifx{opt@#1}\@nnil
  \bbl@csarg\edef{opt@#1}{#2}\
  \else\bbl@error
  {Bad option \texttt{\textquotesingle#1\textquotesingle=\textquotesingle#2\textquotesingle}. Either you have misspelled the\%
    key or there is a previous setting of \texttt{\textquotesingle#1\textquotesingle}. Valid\%
    keys are, among others, \texttt{\textquotesingleshorthands\textquotesingle}, \texttt{\textquotesinglemain\textquotesingle}, \texttt{\textquotesinglebidi\textquotesingle},\%
    \texttt{\textquotesinglestrings\textquotesingle}, \texttt{\textquotesingleconfig\textquotesingle}, \texttt{\textquotesingleheadfoot\textquotesingle}, \texttt{\textquotesinglesafe\textquotesingle}, \texttt{\textquotesinglemath\textquotesingle}.}%
  \{See the manual for further details.\}
  \\fi};
\}

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

\let\bbl@language@opts\@empty
\DeclareOption*{%
  \bbl@xin\{\string=\}{\CurrentOption}\
  \ifin@
  \expandafter\bbl@tempa\CurrentOption\bbl@tempa\
  \else\bbl@add@list\bbl@language@opts{\CurrentOption}\
  \fi};
\}

Now we finish the first pass (and start over).

\ProcessOptions*
\let\bbl@opt@provide\@nnil
\let\bbl@opt@provide\@empty % %%% MOVE above
\else\chardef\bbl@iniflag\@ne
\bbl@exp\{\bl@forkv{\@nameuse{@raw@opt@babel.sty}}\}{%
  \in\{\string=\},\CurrentOption\}{\CurrentOption}\
  \\fi};
\}

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.

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

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

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 – \bbl@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.
\expandafter\@firstoftwo
\else
\expandafter\@secondoftwo
\fi}
\fi
⟨/package⟩
(*core)

7.6 Interlude for Plain

Because of the way \texttt{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 \emph{Emulate \LaTeX}.

\ifx\ldf@quit\@undefined\else
\endinput\fi % Same line!
⟨⟨
MakesureProvidesFileisdefined
⟩⟩
\ProvidesFile{babel.def}[
⟨⟨
date
⟩⟩ ⟨⟨
version
⟩⟩
Babelcommondefinitions]
\ifx\AtBeginDocument\@undefined % TODO. change test.
⟨⟨
Emulate\LaTeX
⟩⟩
\fi
That is all for the moment. Now follows some common stuff, for both Plain and \LaTeX. After it, we will resume the \LaTeX-only stuff.

(*core)
(*package | core)

8 Multiple languages

This is not a separate file (\texttt{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.

\def\bbl@version{
⟨⟨
version
⟩⟩
}
\def\bbl@date{
⟨⟨
date
⟩⟩
}
⟨⟨
Define coreswitchingmacros
⟩⟩
\adddialect

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

\def\adddialect#1#2{%
\global\chardef#1#2\relax
\bbl@usehooks{adddialect}{#1}{#2}%
\begingroup
\count@#1\relax
\def\bbl@elt##1##2##3##4{%
\ifnum\count@=##2\relax
\edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
\bbl@info{Hyphen rules for '\\expandafter\@gobble\bbl@tempa' set to \string\language\the\count@. Reported}%
\def\bbl@elt####1####2####3####4{}%
\fi}%
\bbl@cs{languages}%
\endgroup}
\bbl@iflanguage
executes code only if the language \texttt{l@} exists. Otherwise raises an error.
The argument of \texttt{\bbl@iflanguage} has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s an attempt to fix a long-standing bug when \texttt{\foreignlanguage} and the like appear in a \texttt{\MakeXXXcase}. However, a lowercase form is not imposed to improve backward compatibility.
(perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note 1@ is encapsulated, so that its case does not change.

\def\bbl@fixname#1{%
  \begingroup
  \def\bbl@tempe{l@}%
  \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
  \bbl@tempd
  \{\lowercase\expandafter{\bbl@tempd}%
  \{\uppercase\expandafter{\bbl@tempd}%
  \@empty
  \{\edef\bbl@tempd{\def\noexpand#1{#1}}%
  \uppercase\expandafter{\bbl@tempd}}%
  \@empty
  \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
  \bbl@tempd
  \bbl@exp{\"\bbl@usehooks{languagename}{{\languagename}{#1}}}}%
\def\bbl@iflanguage#1{%
  \@ifundefined{l@#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 look up. 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
    \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \fi}
\def\bbl@bcpcase#1#2#3#4\@@#5{%
  \ifx\empty\empty#3%
    \bbl@bcpcase#2\empty\empty@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \else
    \bbl@bcpcase#2\empty\empty\@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \fi}
\def\bbl@bcpcase#1#2#3#4\@@#5{%
  \ifx\empty\empty#3%
    \bbl@bcpcase#2\empty\empty@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \else
    \bbl@bcpcase#2\empty\empty\@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \fi}
\def\bbl@bcpcase#1#2#3#4\@@#5{%
  \ifx\empty\empty#3%
    \bbl@bcpcase#2\empty\empty@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \else
    \bbl@bcpcase#2\empty\empty\@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \fi}
\def\bbl@bcpcase#1#2#3#4\@@#5{%
  \ifx\empty\empty#3%
    \bbl@bcpcase#2\empty\empty@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \else
    \bbl@bcpcase#2\empty\empty\@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \fi}
\def\bbl@bcpcase#1#2#3#4\@@#5{%
  \ifx\empty\empty#3%
    \bbl@bcpcase#2\empty\empty@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \else
    \bbl@bcpcase#2\empty\empty\@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \fi}
\def\bbl@bcpcase#1#2#3#4\@@#5{%
  \ifx\empty\empty#3%
    \bbl@bcpcase#2\empty\empty@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \else
    \bbl@bcpcase#2\empty\empty\@\@\bbl@tempb
    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini} %{\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
  \fi}
512 \{\edef\bbl@bcp\{\bbl@tempa-\bbl@tempc\}\}
513 \}%
514 \fi
515 \ifx\bbl@bcp\relax
516 \iffileexists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}\fi
517 \fi
518 \fi\fi\}
519 \let\bbl@initoload\relax
520 \edef\bbl@provide@locale{%
521 \ifx\babelprovide\@undefined
522 \bbl@error{For a language to be defined on the fly 'base'\
523 is not enough, and the whole package must be\%
524 loaded. Either delete the 'base' option or\%
525 request the languages explicitly}\%
526 {See the manual for further details.}%
527 \fi
528 % TODO. Option to search if loaded, with \LocaleForEach
529 \let\bbl@auxname\languagename % Still necessary. TODO
530 \ifbbl@bcpallowed
531 \expandafter\ifx\csname date\languagename\endcsname\relax
532 \bbl@bcplookup\languagename-@empty-@empty-@empty@@
533 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
534 \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
535 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
536 \expandafter\ifx\csname date\languagename\endcsname\relax
537 \let\bbl@initoload\bbl@bcp
538 \bbl@exp{\bbl@provide{\bbl@autoload@bcpoptions}{\languagename}}%
539 \let\bbl@initoload\relax
540 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
541 \fi
542 \fi
543 \expandafter\ifx\csname date\languagename\endcsname\relax
544 \let\bbl@initoload\relax
545 \fi
546 \fi
547 \fi
548 \expandafter\ifx\csname date\languagename\endcsname\relax
549 \iffileexists{babel-\languagename.tex}%
550 \bbl@exp{\bbl@provide{\bbl@autoload@options}{\languagename}}%
551 \}%
552 \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.

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

8.1 Selecting the language
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, arabic, 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 \TeXX’s aftergroup mechanism to help us. The command aftergroup stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence \bbl@pop@language 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:

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.

This macro stores its first element (which is delimited by the ‘+’-sign) in \languagename and stores the rest of the string in \bbl@language@stack.

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).
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).

The unprotected part of \selectlanguage.

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

We also write a command to change the current language in the auxiliary files. \bbl@savelastskip is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from hyperref, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in luatex, is to avoid the \write altogether when not needed).
\expandafter\ifx\csname date\languagename\endcsname\relax
  \edef\languagename{#1}\
  \let\localename\languagename
\else
  \bbl@info{Using `\string\language' instead of `language' is deprecated. If what you want is to use a macro containing the actual locale, make sure it does not not match any language. Reported}\
  \if\scantokens\undefined
    \def\localename{??}\
  \else
    \scantokens\expandafter{\def\expandafter\localename\expandafter{\languagename}}\
  \fi
\fi
\else
  \def\localename{#1}% This one has the correct catcodes
\fi
\select@language{\languagename}\
% write to auxs
\expandafter\ifx\csname date\languagename\endcsname\relax\else
  \if@filesw
    \if\babel@aux\@gobbletwo\else % Set if single in the first, redundant
      \bbl@savelastskip
      \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}\%
    \bbl@restorelastskip
  \fi
  \bbl@usehooks{write}{%}
  \fi
\fi
\let\bbl@savelastskip\relax
\let\bbl@restorelastskip\relax
%\newif\if\bbl@bcpallowed
\bbl@bcpallowedfalse
\def\select@language#1{% from set@, babel@aux
  % set hymap
  \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
  % set name
  \edef\languagename{#1}\
  \bbl@fixname\languagename
  % TODO. name@map must be here?
  \bbl@provide@locale
  \if\bbl@iflanguage\languagename{%\% from set@, babel@aux
    % set type
    \let\bbl@select@type\z@\%
    \expandafter\bbl@switch\expandafter{\languagename}\
  \fi
\else
  % set type
  \let\bbl@select@type\z@\%
  \expandafter\bbl@switch\expandafter{\languagename}\
\fi}
First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring \TeX in a certain pre-defined state.

The name of the language is stored in the control sequence \languagename. Then we have to redefine \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the \noextras⟨lang⟩ command at definition time by expanding the \csname primitive. Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of \selectlanguage, and calling these macros.

The switching of the values of \lefthyphenmin and \righthyphenmin is somewhat different. First we save their current values, then we check if \⟨lang⟩hyphenmins is defined. If it is not, we set default values (2 and 3), otherwise the values in \⟨lang⟩hyphenmins will be used.

% make sure there is info for the language if so requested
\bbl@ensureinfo[1]
% restore
\originalTeX
\expandafter\def\expandafter\originalTeX\expandafter{%
\csname noextras#1\endcsname
\let\originalTeX\@empty
\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
% switch extras
\bbl@usehooks{beforeextras}
\csname extras#1\endcsname
\bbl@usehooks{afterextras}
% > babel-ensure
% > babel-sh-<short>
% > babel-bidi
% > babel-fontspec
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.

The `\endotherlanguage` part of the environment tries to hide itself when it is called 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.

\expandafter\def\csname otherlanguage*\endcsname{%
  \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[\]]}}
\def\bbl@otherlanguage@s[#1]#2{%
  \ifnum\bbl@hymapsel=@ccliv\chardef\bbl@hymapsel=\relax\fi
  \def\bbl@select@opts{#1}%
  \foreignlanguage{#2}}

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

\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 \extrastext{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.

\providecommand\bbl@beforeforeign{}
\edef\foreignlanguage{%
  \noexpand\protect
  \expandafter\noexpand\csname foreignlanguage \endcsname}
\expandafter\def\csname foreignlanguage \endcsname{%
  \@ifstar\bbl@foreign@s\bbl@foreign@x}{%}

\begin{group}
\def\bbl@select@opts{#1}%
\let\BabelText=\firstofone
\bbl@beforeforeign
\foreignlanguage{#2}%
\bbl@usehooks{foreign}{}
\BabelText{#3} Now in horizontal mode!
\end{group}
\edef\bbl@foreign@x[#1]{% TODO - \shapemode, \@setpar, ?\@@par
\par}
\let\bbl@select@opts=\empty
\let\BabelText=\firstofone
\foreignlanguage{#1}%
\bbl@usehooks{foreign*}{}
\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.

\def\foreignlanguage#1{%
  % set name
  \edef\languagename{#1}%
  \ifbbl@usedategroup
    \bbl@add\bbl@select@opts{,date,}%
  \fi
  \bbl@fixname\languagename%
  \if\bbl@providelocale
    \bbl@iflanguage\languagename{%
      \expandafter\ifx\csname date\languagename\endcsname\relax
        \bbl@warning % TODO - why a warning, not an error?
          {Unknown language '#1'. Either you have\\
            misspelled its name, it has not been installed,\\
            or you requested it in a previous run. Fix its name,\\
            install it or just rerun the file, respectively. In\\
            some cases, you may need to remove the aux file.\\
            I'll proceed, but expect wrong results.\\%
            Reported}%
      \fi
      % set type
      \let\bbl@select@type@one
      \expandafter\bbl@switch\expandafter{\languagename}}}
\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 has been set, too). \bbl@hyphenation@ is set to relax until the very first \bbl@hyphenation, 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.

\let\bbl@hyphlist\@empty
\let\bbl@hyphenation\@relax
\let\bbl@pttnlist\@empty
\let\bbl@patterns\@relax
\let\bbl@hymapsel=\@cclv
\def\bbl@patterns#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
  \expandafter\@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
\@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
\begingroup
\edef\bbl@hyphenation@{}%
The environment \texttt{hyphenrules} can be used to select just the hyphenation rules. This environment does not change \texttt{language} and when the hyphenation rules specified were not loaded it has no effect. Note however, \texttt{lccode} and font encodings are not set at all, so in most cases you should use other language.*.

\begin{verbatim}
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\languageshorthands\{none\}\fi\expandafter\expandafter\expandafter\set@hyphenmins\csname\bbl@tempf hyphenmins\endcsname\relax\set@hyphenmins\tw@\thr@@\relax\else\expandafter\expandafter\expandafter\set@hyphenmins\csname\bbl@tempf hyphenmins\endcsname\relax\fi}}}
\end{verbatim}

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

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

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

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

\texttt{\ProvidesLanguage} The identification code for each file is something that was introduced in \LaTeX{}2ε. When the command \texttt{\ProvidesFile} does not exist, a dummy definition is provided temporarily. For use in the language definition file the command \texttt{\ProvidesLanguage} is defined by babel. Depending on the format, ie, on if the former is defined, we use a similar definition or not.

\begin{verbatim}\if\ProvidesFile\@undefined\def\ProvidesLanguage#1[#2 #3 #4]{\wlog{Language: #1 #4 #3 <#2>}}\else\@namedef{#1hyphenmins}{#2}\fi\end{verbatim}
\originalTeX  The macro \originalTeX should be known to \TeX at this moment. As it has to be expandable we \let it to \@empty instead of \relax.

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

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

\ifx\babel@beginsave\@undefined\let\babel@beginsave\relax\fi

A few macro names are reserved for future releases of \babel, which will use the concept of 'locale':

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.
Some functions for '#1' are tentative.\%
They might not work as expected and their behavior\%
could change in the future.\%
Reported}}
\def\nolanerr#1{%\bbl@error
{You haven't defined the language '#1' yet.\%
Perhaps you misspelled it or your installation\%
is not complete}%
{Your command will be ignored, type <return> to proceed}}
\def\nopatterns#1{%\bbl@warning
{No hyphenation patterns were preloaded for\%the language '#1' into the format.\%
Please, configure your TeX system to add them and\%
rebuild the format. Now I will use the patterns\%
preloaded for \bbl@nulllanguage\space instead}}
\let\bbl@usehooks\@gobbletwo
\ifx\bbl@onlyswitch\@empty\endinput\fi
% Here ended switch.def
Here ended the now discarded switch.def. Here also (currently) ends the base option.
\ifx\directlua\@undefined\else\ifx\bbl@luapatterns\@undefined
\input luababel.def\fi\fi\fi

\addto It takes two arguments, a \texttt{\langle control sequence \rangle} and \TeX-code to be added to the \texttt{\langle control sequence \rangle}.
If the \texttt{\langle control sequence \rangle} has not been defined before it is defined now. The control sequence could also expand to \texttt{\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.
\def\addtolist{\endgroup
The macro \texttt{\initiate@active@char} below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. TODO. Always used with additional expansions. Move them here? Move the macro to basic?

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

\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 \texttt{\LaTeX} macros completely in case their definitions change (they have changed in the past). A macro named \texttt{\macro} will be saved new control sequences named \texttt{\org@\macro}.

\begin{verbatim}
def \bbl@redefine#1{%
  \edef \bbl@tempa{\bbl@stripslash#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1%
  \expandafter\def\csname\bbl@tempa\endcsname}
\end{verbatim}

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

\begin{verbatim}
def \bbl@redefine@long#1{%
  \edef \bbl@tempa{\bbl@stripslash#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1%
  \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname
\end{verbatim}

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

\begin{verbatim}
def \bbl@redefinerobust#1{%
  \edef \bbl@tempa{\bbl@stripslash#1}%
  \bbl@ifunset{\bbl@tempa\space}{}%
  \def \bbl@tempa{\bbl@stripslash#1}%
  \expandafter\edef\csname \bbl@tempa\space\endcsname{\bbl@tempa\space}%
  \bbl@exp{\def\#1{\protect\bbl@tempa\space}}%
  \bbl@exp{\let\org@\bbl@tempa\bbl@tempa\space}%
  \namedef{\bbl@tempa\space}{}
\end{verbatim}

8.3 Hooks

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

\begin{verbatim}
def \bbl@trace{Hooks} newcommand\AddBabelHook[3][]{%
  \bbl@ifunset{\bbl@hk@#2}{\EnableBabelHook{#2}}%
  \def \bbl@tempa{\bbl@tempa\space}%
  \expandafter\edef\csname \bbl@tempa\space\endcsname{\bbl@tempa\space}%
  \bbl@exp{\def\#1{\protect\bbl@tempa\space}}%
  \bbl@exp{\let\org@\bbl@tempa\bbl@tempa\space}%
  \namedef{\bbl@tempa\space}{}
\end{verbatim}
To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for `hyphen.cfg` are also loaded (just in case you need them for some reason).

