Babel

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and internationalization

Unicode
TEX
pdfTEX
LuaTEX
XeTEX

Contents

Troubleshoooting

Part I
User guide

What is this document about? This user guide focuses on internationalization and localization with LaTeX and pdftex, xetex and luatex with the babel package. There are also some notes on its use with e-Plain and pdf-Plain TeX. 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 LaTeX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in LaTeX for an option – in this case a language – to be recognized by several packages.

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

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

```
PDFTEX
    \documentclass{article}

    \usepackage[T1]{fontenc}
```

```
\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

The document will be typeset, but very likely the text will not be correctly hyphenated. Some
languages 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 (MacTeX, MikTeX, TeXLive, etc.) for
further info about how to configure it.

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In LaTeX, the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

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

```
PDFTEX
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

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

\end{document}
```

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

```
LUATEX/XETEX
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename, \alsoname, \today.

\selectlanguage{vietnamese}

\prefacename, \alsoname, \today.

\end{document}
```

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

1.3   Mostly monolingual documents

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

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

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

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4   Modifiers

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

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

Attributes (described below) are considered modifiers, ie, you can set an attribute by including it in the list of modifiers. 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:[2]

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

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

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

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

---

[1] No predefined "axis" for modifiers are provided because languages and their scripts have quite different needs.
[2] In old versions the error read "You have used an old interface to call babel", not very helpful.
[3] In old versions the error read "You haven't loaded the language LANG yet".

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.

<code>\selectlanguage</code> {⟨*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.

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

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

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

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

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

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

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

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

The command \foreignlanguage takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one.

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

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

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

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

1.8 Auxiliary language selectors

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

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

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

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

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

Spaces after the environment are ignored.

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

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

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

1.9 More on selection

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

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

---

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

1.10 Shorthands

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

There are four levels of shorthands: *user*, *language*, *system*, and *language user* (by order of precedence). In most cases, you will use only shorthands provided by languages.

NOTE Keep in mind the following:

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

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

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

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

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

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

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

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

\useshorthands `*{⟨char⟩}`

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

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

The command \defineshorthand takes two arguments: the first is a one- or two-character
shorthand sequence, and the second is the code the shorthand should expand to.
New 3.9a An optional argument allows to (re)define language and system shorthands
(some languages do not activate shorthands, so you may want to add
\languageshorthands{⟨lang⟩} to the corresponding \extras⟨lang⟩, as explained below).
By default, user shorthands are (re)defined.
User shorthands override language ones, which in turn override system shorthands.
Language-dependent user shorthands (new in 3.9) take precedence over "normal" user
shorthands.

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

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

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

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

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

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

\languageshorthands `{⟨language⟩}`

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

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

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

---

[5] 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.

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

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

`\babelshorthand` {⟨*shorthand*⟩}

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

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

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

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

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

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

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

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

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

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

`\aliasshorthand` {⟨*original*⟩}{⟨*alias*⟩}

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

---

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

character / over " in typing Polish texts, this can be achieved by entering
\aliasshorthand{"}{/}. For the reasons in the warning below, usage of this macro is not
recommended.

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

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

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

WARNING Shorthands remember somehow the original character, and the fallback value is that of
the latter. So, in this example, if no shorthand if found, ^ expands to a non-breaking space,
because this is the value of ~ (internally, ^ still calls \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 εTeX 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.[8]

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

Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional TeX, LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal LaTeX 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.[9] It can take the following values:

off deactivates this feature and no case mapping is applied;
first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at `\begin{document}`, but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated;[10]
select sets it only at `\selectlanguage`;
other also sets it at `otherlanguage`;
other* also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.[11]

---

[8] You can use alternatively the package silence.
[9] Turned off in plain.
[10] Duplicated options count as several ones.
[11] Providing foreign is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn't really useful, but who knows.

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

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

layout=

New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.24.

provide= *

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

1.12 The base **option**

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

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

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

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

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

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

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

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

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

1.13 ini **files**

An alternative approach to define a language (or, more precisely, a *locale*) is by means of an ini file. Currently babel provides about 250 of these files containing the basic data required for a locale, plus basic templates for 500 about locales.
ini files are not meant only for babel, and they has been devised as a resource for other packages. To easy interoperability between TeX and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the \...name strings).
Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward

compatibility is important). The following section shows how to make use of them by means of \babelprovide. In other words, \babelprovide is mainly meant for auxiliary tasks, and as alternative when the ldf, for some reason, does work as expected.

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

LUATEX/XETEX

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უძველესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

Or also:

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

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

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

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

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

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

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

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

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1ກ 1ຍ 1ອ 1ງ 1ກ 1�} % Random
```

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

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

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

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

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

af	Afrikaans[ul]	asa	Asu
agq	Aghem	ast	Asturian[ul]
ak	Akan	az-Cyrl	Azerbaijani
am	Amharic[ul]	az-Latn	Azerbaijani
ar	Arabic[ul]	az	Azerbaijani[ul]
ar-DZ	Arabic[ul]	bas	Basaa
ar-EG	Arabic[ul]	be	Belarusian[ul]
ar-IQ	Arabic[ul]	bem	Bemba
ar-JO	Arabic[ul]	bez	Bena
ar-LB	Arabic[ul]	bg	Bulgarian[ul]
ar-MA	Arabic[ul]	bm	Bambara
ar-PS	Arabic[ul]	bn	Bangla[ul]
ar-SA	Arabic[ul]	bo	Tibetan[u]
ar-SY	Arabic[ul]	brx	Bodo
ar-TN	Arabic[ul]	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian[ul]

Code	Language	Code	Language
bs	Bosnian[ul]	ha-GH	Hausa
ca	Catalan[ul]	ha-NE	Hausa[l]
ce	Chechen	ha	Hausa
cgg	Chiga	haw	Hawaiian
chr	Cherokee	he	Hebrew[ul]
ckb	Central Kurdish	hi	Hindi[u]
cop	Coptic	hr	Croatian[ul]
cs	Czech[ul]	hsb	Upper Sorbian[ul]
cu	Church Slavic	hu	Hungarian[ul]
cu-Cyrs	Church Slavic	hy	Armenian[u]
cu-Glag	Church Slavic	ia	Interlingua[ul]
cy	Welsh[ul]	id	Indonesian[ul]
da	Danish[ul]	ig	Igbo
dav	Taita	ii	Sichuan Yi
de-AT	German[ul]	is	Icelandic[ul]
de-CH	Swiss High German[ul]	it	Italian[ul]
de	German[ul]	ja	Japanese[u]
dje	Zarma	jgo	Ngomba
dsb	Lower Sorbian[ul]	jmc	Machame
dua	Duala	ka	Georgian[ul]
dyo	Jola-Fonyi	kab	Kabyle
dz	Dzongkha	kam	Kamba
ebu	Embu	kde	Makonde
ee	Ewe	kea	Kabuverdianu
el	Greek[ul]	khq	Koyra Chiini
el-polyton	Polytonic Greek[ul]	ki	Kikuyu
en-AU	English[ul]	kk	Kazakh
en-CA	English[ul]	kkj	Kako
en-GB	English[ul]	kl	Kalaallisut
en-NZ	English[ul]	kln	Kalenjin
en-US	English[ul]	km	Khmer
en	English[ul]	kmr	Northern Kurdish[u]
eo	Esperanto[ul]	kn	Kannada[ul]
es-MX	Spanish[ul]	ko	Korean[u]
es	Spanish[ul]	kok	Konkani
et	Estonian[ul]	ks	Kashmiri
eu	Basque[ul]	ksb	Shambala
ewo	Ewondo	ksf	Bafia
fa	Persian[ul]	ksh	Colognian
ff	Fulah	kw	Cornish
fi	Finnish[ul]	ky	Kyrgyz
fil	Filipino	lag	Langi
fo	Faroese	lb	Luxembourgish[ul]
fr	French[ul]	lg	Ganda
fr-BE	French[ul]	lkt	Lakota
fr-CA	French[ul]	ln	Lingala
fr-CH	French[ul]	lo	Lao[ul]
fr-LU	French[ul]	lrc	Northern Luri
fur	Friulian[ul]	lt	Lithuanian[ul]
fy	Western Frisian	lu	Luba-Katanga
ga	Irish[ul]	luo	Luo
gd	Scottish Gaelic[ul]	luy	Luyia
gl	Galician[ul]	lv	Latvian[ul]
grc	Ancient Greek[ul]	mas	Masai
gsw	Swiss German	mer	Meru
gu	Gujarati	mfe	Morisyen
guz	Gusii	mg	Malagasy
gv	Manx	mgh	Makhuwa-Meetto

Code	Language	Code	Language
mgo	Meta'	shi-Tfng	Tachelhit
mk	Macedonian[ul]	shi	Tachelhit
ml	Malayalam[ul]	si	Sinhala
mn	Mongolian	sk	Slovak[ul]
mr	Marathi[ul]	sl	Slovenian[ul]
ms-BN	Malay[l]	smn	Inari Sami
ms-SG	Malay[l]	sn	Shona
ms	Malay[ul]	so	Somali
mt	Maltese	sq	Albanian[ul]
mua	Mundang	sr-Cyrl-BA	Serbian[ul]
my	Burmese	sr-Cyrl-ME	Serbian[ul]
mzn	Mazanderani	sr-Cyrl-XK	Serbian[ul]
naq	Nama	sr-Cyrl	Serbian[ul]
nb	Norwegian Bokmål[ul]	sr-Latn-BA	Serbian[ul]
nd	North Ndebele	sr-Latn-ME	Serbian[ul]
ne	Nepali	sr-Latn-XK	Serbian[ul]
nl	Dutch[ul]	sr-Latn	Serbian[ul]
nmg	Kwasio	sr	Serbian[ul]
nn	Norwegian Nynorsk[ul]	sv	Swedish[ul]
nnh	Ngiemboon	sw	Swahili
no	Norwegian	ta	Tamil[u]
nus	Nuer	te	Telugu[ul]
nyn	Nyankole	teo	Teso
om	Oromo	th	Thai[ul]
or	Odia	ti	Tigrinya
os	Ossetic	tk	Turkmen[ul]
pa-Arab	Punjabi	to	Tongan
pa-Guru	Punjabi	tr	Turkish[ul]
pa	Punjabi	twq	Tasawaq
pl	Polish[ul]	tzm	Central Atlas Tamazight
pms	Piedmontese[ul]	ug	Uyghur
ps	Pashto	uk	Ukrainian[ul]
pt-BR	Portuguese[ul]	ur	Urdu[ul]
pt-PT	Portuguese[ul]	uz-Arab	Uzbek
pt	Portuguese[ul]	uz-Cyrl	Uzbek
qu	Quechua	uz-Latn	Uzbek
rm	Romansh[ul]	uz	Uzbek
rn	Rundi	vai-Latn	Vai
ro	Romanian[ul]	vai-Vaii	Vai
ro-MD	Moldavian[ul]	vai	Vai
rof	Rombo	vi	Vietnamese[ul]
ru	Russian[ul]	vun	Vunjo
rw	Kinyarwanda	wae	Walser
rwk	Rwa	xog	Soga
sa-Beng	Sanskrit	yav	Yangben
sa-Deva	Sanskrit	yi	Yiddish
sa-Gujr	Sanskrit	yo	Yoruba
sa-Knda	Sanskrit	yue	Cantonese
sa-Mlym	Sanskrit	zgh	Standard Moroccan Tamazight
sa-Telu	Sanskrit	zh-Hans-HK	Chinese[u]
sa	Sanskrit	zh-Hans-MO	Chinese[u]
sah	Sakha	zh-Hans-SG	Chinese[u]
saq	Samburu	zh-Hans	Chinese[u]
sbp	Sangu	zh-Hant-HK	Chinese[u]
se	Northern Sami[ul]	zh-Hant-MO	Chinese[u]
seh	Sena	zh-Hant	Chinese[u]
ses	Koyraboro Senni	zh	Chinese[u]
sg	Sango	zu	Zulu
shi-Latn	Tachelhit		

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.

aghem	chechen
akan	cherokee
albanian	chiga
american	chinese-hans-hk
amharic	chinese-hans-mo
ancientgreek	chinese-hans-sg
arabic	chinese-hans
arabic-algeria	chinese-hant-hk
arabic-DZ	chinese-hant-mo
arabic-morocco	chinese-hant
arabic-MA	chinese-simplified-hongkongsarchina
arabic-syria	chinese-simplified-macausarchina
arabic-SY	chinese-simplified-singapore
armenian	chinese-simplified
assamese	chinese-traditional-hongkongsarchina
asturian	chinese-traditional-macausarchina
asu	chinese-traditional
australian	chinese
austrian	churchslavic
azerbaijani-cyrillic	churchslavic-cyrs
azerbaijani-cyrl	churchslavic-oldcyrillic[12]
azerbaijani-latin	churchsslavic-glag
azerbaijani-latn	churchsslavic-glagolitic
azerbaijani	colognian
bafia	cornish
bambara	croatian
basaa	czech
basque	danish
belarusian	duala
bemba	dutch
bena	dzongkha
bangla	embu
bodo	english-au
bosnian-cyrillic	english-australia
bosnian-cyrl	english-ca
bosnian-latin	english-canada
bosnian-latn	english-gb
bosnian	english-newzealand
brazilian	english-nz
breton	english-unitedkingdom
british	english-unitedstates
bulgarian	english-us
burmese	english
canadian	esperanto
cantonese	estonian
catalan	ewe
centralatlastamazight	ewondo
centralkurdish	faroese

---

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

filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini

kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese

polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi

sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
tachelhit
taita
tamil
tasawaq
telugu
teso
thai
tibetan
tigrinya
tongan
turkish
turkmen
ukenglish
ukrainian
uppersorbian
urdu
usenglish
usorbian
uyghur
uzbek-arab
uzbek-arabic
uzbek-cyrillic
uzbek-cyrl
uzbek-latin
uzbek-latn
uzbek
vai-latin
vai-latn
vai-vai
vai-vaii
vai
vietnam
vietnamese
vunjo
walser
welsh
westernfrisian
yangben
yiddish
yoruba
zarma
zulu afrikaans

Modifying and adding values to `ini` **files**

New 3.39 There is a way to modify the values of `ini` files when they get loaded with `\babelprovide` and import. To set, say, `digits.native` in the `numbers` section, use something like `numbers/digits.native=abcdefghij`. Keys may be added, too. Without `import` you may modify the identification keys.