The user command just parses the optional argument and creates a new macro named `\bbl@e@[language]`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro `\bbl@e@[language]` contains `\bbl@ensure{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the `fontenc` is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the `include` list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

The user command just parses the optional argument and creates a new macro named `\bbl@e@[language]`. We register a hook at the `afterextras` event which just executes this macro in a “complete” selection (which, if undefined, is `\relax` and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro `\bbl@e@[language]` contains `\bbl@ensure{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}`, which in turn loops over the macros names in `\bbl@captionslist`, excluding (with the help of `\in@`) those in the exclude list. If the `fontenc` is given (and not `\relax`), the `\fontencoding` is also added. Then we loop over the include list, but if the macro already contains `\foreignlanguage`, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.
8.4 Setting up language files

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

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

Another character that needs to have the correct category code during processing of language definition files is the equals sign, `=`, because it is sometimes used in constructions with the `\let` primitive. Therefore we store its current catcode and restore it later on.
Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to \LdfInit is a control sequence. We do that by looking at the first token after passing #2 through string. When it is equal to \@backslashchar we are dealing with a control sequence which we can compare with \@undefined.

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

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

Finally we check \originalTeX.

\def\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`@=11\relax
\chardef\eqcatcode=\catcode`\=
\catcode`\==12\relax
\expandafter\if\expandafter\@backslashchar
\expandafter\@car\string#2\@nil
\ifx#2\@undefined
\ldf@quit{#1}\
\else
\expandafter\ifx\csname#2\endcsname\relax
\ldf@quit{#1}\
\fi
\fi
\else
\expandafter\ifx\csname#2\endcsname\relax\ldf@quit{#1}\
\fi
\fi
\bbl@ldfinit}
\ldf@quit This macro interrupts the processing of a language definition file.
\def\ldf@quit#1{%
\expandafter\main@language\expandafter{#1}\
\catcode`@=\atcatcode \let\atcatcode\relax
\catcode`\==\eqcatcode \let\eqcatcode\relax
\endinput}
\ldf@finish This macro takes one argument. It is the name of the language that was defined in the language definition file.

We load the local configuration file if one is present, we set the main language (taking into account that the argument might be a control sequence that needs to be expanded) and reset the category code of the @-sign.
\def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
\bbl@afterlang
\let\bbl@afterlang\relax
\let\BabelModifiers\relax
\let\bbl@screset\relax
\def\ldf@finish#1{%
\loadlocalcfg{#1}\
\bbl@afterldf{#1}\
\expandafter\main@language\expandafter{#1}\
\catcode`@=\atcatcode \let\atcatcode\relax
\catcode`\==\eqcatcode \let\eqcatcode\relax
89
After the preamble of the document the commands \LdfInit, \ldf@quit and \ldf@finish are no longer needed. Therefore they are turned into warning messages in \TeX.

\main@language
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.

\main@language
\def\main@language#1{\def\bbl@main@language{#1}\let\languagename\bbl@main@language}
\main@language
\bbl@id@assign\bbl@patterns{\languagename}
\main@language

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.

\bbl@beforestart
\def\bbl@beforestart{%\def\@nolanerr##1{%\bbl@warning{Undefined language '##1' in aux. \Reported}}%\bbl@usehooks{beforestart}{}\global\let\bbl@beforestart\relax\AtBeginDocument{%\if@filesw\providecommand\babel@aux[2]{}%\immediate\write\@mainaux{\string\providecommand\string\babel@aux[2]{}}\immediate\write\@mainaux{%\@nameuse{bbl@beforestart}}\fi\expandafter\selectlanguage\expandafter{\bbl@main@language}\ifbbl@single % must go after the line above.\renewcommand\selectlanguage[1]{}\renewcommand\foreignlanguage[2]{#2}\global\let\babel@aux\@gobbletwo % Also as flag\fi\ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

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

\select@language@x#1{\ifcase\bbl@select@type\bbl@ifsamestring\languagename{#1}{}\select@language{#1}\fi}
\select@language@x#1

8.5 Shorthands

\bbl@add@special
The macro \bbl@add@special is used to add a new character (or single character control sequence) to the macro \dospecials (and \@sanitize if \TeX is used). It is used only at one place, namely when \initiateactive@char is called (which is ignored if the char has been made active before). Because \@sanitize can be undefined, we put the definition inside a conditional. Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It’s already done with \nfss@catcodes, added in 3.10.

\bbl@trace{Shorhands}
\def\bbl@add@special#1{%\bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.\bbl@ifunset{@sanitize}{}\bbl@add\@sanitize{\@makeother#1}
The companion of the former macro is \bbl@remove@special. It removes a character from the set macros \dospecials and \@sanitize, but it is not used at all in the babel core.

\begin{verbatim}
\def\bbl@remove@special#1{\begingroup\def\x##1##2{\ifnum`#1=`##2\noexpand\@empty\else\noexpand##1\noexpand##2\fi}\def\do{\x\do}\def\@makeother{\x\@makeother}\edef\x{\endgroup\def\noexpand\dospecials{\dospecials}\def\noexpand\@sanitize{@sanitize}\relax\else\edef\noexpand\@sanitize{@sanitize}\fi}\x}
\end{verbatim}

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

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

\begin{verbatim}
\def\bbl@active@def#1#2#3#4{% 
  \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\endgroup
\end{verbatim}

When there is also no current-level shorthand with an argument we will check whether there is a next-level defined shorthand for this active character.
\begin{verbatim}
\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 ('ed) and the original one. This trick simplifies the code a lot.

\def\initiate@active@char#1{\bbl@ifunset{active@char\string#1}{\bbl@withactive
\expandafter\@initiate@active@char\expandafter}#1\string#1#1}{}

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

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

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

\ifx#1#3\relax\expandafter\let\csname normal@char#2\endcsname#3\else\bbl@info{Making #2 an active character}\nfnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi\namedef{normal@char#2}{\textormath{#3}{\csname bbl@oridef@@#2\endcsname}}\else\namedef{normal@char#2}{#3}\fi}

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

\bbl@restoreactive{#2}\
\AtBeginDocument{\if@filesw\immediate\write\@mainaux{\catcode`#2=\active}\fi}\bbl@add@special\csname#2\endcsname\catcode`#2=\active\fi}
\end{verbatim}

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

\let\bbl@tempa@firstoftwo
\if\string^#2%
  \def\bbl@tempa{\noexpand\textormath}%
\else
  \ifx\bbl@mathnormal@undefined\else
    \let\bbl@tempa\bbl@mathnormal
  \fi
\fi
\expandafter\edef\csname active@char#2\endcsname{%
  \bbl@tempa{
    \if@safe@actives
      \expandafter
      \expandafter
      \expandafter\csname normal@char#2\endcsname
    \else
      \expandafter
      \expandafter
      \expandafter\csname bbl@doactive#2\endcsname
    \fi
  }{%
\bbl@csarg\edef{doactive#2}{%\expandafter\noexpand\csname user@active#2\endcsname}%

We now define the default values which the shorthand is set to when activated or deactivated. It is set to the deactivated form (globally), so that the character expands to

\begin{verbatim}
  \active@prefix (char) \normal@char
\end{verbatim}

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

\bbl@csarg\edef{active@#2}{%\noexpand\active@prefix\noexpand#1%\expandafter\noexpand\csname active@char#2\endcsname}%
\bbl@csarg\edef{normal@#2}{%\noexpand\active@prefix\noexpand#1%\expandafter\noexpand\csname normal@char#2\endcsname}%
\expandafter\let\expandafter#1\csname bbl@normal@#2\endcsname

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

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

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

\expandafter\edef\csname user@group@sh@#2@@\endcsname{%\expandafter\noexpand\csname normal@char#2\endcsname}%
\expandafter\edef\csname user@group@sh@#2@\string\protect\endcsname{%\expandafter\noexpand\csname user@active#2\endcsname}%

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

\begin{itemize}
\item \texttt{math=active}
\item \texttt{math=normal}
\end{itemize}

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.

\begin{itemize}
\item \texttt{KeepShorthandsActive}
\end{itemize}

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

\begin{itemize}
\item \texttt{\bbl@sh@select}\!
\end{itemize}

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

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.

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.

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

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 ldf files.
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
\def\textormath{% 
  \ifmmode  
  \expandafter\@secondoftwo
  \else  
  \expandafter\@firstoftwo
  \fi
}

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

The current concept of 'shorthands' supports three levels or groups of shorthands. For each level the name of the level or group is stored in a macro. The default is to have a user group; use language group 'english' and have a system group called 'system'.

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

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

\def\useshorthands{% 
  \@ifstar\bbll@usesh@s{% \@bbll@usesh@x{}}%
  \bbll@usesh@s{#1}{% 
    \bbll@usesh@x{#1}{% 
      \AddBabelHook{babel-sh-#1}{afterextras}{\bbll@activate{#1}}%
        {#1}}%
  \bbll@usesh@s{#2}{% 
  \bbll@usesh@s{#3}{% 
    \bbll@usesh@s{#4}{% 
  \fi
}
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.

\defineshorthand

\languageshorthand

A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing 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 \aliasshorthands{"} is \active@prefix /\active@char/, so we still need to let the latest to \active@char".

97
\@notshorthand
\def\@notshorthand#1{%
  \bbLError{%
    The character \string #1 should be made a shorthand character;\%
    add the command \string\useshorthands\string(#1\string) to\%
    the preamble.\%
    I will ignore your instruction}%
  {You may proceed, but expect unexpected results}}
\shorthandon The first level definition of these macros just passes the argument on to \bbl@switch@sh, adding\@nil at the end to denote the end of the list of characters.
\shorthandoff \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
\DeclareRobustCommand*\shorthandoff{\ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
\def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2\@nnil}

\bbl@switch@sh The macro \bbl@switch@sh takes the list of characters apart one by one and subsequently switches the category code of the shorthand character according to the first argument of \bbl@switch@sh. But before any of this switching takes place we make sure that the character we are dealing with is known as a shorthand character. If it is, a macro such as \active@char” should exist. Switching off 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.
\def\bbl@switch@sh#1#2{%\ifx#2\@nnil\else\bbIffalse{bbl@active\string#2}%
  {\bbLError{%
    I can't switch \string#2 on or off--not a shorthand}%
    {This character is not a shorthand. Maybe you made\%
    a typing mistake? I will ignore your instruction.}}%
  {\ifcase#1% off, on, off*
    \catcode`#212\relax
    \catcode`#2\active
    \bbl@ifunset{bbl@shdef\string#2}{}%
    {\bbl@withactive{\let#2\csname bbl@shdef\string#2\endcsname}
      \csname bbl@oridef\string#2\endcsname}%
    \ifcase\bbl@activated\or
      \bbl@activate(#2)%
    \else
      \bbl@deactivate(#2)%
    \fi
  }%}
  \bbl@afterfi\bbl@switch@sh#1%\fi}
\babelfile{shorthand}{\active@prefix\babelfile{shorthand}{\bbl@putsh}}
\def\bbl@putsh#1{%\bbIffalse{bbl@active\string#1}%
  \bbIffalse{bbl@active\string#1}%
}
\babelfile{shorthand}{\active@prefix\babelfile{shorthand}{\bbl@putsh}}
\def\bbl@putsh#1{%\bbIffalse{bbl@active\string#1}%
  \bbIffalse{bbl@active\string#1}%

Note the value is that at the expansion time; eg, in the preamble shorthands are usually deactivated.
You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on or off.

\\@prim@s
\\@pr@m@s

One of the internal macros that are involved in substituting \texttt{\textbackslash prime} for each right quote in math mode is \texttt{\textbackslash prim@s}. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.

\def\\@prim@s{}\futurelet\@let@token\@pr@m@s

\def\@pr@m@s{}\if\@let@token\expandafter\@firstoftwo\else\if\@let@token\bbl@afterelse\expandafter\@firstoftwo\else\bbl@afterfi\expandafter\@secondoftwo\fi\fi

\begin{group}
\catcode`^=7 \catcode`*=\active \lccode`*=\^ \lowercase{
\gdef\@pr@m@s{}\if\@let@token\bbl@if@primes"\pr@@@s{\bbl@if@primes^\pr@@@t\egroup}}
\end{group}

Usually the ~ is active and expands to \texttt{\penalty\@M\penalty\@M\penalty\@M}. When it is written to the .aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).

\initiate@active@char{	exttt{~}}
\declare@shorthand{system}{~}{\leavevmode\nobreak\ }
The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the \f@encoding macro. Therefore we define two macros here to store the position of the character in these encodings.

\expandafter\def\csname OT1dqpos\endcsname{127}
\expandafter\def\csname T1dqpos\endcsname{4}

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

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.

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.

\languageattribute
\newcommand\languageattribute[2]{%
\bbl@trace{Language attributes}
\newcommand\languageattribute[2]{%
\def\bbl@tempc{#1}%
\bbl@fixname\bbl@tempc
\bbl@iflanguage\bbl@tempc{%
\bbl@vforeach{#2}{%
Wewant to make sure that each attribute is selected only once; therefore we store the already selected attributes in \bbl@known@attribs. When that control sequence is not yet defined this attribute is certainly not selected before.
\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
When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated \TeX-code.
\bbl@exp{%}
\\\bbl@add@list\bbl@known@attribs{\bbl@tempc-##1})%
\edef\bbl@tempa{\bbl@tempc-##1}%
\expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes{\csname\bbl@tempc @attr@##1\endcsname}{\@attrerr{\bbl@tempc}{##1}}%
\fi}}}
\@onlypreamble\languageattribute

The error text to be issued when an unknown attribute is selected.
\newcommand*{\@attrerr}[2]{%
\bbl@error{The attribute #2 is unknown for language #1.}%
\{Your command will be ignored, type <return> to proceed}}
\bbl@declare@ttribute \hspace{1em} 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 \bbl@\texttt{extr...} for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \texttt{\begin{document}}.

\begin{verbatim}
1572 \def\bbl@declare@ttribute#1#2#3{%
1573 \bbl@xin{,#2,}{,\BabelModifiers,}%
1574 \ifin@
1575 \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1576 \fi
1577 \bbl@add@list\bbl@attributes{#1-#2}%
1578 \expandafter\def\csname#1@attr@#2\endcsname{#3}}
\end{verbatim}

\bbl@ifattributeset \hspace{1em} This internal macro has 4 arguments. It can be used to interpret \TeX\ code based on whether a certain attribute was set. This command should appear inside the argument to \texttt{\AtBeginDocument} because the attributes are set in the document preamble, after \texttt{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.

\begin{verbatim}
1579 \def\bbl@ifattributeset#1#2#3#4{% 
1580 \ifx\bbl@known@attribs\@undefined
1581 \in@false 
1582 \else 
1583 \bbl@xin{,#1-#2,}{,\bbl@known@attribs,}%
1584 \fi 
1585 \ifin@
1586 \bbl@afterelse#3%
1587 \else 
1588 \bbl@afterfi#4%
1589 \fi
\end{verbatim}

\bbl@ifknown@trib \hspace{1em} 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.

\begin{verbatim}
1590 \def\bbl@ifknown@trib#1#2{% 
1591 \let\bbl@tempa@secondoftwo
1592 \bbl@loopx\bbl@tempb{#2}{%
1593 \expandafter\in@\expandafter{\expandafter{,\bbl@tempb,}{,#1,}}%
1594 \ifin@
1595 \let\bbl@tempa@firstoftwo
1596 \else 
1597 \fi}%
1598 \bbl@tempa}
\end{verbatim}

\bbl@clear@ttribs \hspace{1em} This macro removes all the attribute code from \TeX\-\texttt{\begin{document}} time (if any is present).

\begin{verbatim}
1599 \def\bbl@clear@ttribs{%
1600 \ifx\bbl@attributes\@undefined\else 
1601 \bbl@loopx\bbl@tempa{\bbl@attributes}{% 
1602 \expandafter\bbl@clear@trib\bbl@tempa. %
1603 \} %
1604 \let\bbl@attributes\@undefined
1605 \fi
1606 \def\bbl@clear@trib#1-#2{.%
1607 \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1608 \AtBeginDocument{\bbl@clear@ttribs}
\end{verbatim}
8.7 Support for saving macro definitions

To save the meaning of control sequences using \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). Note undefined macros are not undefined any more when saved – they are \relax’ed.