This can be used to create private variants easily. All you need is to import the same `ini` file with a different locale name and different parameters.

1.14 Selecting fonts

New 3.15 Babel provides a high level interface on top of `fontspec` to select fonts. There is no need to load fontspec explicitly – babel does it for you with the first `\babelfont`.[13]

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

---

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

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

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

EXAMPLE Here is how to do it:

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

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

NOTE You may load fontspec explicitly. For example:

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

EXAMPLE   If you need a language named `arhinish`:

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

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

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

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

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

import= ⟨*language-tag*⟩

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

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

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

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

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

captions= ⟨*language-tag*⟩

Loads only the strings. For example:

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

hyphenrules= ⟨*language-list*⟩

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

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

If none of the listed hyphenrules exist, the default behavior applies. Note in this example we set `chavacano` as first option – without it, it would select `spanish` even if `chavacano` exists.
A special value is +, which allocates a new language (in the TEX sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with luatex, because you can add some patterns with `\babelpatterns`, as for example:

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

In other engines it just suppresses hyphenation (because the pattern list is empty).
New 3.58 Another special value is `unhyphenated`, which activates a line breking 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.

> **EXAMPLE** Let's assume your document (xetex or luatex) is mainly in Polytonic Greek with but with some sections in Italian. Then, the first attempt should be:
>
> ```
> \usepackage[italian, greek.polutonic]{babel}
> ```
>
> But if, say, accents in Greek are not shown correctly, you can try
>
> ```
> \usepackage[italian, polytonicgreek, provide=*]{babel}
> ```
>
> Remerber there is an alternative syntax for the latter:
>
> ```
> \usepackage[italian]{babel}
> \babelprovide[import, main]{polytonicgreek}
> ```
>
> Finally, also remember you might not need to load `italian` at all if there are only a few word in this language (see 1.3).

script= ⟨*script-name*⟩

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

language= ⟨*language-name*⟩

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

alph= ⟨*counter-name*⟩

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

Alph= ⟨*counter-name*⟩

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

onchar= `ids` | `fonts` | `letters`

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

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

<dl>

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

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

intrapenalty= ⟨*penalty*⟩

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

transforms= ⟨*transform-list*⟩

See section 1.21.

justification= kashida | elongated | unhyphenated | padding

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.
New 3.81   The option `padding` has been devised primarily for Tibetan. It's still somewhat experimental. Again, there is an explanation in the babel site.

linebreaking=   New 3.59   Just a synonymous for `justification`.

mapfont= direction

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

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

1.17   Digits and counters

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

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

</dl>

Languages providing native digits in all or some variants are:

Arabic	Persian	Lao	Odia	Urdu
Assamese	Gujarati	Northern Luri	Punjabi	Uzbek
Bangla	Hindi	Malayalam	Pashto	Vai
Tibetar	Khmer	Marathi	Tamil	Cantonese
Bodo	Kannada	Burmese	Telugu	Chinese
Central Kurdish	Konkani	Mazanderani	Thai	
Dzongkha	Kashmiri	Nepali	Uyghur	

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.

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

New 3.41 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 availabe styles in each language, see the list below):

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

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

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

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

The styles are:

Ancient Greek `lower.ancient`, `upper.ancient`
Amharic `afar`, `agaw`, `ari`, `blin`, `dizi`, `gedeo`, `gumuz`, `hadiyya`, `harari`, `kaffa`, `kebena`, `kembata`, `konso`, `kunama`, `meen`, `oromo`, `saho`, `sidama`, `silti`, `tigre`, `wolaita`, `yemsa`
Arabic `abjad`, `maghrebi.abjad`
Armenian `lower.letter`, `upper.letter`
Belarusan, Bulgarian, Church Slavic, Macedonian, Serbian `lower`, `upper`
Bangla `alphabetic`
Central Kurdish `alphabetic`
Chinese `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`, `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`
Church Slavic (Glagolitic) `letters`
Coptic `epact`, `lower.letters`
French `date.day` (mainly for internal use).
Georgian `letters`
Greek `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with keraia)
Hebrew `letters` (neither geresh nor gershayim yet)
Hindi `alphabetic`
Italian `lower.legal`, `upper.legal`

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

Khmer `consonant`

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

Marathi `alphabetic`

Persian `abjad`, `alphabetic`

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

Syriac `letters`

Tamil `ancient`

Thai `alphabetic`

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

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

1.18   Dates

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

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

By default the calendar is the Gregorian, but an `ini` file may define strings for other calendars (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` and `calendar=coptic`). However, with the option `convert` it's converted (using internally the following command).

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

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

New 3.76   Although calendars aren't the primary concern of babel, the package should be able to, at least, generate correctly the current date in the way users would expect in their own culture. Currently, `\localedate` can print dates in a few calendars (provided the ini locale file has been imported), but year, month and day had to be entered by hand, which is very inconvenient. With this macro, the current date is converted and stored in the three last arguments, which must be macros: allowed calendars are `buddhist`, `coptic`, `hebrew`, `islamic-civil`, `islamic-umalqura`, `persian`. The optional argument converts the given date, in the form '⟨*year*⟩-⟨*month*⟩-⟨*day*⟩'. Please, refer to the page on the news for 3.76 in the babel site for further details.

1.19   Accessing language info

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

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

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

If more than one language is used, it might be necessary to know which language is active at a specific time. This can be checked by a call to `\iflanguage`, but note here "language" is

used in the TeX sense, as a set of hyphenation patterns, and *not* as its babel name. This macro takes three arguments. The first argument is the name of a language; the second and third arguments are the actions to take if the result of the test is true or false respectively.

\localeinfo `*`{⟨*field*⟩}

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

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

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

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

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

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

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

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

\localeid Each language in the babel sense has its own unique numeric identifier, which can be retrieved with `\localeid`.
The `\localeid` is not the same as the `\language` identifier, which refers to a set of hyphenation patters (which, in turn, is just a component of the line breaking algorithm

described in the next section). The data about preloaded patterns are store in an internal macro named \bbl@languages (see the code for further details), but note several locales may share a single \language, so they are separated concepts. In luatex, the \localeid is saved in each node (when it makes sense) as an attribute, too.