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

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

The macro \save⟨csname⟩ saves the current meaning of the control sequence ⟨csname⟩ to \Original. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to \Original and the counter is incremented. The macro \savevariable⟨variable⟩ saves the value of the variable. ⟨variable⟩ can be anything allowed after the \the primitive.

Some languages need to have \frenchspacing in effect. Others don’t want that. The command \frenchspacing switches it on when it isn’t already in effect and \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 \provide. This new method should be ideally the default one.

\Original has to be expandable, i.e. you shouldn’t let it to \relax.

1629 \let\bbl@nonfrenchspacing%
8.8 Short tags
\bbltags This macro is straightforward. After zapping spaces, we loop over the list and define the macros \text{$\langle$tag$\rangle$} and \$\langle$tag$\rangle$. Definitions are first expanded so that they don’t contain \csname but the actual macro.

8.9 Hyphens
\bblhyphenation 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.
This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hspace 0pt plus 0pt\nobreak 32.\nobreak

Macro 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 shorthand, with \active@prefix.\nobreak

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.\nobreak

Finally, we define the hyphen “types”. Their names will not change, so you may use them in \ldf’s. After a space, the \mbox in \bl@hy@nobreak is redundant.\nobreak
\bbl@disc 
For some languages the macro \bbl@disc is used to ease the insertion of discretionary for letters that behave ‘abnormally’ at a breakpoint.

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

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?
    \ifnum\@tempcnta="FF\else
        \catcode\@tempcnta=#1\relax
        \advance\@tempcnta\@one
        \expandafter\bbl@tempa
    \fi
}\bl@tempa

These second one. We need to patch \@uclclist, 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 \@uclclist 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@uclc. The parser is restarted inside \langle lang⟩\bbl@uclc because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when upercasing, we have:
\let\bbl@tolower\empty\bbl@toupper\empty

and starts over (and similarly when lowercasing).
\ifpackagewith{babel}{nocase}%
{\let\bbl@patchuclc\relax}%
{\def\bbl@patchuclc{\global\let\bbl@patchuclc\relax
    \g@addto@macro\@uclclist{\reserved@a{\reserved@b\bbl@uclc}\
    \gdef\bbl@uclc#1{%
    \let\bbl@encoded\bbl@encoded@uclc
    \bbl@ifunset{\languagename @bbl@uclc}\% and resumes it
    {#1}\
}}

105
The following package options control the behavior of \SetString.

\def\StartBabelCommands{%
\begingroup
\bbl@recatcode{11}\
\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\bbl@startcmds
\if\bbl@screset\@nnil\else
\bbl@usehooks{stopcommands}{}%
\fi
\endgroup
\begingroup
\@ifstar
{\ifx\bbl@opt@strings\@nnil
\let\bbl@opt@strings\BabelStringsDefault
\fi
\bbl@startcmds@i}%
\bbl@startcmds@i}
\def\bbl@startcmds@i#1#2{%
\edef\bbl@L{\zap@space#1 \@empty}%
\edef\bbl@G{\zap@space#2 \@empty}%
\bbl@startcmds@ii}
\let\bbl@startcommands\StartBabelCommands

\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.

\parse the encoding info to get the label, input, and font parts.
Select the behavior of \SetString. There are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the
strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```latex
\newcommand{\bbl@startcmds@ii}[1][\empty]{%
  \let\SetString@gobbletwo
  \let\bbl@stringdef@gobbletwo
  \let\AfterBabelCommands@gobble
  \ifx\empty#1%
    \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
    \let\bbl@sc@fontenc space % <- " "
    \def\bbl@tempa##1=##2\@nil{%\bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }%}
    \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
    \def\bbl@tempa##1 ##2{% space -> comma
      ##1%
      \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
    \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
    \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
    \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
    \def\bbl@encstring##1##2{%\bbl@foreach\bbl@sc@fontenc{%\bbl@ifunset{T@####1}{}%{\ProvideTextCommand##1{####1}{##2}%
      \bbl@toglobal##1%
      \expandafter\bbl@toglobal\csname####1\string##1\endcsname}}}%
    \def\bbl@sctest{%\bbl@xin@{},\bbl@opt@strings,{},\bbl@@sc@label,\bbl@@sc@fontenc,}%
  \fi
  \ifx\bbl@opt@strings\@nnil % ie, no strings key -> defaults
    \else\ifx\bbl@opt@strings\relax % ie, strings=encoded
      \let\AfterBabelCommands\bbl@aftercmds
      \let\SetString\bbl@setstring
      \let\bbl@stringdef\bbl@provstring
    \else % ie, strings=value
      \let\bbl@sctest
    \fi\fi\fi
  \fi
\bbl@scswitch
\if\bbl@G\@empty
  \def\error{Missing group for string \string##1}%
  {You must assign strings to some category, typically\%
    captions or extras, but you set none})%
\fi
```
There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes sure \(\langle group\rangle\langle language\rangle\) is reset, but only once (\bbl@screset is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing. The macro \bbl@forlang loops \bbl@L but its body is executed only if the value is in \texttt{\BabelLanguages} (inside babel) or \texttt{\date\langle language\rangle} is defined (after babel has been loaded). There are also two version of \bbl@forlang. The first one skips the current iteration if the language is not in \texttt{\BabelLanguages} (used in ldfs), and the second one skips undefined languages (after babel has been loaded).

\def\bbl@forlang#1#2{%
  \bbl@for#1\bbl@L{%\bbl@xin@{,#1,}{,\BabelLanguages,}%
    \ifin@#2\relax\fi}}
\def\bbl@scswitch{%
  \bbl@forlang\bbl@tempa{%\bbl@for#1\bbl@GL{%\bbl@xin@{\bbl@GL,}{,\bbl@screset,}%
    \ifin@else
      \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
      \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
    \fi
  }%}
\AtEndOfPackage{%
  \def\bbl@forlang#1#2{%{\bbl@for#1\bbl@L{%{\bbl@ifunset{date#1}{}{#2}}}%}
  \let\bbl@scswitch\relax}
\@onlypreamble\EndBabelCommands
\def\EndBabelCommands{%
  \bbl@usehooks{stopcommands}{}%
  \endgroup}
\let\bbl@scafter\
\let\bbl@endcommands\EndBabelCommands
Now we define commands to be used inside \texttt{\StartBabelCommands}.

**Strings** The following macro is the actual definition of \texttt{\SetString} when it is “active” First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like \texttt{\providescommand}). With the event \texttt{stringprocess} you can preprocess the string by manipulating the value of \texttt{\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@tiempa{%\bbl@for#1\bbl@LC{%\bbl@xinslash{\bbl@tiempa}{\bbl@stripslash#1}%
    \ifin@\bbl@exp{%\bbl@iunset{\bbl@tiempa}{\bbl@LC}{eg, \germanchaptername}
      \{\bbl@exp%
        \global\bbl@add\langle\bbl@G\bbl@tiempa{\bbl@add\langle\bbl@LC\rangle}{\bbl@iunset{\bbl@set\#1}{\langle\bbl@LC\rangle}}%
      }%
  }%}
\def\BabelString{#2}%
\bbl@usehooks{stringprocess}{}%
Now, some additional stuff to be used when encoded strings are used. Captions then include \\
\texttt{\textbackslash encoded} for string to be expanded in case transformations. It is \texttt{\textbackslash relax} by default, but in \texttt{Make\texttt{\textbackslash Uppercase} and \texttt{Make\texttt{\textbackslash Lowercase}}} its value is a modified expandable \texttt{\textbackslash @changed@cmd}.

\begin{verbatim}
1899 \texttt{\cename\bbl@LC\expandafter\endcsname\expandafter{\textbackslash Babel\texttt{\textbackslash String}}}\}

1900 \texttt{\ifx\bbl@opt@strings\texttt{\textbackslash relax}}
1901 \texttt{\def\bbl@scset##1##2{\def##1{\bbl@encoded##2}}}
1902 \texttt{\bbl@patchuclc}
1903 \texttt{\let\bbl@encoded\texttt{\textbackslash relax}}
1904 \texttt{\def\bbl@encoded@uclc#1{\@inmathwarn#1}}
1905 \texttt{\expandafter\ifx\csname \cf@encoding\string#1\endcsname\texttt{\textbackslash TextSymbolUnavailable\texttt{\textbackslash number}}}\}
1906 \texttt{\else}
1907 \texttt{\csname \cf@encoding\string#1\endcsname\texttt{\textbackslash endcsname\texttt{\textbackslash string}}}\}
1908 \texttt{\fi}
1909 \texttt{\else}
1910 \texttt{\cename \texttt{\textbackslash string}\}
1911 \texttt{\fi}
1912 \texttt{\else}
1913 \texttt{\cename \texttt{\textbackslash cf@encoding}\texttt{\textbackslash string}\}
1914 \texttt{\fi}
1915 \texttt{\else}
1916 \texttt{\def\bbl@scset##1##2{\def##1{##2}}}
1917 \texttt{\fi}
\end{verbatim}

Define \texttt{\textbackslash Set\texttt{\textbackslash StringLoop}}, which is actually set inside \texttt{\Start\textbackslash Babel\texttt{\textbackslash Commands}}. The current definition is somewhat complicated because we need a count, but \texttt{\count@} is not under our control (remember \texttt{\textbackslash Set\texttt{\textbackslash String}} may call hooks). Instead of defining a dedicated count, we just “pre-expand” its value.

\begin{verbatim}
1918 (\*Macros local to Babel\texttt{\textbackslash Commands}\}) ≡
1919 \def\Set\texttt{\textbackslash StringLoop}##1##2{\%
1920 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}\}
1921 \count@\z@
1922 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1923 \advance\count@\texttt{\textbackslash one}
1924 \toks@\expandafter{\bbl@tempa}\%
1925 \bbl@exp{\%
1926 \texttt{\\Set\textbackslash String}\bbl@templ{\texttt{\textbackslash roman\texttt{\textbackslash numeral}}\texttt{\textbackslash count}@\texttt{\textbackslash the}\texttt{\toks@} }\%
1927 \count@@\texttt{\textbackslash the}\texttt{\textbackslash count}@\texttt{\textbackslash relax}\})\%
1928 (\*Macros local to Babel\texttt{\textbackslash Commands}\})
\end{verbatim}

\textbf{Delaying code} Now the definition of \texttt{\\After\textbackslash Babel\texttt{\textbackslash Commands}} when it is activated.

\begin{verbatim}
1929 \def\bbl@aftercmds#1{%
1930 \toks@\expandafter{\bbl@scafter#1}%
1931 \def\bbl@scafter{\texttt{\the\toks@}}
\end{verbatim}

\textbf{Case mapping} The command \texttt{\textbackslash Set\texttt{\textbackslash Case}} provides a way to change the behavior of \texttt{\textbackslash Make\texttt{\textbackslash Uppercase} and \texttt{Make\texttt{\textbackslash Lowercase}}} \texttt{\bbl@tempa} is set by the patched \texttt{\textbackslash uclclist} to the parsing command.

\begin{verbatim}
1932 (\*Macros local to Babel\texttt{\textbackslash Commands}\}) ≡
1933 \newcommand\Set\texttt{\textbackslash Case}[3][]{\%
1934 \bbl@patchuclc
1935 \bbl@forlang\bbl@tempa{%
1936 \expandafter{\bbl@tempa}\%
1937 \cename\bbl@tempa@\bbl@uc\texttt{\textbackslash lclist}\texttt{\textbackslash endcsname}\texttt{\textbackslash bbl@tempa}\%
1938 \expandafter{\bbl@tempa}\%
1939 \cename\bbl@tempa@\bbl@uc\texttt{\textbackslash lclist}\texttt{\textbackslash endcsname}\%
1940 \expandafter{\bbl@tempa}\%
1941 \cename\bbl@tempa@\bbl@lc\texttt{\textbackslash endcsname}\%
1942 (\*Macros local to Babel\texttt{\textbackslash Commands}\})
\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.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.

The following package options control the behavior of hyphenation mapping.
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.
\def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}\%}
\dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@\%
\setbox\z@\hbox{\lower\dimen\z@ \box\z@\hbox{\ht\z@\@ht\tw@ \dp\z@\dp\tw@}}

\save@sf@q The macro \save@sf@q is used to save and reset the current space factor.
\def\save@sf@q#1{\leavevmode
  \begingroup
    \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
  \endgroup}

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

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

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

\UseTextSymbol{OT1}{\quotedblbase}

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

\ProvideTextCommandDefault{\quotedblbase}{%}
\UseTextSymbol{OT1}{\quotedblbase}

\texttt{\quotesinglbase} We also need the single quote character at the baseline.

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

\UseTextSymbol{OT1}{\quotesinglbase}

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

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

\texttt{\guillemetleft} 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}

\UseTextSymbol{OT1}{\guillemetleft}
\UseTextSymbol{OT1}{\guillemetright}

\save@sf@q

112
\raise.2ex\hbox{$\scriptscriptstyle<\$}\bbl@allowhyphens}%
\fi
\ProvideTextCommand{\guillemotright}{OT1}{%\ifmmode
\gg
\else
\save@sf@q{\nobreak
\raise.2ex\hbox{$\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
\guilsinglright

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

\ProvideTextCommand{\guilsinglleft}{OT1}{%\ifmmode
<%
\else
\save@sf@q{\nobreak
\raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
\fi}
\ProvideTextCommand{\guilsinglright}{OT1}{%\ifmmode
>%
\else
\save@sf@q{\nobreak
\raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
\fi}

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

\provideTextCommandDefault{\guilsinglleft}{%\UseTextSymbol{OT1}{\guilsinglleft}}
\provideTextCommandDefault{\guilsinglright}{%\UseTextSymbol{OT1}{\guilsinglright}}

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.
\IJ

\DeclareTextCommand{\ij}{OT1}{%\i\kern-0.02em}\bbl@allowhyphens j}
\DeclareTextCommand{\IJ}{OT1}{\I\kern-0.02em}\bbl@allowhyphens J}
\DeclareTextCommand{\ij}{T1}{\char188}
\DeclareTextCommand{\IJ}{T1}{\char156}

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

\provideTextCommandDefault{\ij}{%\UseTextSymbol{OT1}{\ij}}
\provideTextCommandDefault{\IJ}{%\UseTextSymbol{OT1}{\IJ}}

113
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).

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.

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.

The ‘German’ single quotes.

The definition of \grq depends on the fontencoding. With T1 encoding no extra kerning is needed.
The definition of \texttt{\textdoublequote} depends on the font encoding. With T1 encoding no extra kerning is needed.

\ProvideTextCommand{\textdoublequote}{T1}{% \texttt{\textquotedblleft}{\mbox{\textquotedblleft}}}\ProvideTextCommand{\textdoublequote}{TU}{% \texttt{\textquotedblleft}{\mbox{\textquotedblleft}}}\ProvideTextCommand{\textdoublequote}{OT1}{% \save@sf@q{\kern-.07em \texttt{\textquotedblleft}{\mbox{\textquotedblleft}} \kern.07em \relax}}\ProvideTextCommandDefault{\textdoublequote}{\UseTextSymbol{OT1}\textdoublequote}

\flq The 'french' single guillemets.
\frq
\flqq The 'french' double guillemets.
\frqq

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

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

\lower@umlaut

The following code fools T\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 \texttt{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 \texttt{\accent} primitive, reset the old x-height and insert the base character in the argument.
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).

\AtBeginDocument{\DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}\DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}\DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}\DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{\i}}\DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}\DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}\DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}\DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}\DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}\DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}\DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}}

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

\ifx\l@english\undefined\chardef\l@english\z@\fi
\newlanguage\l@unhyphenated\ifx\l@unhyphenated\undefined\newlanguage\l@unhyphenated\fi

\AtBeginDocument{\DeclareTextCompositeCommand{"}{OT1}{a}{\bbl@umlauta{a}}\DeclareTextCompositeCommand{"}{OT1}{e}{\bbl@umlaute{e}}\DeclareTextCompositeCommand{"}{OT1}{i}{\bbl@umlaute{i}}\DeclareTextCompositeCommand{"}{OT1}{\i}{\bbl@umlaute{\i}}\DeclareTextCompositeCommand{"}{OT1}{o}{\bbl@umlauta{o}}\DeclareTextCompositeCommand{"}{OT1}{u}{\bbl@umlauta{u}}\DeclareTextCompositeCommand{"}{OT1}{A}{\bbl@umlauta{A}}\DeclareTextCompositeCommand{"}{OT1}{E}{\bbl@umlaute{E}}\DeclareTextCompositeCommand{"}{OT1}{I}{\bbl@umlaute{I}}\DeclareTextCompositeCommand{"}{OT1}{O}{\bbl@umlauta{O}}\DeclareTextCompositeCommand{"}{OT1}{U}{\bbl@umlauta{U}}}

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

8.13 Layout

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

\input txtbabel.def
\input luababel.def
\input xebabel.def

8.15 Creating and modifying languages

\bbl@trace{Creating languages and reading ini files}
\def\bbl@KVP@captions{\nil}
\def\bbl@KVP@date{\nil}
\def\bbl@KVP@import{\nil}
\def\bbl@KVP@language{\nil}
\def\bbl@KVP@script{\nil}
\def\bbl@KVP@Hyphenrules{\nil}
\def\bbl@KVP@linebreaking{\nil}
\def\bbl@KVP@justification{\nil}
\def\bbl@KVP@mapfont{\nil}
\def\bbl@KVP@maparabic{\nil}
\def\bbl@KVP@mapdigits{\nil}
\def\bbl@KVP@intraspace{\nil}
\def\bbl@KVP@intrapenalty{\nil}
\let\bbl@KVP@captions\nil
\let\bbl@KVP@date\nil
\let\bbl@KVP@import\nil
\let\bbl@KVP@language\nil
\let\bbl@KVP@script\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
\else
\bbl@ifblank{#1}%
{\% With \bbl@load@basic below
{\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{\
\babelensure[exclude=\today,\include=\bbl@extracaps@#2]\{#2\}}%}
\bbl@exp{\bbl@exp{\
\DeclareRobustCommand\bbl@ensure@\languagename\space[1]{%\bbl@ensure@\languagename\space}\{##1\}\}}%}
{\%}
\bbl@exp{\bbl@exp{\bbl@toglobal<\bbl@ensure@\languagename>\bbl@toglobal<\bbl@ensure@\languagename\space>}}%
\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
\ifx\bbl@KVP@script\@nil\else
\bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
\fi
\ifx\bbl@KVP@language\@nil\else
\bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
\fi
% == onchar ==
\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}%
\fi

\ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
\AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
\fi
\bbl@exp{\bbl@add<\bbl@starthyphens>{\bbl@patterns@lua\{\languagename\}}}%
% TODO - error/warning if no script
\directlua{
  if Babel.script_blocks["\bbl@cl{sbcp}"] then
    Babel.loc_to_scr[\the\localeid] = Babel.script_blocks["\bbl@cl{sbcp}"]
    Babel.locale_props[\the\localeid].lc = \the\localeid\space
    Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
  end
}%
\fi
\bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}\
\ifin@
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}
\directlua{
  if Babel.script_blocks["\bbl@cl{sbcp}"] then
    Babel.loc_to_scr[\the\localeid] = Babel.script_blocks["\bbl@cl{sbcp}"]
  end}%
\ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
  \AtBeginDocument{%
    \bbl@patchfont{{\bbl@mapselect}}%
    \selectfont}%
  \def\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]%'\bbl@prefontid' = \fontid\font\space}}%}
\fi
\bbl@exp{\\bbl@add\\bbl@mapselect{\\bbl@mapdir{\languagename}}}%
\fi
% TODO - catch non-valid values
\fi
% == mapfont ==
% For bidi texts, to switch the font based on direction
\ifx\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}}{}
\ifx\bbl@mapselect\@undefined % TODO. See onchar.
  \AtBeginDocument{%
    \bbl@patchfont{{\bbl@mapselect}}%
    \selectfont}%
  \def\bbl@mapselect{%
    \let\bbl@mapselect\relax
    \edef\bbl@prefontid{\fontid\font}%
  }\def\bbl@mapdir##1{%
  \def\languagename{##1}%
  \let\bbl@ifrestoring\@firstoftwo % avoid font warning
  \bbl@switchfont
  \directlua{Babel.fontmap
    [\the\csname bbl@wdir@@##1\endcsname]%.}}%}
\fi
% == mapfont ==
Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.

\def\bbl@provide@new#1{%
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.)

\def\bbl@load@basic#1{%
  \ifcase\bbl@howloaded\or\or
    \ifcase\csname bbl@llevel@\languagename\endcsname
      \bbl@csarg\let\{lname@\languagename}\relax
    \fi
  \fi
  \bbl@ifunset{bbl@lname@#1}%
  \{\def\BabelBeforeIni###1.###2.###3.###4\relax ###5###6{}%
    \bbl@read@ini{#1}1%
    \ifx\bbl@initoload\relax\endinput\fi
  \}\endgroup%
\begingroup % boxed, to avoid extra spaces:
  \ifx\bbl@initoload\relax
    \bbl@input@texini{#1}%
  \else
    \setbox\z\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
  \fi
\endgroup}{}

The hyphenrules option is handled with an auxiliary macro.

\def\bbl@provide@hyphens#1{%
  \let\bbl@tempa\relax
  \ifx\bbl@KVP@hyphenrules\@nil\else
    \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
    \bbl@foreach\bbl@KVP@hyphenrules{%
      \ifx\bbl@tempa\relax % if not yet found
        \bbl@ifsamestring{##1}{+}%
        \ifx\bbl@tempa\relax % if no opt or no language in opt found
          \bbl@ifblank{\bbl@cs{hyphr@#1}}% \addlanguage<\l@##1>
        \else % if importing
          \bbl@ifblank{\bbl@cs{hyphr@#1}}% \addlanguage<\l@\bbl@cl{hyphr}>}
        \fi
      \fi}
  \fi
  \ifx\bbl@tempp% if no opt or no language in opt found
    \bbl@import\atnil
    \ifx\bbl@KVP@import\atnil
      \bbl@exp{\let\bbl@tempp\relax<}\l@\bbl@cl{hyphr}>}
    \else
      \setbox\z\hbox{\BabelBeforeIni{\bbl@tempp}{}}%
    \fi
  \fi
  \ifx\bbl@tempp% if importing
    \bbl@exp{% and hyphenrules is not empty
      \bbl@ifblank{\bbl@cs{hyphr@#1}}% \addlanguage<\l@\bbl@cl{hyphr}>}
  \fi
  \else % if importing
    \bbl@exp{% and hyphenrules is not empty
      \bbl@ifblank{\bbl@cs{hyphr@#1}}% \addlanguage<\l@\bbl@cl{hyphr}>}
  \fi
  \bbl@ifunset{bbl@tempp} ie, relax or undefined
  \{\def\bbl@funset{\l@}% no hyphenrules found - fallback
    \bbl@exp{\let\bbl@tempp<}\l@<\l@\bbl@cl{hyphr}>}
  \}
\}
The reader of babel-...tex files. We reset temporarily some catcodes.

\def\bbl@input@texini#1{\bbl@bsphack\\bbl@exp{\catcode`\%=14 \catcode`\\=0\catcode`\{=1 \catcode`\}=2\lowercase{\InputIfFileExists{babel-#1.tex}{}{}}\catcode`\%=	he\catcode`\%\relax\catcode`\\=	he\catcode`\\\relax\catcode`\{=	he\catcode`\}\relax}\bbl@esphack}

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.

\def\bbl@iniline#1\bbl@iniline{\@ifnextchar[\bbl@inisect\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@% \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}\def\bbl@iniskip#1\@@{}% if starts with ;; \def\bbl@inistore#1=#2\@@{% full (default) \bbl@trim@def\bbl@tempa{#1}% \bbl@trim\toks@{#2}% \bbl@xin@{;\bbl@section/\bbl@tempa}{\bbl@key@list}% \ifin@% \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini) \bbl@trim@def\bbl@tempa{#1}% \bbl@trim\toks@{#2}% \bbl@xin@{.identification.}{.;\bbl@section.}% \ifin@% \bbl@exp{\g@addto@macro\bbl@inidata{\bbl@section}{\bbl@tempa}{\the\toks@}}}% \fi}% 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.

\ifx\bbl@readstream\undefined\csname newread\endcsname\bbl@readstream\fi% \def\bbl@read@ini#1#2{% \global\let\bbl@extend@ini\@gobble \openin\bbl@readstream=babel-#1.ini\ifeof\bbl@readstream\bbl@error{There is no ini file for the requested language#1. Perhaps you misspelled it or your installation is not complete.}% \else\fi% \def\bbl@read@ini#1#2{% (There is no ini file for the requested language#1. Perhaps you misspelled it or your installation is not complete.}% \else
% Store ini data in \bbl@inidata =
\catcode`\[=12 \catcode`\]=12 \catcode`\|=12 \catcode`\%=12 \catcode`\&=12
\catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
\bbl@info{Importing \ifcase#2 font and identification \or basic \fi data for \languagename\%
from babel-#1.ini. Reported}\%
\ifnum#2=\z@
  \global\let\bbl@inidata\@empty
  \let\bbl@inistore\bbl@inistore@min % Remember it's local
\fi
\def\bbl@section{identification}%
\bbl@exp{\bbl@inistore tag.ini=#1\@@}
\bbl@inistore load.level=#2\@@
\loop
  \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
  \endlinechar\m@ne
  \read\bbl@readstream to \bbl@line
  \endlinechar`^^M
  \ifx\bbl@line\@empty\else
    \expandafter\bbl@iniline\bbl@line\bbl@iniline
  \fi
\repeat
% Process stored data ==
\bbl@csarg\xdef{lini@\languagename}{#1}%
\bbl@read@ini@aux
% 'Export' data ==
\bbl@exp{\def\bbl@stringdef\bbl@exportkey\bbl@inidata}
\global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
\global\let\bbl@inidata\@empty
\bbl@exp{\bbl@add@list\bbl@ini@loaded{\languagename}}%
\bbl@toglobal\bbl@ini@loaded
\def\bbl@read@ini@aux{%
  \let\bbl@savestrings\@empty
  \let\bbl@savetoday\@empty
  \let\bbl@savedate\@empty
  \def\bbl@elt##1##2##3{%
    \def\bbl@section{##1}%
    \in@{=date.}{=##1}% Find a better place
    \ifin@
      \bbl@ini@calendar{##1}%
    \fi
    \bbl@ifunset{bbl@inikv@##1}{}%
    \csname bbl@inikv@##1\endcsname{##2}{##3}}}%
  \bbl@inidata}
A variant to be used when the ini file has been already loaded, because it's not the first
\babelprovide for this language.
\def\bbl@extend@ini@aux{%
  \let\bbl@datestrings\empty
  \let\bbl@today\empty
  \let\bbl@date\empty
  \def\bbl@elt###1###2###3{%
    \def\bbl@section{###1}%
    \in@{=date.}{=###1}% Find a better place
    \ifin@
      \bbl@ini@calendar{###1}%
    \fi
    \bbl@ifunset{bbl@@kv@###2}{}%
    \csname bbl@@kv@###2\endcsname{###3}{###3}}%
  \bbl@inidata}
\def\bbl@startcommands*{#1}{captions}%
% Activate captions/... and modify exports
\bbl@csarg\def{linikv@captions.licr}{#1##2}{#2}{#3}{#4}{#5}{#6}(\gdef{#1}{#2}){#3}{#4}{#5}{#6}%
\setlocalecaption{#1}{#2}{#3}\%
\bbl@ini@captions@aux{#1}{#2}{#3}{#4}{#5}{#6}
As a somewhat hackish tool to handle calendar sections. To be improved.

A key with a slash in \texttt{babel} 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 \texttt{\bbl@inistore} above).

The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.
Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@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@exports#1{%
\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}{}%
% Identification always exported
\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{htoc}{typography.hyphenate.other.locale}{}%
\bbl@exportkey{htos}{typography.hyphenate.other.script}{}%
\bbl@exportkey{in tsp}{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}{}% 0 = only info, 1, 2 = basic, (re)new
\bbl@exportkey{rqtex}{identification.require.babel}{}%
\bbl@savedate
\bbl@savetoday
% Conditional
\ifnum#1=\tw@ % only (re)new
\bbl@exportkey{rqtex}{identification.require.babel}{}%
A shared handler for key=val lines to be stored in \bbl@section.<key>.

\def\bbl@inikv#1#2{% key=value
\toks@{#2}% This hides #'s from ini values
\bbl@csargedef{kv@\bbl@section.#1}{\the\toks@}
}

By default, the following sections are just read. Actions are taken later.

\let\bbl@inikv@identification\bbl@inikv
\let\bbl@inikv@typography\bbl@inikv
\let\bbl@inikv@characters\bbl@inikv
\let\bbl@inikv@numbers\bbl@inikv

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

\def\bbl@inikv@counters#1#2{% \bbl@ifsamestring{#1}{digits}% {\bbl@error{The counter name 'digits' is reserved for mapping\%}
{Use another name.}}%
{}% \def\bbl@tempc{#1}% \bbl@trim@def{\bbl@tempb*}{#2}% \in@{.1$}{#1$}% \if\in@
\bbl@replace\bbl@tempc{.1}{}%
\bbl@csarg{\protected@xdef}{cntr@\bbl@tempc @\languagename}{\bbl@alphnumeral{\bbl@tempc}}%
\in@{.F.}{#1}%% \if\in@
\bbl@csarg{\protected@xdef}{cntr@#1@\languagename}{\bbl@tempb*}
\else
\toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
\expandafter\bbl@buildifcase\bbl@tempb* \ % Space after \%
\bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}{\bbl@tempa}
\fi}

Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

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

The auxiliary macro for captions define \<caption>name.

\def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
\bbl@replace\bbl@tempa{.template}{%\bbl@torreplace{#1}{#2}}
\def\bbl@ini@captions@aux#1#2{%\bbl@torreplace{#1}{#2}}
\fi

The auxiliary macro for captions define \<caption>name.
Labels. Captions must contain just strings, no format at all, so there is new group in ini files.
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 replacements work in many classes, but not all. Actually, the following lines are somewhat tentative.

```
def \bbl@chaptype{chapter}
\ifx \@makechapterhead \@undefined
  \let \bbl@patchchapter \relax
\else
  \ifx \thechapter \@undefined
    \let \bbl@patchchapter \relax
  \else
    \def \bbl@patchchapter {
      \global \let \bbl@patchchapter \relax
      \gdef \bbl@chfmt {
        \bbl@ifunset { bbl@chaptype fmt@languagename } {
          \@chapapp space \thechapter
        }{ bbl@chaptype fmt@languagename }
      }
      \bbl@add \appendix {
        \def \bbl@chaptype { appendix }
        \bbl@sreplace \ps@headings {
          \@chapapp \space \thechapter
        }{ \bbl@chfmt }
        \bbl@sreplace \chaptermark {
          \@chapapp \space \thechapter
        }{ \bbl@chfmt }
        \bbl@sreplace \@makechapterhead {
          \@chapapp \space \thechapter
        }{ \bbl@chfmt }
      }
      \bbl@toglobal \appendix
      \bbl@toglobal \ps@headings
    }
  \fi
\fi
```
Date. TODO. Document

% Arguments are _not_ protected.
\let\bbl@calendar\@empty
\DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
\def\bbl@localedate#1#2#3#4\relax#5#6{% TODO - ignore with 'captions'
  \bbl@trim@def\bbl@tempa{#1.#2}\
  \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
  {\bbl@trim@def\bbl@toreplace{#5}\
    \bbl@TG@@date
    \bbl@ifunset{bbl@date\@language@name @}\
    {\bbl@exp% TODO. Move to a better place.
      \gdef<\language@name date>{\bbl@usedategrouptrue
        \bbl@exp% TODO. Move to a better place.
        \gdef<\language@name date >####1####2####3{\bbl@usedategroupfalse}
Dates will require some macros for the basic formatting. They may be redefined by language, so "semi-public" names (camel case) are used. Oddly enough, the CLDR places particles like “de” inconsistently in either in the date or in the month name. 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} % TODO. \let instead of repeating
\newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
\newcommand\BabelDated[1]{{\number#1}}
\newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
\newcommand\BabelDateM[1]{{\number#1}}
\newcommand\BabelDateMM[1]{{\csname month\romannumeral#1\bbl@calendar name\endcsname}}
\newcommand\BabelDatey[1]{{\number#1}}
\newcommand\BabelDateyy[1]{{\ifnum#1<10 0\fi\number#1 \else\ifnum#1<100 \number#1 \else\ifnum#1<1000 \expandafter\@gobble\number#1 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 \else\bbl@error{Currently two-digit years are restricted to the\range 0-9999.} \fi\fi\fi\fi}}
\newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
\def\bbl@replace@finish@iii#1{\bbl@exp{\def\#1####1####2####3{\the\toks@}}}
\def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
\def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
Transforms.

\let\bbl@release@transforms@empty
\namedef{bbl@inikv@transforms.prehyphenation}{\bbl@transforms\babelprehyphenation}
\namedef{bbl@inikv@transforms.posthyphenation}{\bbl@transforms\babelposthyphenation}
\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\\
\directlua{
str = [[#2]]
str = str:gsub('%.%d+%.%d+$', '')
tex.print([[\def\string\babeltempa{\}
[[ .. str .. [[]])
}\] &
\bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&
\ife\in@{,0$}{#2$}&
\ife\in@{&
\g@addto@macro\bbl@release@transforms{&
\relax\bbl@transforms@aux#1{\languagename}{#3}}&
\else
\g@addto@macro\bbl@release@transforms{, {#3}}&
\fi
\fi
\fi}
\endgroup

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}{}%}{Script=\bbl@cs{sname@#1}}%
\bbl@csarg\let{lsys@#1}\@empty
\bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
\bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
\bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}%
\ifcase\bbl@engine\or\or
\bbl@ifunset{bbl@prehc@#1}{%}
\let\bbl@xenohyph\bbl@xenohyph@d
\ifx\AtBeginDocument\@notprerr
\expandafter\@secondoftwo % to execute right now
\AtBeginDocument{%
\bbl@patchfont{\bbl@xenohyph}%
\expandafter\selectlanguage\expandafter{\languagename}}%
\fi
\fi
\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{sname@#1}}%
\ifcase\bbl@engine\or\or
\bbl@ifunset{bbl@prehc@#1}{%}
\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}{%}
\bbl@xxenohyph\undefined
\let\bbl@xenohyph\bbl@xenohyph@d
\ifx\AtBeginDocument\@notprerr
\expandafter\@secondoftwo % to execute right now
\AtBeginDocument{%
\expandafter\selectlanguage\expandafter{\languagename}}%
\fi
\fi
\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}%
\def\bbl@xxenohyph\undefined
\let\bbl@xenohyph\bbl@xenohyph@d
\bbl@ifset{bbl@prehc@\languagename}{%}
\def\bbl@xenohyph\undefined
\let\bbl@xenohyph\bbl@xenohyph@d
135
The following ini reader ignores everything but the identification section. It is called when a font is defined (i.e., when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy .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 a special case, for a fixed form (see babel-he.ini, for example).

The information in the identification section can be useful, so the following macro just exposes it with a user command.
More general, but non-expandable, is \getlocaleproperty. To inspect every possible loaded ini, we define \LocaleForEach, where \bbl@ini@loaded is a comma-separated list of locales, built by \bbl@readini.

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@cs{ADJ@##1@##2}}%
  \let\bbl@ini@loaded\@empty
  \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

% 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@cs{ADJ@##1@##2}}%
  \let\bbl@ini@loaded\@empty
  \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

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@cs{ADJ@##1@##2}}%
  \let\bbl@ini@loaded\@empty
  \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

% 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@cs{ADJ@##1@##2}}%
  \let\bbl@ini@loaded\@empty
  \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

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@cs{ADJ@##1@##2}}%
  \let\bbl@ini@loaded\@empty
  \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
\ifvmode
  \ifnum\currentgrouplevel=\z@
    \directlua{ Babel.#2 }%
  \fi
\fi
\expandafter\expandafter\expandafter\@gobble
\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}}
\@namedef{bbl@ADJ@linebreak.tabular@on}{\bbl@adjust@lua{\let\@tabular\bbl@NL@@tabular}}
\@namedef{bbl@ADJ@linebreak.tabular@off}{\bbl@adjust@lua{\let\@tabular\bbl@OL@@tabular}}
\@namedef{bbl@ADJ@linebreak.tabular@on}{\bbl@adjust@lua{\let\@tabular\bbl@NL@@tabular}}
\@namedef{bbl@ADJ@linebreak.tabular@off}{\bbl@adjust@lua{\let\@tabular\bbl@OL@@tabular}}
\@namedef{bbl@ADJ@linebreak.lists@on}{\bbl@adjust@lua{\let\list\bbl@NL@list}}
\@namedef{bbl@ADJ@linebreak.lists@off}{\bbl@adjust@lua{\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}
\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}
As the final task, load the code for lua. TODO: use babel name, override

As the final task, load the code for lua. TODO: use babel name, override

\ifx\directlua@undefined\else
\ifx\bbl@luapatterns@undefined\else
\input luababel.def
\fi
\fi

Continue with \LaTeX.

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.

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

\newl@bel 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.