\LocaleForEach {⟨code⟩}

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

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

1.20 Hyphenation and line breaking

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

\babelhyphen *{⟨type⟩}
\babelhyphen *{⟨text⟩}

New 3.9a It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in TeX are entered as -, and (2) *optional* or *soft hyphens*, which are entered as \-. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in TeX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.
In TeX, - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, "- in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic "hyphens" which can be used by themselves, to define a user shorthand, or even in language files.

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

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

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

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

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

With all of them, hyphenation in the rest of the word is enabled. If you don't want to enable it, there is a starred counterpart: \babelhyphen*{soft} (which in most cases is equivalent to the original \-), \babelhyphen*{hard}, etc.
Note hard is also good for isolated prefixes (eg, *anti-*) and nobreak for isolated suffixes (eg, *-ism*), but in both cases \babelhyphen*{nobreak} is usually better.

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

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

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

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

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

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

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

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

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

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

New 3.9m *In luatex only*,[14] adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.
It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \lccodes's done in \extras⟨*lang*⟩ as well as the language-specific encoding (not set in the preamble by default). Multiple \babelpatterns's are allowed.
Listed patterns are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.
New 3.31 (Only luatex.) With \babelprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (New 3.32 it is disabled in verbatim mode, or more precisely when the

---

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

hyphenrules are set to nohyphenation). It can be activated alternatively by setting explicitly the intraspace.

New 3.27   Interword spacing for Thai, Lao and Khemer is activated automatically if a language with one of those scripts are loaded with \babelprovide. See the sample on the babel repository. With both Unicode engines, spacing is based on the "current" em unit (the size of the previous char in luatex, and the font size set by the last \selectfont in xetex).

1.21   Transforms

Transforms (only luatex) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.[15]
It currently embraces \babelprehyphenation and \babelposthyphenation.

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

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

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

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

This applies transliteration.omega always, but sigma.final only when \withsigmafinal is set.
Here are the transforms currently predefined. (A few may still require some fine-tuning. More to follow in future releases.)

Arabic	transliteration.dad	Applies the transliteration system devised by Yannis Haralambous for dad (simple and TeX-friendly). Not yet complete, but sufficient for most texts.
Croatian	digraphs.ligatures	Ligatures *DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj*. It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	hyphen.repeat	Explicit hyphens behave like \babelhyphen {repeat}.
Czech, Polish, Slovak	oneletter.nobreak	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Finnish	prehyphen.nobreak	Line breaks just after hyphens prepended to words are prevented, like in "pakastekaapit ja -arkut".

---

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

Greek	`diaeresis.hyphen`	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	`transliteration.omega`	Although the provided combinations are not the full set, this transform follows the syntax of Omega: = for the circumflex, v for digamma, and so on. For better compatibility with Levy's system, ~ (as 'string') is an alternative to =. ' is tonos in Monotonic Greek, but oxia in Polytonic and Ancient Greek.
Greek	`sigma.final`	The transliteration system above does not convert the sigma at the end of a word (on purpose). This transforms does it. To prevent the conversion (an abbreviation, for example), write "s.
Hindi, Sanskrit	`transliteration.hk`	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	`punctuation.space`	Inserts a space before the following four characters: *!?:;* .
Hungarian	`digraphs.hyphen`	Hyphenates the long digraphs *ccs*, *ddz*, *ggy*, *lly*, *nny*, *ssz*, *tty* and *zzs* as *cs-cs*, *dz-dz*, etc.
Indic scripts	`danda.nobreak`	Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.
Latin	`digraphs.ligatures`	Replaces the groups *ae*, *AE*, *oe*, *OE* with *œ*, *Æ*, *œ*, *Œ*.
Latin	`letters.noj`	Replaces *j*, *J* with *i*, *I*.
Latin	`letters.uv`	Replaces *v*, *U* with *u*, *V*.
Sanskrit	`transliteration.iast`	The IAST system to romanize Devanagari.[16]
Serbian	`transliteration.gajica`	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.
Arabic, Persian	`kashida.plain`	Experimental. A very simple and basic transform for 'plain' Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.

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

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

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

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

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

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

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

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

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

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

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

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

See the description above for the optional argument.

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

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

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

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

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

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

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

1.22 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

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

autoload.bcp47 with values on and off.

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

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

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

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

(You can deactivate it with off.) So, if dutch is one of the package (or class) options, you can write \selectlanguage{nl}. Note the language name does not change (in this

example is still `dutch`), but you can get it with `\localeinfo` or `\getlanguageproperty`. It must be turned on explicitly for similar reasons to those explained above.

1.23   Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.[17]

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.[18]

`\ensureascii` {⟨*text*⟩}

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

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

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

1.24   Selecting directions

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

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 are progresses in the latter, including amsmath and mathtools too, but for example `gathered` may fail).

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

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

---

[17]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.

[18]But still defined for backwards compatibility.

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

There are some package options controlling bidi writing.

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

> New 3.14   Selects the bidi algorithm to be used. With default the bidi mechanism is just activated (by default it is not), but every change must be marked up. In xetex and pdftex this is the only option.
>
> In luatex, basic-r provides a simple and fast method for R text, which handles numbers and unmarked L text within an R context many in typical cases. New 3.19   Finally, basic supports both L and R text, and it is the preferred method (support for basic-r is currently limited). (They are named basic mainly because they only consider the intrinsic direction of scripts and weak directionality.)
>
> New 3.29   In xetex, bidi-r and bidi-l resort to the package bidi (by Vafa Khalighi). Integration is still somewhat tentative, but it mostly works. For RL documents use the former, and for LR ones use the latter.
>
> There are samples on GitHub, under /required/babel/samples. See particularly lua-bidibasic.tex and lua-secenum.tex.
>
> **EXAMPLE** The following text comes from the Arabic Wikipedia (article about Arabia). Copy-pasting some text from the Wikipedia is a good way to test this feature. Remember basic is available in luatex only.

```
    \documentclass{article}

    \usepackage[bidi=basic]{babel}

    \babelprovide[import, main]{arabic}

    \babelfont{rm}{FreeSerif}

    \begin{document}

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

    \end{document}
```

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

```
    \documentclass{book}

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

    \babelprovide[onchar=ids fonts]{arabic}

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

    \begin{document}

    Most Arabic speakers consider the two varieties to be two registers
```