```
\bbl@trace{Cross referencing macros}
\ifx\bbl@opt@safe\@empty\else
\def\@newl@bel#1#2#3{%
\@safe@activestrue
\bbl@ifunset{#1@#2}{}
\relax
\gdef\@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 \LaTeX macro used to test if the labels that have been written on the .aux file have changed. It is called by the \enddocument macro.

```
\CheckCommand*\@testdef[3]{%
\def\reserved@a{#3}%
\expandafter\ifx\csname#1@#2\endcsname\reserved@a\else
\relax
\gdef\@multiplelabels{\@latex@warning@no@line{There were multiply-defined labels}}%
\@latex@warning@no@line{Label `#2' multiply defined}}%
\global\@namedef{#1@#2}[#3]{%}
```

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

```
\def\@testdef#1#2#3{% TODO. With @samestring?
\@safe@activestrue
\expandafter\let\expandafter\bbl@tempa\csname #1#2\endcsname
\def\bbl@tempb{#3}%
\@safe@activesfalse
\expandafter\ifx\csname#1@#2\endcsname\bbl@tempb\relax
\else
\edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
\fi
\edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
\ifx\bbl@tempa\bbl@tempb
\@tempswarue
\fi
```

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.

\begin{verbatim}
\bbl@xin@{R}\bbl@opt@safe
\ifin@
\bbl@redefinerobust\ref#1{%
  \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
\bbl@redefinerobust\pageref#1{%
  \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
\else
\let\org@ref\ref
\let\org@pageref\pageref
\fi
\end{verbatim}

The macro used to cite from a bibliography, \cite, uses an internal macro, \@citex. It is this internal macro that picks up the argument(s), so we redefine this internal macro and leave \cite alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the second argument.

\begin{verbatim}
\bbl@xin@{B}\bbl@opt@safe
\ifin@
\bbl@redefine\@citex[#1]#2{%
  \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
  \org@@citex[#1]{\@tempa}}%
\else
  \def\@citex[#1]{\@tempa}{}
\fi
\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}
\AtBeginDocument{%
  \@ifpackageloaded{natbib}{%
    \def\@citex[#1]#2{%
      \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
      \org@@citex[#1]{{\@tempa}}%
    }}%
\else
  \def\@citex[#1]#2{%
    \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
    \org@@citex[#1]{{\@tempa}}%
  }}%
\AtBeginDocument{%
  \@ifpackageloaded{cite}{%
    \def\@citex[#1]#2{%
      \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
      \org@@citex[#1]{{\@tempa}}%
    }}%
\else
  \def\@citex[#1]#2{%
    \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
    \org@@citex[#1]{{\@tempa}}%
  }}%
\end{verbatim}

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

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

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

\begin{verbatim}
\bbl@redefine\nocite#1{%
  \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
\end{verbatim}

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

142
\@bibitem One of the two internal \LaTeX macros called by \@bibitem that write the citation label on the .aux file.
\markright Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \markright and \markboth somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used. We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.
\markboth The definition of \markboth is equivalent to that of \markright, except that we need two token registers. The documentclasses 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.)

\@mkboth
\bbl@restore@actives
\the\toks@}}}}%
\org@markright{{\protect\bbl@restore@actives\the\toks@}}}}%
\markboth
\@mkboth
\bbl@restore@actives
\the\toks@}}}}%

\ifx\@mkboth\markboth
\def\bbl@tempc{\let\@mkboth\markboth}
\else
\def\bbl@tempc{}
\fi
\bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
\markboth#1#2{%\protected@edef\bbl@tempb##1{%\protect\foreignlanguage{\languagename}{\protect\bbl@restore@actives##1}}%\bbl@ifblank{#1}{}\toks@{}}%\bbl@ifblank{#2}{}\@temptokena{}\@temptokena\expandafter{\bbl@tempb{#2}}%\bbl@exp{\protect\bbl@markboth{\the\toks@}{\the@temptokena}}}
\bbl@tempc
\fi % end ifbbl@single, end IfBabelLayout

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:

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

In order for this to work the argument of \isodd needs to be fully expandable. With the above redefinition of \pageref it is not in the case of this example. To overcome that, we add some code to the definition of \ifthenelse to make things work.

We want to revert the definition of \pageref and \ref to their original definition for the first argument of \ifthenelse, so we first need to store their current meanings.

Then we can set the \@safe@actives switch and call the original \ifthenelse. In order to be able to use shorthands in the second and third arguments of \ifthenelse the resetting of the switch and the definition of \pageref happens inside those arguments.
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`.

\begin{verbatim}
\AtBeginDocument{\IfPackageLoaded{varioref}{\bbl@redefine\@@vpageref#1[#2]{\@safe@activetrue\org\@@@vpageref{#1}[#2]{#3}}\@safe@activesfalse}{\\bbl@redefine\vrefpagenum#1#2{\@safe@activetrue\org@vrefpagenum{#1}{#2}}\@safe@activesfalse}}
\end{verbatim}

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.

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

9.3.3 hhline

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

\begin{verbatim}
\AtEndOfPackage{\AtBeginDocument{\IfPackageLoaded{hhline}\{\expandafter\ifx\csname normal@char\string::\endcsname\relax\protect@edef\@tempa{\org@ref[#1]}\expandafter\MakeUppercase\@tempa\}\else\makeatletter\\def\@currname{hhline}\input{hhline.sty}\makeatother\fi\}}}
\end{verbatim}
Deprecated. Use the tools provided by \TeX. 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}\[
\the\year/\two@digits{\the\month}/\two@digits{\the\day}\space generated font description file]^^J
\string\DeclareFontFamily{#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
\string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
\string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
\string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
\string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
}%
\closeout15
}
}\@onlypreamble\substitutefontfamily

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 \ensureasci. The default ASCII encoding is set, too (in reverse order): the “main” encoding (when the document begins), the last loaded, or OT1.

\ensureasci
\bbl@trace{Encoding and fonts}
\newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
\newcommand\BabelNonText{TS1,T3,TS3}
\let\org@TeX\TeX
\let\org@LaTeX\LaTeX
\let\ensureascii\@firstofone
\AtBeginDocument{%
\def\@elt#1{,#1,}%
\edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
\let\@elt\relax
\let\bbl@tempb\@empty
\def\bbl@tempc{OT1}%
\bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
\bbl@ifunset{T@#1}{\def\bbl@tempb{#1}}% Store last non-ascii
\bbl@foreach\bbl@tempa{% Pass
\bbl@xin@{#1}{\BabelNonASCII}\ifin@%
\bbl@xin@{#1}{\BabelNonText}\ifin@% Store last ascii
\fi
\fi}%
\ifx\bbl@tempb\@empty\else
\bbl@xin@{\cf@encoding,}{\BabelNonASCII,\BabelNonText,}%
\ifin@\else
\bbl@xin@{,}{\BabelNonASCII,\BabelNonText}%
\fi
\fi}
Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at \begin{document}, which Latin fontencoding to use.

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

\AtEndOfPackage{\edef\latinencoding{\cf@encoding}}

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.

\AtBeginDocument{\if@ifpackageloaded{fontspec}{\xdef\latinencoding{\if\UTFencname@undefined\EU\case\bbl@engine\or2\or1\fi}\else\UTFencname\fi}}

\latinencoding Then we can define the command \latinencoding which is a declarative switch to a Latin font-encoding. Usage of this macro is deprecated.

\DeclareRobustCommand{\latinencoding}{\fontencoding{\latinencoding}\selectfont}\def\encodingdefault{\latinencoding}

\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.

\ifx\@undefined\DeclareTextFontCommand
\DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latinencoding #1}}\else
\DeclareTextFontCommand{\textlatin}[1]{\latinencoding #1}\fi

For several functions, we need to execute some code with \selectfont. With \LaTeX{} 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).

\bbl@iffformatlater{2021-06-01}
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 rlbabel did), and by introducing a “middle layer” just below the user interface (sectioning, footnotes).

- pdftex provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- xetex is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour \TeX\ grouping.
- luatex can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As Lua\TeX\-ja shows, vertical typesetting is possible, too.
% TODO? Separate:
\ifnum\bbl@bidimode=\@ne
\let\bbl@beforeforeign\leavevmode
\ifodd\bbl@engine
\newattribute\bbl@attr@dir
\directlua{ Babel.attr_dir = luatexbase.registernumber'\bbl@attr@dir' }
\bbl@exp{\output{\bodydir\pagedir\the\output}}
\fi
\AtEndOfPackage{%
\EnableBabelHook{babel-bidi}%
\ifodd\bbl@engine\else
\bbl@xebidipar
\fi}
\fi

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

\bbl@trace{Macros to switch the text direction}
\def\bbl@alscripts{\Arabic,\Syriac,\Thaana,}
\def\bbl@rscripts{% TODO. Base on codes ??
,\ImperialAramaic,\Avestan,\Cypriot,\Hatran,\Hebrew,%
\OldHungarian,\OldHungarian,\Lydian,\Manaeæan,\Manichaean,%
\Manichaean,\MeroiticCursive,\Meroitic,\OldNorthArabian,%
\Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
\PsalterPahlavi,\Pheonic,\InscriptionalParthian,\Samaritan,%
\OldSouthArabian,}%
\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
\else%
\global\bbl@csarg\chardef{wdir@#1}\z@%
\fi
\fi
\ifodd\bbl@engine
\bbl@csarg\ifcase{wdir@#1}%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
\or%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
\or%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
\fi
\fi
\def\bbl@switchdir{%
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
\bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}
\def\bbl@setdirs#1{%
\ifcase\bbl@select@type %
\bbl@bodydir{#1}%
\bbl@pardir{#1}%
\fi
\ifodd\bbl@engine
\bbl@csarg\ifcase{wdir@#1}%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
\or%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
\or%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
\fi
\fi
\def\bbl@switchdir{%
\bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
\bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
\bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}
\def\bbl@setdirs#1{%
\ifcase\bbl@select@type %
\bbl@bodydir{#1}%
\bbl@pardir{#1}%
\fi
\ifodd\bbl@engine
\bbl@csarg\ifcase{wdir@#1}%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
\or%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
\or%
\directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
\fi
\fi
\ifcase\bbl@select@type % TODD - math
\bbl@bodydir{#1}%
\bbl@pardir{#1}%
\fi
\bbl@textdir{#1}
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).
A tool for weak L (mainly digits). We also disable warnings with hyperref.

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.

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.
in \texttt{bbl@language@opts} are assumed to be languages (note this list also contains the language given with \texttt{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 \texttt{ldf} exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.
If a main language has been set, store it for the third pass.

And we are done, because all options for this pass have 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):

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.
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.

10 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

(+kernel)
\let\bbl@onlyswitch=\empty
\input babel.def
\let\bbl@onlyswitch=\undefined
(+patterns)
11 Loading hyphenation patterns

The following code is meant to be read by \iniTeX because it should instruct \TeX to read hyphenation patterns. To this end the docstrip option \texttt{patterns} is used to include this code in the file \texttt{hyphen.cfg}. Code is written with lower level macros.

\begin{verbatim}
\ProvidesFile{hyphen.cfg}[\langle\langle date\rangle\rangle \langle\langle version\rangle\rangle Babel hyphens]
\xdef\bbl@format{\jobname}
\def\bbl@version{\langle\langle version\rangle\rangle}
\def\bbl@date{\langle\langle date\rangle\rangle}
\ifx\AtBeginDocument\@undefined
\def\@empty{}
\fi

\Definecoreswitchingmacros
\process@line
\Eachlineinthefile\texttt{language.dat}isprocessedby \process@line after it is read. The first thing this macro does is to check whether the line starts with =. When the first token of a line is an =, the macro \process@synonym is called; otherwise the macro \process@language will continue.

\process@synonym
\Thismacrotakescareofthelineswhichstartwithan =. It needs an empty token register to begin with. \bbl@languages is also set to empty.

\process@language
\Themacro \process@language is used to process a non-empty line from the \texttt{configuration file}. It has three arguments, each delimited by white space. The first argument is the \texttt{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.

\end{verbatim}

The first thing to do is call \texttt{\addlanguage} to allocate a pattern register and to make that register \texttt{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 \texttt{language.dat} by adding for instance \texttt{:T1} to the name of the language. The macro \texttt{\bbl@get@enc} extracts the font encoding from the language name and stores it in \texttt{\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\{(language-name)\}{\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}\
% > luatex
\bbl@get@enc#1::
\begingroup
\lefthyphenmin\m@ne
\bbl@hook@loadpatterns{#2}\
% > luatex
\ifnum\lefthyphenmin=\m@ne
\else
\expandafter\xdef\csname #1hyphenmins\endcsname{%
\the\lefthyphenmin\the\righthyphenmin}\
\fi
\endgroup
\def\bbl@tempa{#3}\
\ifx\bbl@tempa\@empty\else
\bbl@hook@loadexceptions{#3}\
% > luatex
\fi
\let\bbl@elt\relax
\edef\bbl@languages{%
\bbl@languages\bbl@elt\{(language-name)\}{\langle number\rangle}{\langle patterns-file\rangle}{\langle exceptions-file\rangle}\
\ifnum\the\language=\z@
\expandafter\ifx\csname #1hyphenmins\endcsname\relax
\set@hyphenmins\tw@\thr@@\relax
\else
\expandafter\expandafter\expandafter\set@hyphenmins\csname #1hyphenmins\endcsname\relax
\fi
\expandafter\expandafter\expandafter\set@hyphenmins\csname #1hyphenmins\endcsname\relax
\fi
\the\toks@\toks@}%
\fi}
```

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.

```
\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.
\def\bbl@hook@everylanguage#1{}
\def\bbl@hook@loadpatterns#1\{\input #1\relax\}
\let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
\def\bbl@hook@loadkernel#1\{%
  \def\addlanguage\{\csname newlanguage\endcsname\}%
  \def\adddialect#1#2\%{
    \global\chardef#1#2\relax
    \wlog{\string#1 = a dialect from \string#2}\%
  }
  \def\iflanguage#1\{%
    \expandafter\ifx\csname l@#1\endcsname\relax
    \@nolanerr{#1}\%
    \else
    \expandafter\expandafter\expandafter\@firstoftwo
    \else
    \expandafter\expandafter\expandafter\@secondoftwo
    \fi
  \}
  \def\providehyphenmins#1#2\%
  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
  \@namedef{#1hyphenmins}{#2}\%
  \fi\%
  \def\set@hyphenmins#1#2\%
  \lefthyphenmin#1\relax
  \righthyphenmin#2\relax\%
  \def\selectlanguage\%
  \errhelp{Selecting a language requires a package supporting it}\%
  \errmessage{Not loaded}\%
  \let\foreignlanguage\selectlanguage
  \let\otherlanguage\selectlanguage
  \let\otherlanguage*\selectlanguage
  \let\languagetext\selectlanguage
  \let\textlanguage\selectlocale
  \let\textlocale\selectlocale
  \begingroup
  \def\AddBabelHook#1#2\%
  \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
    \def\next\{\toks1\%
  \else
    \def\next\{\expandafter\gdef\csname bbl@hook@#2\endcsname\%
  \fi
  \next\%
  \expandafter\ifx\directlua\@undefined
    \expandafter\ifx\XeTeXinputencoding\@undefined\else
      \input xebabel.def
    \fi
  \else
    \input luababel.def
  \fi
  \openin1 = babel-\bbl@format.cfg
  \ifeof1

158
The configuration file can now be opened for reading.

```tex
\openin1 = language.dat
See if the file exists, if not, use the default hyphenation file hyphen.tex. The user will be informed about this.
```

```tex
\def\languagename{english}\
\ifeof1
\message{I couldn’t find the file language.dat, space I will try the file hyphen.tex}\n\input hyphen.tex\relax\n\chardef\l@english\z@
\else
Pattern registers are allocated using count register \last@language. Its initial value is 0. The definition of the macro \newlanguage is such that it first increments the count register and then defines the language. In order to have the first patterns loaded in pattern register number 0 we initialize \last@language with the value $-1$.
```

```tex\last@language\m@ne
We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.
```

```tex\loop
\endlinechar\m@ne\n\read1 to \bbl@line\n\endlinechar`\^^M
If the file has reached its end, exit from the loop here. If not, empty lines are skipped. Add 3 space characters to the end of \bbl@line. This is needed to be able to recognize the arguments of \process@line later on. The default language should be the very first one.
```

```tex\if T\ifeof1\fi T\relax\n\ifx\bbl@line\@empty\else\n\edef\bbl@line{\bbl@line\space\space\space}\n\expandafter\process@line\bbl@line\relax\n\fi\n\repeat
Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns, and close the configuration file.
```

```tex\begingroup\n\def\bbl@elt#1#2#3#4{%\n\global\language=#2\relax\n\gdef\languagename{#1}%\n\def\bbl@elt##1##2##3##4{}}\n\bbl@languages\n\endgroup\nWe add a message about the fact that babel is loaded in the format and with which language patterns to the \everyjob register.
```

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

\let\bbl@line\@undefined
\let\process@line\@undefined
\let\process@synonym\@undefined
\let\process@language\@undefined
\let\bbl@get@enc\@undefined
\let\bbl@hyph@enc\@undefined
\let\bbl@tempa\@undefined
\let\bbl@hook@loadkernel\@undefined
\let\bbl@hook@everylanguage\@undefined
\let\bbl@hook@loadpatterns\@undefined
\let\bbl@hook@loadexceptions\@undefined
\langle /patterns \rangle

Here the code for init\TeX{} ends.

## 12 Font handling with fontspec

Add the bidi handler just before luatoload, 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].

\langle More package options \rangle ≡
\chardef\bbl@bidimode\z@
\DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
\DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101}
\DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102}
\DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201}
\DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203}
\langle More package options \rangle

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

At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to \texttt{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.

\langle More package options \rangle ≡
\bbl@trace{Font handling with fontspec}
\ifx\ExplSyntaxOn\@undefined\else
\ExplSyntaxOn
\catcode\ =10
\def\bbl@loadfontspec{\usepackage{fontspec}% TODO. Apply patch always
 expandafter
 \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{Font \l_fontspec_fontname_tl is using the\%
 default features for language \texttt{##1}.\%
 That’s usually fine, because many languages\%
 require no specific features, but if the output is\%
 not as expected, consider selecting another font.)\%
 expandafter
 \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{Font \l_fontspec_fontname_tl is using the\%
}
default features for script '#2'.
That's not always wrong, but if the output is not as expected, consider selecting another font.}

\ExplSyntaxOff
\fi
\onlypreamble\babelfont
\newcommand\babelfont[2][]{1=langs/scripts 2=fam
\bbl@foreach{#1}{\expandafter\ifx\csname date##1\endcsname\relax
\IfFileExists{babel-##1.tex}{\babelprovide{##1}}{}\fi}
\edef\bbl@tempa{#1}
\def\bbl@tempb{#2}% Used by \bbl@bblfont
\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}}{}
% For the default font, just in case:
\bbl@ifunset{bbl@lsys@languagename}{\bbl@provide@lsys{\languagename}}{}
\expandafter\bbl@ifblank\expandafter{\bbl@tempa}{}{\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
\bbl@csarg\edef{\bbl@tempb dflt@##1}{<>{#1}{#2}}% save bbl@rmdflt@
\bbl@exp%
  \let<br\bbl@tempb dflt@languagename>&<\bbl@tempb dflt@languagename>\%
  \bbl@font@set<br\bbl@tempb dflt@languagename><\bbl@tempb dflt@languagename>\%
  \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%
  \\\newcommand#1default{\}
  \\\DeclareRobustCommand\#1family{%
    \not@math@alphabet\#1family\relax
    \\prepare@family@series@update{#1}\#1default% TODO. Fails
    \fontfamily#1default\%
    % \UseHooks\#1family\%
    % % %
    \\\selectfont%
    \\DeclareTextFontCommand{\#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@exp%
  \\\newcommand\#1default{\}
  \\\DeclareRobustCommand\#1family{%
    \not@math@alphabet\#1family\relax
    \\prepare@family@series@update{#1}\#1default% TODO. Fails
    \fontfamily\#1default\%
    % \UseHooks\#1family\%
    % % %
    \\\selectfont%
    \\DeclareTextFontCommand{\#1}{\#1family}{}
  }
% Flag, to avoid dupl warns
\bbl@provide@lsys{\languagename}
% The current font is not a babel standard family:
% #1%
\fontname\font\%
There is nothing intrinsically wrong with this warning, and you can ignore it altogether if you do not need these families. But if they are used in the document, you should be...
aware 'babel' will no set Script and Language for them, so\%
you may consider defining a new family with \string\babelfont.\%
See the manual for further details about \string\babelfont.\%
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@foreach\bbl@font@fams{%
\bbl@ifunset{bbl@##1dflt@\languagename}{% (1) language?
    \{\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}% (2) from script?
    \{\bbl@ifunset{bbl@##1dflt@}% 2=F - (3) from generic?
    {}% 123=F - nothing!
        \{\bbl@exp{% 3=T - from generic
            \global\let\bbl@##1dflt@\languagename>\bbl@##1dflt@}}}%
    \{\bbl@exp{% 2=T - from script
        \global\let<\bbl@##1dflt@\languagename>\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@*\bbl@tempa>% the main part!
            \bbl@add\originalTeX{%
                \bbl@font@rst{\bbl@cl{##1dflt}}%
                    \#1default\<\#1family>{##1}%
            \}<\bbl@##1dflt@*\bbl@tempa}>}}%
    \bbl@ifrestoring{\{\bbl@tempa}}%}
}\ifx\f@family\@undefined\else % if latex
    \ifcase\bbl@engine % if pdftex
        \let\bbl@ckeckstdfonts\relax
    \else
        \def\bbl@ckeckstdfonts{%
            \begingroup
                \\bbl@font@rst{\bbl@cl{##1dflt}}%
                    \#1default\<\#1family>{##1}%
            \}<\bbl@##1dflt@*\bbl@tempa>}}%
    \ifx\bbl@tempa\@empty\else
        \bbl@infowarn{The following font families will use the default\%
            settings for all or some languages:\%
            \bbl@tempa}
        \bbl@infowarn{There is nothing intrinsically wrong with it, but\%
            'babel' will no set Script and Language, which could\%

The following is executed at the beginning of the aux file or the document to warn about fonts not
defined with \babelfont.
be relevant in some languages. If your document uses these families, consider redefining them with \string\babelfont. Reported%
\endgroup\fi
\fi
\fi

Now the macros defining the font with fontspec.
When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.
\def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \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}{\bfseries}{}
\\let\\@tempa\relax}
{}}}
\def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
\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}}%
\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}%
\\<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}%
\\\renewfontfamily\#4%
{[\bbl@cl{lsys},#2]}{#3} % ie \bbl@exp{..}{#3}
\begingroup
\newcommand\babelfSstore[2][2]{%}

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont. The old tentative way. Short and preverved 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{\ife\bbl@fn@footnote\@undefined\let\bbl@fn@footnote\footnote\fi\bbl@ifblank{#2}{#1\bbl@footnote@o{#2}{#3}{#4}{\ignorespaces#5}{#6}}{#1\bbl@exp{#2}{#3}{#4}}\@namedef{\bbl@stripslash#1text}{#5}}
\end{verbatim}

Now, the code.
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 \texttt{txt babel} as a shorthand for \texttt{tex--xet babel}, 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 ctablestack). FIX - This isn’t true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated. This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).
\count@\z@ % 0=start, 1=0th, 2=normal
\def\bbl@process@line#1#2 #3 #4 {%\count@=1\%
 \bbl@process@synonym(#2)\%
 \else
 \bbl@process@language{#1}{#2}{#3}{#4}\%
 \fi
 \ignorespaces}
\def\bbl@manylang{%\fnnum\bbl@last>1\one
 \bbl@info{Non-standard hyphenation setup}\%
 \fi
 \let\bbl@manylang\relax}
\def\bbl@process@language#1#2{\ifcase\count@
 \ifundefined{zth@#1}{\count@@tw@}{\count@\@ne}
 \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}
\if\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\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>1\z@\else
 \noexpand\bbl@elt{#1}{#2}{#3}{#4}\%
 \fi}%
 \xdef\bbl@languages{\bbl@languages\bbl@languages}%
 \fi
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

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

if\n
\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}%\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@tempa{utf8}\def\bbl@tempb{#1}%\ifx\bbl@tempa\bbl@tempb\empty\else % if not a synonymous\def\bbl@tempc{{#3}{#4}}%\fi\fi}}%\fi\fi\fi\fi}\fi\fi\fi\fi

\directlua{Babel.attr_locale = luatexbase.registernumber'\bbl@attr@locale'}
\AddBabelHook{luatex}{beforeextras}{%\setattribute\bbl@attr@locale\localeid}
\AddBabelHook{luatex}{stopcommands}{%\luabbl@stop}%\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@tempa{utf8}\def\bbl@tempb{#1}%\ifx\bbl@tempa\bbl@tempb\empty\else % if not a synonymous\def\bbl@tempc{{#3}{#4}}%\fi\fi}}%\fi\fi\fi\fi

172
This macro adds patterns. Two macros are used to store them: \texttt{\bbl@patterns@} for the global ones and \texttt{\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 \bblposthyphenation. Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

\% TODO - to a lua file
\directlua{
Babel = Babel or {}% 5034
Babel.linebreaking = Babel.linebreaking or {}% 5035
Babel.linebreaking.before = {}% 5036
Babel.linebreaking.after = {}% 5037
Babel.locale = {} % Free to use, indexed by \localeid% 5038
function Babel.linebreaking.add_before(func)% 5039
tex.print([^\noexpand\csname bbl@luahyphenate\endcsname\]])% 5040
table.insert(Babel.linebreaking.before, func)% 5041
end% 5042
function Babel.linebreaking.add_after(func)% 5043
tex.print([^\noexpand\csname bbl@luahyphenate\endcsname\]])% 5044
table.insert(Babel.linebreaking.after, func)% 5045
end% 5046
}
\def\bbl@intraspace#1 #2 #3\@@ {% 5047
\directlua{
Babel = Babel or {}% 5048
Babel.intraspaces = Babel.intraspaces or {}% 5049
Babel.intraspaces[^\csname bbl@sbcp@\languagename\endcsname\] = % 5050
\{b = #1, p = #2, m = #3\}% 5051
Babel.locale_props[^\the\localeid\].intraspace = % 5052
\{b = #1, p = #2, m = #3\}% 5053
}% 5054
\def\bbl@intrapenalty#1\@@{% 5055
\directlua{
Babel = Babel or {}% 5056
Babel.intrapenalties = Babel.intrapenalties or {}% 5057
Babel.intrapenalties[^\csname bbl@sbcp@\languagename\endcsname\] = #1% 5058
Babel.locale_props[^\the\localeid\].intrapenalty = #1% 5059
}% 5060
\begingroup
\catcode`\%=12% 5061
\catcode`^=14% 5062
\catcode`\'=12% 5063
\catcode`\^-12% 5064
\gdef\bbl@seaintraspace:^% 5065
\let\bbl@seaintraspace\relax% 5066
\beginlualatex% 5067
Babel = Babel or {}% 5068
Babel.sea_enabled = true% 5069
Babel.sea_ranges = Babel.sea_ranges or {}% 5070
function Babel.set_chranges (script, chrng)% 5071
local c = 0% 5072
for s, e in string.gmatch(chrng..\', \', \',\%(.\%(.\%)s\)\)\) do% 5073
Babel.sea_ranges[script..c] =\{tonumber(s,16), tonumber(e,16)\}% 5074
c = c + 1% 5075
end% 5076
end% 5077
function Babel.sea_disc_to_space (head)% 5078
local sea_ranges = Babel.sea_ranges% 5079
local last_char = nil% 5080

174
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 intraspace = Babel.intraspaces[lg]
                local intrapenalty = Babel.intrapenalties[lg]
                local n
                if intrapenalty ~= 0 then
                    n = node.new(14, 0) % penalty
                    n.penalty = intrapenalty
                    node.insert_before(head, item, n)
                end
                n = node.new(12, 13) % (glue, spaceskip)
                node.setglue(n, intraspace.b * quad,
                             intraspace.p * quad,
                             intraspace.m * quad)
                node.insert_before(head, item, n)
                node.remove(head, item)
            end
        end
    end
    last_class = nil
    last_lang = nil
end

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@cjkintraspaces%
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

end

lang.hyphenate(head)

\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
    end
}
5195  end
5196  if Babel.cjk_enabled then
5197      Babel.cjk_linebreak(head)
5198  end
5199  lang.hyphenate(head)
5200  if Babel.linebreaking.after then
5201      for k, func in ipairs(Babel.linebreaking.after) do
5202          func(head)
5203      end
5204  end
5205  if Babel.sea_enabled then
5206      Babel.sea_disc_to_space(head)
5207  end,
5208  end,
5209  'Babel.hyphenate')
5210  }
5211 }
5212 \endgroup
5213 \def\bbl@provide@intraspace{%
5214  \bbl@ifunset{bbl@intsp@\languagename}{%}%
5215  \{\expandafter\ifx\csname bbl@intsp@\languagename\endcsname@empty\else
5216    \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5217    \ifin@ % cjk
5218    \bbl@cjkintraspace
5219    \directlua{
5220         Babel = Babel or {}
5221         Babel.locale_props = Babel.locale_props or {}
5222         Babel.locale_props[\the\localeid].linebreak = 'c'
5223     }%
5224    \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\@@}%
5225    \ifx\bbl@KVP@intrapenalty\@nil
5226       \bbl@intrapenalty0@@
5227    \fi
5228  \else % sea
5229    \bbl@seaintraspace
5230    \bbl@exp{\\bbl@intraspace\bbl@cl{intsp}\@@}%
5231    \directlua{
5232         Babel = Babel or {}
5233         Babel.locale_props = Babel.locale_props or {}
5234         Babel.locale_props[\the\localeid].linebreak = 'c'
5235     }%
5236    \ifx\bbl@KVP@intrapenalty\@nil
5237       \bbl@intrapenalty0@@
5238    \fi
5239  \else
5240    \fi
5241  \fi
5242  \ifx\bbl@KVP@intrapenalty\@nil\else
5243    \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty@@
5244  \fi}%