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

    \end{document}
```

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

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

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

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

---

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

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

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

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

With counters, \arabic is not only considered L text always (with \babelsublr, see below), but also an "isolated" block which does not interact with the surrounding chars. So, while 1.2 in R text is rendered in that order with bidi=basic (as a decimal number), in \arabic{c1}.\arabic{c2} the visual order is *c2.c1*. Of course, you may always adjust the order by changing the language, if necessary.[19]

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

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

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

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

---

[19] Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr {⟨*lr-text*⟩}

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

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

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

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

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

\BabelPatchSection {⟨*section-name*⟩}

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.25 Language attributes

\languageattribute

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

1.26 Hooks

New 3.9a A hook is a piece of code to be executed at certain events. Some hooks are predefined when luatex and xetex are used.
New 3.64 This is not the only way to inject code at those points. The events listed below can be used as a hook name in `\AddToHook` in the form
`babel/`⟨*language-name*⟩`/`⟨*event-name*⟩ (with * it's applied to all languages), but there is a limitation, because the parameters passed with the babel mechanism are not allowed. The `\AddToHook` mechanism does *not* replace the current one in 'babel'. Its main advantage is you can reconfigure 'babel' even before loading it. See the example below.

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

The same name can be applied to several events. Hooks with a certain {⟨*name*⟩} may be enabled and disabled for all defined events with \EnableBabelHook{⟨*name*⟩}, \DisableBabelHook{⟨*name*⟩}. Names containing the string babel are reserved (they are used, for example, by \useshortands* to add a hook for the event afterextras). New 3.33 They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

defaultcommands Used (locally) in \StartBabelCommands.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

everylanguage (language) Executed before every language patterns are loaded.

loadkernel (file) By default just defines a few basic commands. It can be used to define different versions of them or to load a file.

loadpatterns (patterns file) Loads the patterns file. Used by luababel.def.

loadexceptions (exceptions file) Loads the exceptions file. Used by luababel.def.

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

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

46

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

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

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

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

1.27 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)[20]
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish

---

[20]The two last name comes from the times when they had to be shortened to 8 characters

Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppersorbian
Welsh welsh

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

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

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

1.28 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
```

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

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

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

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

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

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

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

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

- For the hyphenation to work correctly, lccodes cannot change, because TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.[21] So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of TeX, not of babel. Alternatively, you may use `\useshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

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

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

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

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

csquotes Logical markup for quotes.

---

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

iflang Tests correctly the current language.

hyphsubst Selects a different set of patterns for a language.

translator An open platform for packages that need to be localized.

siunitx Typesetting of numbers and physical quantities.

biblatex Programmable bibliographies and citations.

bicaption Bilingual captions.

babelbib Multilingual bibliographies.

microtype Adjusts the typesetting according to some languages (kerning and spacing). Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

ucharclasses (xetex) Switches fonts when you switch from one Unicode block to another.

zhspacing Spacing for CJK documents in xetex.

1.31   Current and future work

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

Useful additions would be, for example, time, currency, addresses and personal names.[22]. But that is the easy part, because they don't require modifying the LaTeX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

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

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

1.32   Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2   Loading languages with `language.dat`

TeX and most engines based on it (pdfTeX, xetex, ϵ-TeX, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, LaTeX, XeLaTeX, pdfLaTeX). babel provides a tool which has become standard in many distributions and based on a "configuration file" named `language.dat`. The exact way this file is used

---

[22]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.

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).[23] 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).[24]

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[25]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

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

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.[26] 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

---

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

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

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

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

the macros that produce texts. Also the language-switching possibility which has been built into the babel system has its implications.

The following assumptions are made:

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

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

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

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

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

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

Some recommendations:

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

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

- Avoid adding things to \noextras⟨*lang*⟩ except for umlauthigh and friends, \bbl@deactivate, \bbl@(non)frenchspacing, and language-specific macros. Use always, if possible, \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 \latintext is deprecated.[27]

- 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.

---

[27]But not removed, for backward compatibility.

3.1 Guidelines for contributed languages

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

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

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

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

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

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

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

The following page provides a starting point for `ldf` files: `http://www.texnia.com/incubator.html`. See also `https://latex3.github.io/babel/guides/list-of-locale-templates.html`.

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

3.2 Basic macros

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

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

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

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

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

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

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

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

\date⟨*lang*⟩	The macro \date⟨*lang*⟩ defines \today.
\extras⟨*lang*⟩	The macro \extras⟨*lang*⟩ contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
\noextras⟨*lang*⟩	Because we want to let the user switch between languages, but we do not know what state TEX might be in after the execution of \extras⟨*lang*⟩, a macro that brings TEX into a predefined state is needed. It will be no surprise that the name of this macro is \noextras⟨*lang*⟩.
\bbl@declare@ttribute	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
\main@language	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use \main@language instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.
\ProvidesLanguage	The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the LATEX command \ProvidesPackage.
\LdfInit	The macro \LdfInit performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the .ldf file from being processed twice, etc.
\ldf@quit	The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.
\ldf@finish	The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.
\loadlocalcfg	After processing a language definition file, LATEX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to \captions⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.
\substitutefontfamily	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .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}
\SetString\monthiname{<name of first month>}
% More strings

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

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

\EndBabelCommands

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

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

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

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

3.4 Support for active characters

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

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

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

\bbl@deactivate \bbl@activate 'switches on' the active behavior of the character. \bbl@deactivate lets the active character expand to its former (mostly) non-active self.

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

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

3.5   Support for saving macro definitions

Language definition files may want to *re*define macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this[28].

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

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

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

3.6   Support for extending macros

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

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

3.7   Macros common to a number of languages

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

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

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

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

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

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

3.8   Encoding-dependent strings

New 3.9a   Babel 3.9 provides a way of defining strings in several encodings, intended mainly for luatex and xetex. This is the only new feature requiring changes in language files if you want to make use of it.

Furthermore, it must be activated explicitly, with the package option strings. If there is no strings, these blocks are ignored, except \SetCases (and except if forced as described below). In other words, the old way of defining/switching strings still works and it's used by default.

It consist is a series of blocks started with \StartBabelCommands. The last block is closed with \EndBabelCommands. Each block is a single group (ie, local declarations apply until

---

[28]This mechanism was introduced by Bernd Raichle.

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

The ⟨*language-list*⟩ specifies which languages the block is intended for. A block is taken into account only if the \CurrentOption is listed here. Alternatively, you can define \BabelLanguages to a comma-separated list of languages to be defined (if undefined, \StartBabelCommands sets it to \CurrentOption). You may write \CurrentOption as the language, but this is discouraged – a explicit name (or names) is much better and clearer.

A "selector" is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name unicode must be used for xetex and luatex (the key strings has also other two special values: generic and encoded).

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

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

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

Blocks without a selector are read always if the key strings has been used. They provide fallback values, and therefore must be the last blocks; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks (mainly LGC scripts using the LICR). Blocks without a selector can be activated explicitly with strings=generic (no block is taken into account except those). With strings=encoded, strings in those blocks are set as default (internally, ?). With strings=encoded strings are protected, but they are correctly expanded in \MakeUppercase and the like. If there is no key strings, string definitions are ignored, but \SetCases are still honored (in a encoded way).