13.6 Arabic justification
5245  \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5246  \def\bblar@chars{%
5247  0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5248  0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5249  0640,0641,0642,0643,0644,0645,0646,0647,0649}%
5250  \def\bblar@elongated{%
\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{
\quad Babel.arabic.elong_map = Babel.arabic.elong_map or {};
Babel.arabic.elong_map[\the\localeid] = {};
luatexbase.add_to_callback('post_linebreak_filter',
\quad Babel.arabic.justify, 'Babel.arabic.justify')
\quad luatexbase.add_to_callback('hpack_filter',
\quad Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox'
)}%
% Save both node lists to make replacement. TODO. Save also widths to
% make computations
\def\bblar@fetchjalt#1#2#3#4{%
\bbl@foreach{#1}{
\bbl@ifunset{bblar@JE@##1}{
\bbl@foreach{#1}{
\bbl@ifunset{bblar@JE@##1}{
\directlua{
local last = nil
for item in node.traverse(tex.box[0].head) do
if item.id == node.id'glyph' and item.char > 0x600 and
not (item.char == 0x200D) then
last = item
end
end

Babel.arabic.#3['##1#4'] = last.char
}}}
% Brute force. No rules at all, yet. The ideal: look at jalt table. And
% perhaps other tables (falt?, cswh?). What about kaf? And diacritic
% positioning?
%}
\gdef\bbl@parsejalt{%
\ifx\addfontfeature\@undefined\else
\bbl@xin{/e}{/\bbl@cl{lnbrk}}%
\ifin@
\directlua{
if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
end
\quad tex.print([|\string\csname\space bbl@parsejalti\endcsname|])
}
\fi
\fi
\gdef\bbl@parsejalti{%
\begingroup
\let\bbl@parsejalt\relax % To avoid infinite loop
\edef\bbl@tempb\{ontid\font}
\bblar@nofswarn
\bbl@fetchjalt\bblar@elongated}{from}{%
\texttt{\bblar@fetchjalt\bblar@chars{\^\^\^\^064a}\{from\}{a}\% Alef maksura}\n\texttt{\bblar@fetchjalt\bblar@chars{\^\^\^\^0649}\{from\}{y}\% Yeh}\n\texttt{\addfontfeature{RawFeature=+jalt}}\%
\texttt{% \\namedef{bblar@JE@0643}{06AA}\% todo: catch medial kaf}\n\texttt{\bblar@fetchjalt\bblar@elongated\{dest\}\{}\%
\texttt{\bblar@fetchjalt\bblar@chars{\^\^\^\^064a}\{dest\}{a}}\%
\texttt{\bblar@fetchjalt\bblar@chars{\^\^\^\^0649}\{dest\}{y}}\%
\texttt{\directlua{\% for \texttt{k}, \texttt{v} in pairs(Babel.arabic.from) do}
\texttt{\quad if Babel.arabic.dest[\texttt{k}] and}
\texttt{\quad \quad not (Babel.arabic.from[\texttt{k}] == Babel.arabic.dest[\texttt{k}]) then}
\texttt{\quad \quad Babel.arabic.elong_map[\texttt{the\localeid}][\texttt{\bbl@tempb}]
\texttt{\quad \quad \quad [Babel.arabic.from[\texttt{k}]] = Babel.arabic.dest[\texttt{k}]}\n\texttt{\quad \quad end}
\texttt{\quad end}}\%
\texttt{\begingroup\% \catcode`#=11\texttt{\catcode`~'=11\directlua{\begin{verbatim}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'\texttt{hlist}', head) do\quad Babel.arabic.justify_hlist(head, line)\end\return head\end\function Babel.arabic.justify_hbox(head, gc, size, pack)\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'\texttt{glyph}'\local KASHIDA = Babel.attr_kashida\end{verbatim}}\endgroup}
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
    elong = {}
    % Stores elongated candidates of each line
    k_list = {}
    % And all letters with kashida
    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
        end % of node.traverse_id
    end % of n in node.traverse_id(GLYPH, line.head) do
    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 = elong[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
        % Substitute back if the line would be too wide and break:
        if width > goal then
            curr.char = oldchar
        end % of while (#elongs > 0 and width < goal) do
    end % of #elongs == 0 and #k_list == 0 then goto next_line end
break
end
% If continue, pop the just substituted node from the list:
table.remove(elongs, x)
end
% == Tatwil ==
if #k_list == 0 then goto next_line end

width = node.dimensions(line.head)  % The 'natural' width
k_curr = #k_list
wt_pos = 1

while width < goal do
    subst_done = true
    k_item = k_list[k_curr].node
    if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
d = node.copy(k_item)
d.char = 0x0640
line.head, new = node.insert_after(line.head, k_item, d)
width_new = node.dimensions(line.head)
if width > goal or width == width_new then
    node.remove(line.head, new) % Better compute before
    break
end
    width = width_new
end
if k_curr == 1 then
    k_curr = #k_list
    wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
else
    k_curr = k_curr - 1
end
end

::next_line::

% Must take into account marks and ins, see luatex manual.
% Have to be executed only if there are changes. Investigate
% what's going on exactly.
if subst_done and not gc then
d = node.hpack(line.head, full, 'exactly')
d.shift = shift
node.insert_before(head, line, d)
node.remove(head, line)
end
end % if process line
end

\endgroup
\fi\fi % Arabic just block

13.7 Common stuff

\AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
\AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
\DisableBabelHook{babel-fontspec}
⟨⟨
Fontselection
⟩⟩

181
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 traverses the node list to carry out the replacements. The table `loc_to_scr` gets the locale form as script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionary are handled in a special way.

```
\directlua{
Babel.script_blocks = {
['dflt'] = {},
['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
{0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
['Armn'] = {{0x0550, 0x058F}},
['Beng'] = {{0x0980, 0x09FF}},
['Copt'] = {{0x038E, 0x03EF}, {0x2C80, 0x2CFF}, {0x012E0, 0x01EF},
{0x0260, 0x02FF}, {0xA640, 0xA69F}},
['Deva'] = {{0x0900, 0x097F}, {0x0A8E0, 0xA8FF}},
['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
['Hebr'] = {{0x0590, 0x05FF}},
['Jpan'] = {{0x0300, 0x033F}, {0x0300, 0x030F},
{0x04E0, 0x09AF}, {0xFF00, 0xFFEF}},
['Knda'] = {{0x0C80, 0x0CFF}},
['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
{0xA960, 0xA97F}, {0xAC00, 0x1BFF}},
['Laoo'] = {{0x0E80, 0x0EFF}},
['Latn'] = {{0x0300, 0x03FF}, {0x0300, 0x03FF}, {0x0100, 0x01FF},
{0x0260, 0x026F}, {0x0260, 0x027F}, {0x0270, 0x02FF}, {0x0320, 0x037F}},
['Mahj'] = {{0x01150, 0x0117F}},
['Mlym'] = {{0x0D80, 0x0D7F}},
['Myrr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
['Orya'] = {{0x0B00, 0x0B7F}},
['Sinh'] = {{0x0D80, 0x0DFF}, {0x1111E0, 0x111FF}},
['SyrL'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
['Taml'] = {{0x0B80, 0x0BFF}},
['Telu'] = {{0x0C00, 0xC7F}},
['Tfng'] = {{0x0230, 0x2D7F}},
['Thai'] = {{0x0E00, 0x0E7F}},
['Tibf'] = {{0x0F00, 0x0FFF}},
['Vaii'] = {{0xA500, 0xA63F}},
['Yi'] = {{0xA900, 0xA48F}, {0xA490, 0xA4CF}}
}
```
Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl

function Babel.locale_map(head)
  if not Babel.locale_mapped then return head end
  local LOCALE = Babel.attr_locale
  local GLYPH = node.id('glyph')
  local inmath = false
  local toloc_save
  for item in node.traverse(head) do
    local toloc
    if not inmath and item.id == GLYPH then
      % Optimization: build a table with the chars found
      if Babel.chr_to_loc[item.char] then
        toloc = Babel.chr_to_loc[item.char]
      else
        for lc, maps in pairs(Babel.loc_to_scr) do
          for _, rg in pairs(maps) do
            if item.char >= rg[1] and item.char <= rg[2] then
              Babel.chr_to_loc[item.char] = lc
              toloc = lc
              break
            end
          end
        end
      end
      % 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
  end
return head
end
The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.

\newcommand\babelcharproperty[1]{
\count@=#1\relax
\ifvmode
\expandafter\bb@chprop
\else
\bb@error{\string\babelcharproperty\space can be used only in vertical mode (preamble or between paragraphs)}
{See the manual for further info}
\fi}
\newcommand\bb@chprop[3][3]{\the\count@}{%}
\def\bb@chprop@direction#1{\directlua{
Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
Babel.characters[\the\count@][‘d’] = ‘#1’
}}
\let\bb@chprop@bc\bb@chprop@direction
\def\bb@chprop@mirror#1{\directlua{
Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
Babel.characters[\the\count@][‘m’] = ‘\number#1’
}}
\let\bb@chprop@bmg\bb@chprop@mirror
\def\bb@chprop@linebreak#1{\directlua{
Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
Babel.cjk_characters[\the\count@][‘c’] = ‘#1’
}}
\let\bb@chprop@lb\bb@chprop@linebreak
\def\bb@chprop@locale#1{\directlua{
Babel.chr_to_loc = Babel.chr_to_loc or {}
Babel.chr_to_loc[\the\count@] = \bb@ifblank{#1}{-1000}{\the\bb@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{\bb@chprop@locale}
\def\bb@chprop@locale{\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 \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).

\begin{verbatim}
\begin{group}
\catcode`\=12
\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}% TODO. Ugly trick to preserve {}:
\bbl@replace\bbl@tempa{},{}%&
\expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
\bbl@ifsamestring{##1}{remove}{
\bbl@add@list\babeltempb{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\")
tex.print([[\string\babeltempa\{nil\} .. rep .. \{\}]]})}}&%
\endgroup}
\endverbatim
\begin{verbatim}
\end{group}
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\begin{group}
\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+)%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 } })
\endgroup}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\end{verbatim}
\begin{verbatim}
\% TODO. Copypaste pattern.
\gdef\babelprehyphenation#1#2#3{%&
\bbl@activateprehyphen
\begin{group}
\let\babeltempa=\bbl@add@list\babeltempb&%
\endgroup}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

185
13.9 Bidi

As a first step, add a handler for bidi and digits (and potentially other processes) just before \luaoftload is applied, which is loaded by default by \LaTeX. Just in case, consider the possibility it has
not been loaded.
5748 \def\bbl@activate@preotf{% 5749 \let\bbl@activate@preotf\relax % only once 5750 \directlua{
5751 Babel = Babel or {} 5752 % 5753 function Babel.pre_otfload_v(head) 5754 if Babel.numbers and Babel.digits_mapped then 5755 head = Babel.numbers(head) 5756 end 5757 if Babel.bidi_enabled then 5758 head = Babel.bidi(head, false, dir) 5759 end 5760 return head 5761 end 5762 % 5763 function Babel.pre_otfload_h(head, gc, sz, pt, dir) 5764 if Babel.numbers and Babel.digits_mapped then 5765 head = Babel.numbers(head) 5766 end 5767 if Babel.bidi_enabled then 5768 head = Babel.bidi(head, false, dir) 5769 end 5770 return head 5771 end 5772 % 5773 luatexbase.add_to_callback('pre_linebreak_filter', 5774 Babel.pre_otfload_v, 5775 'Babel.pre_otfload_v', 5776 luatexbase.priority_in_callback('pre_linebreak_filter', 5777 'luaotfload.node_processor') or nil) 5778 % 5779 luatexbase.add_to_callback('hpack_filter', 5780 Babel.pre_otfload_h, 5781 'Babel.pre_otfload_h', 5782 luatexbase.priority_in_callback('hpack_filter', 5783 'luaotfload.node_processor') or nil) 5784 })

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=.

5785 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200 5786 \let\bbl@beforeforeign\leavevmode 5787 \AtEndOfPackage{\EnableBabelHook{babel-bidi}} 5788 \RequirePackage{luatexbase} 5789 \bbl@activate@preotf 5790 \directlua{
5791 require('babel-data-bidi.lua') 5792 \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or 5793 require('babel-bidi-basic.lua') 5794 \or 5795 require('babel-bidi-basic-r.lua') 5796 \fi} 5797 % TODO - to locale_props, not as separate attribute 5798 \newattribute\bbl@attr@dir 5799 \directlua{ \bbl@attr_dir = luatexbase.registernumber'\bbl@attr@dir' } 5800 % TODO. I don't like it, hackish:
5801 \bbl@exp\{\output{\bodydir|\pagedir|the\output}}

187
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 bidi=basic, without having to patch almost any macro where text direction is relevant. \@hangfrom is useful in many contexts and it is redefined always with the layout option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by \bodydir), and when \parbox and \hangindent are involved. Fortunately, latest releases of luatex simplify a lot the solution with \shapemode.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, tabular seems to work (at least
in simple cases) with array, tabularx, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

\begin{verbatim}
5848 \bbl@trace{Redefinitions for bidi layout}
5849 \ifx\@eqnnum\@undefined\else
5850 \else\bbl@attrdir\@undefined\else
5851 \edef\@eqnnum{{%\unexpanded{\ifcase\bbl@attrdir\else\bbl@textdir\@ne\fi}%
5852 \unexpanded\expandafter{%\@eqnnum}}}
5853 \fi
5854 \fi
5855 \fi
5856 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5857 \ifnum\bbl@bidimode>\z@
5858 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5859 \bbl@exp(%
5860 \mathdir\the\bodydir
5861 \#1% Once entered in math, set boxes to restore values
5862 \<ifmmode>%
5863 \everyvbox{%
5864 \the\everyvbox
5865 \bodydir\the\bodydir
5866 \mathdir\the\mathdir
5867 \everyhbox\{\the\everyhbox}\%
5868 \everyvbox\{\the\everyvbox}\%
5869 \everyhbox\%
5870 \the\everyhbox
5871 \bodydir\the\bodydir
5872 \mathdir\the\mathdir
5873 \everyhbox\{\the\everyhbox}\%
5874 \everyvbox\{\the\everyvbox}\%
5875 \<fi>}%
5876 \def@hangfrom#1{%
5877 \setbox\@tempboxa\hbox{{#1}}%
5878 \hangindent\wd\@tempboxa
5879 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5880 \shapemode@ne
5881 \fi
5882 \noindent\box\@tempboxa}
5883 \fi
5884 \IfBabelLayout{tabular}
5885 {\let\bbl@OL@tabular@tabular\bbl@OL}
5886 \bbl@replace@tabular{$}{\bbl@nextfake$}%
5887 \let\bbl@NL@tabular@tabular\bbl@NL
5888 \AtBeginDocument{%
5889 \ifx\bbl@NL@tabular\@tabular\else
5890 \bbl@replace@tabular{$}{\bbl@nextfake$}%
5891 \let\bbl@NL@tabular@tabular
5892 \fi}
5893 }
5894 \IfBabelLayout{lists}
5895 {\let\bbl@OL@list\bbl@OL}
5896 \bbl@replace@list{\parshape}{\bbl@listparshape}%
5897 \let\bbl@NL@list\bbl@NL
5898 \def\bbl@listparshape#1#2#3{%
5899 \parshape #1 #2 #3%
5900 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5901 \shapemode@tw@
5902 \fi}
5903 }
\end{verbatim}
\IfBabelLayout{graphics}
{\let\bbl@pictresetdir\relax
\def\bbl@pictsetdir#1{% 
\ifcase\bbl@thetextdir
\let\bbl@pictresetdir\relax
\else
\ifcase#1\bodydir TLT % Remember this sets the inner boxes
\or\textdir TLT
\else\bodydir TLT \textdir TLT
\fi
% \text|par)dir required in pgf:
\def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
\fi}
\ifx\AddToHook\@undefined\else
\AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
\directlua{
Babel.get_picture_dir = true
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
      end
      if dir and (dir == 'al' or dir == 'r') then
        Babel.picture_has_bidi = 1
      end
    end
  end
  return head
end
luatexbase.add_to_callback("hpack_filter", Babel.picture_dir, "Babel.picture_dir")%
}\AtBeginDocument{%
\long\def\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
}%
\setbox\z@@\ht\z@\z@
\@defaultunitsset\@tempdimc{#1}\unitlength
Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.

\IfBabelLayout{counters}{\let\bbl@OL@@textsuperscript\@textsuperscript
\bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}
\let\bbl@latinarabic=\@arabic
\let\bbl@OL@@arabic\@arabic
\def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}
\let\bbl@asciiroman=\@roman
\let\bbl@OL@@roman\@roman
\def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}
\let\bbl@OL@labelenumii\labelenumii
\def\labelenumii{)
\theenumii(}
\let\bbl@OL@p@enumiii\p@enumiii
\def\p@enumiii{\p@enumii)
\theenumii(}}
\IfBabelLayout{footnotes}{\let\bbl@OL@footnote\footnote
\BabelFootnote\footnote\languagename{}{}%
\BabelFootnote\localfootnote\languagename{}{}%}
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.

\IfBabelLayout{extras}\
\begin{verbatim}
\let\bbl@OL@underline\underline
\bbl@sreplace\underline{$\@\underline}{\bbl@nextfake$\@\underline}\
\let\bbl@OL@LaTeX2e\LaTeX2e
\DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
\if b\expandafter\@car\f@series\@nil\boldmath\fi
\babelsublr{\LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
\end{verbatim}

\langle\text{/luatex}\rangle

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 that 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.

\begin{verbatim}
\langle* transforms \rangle
Babel.linebreaking.replacements = {}  \Babel.linebreaking.replacements[0] = {} \Babel.linebreaking.replacements[1] = {}  -- pre -- post
-- Discretionaries contain strings as nodes
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
\end{verbatim}

192
Merging both functions doesn't seem feasible, because there are too many differences.

```lua
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
      end
      break
    else
      break
    end
    elseif item.id == 12 and item.subtype == 13 then
      word_string = word_string .. ' '
      word_nodes[#word_nodes+1] = item
    end
    -- Ignore leading unrecognized nodes, too.
    elseif item.id == 11 then
      -- pass
    end
  end

  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
    -- (1) Go to next word if nothing was found, and (2) implicitly
    -- remove leading USs.
    elseif word_string == '' then
      -- pass
    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
  end
  word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
end

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)
function Babel.hyphenate_replace(head, mode)
  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
      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
        -- This variable is set in some cases below to the first *byte*
        -- after the match, either as found by u.match (faster) or the
        -- computed position based on sc if w has changed.
        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
          -- 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

local save_last = last -- with A()BCD, points to D

-- 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 n small table the nodes
-- corresponding to the pattern. Used by 'data' to provide a
-- predictable behavior with 'insert' (now 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)
  end

  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)

  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
  node.remove(head, item)
table.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

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

elseif crep and crep.string then
  local str = crep.string(matches)
  if str == '' then -- Gather with nil
    node.remove(head, item)
table.remove(w_nodes, sc)
w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
sc = 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(head, 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(head, 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(head, item)
end -- if ''
last_match = utf8.offset(w, sc+1+step)
goto next

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
    d.penalty = crep.penalty or tex.exhyphenpenalty
end
placeholder = '|'  
head, new = node.insert_before(head, item, d)
if 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
    d = node.new(12, 13) -- (glue, spaceskip)
    local quad = font.getfont(item_base.font).size or 655360
    node.setglue(d, crep.space[1] * quad,
                 crep.space[2] * quad,
                 crep.space[3] * quad)
    if mode == 0 then
        placeholder = ' '  
    end
    head, new = node.insert_before(head, item, d)
elseif crep and crep.spacefactor then
    d = node.new(12, 13) -- (glue, spaceskip)
    local base_font = font.getfont(item_base.font)
    node.setglue(d,
                 crep.spacefactor[1] * base_font.parameters['space'],
                 crep.spacefactor[2] * base_font.parameters['space_stretch'],
                 crep.spacefactor[3] * base_font.parameters['space_shrink'])
    if mode == 0 then
        placeholder = ' '  
    end
    head, new = node.insert_before(head, item, d)
elseif 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)

d

last_match = utf8.offset(w, sc+1+step)
::next::
end -- for each replacement

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

end -- for match

::next::
word_head = nw

end -- for substring

return head

::next::

-- 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)
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 luatex excels, because everything related to bidi writing is under our control.

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

function Babel.bidi(head, ispar)
    local strong = ('TRT' == tex.pardir) and 'r' or 'l'
    local strong_lr = (strong == 'l') and 'l' or 'r'
    local outer = strong
    local num = ''
    local type_n = ''
Next also depends on script/lang (<al>/<r>). To be set by babel. tex.pardir is dangerous, could be (re)set but it should be changed only in vmode. There are two strong's – strong = l/al/r and strong_lr = l/r (there must be a better way):
```
for item in node.traverse(head) do
  -- three cases: glyph, dir, otherwise
  if item.id == node.id'glyph'
    or (item.id == 7 and item.subtype == 2) then
    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
  end
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
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.

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.

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

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

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

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

if dir == 'l' or dir == 'r' then
  if dir ~= outer then
    first_d = first_d or item
    last_d = item
  else first_d and dir ~= strong_lr then
    dir_mark(head, first_d, last_d, outer)
  end
end
Mirroring. Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it’s clearly <r> and <l>, 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.

```lua
6640  if dir and not last_lr and dir == 'l' and outer == 'r' then
6641     item.char = characters[item.char] and
6642         characters[item.char].m or item.char
6643  elseif (dir or new_dir) and last_lr ~= item then
6644      local mir = outer .. strong_lr .. (dir or outer)
6645      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6646         for ch in node.traverse(node.next(last_lr)) do
6647             if ch == item then break end
6648             if ch.id == node.id'glyph' and characters[ch.char] then
6649                 ch.char = characters[ch.char].m or ch.char
6650             end
6651         end
6652      end
6653  end
6654
6655  if dir == 'l' or dir == 'r' then
6656      last_lr = item
6657      strong = dir_real -- Don’t search back - best save now
6658  elseif new_dir then
6659      last_lr = nil
6660  end
6661 end
6662
6663  if last_lr and outer == 'r' then
6664      for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6665          if characters[ch.char] then
6666              ch.char = characters[ch.char].m or ch.char
6667          end
6668      end
6669  end
6670
6671  if first_n then
6672      dir_mark(head, first_n, last_n, outer)
6673  end
6674
6675  if first_d then
6676      dir_mark(head, first_d, last_d, outer)
6677  end
6678
6679  return node.prev(head) or head
6680 end
6681 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```lua
6682 ⟨*basic⟩
6683 Babel = Babel or {}
6684
6685 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6686
6687 Babel.fontmap = Babel.fontmap or {}
```

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

```
6675  return node.prev(head) or head
6676 end
6677 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:
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

local 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 nodes = {}

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

43
local outer_first = nil
local inmath = false
local glue_d = nil
local glue_i = nil
local has_en = false
local first_et = nil

local ATDIR = 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
-- 'al' is only taken into account in the first, current loop
if save_outer == 'al' then save_outer = 'r' end

local fontmap = Babel.fontmap
for item in node.traverse(head) do
  -- In what follows, #node is the last (previous) node, because the
  -- current one is not added until we start processing the neutrals.
  -- three cases: glyph, dir, otherwise
  if item.id == GLYPH
    or (item.id == 7 and item.subtype == 2) then
    local d_font = nil
    local item_r
    if item.id == 7 and item.subtype == 2 then
      item_r = item.replace -- automatic discs have just 1 glyph
    else
      item_r = item
    end
    local chardata = characters[item_r.char]
    d = chardata and chardata.d or nil
    if not d or d == 'nsm' then
      for nn, et in ipairs(ranges) do
        if item_r.char < et[1] then
          break
        elseif item_r.char <= et[2] then
          if not d then d = et[3] end
        end
      end
    end
  end
  if item.id == 7 then
    if item.child[1].id == 7 then
      item_r = item.replace
    end
  end
  if item.id == 6 then
    if item.subtype == 2 and item.child[1].id == 7 then
      item_r = item.replace
    end
  end
end
206
elseif d == 'nsn' then d_font = et[3]
break
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)
  end
  attr_d = attr_d % 3
  if attr_d == 1 then
    outer_first = 'r'
    last = 'r'
  elseif attr_d == 2 then
    outer_first = 'r'
    last = 'al'
  else
    outer_first = 'l'
    last = 'l'
  end
  outer = last
  has_en = false
  first_et = nil
  new_d = false
end

if glue_d then
  if (d == 'l' and 'l' or 'r') ~= glue_d then
    table.insert(nodes, {glue_i, 'on', nil})
  end
  glue_d = nil
  glue_i = nil
end

elseif item.id == DIR then
  d = nil
  new_d = true
endif

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 \text{then} 
    inmath = \text{(item.subtype == 0)}
else
    d = nil
end

-- AL <= EN/ET/ES -- W2 + W3 + W6 
if last == \text{'}al\text{'} and d == \text{'}en\text{'} then 
    d = \text{'}an\text{'} -- W3
elseif last == \text{'}al\text{'} and (d == \text{'}et\text{'} or d == \text{'}es\text{'}) then 
    d = \text{'}on\text{'} -- W6
end

-- EN + CS/ES + EN -- W4 
if d == \text{'}en\text{'} and \#nodes >= 2 then 
    if (nodes[\#nodes][2] == \text{'}es\text{'} or nodes[\#nodes][2] == \text{'}cs\text{'}) 
        and nodes[\#nodes-1][2] == \text{'}en\text{'} then 
        nodes[\#nodes][2] = \text{'}en\text{'}
    end
end

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

-- ET/EN -- W5 + W7->l / W6->on 
if d == \text{'}et\text{'} then 
    first_et = first_et or (\#nodes + 1)
elseif d == \text{'}en\text{'} then 
    has_en = true 
    first_et = first_et or (\#nodes + 1)
else 
    first_et = nil 
    has_en = false 
end 

for e = first_et, \#nodes do 
    if nodes[e][1].id == GLYPH then nodes[e][2] = temp end 
end 

-- Force mathdir in math if ON (currently works as expected only -- with \text{'l'}\text{')} 
if inmath and d == \text{'}on\text{'} then 
    d = (\text{'TRT' == tex.mathdir} \text{') and \text{'r'} or \text{'l'}} 
end
if d then
  if d == 'al' then
    d = 'r'
    last = 'al'
  elseif d == 'l' or d == 'r' then
    last = d
  end
  prev_d = d
  table.insert(nodes, {item, d, outer_first})
end

outer_first = nil
end

-- 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
table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})

--------------- NEUTRAL ----------------
outer = save_outer
last = outer
local first_on = nil
for q = 1, #nodes do
  local item
  local outer_first = nodes[q][3]
  outer = outer_first or outer
  last = outer_first or last
  local d = nodes[q][2]
  if d == 'an' or d == 'en' then d = 'r' end
  if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
  if d == 'on' then
    first_on = first_on or q
  elseif first_on then
    if last == d then
      temp = d
    end
  end
end
else
    temp = outer
end
for r = first_on, q - 1 do
    nodes[r][2] = temp
    item = nodes[r][1] -- MIRRORING
    if Babel.mirroring_enabled and item.id == GLYPH
        and temp == 'r' and characters[item.char]
        then
            local font_mode = font.fonts[item.font].properties.mode
            if font_mode ~= 'harf' and font_mode ~= 'plug' then
                item.char = characters[item.char].m or item.char
            end
        end
    end
end
first_on = nil
end
if d == 'r' or d == 'l' then last = d end
end
------------ IMPLICIT, REORDER --------------
outer = save_outer
last = outer
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 == 'l' 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)
            end
        end
    else
        if d == 'an' or d == 'l' then
            if d == 'r' then state.has_r = nil, nil, false
            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
state.eim = item
elseif d == 'r' and state.sim then
    head, state = insert_implicit(head, state, outer)
elseif d == 'r' then
    state.sim, state.eim = nil, nil
end
if isdir then
    last = d -- Don't search back - best save now
elseif d == 'on' and state.san then
    state.san = state.san or item
    state.ean = item
end
return node.prev(head) or head
end
⟨/*nil*/
\ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩] Nil language
\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.
\IFDEF\l@nil\@undefined
\newlanguage\l@nil
\@namedef{bbl@hyphendata@	he\l@nil}{{}{}}% Remove warning
\let\bbl@elt\relax
\edef\bbl@languages{% Add it to the list of languages

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.
This macro is used to store the values of the hyphenation parameters \lefthyphenmin and \righthyphenmin.

\providehyphenmins{\CurrentOption}{\m@ne\m@ne}

The next step consists of defining commands to switch to (and from) the ‘nil’ language.

\captionnil \datenil
\let\captionsnil\@empty
\let\datenil\@empty

The macro \ldf@finish takes care of looking for a configuration file, setting the main language to be switched on at \begin{document} and resetting the category code of @ to its original value.

\ldf@finish(nil)
\(/nil)

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

16.1 Not renaming hyphen.tex

As Don Knuth has declared that the filename hyphen.tex may only be used to designate his version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based \TeX{}-format. When asked he responded:

That file name is “sacred”, and if anybody changes it they will cause severe upward/downward compatibility headaches.

People can have a file localhyphen.tex or whatever they like, but they mustn’t diddle with hyphen.tex (or plain.tex except to preload additional fonts).

The files bplain.tex and blplain.tex can be used as replacement wrappers around plain.tex and lplain.tex to achieve the desired effect, based on the babel package. If you load each of them with \input{}, you will get a file called either bplain.fmt or blplain.fmt, which you can use as replacements for plain.fmt and lplain.fmt.

As these files are going to be read as the first thing \init\TeX{} sees, we need to set some category codes just to be able to change the definition of \input.

\openin 0 hyphen.cfg
\ifeof0
\else
\let[a]input
\fi

Then \input{} is defined to forget about its argument and load hyphen.cfg instead. Once that’s done the original meaning of \input{} can be restored and the definition of a can be forgotten.
Now that we have made sure that hyphen.cfg will be loaded at the right moment it is time to load plain.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 an alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).
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}}
\@onlypreamble\@onlypreamble
\end{verbatim}

\textbf{Mimick \LaTeX's} \texttt{\textbackslash AtBeginDocument}; for this to work the user needs to add \texttt{\textbackslash begindocument} to his file.

\begin{verbatim}
\def\begindocument{\@begindocumenthook
\global\let\@begindocumenthook\@undefined
\do\@preamblecmds
\global\let\do\noexpand}
\ifx\@begindocumenthook\@undefined
\def\@begindocumenthook{}\fi
\@onlypreamble\@onlypreamble
\end{verbatim}

\textbf{\LaTeX2ε} has the command \texttt{\@onlypreamble} which adds commands to a list of commands that are no longer needed after \texttt{\begin{document}}.

\begin{verbatim}
\ifx\@preamblecmds\@undefined
\def\@preamblecmds{}
\fi
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble
\end{verbatim}

\textbf{\LaTeX2ε} has the command \texttt{\@onlypreamble} which adds commands to a list of commands that are no longer needed after \texttt{\begin{document}}.

\begin{verbatim}
\ifx\@preamblecmds\@undefined
\def\@preamblecmds{}
\fi
\def\@onlypreamble#1{\expandafter\gdef\expandafter\@preamblecmds\expandafter{\@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble
\end{verbatim}

\textbf{Mimick \LaTeX'S} \texttt{\textbackslash AtBeginDocument}; for this to work the user needs to add \texttt{\textbackslash begindocument} to his file.
We also have to mimick \LaTeX's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `@endofldf`.

\begin{verbatim}
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 `\if`. The same trick is applied below.

\begin{verbatim}
catcode`\&=7
\ifx\if\expandafter\let\csname\if\expandafter\csname iffalse\endcsname
\fi
\catcode`\&=4
\end{verbatim}

Mimick \LaTeX's commands to define control sequences.

\begin{verbatim}
def\newcommand{\@star@or@long\new@command}
def\new@command#1{\@testopt{\@newcommand#1}0}
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}}
\long\def\@yargdef#1#2#3{\@tempcnta#3\relax\advance\@tempcnta\@ne\let\@hash@\relax\edef\reserved@a{\ifx#2\tw@\[\@hash@1\]\fi}\@tempcntb#2\@whilenum\@tempcntb<\@tempcnta\do{\edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}}\let\@hash@##\l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
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{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand} 
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}
def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
def\declare@robustcommand#1{\begingroup\escapechar\m@ne\def\@gtempa{\string#1}
\endgroup\begingroup\edef\reserved@a{\escapechar\m@ne}\xdef\@gtempa\endgroup\expandafter\@ifundefined\@gtempa{\def\reserved@a{\new@command\reserved@a}}{\let\reserved@a\relax\def\reserved@a{\new@command\reserved@a}}\reserved@a}{
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 \@if@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\@if@aded#1#2#3#4{}

For the following code we need to make sure that the commands \newcommand and \providecommand exist with some sensible definition. They are not fully equivalent to their \LaTeX{} versions; just enough to make things work in plain \TeX{} environments.

\ifx\@tempcnta\@undefined
\csname newcount\endcsname\@tempcnta\relax
\fi
\ifx\@tempcntb\@undefined
\csname newcount\endcsname\@tempcntb\relax
\fi
To prevent wasting two counters in \TeX (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

\begin{verbatim}
16.4 Encoding related macros

Code from loutenc.dtx, adapted for use in the plain \TeXX environment.

\def\DeclareTextCommand{\@dec@text@cmd\providecommand}
\def\ProvideTextCommand{\@dec@text@cmd\providecommand}
\def\DeclareTextSymbol#1#2#3{\@dec@text@cmd\chardef#1{#2}#3\relax}
\def\@dec@text@cmd#1#2#3{\expandafter\def\expandafter#2\expandafter{\csname#3-cmd\endcsname#2\csname#3\string#2\endcsname}}
\@ifdefinable\@rc@ifdefinable\expandafter#1\csname#3\string#2\endcsname
\end{verbatim}
Currently we only use the \textasciitilde 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 \TeX-control sequence \scriptsize to be available. Because plain \TeX doesn't have such a sophisticated font mechanism as \TeX has, we just \let it to \sevenrm.
A proxy file:

\input babel.def

17 Acknowledgements

I would like to thank all who volunteered as \textit{\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 Spith helped find and repair bugs. During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