The ⟨*category*⟩ is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.[29] 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:

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

\StartBabelCommands{german,austrian}{date}
  [unicode, fontenc=TU EU1 EU2, charset=utf8]
```

---

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

```
    \SetString\monthiiiname{März}

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

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

\StartBabelCommands{german,austrian}{date}
    \SetString\monthiiname{Februar}
    \SetString\monthiiiname{M\"{a}rz}
    \SetString\monthivname{April}
    \SetString\monthvname{Mai}
    \SetString\monthviname{Juni}
    \SetString\monthviiname{Juli}
    \SetString\monthviiiname{August}
    \SetString\monthixname{September}
    \SetString\monthxname{Oktober}
    \SetString\monthxiname{November}
    \SetString\monthxiiname{Dezenber}
    \SetString\today{\number\day.~%
        \csname month\romannumeral\month name\endcsname\space
        \number\year}

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

\EndBabelCommands
```

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

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.[30]

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands {⟨*code*⟩}

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

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

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

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

---

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

58

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

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

#1 is replaced by the roman numeral.

\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 \@uclclist (eg, \bb\BB\cc\CC). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in LaTeX, we can set for Turkish:

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

\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetCase
  {\uccode`i=`İ\relax
   \uccode`ı=`I\relax}
  {\lccode`İ=`i\relax
   \lccode`I=`ı\relax}

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap {⟨*to-lower-macros*⟩}

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

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

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

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

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

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

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

3.9 Executing code based on the selector

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

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

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

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

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`).

4 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 appropiated places in the source code and shown below with ⟨⟨*name*⟩⟩. That brings a little bit of literate programming.

5 `locale` **directory**

A required component of babel is a set of `ini` files with basic definitions for about 200 languages. They are distributed as a separate `zip` file, not packed as `dtx`. With them, babel will fully support Unicode engines.
Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and polytonic Greek,

and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.

This is a preliminary documentation.

ini files contain the actual data; tex files are currently just proxies to the corresponding ini files. Most keys are self-explanatory.

charset the encoding used in the ini file.

version of the ini file

level "version" of the ini specification . which keys are available (they may grow in a compatible way) and how they should be read.

encodings a descriptive list of font encodings.

[captions] section of captions in the file charset

[captions.licr] same, but in pure ASCII using the LICR

date.long fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (eg, MMMM for the month name) and anything outside is text. In addition, [ ] is a non breakable space and [.] is an abbreviation dot.

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

6 Tools

1 ⟨⟨version=3.81.2883⟩⟩
2 ⟨⟨date=2022/10/07⟩⟩

Do not use the following macros in ldf **files. They may change in the future**. This applies mainly to those recently added for replacing, trimming and looping. The older ones, like \bbl@afterfi, will not change.

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

```
3 ⟨⟨*Basic macros⟩⟩ ≡
4 \bbl@trace{Basic macros}
5 \def\bbl@stripslash{\expandafter\@gobble\string}
6 \def\bbl@add#1#2{%
7   \bbl@ifunset{\bbl@stripslash#1}%
8     {\def#1{#2}}%
9     {\expandafter\def\expandafter#1\expandafter{#1#2}}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@carg#1#2{\expandafter#1\csname#2\endcsname}%
12 \def\bbl@ncarg#1#2#3{\expandafter#1\expandafter#2\csname#3\endcsname}%
13 \def\bbl@ccarg#1#2#3{%
14   \expandafter#1\csname#2\expandafter\endcsname\csname#3\endcsname}%
15 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
16 \def\bbl@cs#1{\csname bbl@#1\endcsname}
17 \def\bbl@cl#1{\csname bbl@#1@\languagename\endcsname}
18 \def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
19 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
20 \def\bbl@@loop#1#2#3,{%
21   \ifx\@nnil#3\relax\else
22     \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
23   \fi}
24 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
```

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

```
25 \def\bbl@add@list#1#2{%
26   \edef#1{%
```

```
27     \bbl@ifunset{\bbl@stripslash#1}%
28       {}%
29       {\ifx#1\@empty\else#1,\fi}%
30     #2}}
```

\bbl@afterelse   Because the code that is used in the handling of active characters may need to look ahead, we take
\bbl@afterfi   extra care to 'throw' it over the \else and \fi parts of an \if-statement[31]. These macros will break
                  if another \if...\fi statement appears in one of the arguments and it is not enclosed in braces.

```
31 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
32 \long\def\bbl@afterfi#1\fi{\fi#1}
```

\bbl@exp   Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple and
            readable. Here \\ stands for \noexpand, \<..> for \noexpand applied to a built macro name (which
            does not define the macro if undefined to \relax, because it is created locally), and \[..] for
            one-level expansion (where .. is the macro name without the backslash). The result may be
            followed by extra arguments, if necessary.

```
33 \def\bbl@exp#1{%
34   \begingroup
35     \let\\\noexpand
36     \let\<\bbl@exp@en
37     \let\[\bbl@exp@ue
38     \edef\bbl@exp@aux{\endgroup#1}%
39   \bbl@exp@aux}
40 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
41 \def\bbl@exp@ue#1]{%
42   \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%
```

\bbl@trim   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.

```
43 \def\bbl@tempa#1{%
44   \long\def\bbl@trim##1##2{%
45     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
46   \def\bbl@trim@c{%
47     \ifx\bbl@trim@a\@sptoken
48       \expandafter\bbl@trim@b
49     \else
50       \expandafter\bbl@trim@b\expandafter#1%
51     \fi}%
52   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}%
53 \bbl@tempa{ }
54 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
55 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}
```

\bbl@ifunset   To check if a macro is defined, we create a new macro, which does the same as \@ifundefined.
                However, in an ϵ-tex engine, it is based on \ifcsname, which is more efficient, and does not waste
                memory. Defined inside a group, to avoid \ifcsname being implicitly set to \relax by the \csname
                test.

```
56 \begingroup
57   \gdef\bbl@ifunset#1{%
58     \expandafter\ifx\csname#1\endcsname\relax
59       \expandafter\@firstoftwo
60     \else
61       \expandafter\@secondoftwo
62     \fi}
63   \bbl@ifunset{ifcsname}%
64     {}%
65     {\gdef\bbl@ifunset#1{%
66       \ifcsname#1\endcsname
```

---

[31]This code is based on code presented in TUGboat vol. 12, no2, June 1991 in "An expansion Power Lemma" by
Sonja Maus.

```
67            \expandafter\ifx\csname#1\endcsname\relax
68              \bbl@afterelse\expandafter\@firstoftwo
69            \else
70              \bbl@afterfi\expandafter\@secondoftwo
71            \fi
72          \else
73            \expandafter\@firstoftwo
74          \fi}}
75 \endgroup
```

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

```
76 \def\bbl@ifblank#1{%
77   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
78 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
79 \def\bbl@ifset#1#2#3{%
80   \bbl@ifunset{#1}{#3}{\bbl@exp{\\\bbl@ifblank{#1}}{#3}{#2}}}
```

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).

```
81 \def\bbl@forkv#1#2{%
82   \def\bbl@kvcmd##1##2##3{#2}%
83   \bbl@kvnext#1,\@nil,}
84 \def\bbl@kvnext#1,{%
85   \ifx\@nil#1\relax\else
86     \bbl@ifblank{#1}{}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
87     \expandafter\bbl@kvnext
88   \fi}
89 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
90   \bbl@trim@def\bbl@forkv@a{#1}%
91   \bbl@trim{\expandafter\bbl@kvcmd\expandafter{\bbl@forkv@a}}{#2}{#4}}
```

A *for* loop. Each item (trimmed), is #1. It cannot be nested (it's doable, but we don't need it).

```
92 \def\bbl@vforeach#1#2{%
93   \def\bbl@forcmd##1{#2}%
94   \bbl@fornext#1,\@nil,}
95 \def\bbl@fornext#1,{%
96   \ifx\@nil#1\relax\else
97     \bbl@ifblank{#1}{}{\bbl@trim\bbl@forcmd{#1}}%
98     \expandafter\bbl@fornext
99   \fi}
100 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

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

```
101 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
102   \toks@{}%
103   \def\bbl@replace@aux##1#2##2#2{%
104     \ifx\bbl@nil##2%
105       \toks@\expandafter{\the\toks@##1}%
106     \else
107       \toks@\expandafter{\the\toks@##1#3}%
108       \bbl@afterfi
109       \bbl@replace@aux##2#2%
110     \fi}%
111   \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
112   \edef#1{\the\toks@}}
```

An extension to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace elax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not* work is in \bbl@TG@@date, and also fails if there are macros with spaces, because they are

retokenized). It may change! (or even merged with \bbl@replace; I'm not sure ckecking the replacement is really necessary or just paranoia).

```
113 \ifx\detokenize\@undefined\else % Unused macros if old Plain TeX
114   \bbl@exp{\def\\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
115     \def\bbl@tempa{#1}%
116     \def\bbl@tempb{#2}%
117     \def\bbl@tempe{#3}}
118   \def\bbl@sreplace#1#2#3{%
119     \begingroup
120       \expandafter\bbl@parsedef\meaning#1\relax
121       \def\bbl@tempc{#2}%
122       \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
123       \def\bbl@tempd{#3}%
124       \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
125       \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
126       \ifin@
127         \bbl@exp{\\\bbl@replace\\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
128         \def\bbl@tempc{%      Expanded an executed below as 'uplevel'
129            \\\makeatletter % "internal" macros with @ are assumed
130            \\\scantokens{%
131              \bbl@tempa\\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
132            \catcode64=\the\catcode64\relax}%  Restore @
133       \else
134         \let\bbl@tempc\@empty  % Not \relax
135       \fi
136       \bbl@exp{%       For the 'uplevel' assignments
137     \endgroup
138       \bbl@tempc}}  % empty or expand to set #1 with changes
139 \fi
```

Two further tools. \bbl@ifsamestring 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 pdfTeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

```
140 \def\bbl@ifsamestring#1#2{%
141   \begingroup
142     \protected@edef\bbl@tempb{#1}%
143     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
144     \protected@edef\bbl@tempc{#2}%
145     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
146     \ifx\bbl@tempb\bbl@tempc
147       \aftergroup\@firstoftwo
148     \else
149       \aftergroup\@secondoftwo
150     \fi
151   \endgroup}
152 \chardef\bbl@engine=%
153   \ifx\directlua\@undefined
154     \ifx\XeTeXinputencoding\@undefined
155       \z@
156     \else
157       \tw@
158     \fi
159   \else
160     \@ne
161   \fi
```

A somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.

```
162 \def\bbl@bsphack{%
163   \ifhmode
164     \hskip\z@skip
165     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
166   \else
167     \let\bbl@esphack\@empty
```

```
168     \fi}
```

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

```
169 \def\bbl@cased{%
170   \ifx\oe\OE
171     \expandafter\in@\expandafter
172       {\expandafter\OE\expandafter}\expandafter{\oe}%
173     \ifin@
174       \bbl@afterelse\expandafter\MakeUppercase
175     \else
176       \bbl@afterfi\expandafter\MakeLowercase
177     \fi
178   \else
179     \expandafter\@firstofone
180   \fi}
```

An alternative to \IfFormatAtLeastTF for old versions. Temporary.

```
181 \ifx\IfFormatAtLeastTF\@undefined
182   \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
183 \else
184   \let\bbl@ifformatlater\IfFormatAtLeastTF
185 \fi
```

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).

```
186 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
187   \toks@\expandafter\expandafter\expandafter{%
188     \csname extras\languagename\endcsname}%
189   \bbl@exp{\\\in@{#1}{\the\toks@}}%
190   \ifin@\else
191     \@temptokena{#2}%
192     \edef\bbl@tempc{\the\@temptokena\the\toks@}%
193     \toks@\expandafter{\bbl@tempc#3}%
194     \expandafter\edef\csname extras\languagename\endcsname{\the\toks@}%
195   \fi}
196 ⟨⟨/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.

```
197 ⟨⟨*Make sure ProvidesFile is defined⟩⟩ ≡
198 \ifx\ProvidesFile\@undefined
199   \def\ProvidesFile#1[#2 #3 #4]{%
200     \wlog{File: #1 #4 #3 <#2>}%
201     \let\ProvidesFile\@undefined}
202 \fi
203 ⟨⟨/Make sure ProvidesFile is defined⟩⟩
```

6.1 Multiple languages

\language Plain TeX version 3.0 provides the primitive \language that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in switch.def and hyphen.cfg; the latter may seem redundant, but remember babel doesn't requires loading switch.def in the format.

```
204 ⟨⟨*Define core switching macros⟩⟩ ≡
205 \ifx\language\@undefined
206   \csname newcount\endcsname\language
207 \fi
208 ⟨⟨/Define core switching macros⟩⟩
```

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

\addlanguage This macro was introduced for TEX < 2. Preserved for compatibility.

```
209 ⟨⟨*Define core switching macros⟩⟩ ≡
210 \countdef\last@language=19
211 \def\addlanguage{\csname newlanguage\endcsname}
212 ⟨⟨/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 \orig@dump is used as a flag. Otherwise, we need to use the second part, so \orig@dump is not defined (plain.def undefines it).

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

6.2 The Package File (LATEX, `babel.sty`)

```
213 ⟨*package⟩
214 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
215 \ProvidesPackage{babel}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ The Babel package]
```

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

```
216 \@ifpackagewith{babel}{debug}
217   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
218    \let\bbl@debug\@firstofone
219    \ifx\directlua\@undefined\else
220      \directlua{ Babel = Babel or {}
221        Babel.debug = true }%
222      \input{babel-debug.tex}%
223    \fi}
224   {\providecommand\bbl@trace[1]{}%
225    \let\bbl@debug\@gobble
226    \ifx\directlua\@undefined\else
227      \directlua{ Babel = Babel or {}
228        Babel.debug = false }%
229    \fi}
230 \def\bbl@error#1#2{%
231   \begingroup
232     \def\\{\MessageBreak}%
233     \PackageError{babel}{#1}{#2}%
234   \endgroup}
235 \def\bbl@warning#1{%
236   \begingroup
237     \def\\{\MessageBreak}%
238     \PackageWarning{babel}{#1}%
239   \endgroup}
240 \def\bbl@infowarn#1{%
241   \begingroup
242     \def\\{\MessageBreak}%
243     \PackageNote{babel}{#1}%
244   \endgroup}
245 \def\bbl@info#1{%
246   \begingroup
247     \def\\{\MessageBreak}%
248     \PackageInfo{babel}{#1}%
249   \endgroup}
```

This file also takes care of a number of compatibility issues with other packages an defines a few aditional 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.

```
250 ⟨⟨Basic macros⟩⟩
```

```
251 \@ifpackagewith{babel}{silent}
252   {\let\bbl@info\@gobble
253    \let\bbl@infowarn\@gobble
254    \let\bbl@warning\@gobble}
255   {}
256 %
257 \def\AfterBabelLanguage#1{%
258   \global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}%
```

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 avaliable with base, because it just shows info.

```
259 \ifx\bbl@languages\@undefined\else
260   \begingroup
261     \catcode`\^^I=12
262     \@ifpackagewith{babel}{showlanguages}{%
263       \begingroup
264         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
265         \wlog{<*languages>}%
266         \bbl@languages
267         \wlog{</languages>}%
268       \endgroup}{}
269   \endgroup
270   \def\bbl@elt#1#2#3#4{%
271     \ifnum#2=\z@
272       \gdef\bbl@nulllanguage{#1}%
273       \def\bbl@elt##1##2##3##4{}%
274     \fi}%
275   \bbl@languages
276 \fi%
```

6.3 base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets ver@babel.sty so that LATEXforgets 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 interesed in the rest of babel.

```
277 \bbl@trace{Defining option 'base'}
278 \@ifpackagewith{babel}{base}{%
279   \let\bbl@onlyswitch\@empty
280   \let\bbl@provide@locale\relax
281   \input babel.def
282   \let\bbl@onlyswitch\@undefined
283   \ifx\directlua\@undefined
284     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
285   \else
286     \input luababel.def
287     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
288   \fi
289   \DeclareOption{base}{}%
290   \DeclareOption{showlanguages}{}%
291   \ProcessOptions
292   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
293   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
294   \global\let\@ifl@ter@@\@ifl@ter
295   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
296   \endinput}{}%
```

6.4 key=value **options and other general option**

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

```
297 \bbl@trace{key=value and another general options}
298 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
299 \def\bbl@tempb#1.#2{%  Remove trailing dot
300    #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
301 \def\bbl@tempd#1.#2\@nnil{%  TODO. Refactor lists?
302   \ifx\@empty#2%
303     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
304   \else
305     \in@{,provide=}{,#1}%
306     \ifin@
307       \edef\bbl@tempc{%
308         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
309     \else
310       \in@{=}{#1}%
311       \ifin@
312         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
313       \else
314         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
315         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
316       \fi
317     \fi
318   \fi}
319 \let\bbl@tempc\@empty
320 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
321 \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* the default as it can cause problems with other packages, but for those who want to use the shorthand characters in the preamble of their documents this can help.

```
322 \DeclareOption{KeepShorthandsActive}{}
323 \DeclareOption{activeacute}{}
324 \DeclareOption{activegrave}{}
325 \DeclareOption{debug}{}
326 \DeclareOption{noconfigs}{}
327 \DeclareOption{showlanguages}{}
328 \DeclareOption{silent}{}
329 % \DeclareOption{mono}{}
330 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
331 \chardef\bbl@iniflag\z@
332 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}     % main -> +1
333 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@}    % add = 2
334 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
335 % A separate option
336 \let\bbl@autoload@options\@empty
337 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
338 % Don't use. Experimental. TODO.
339 \newif\ifbbl@single
340 \DeclareOption{selectors=off}{\bbl@singletrue}
341 ⟨⟨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.

```
342 \let\bbl@opt@shorthands\@nnil
343 \let\bbl@opt@config\@nnil
344 \let\bbl@opt@main\@nnil
345 \let\bbl@opt@headfoot\@nnil
346 \let\bbl@opt@layout\@nnil
347 \let\bbl@opt@provide\@nnil
```

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

```
348 \def\bbl@tempa#1=#2\bbl@tempa{%
349   \bbl@csarg\ifx{opt@#1}\@nnil
```

68

```
350     \bbl@csarg\edef{opt@#1}{#2}%
351   \else
352     \bbl@error
353       {Bad option '#1=#2'. Either you have misspelled the\\%
354        key or there is a previous setting of '#1'. Valid\\%
355        keys are, among others, 'shorthands', 'main', 'bidi',\\%
356        'strings', 'config', 'headfoot', 'safe', 'math'.}%
357       {See the manual for further details.}
358   \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.

```
359 \let\bbl@language@opts\@empty
360 \DeclareOption*{%
361   \bbl@xin@{\string=}{\CurrentOption}%
362   \ifin@
363     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
364   \else
365     \bbl@add@list\bbl@language@opts{\CurrentOption}%
366   \fi}
```

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

```
367 \ProcessOptions*

368 \ifx\bbl@opt@provide\@nnil
369   \let\bbl@opt@provide\@empty   % %%% MOVE above
370 \else
371   \chardef\bbl@iniflag\@ne
372   \bbl@exp{\\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
373     \in@{,provide,}{,#1,}%
374     \ifin@
375       \def\bbl@opt@provide{#2}%
376       \bbl@replace\bbl@opt@provide{;}{,}%
377     \fi}
378 \fi
379 %
```

6.5   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=....

```
380 \bbl@trace{Conditional loading of shorthands}
381 \def\bbl@sh@string#1{%
382   \ifx#1\@empty\else
383     \ifx#1t\string~%
384     \else\ifx#1c\string,%
385     \else\string#1%
386     \fi\fi
387     \expandafter\bbl@sh@string
388   \fi}
389 \ifx\bbl@opt@shorthands\@nnil
390   \def\bbl@ifshorthand#1#2#3{#2}%
391 \else\ifx\bbl@opt@shorthands\@empty
392   \def\bbl@ifshorthand#1#2#3{#3}%
393 \else
```

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

```
394   \def\bbl@ifshorthand#1{%
395     \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
396     \ifin@
397       \expandafter\@firstoftwo
```

```
398     \else
399       \expandafter\@secondoftwo
400     \fi}
```

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

```
401   \edef\bbl@opt@shorthands{%
402     \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%
```

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

```
403   \bbl@ifshorthand{'}%
404     {\PassOptionsToPackage{activeacute}{babel}}{}
405   \bbl@ifshorthand{`}%
406     {\PassOptionsToPackage{activegrave}{babel}}{}
407 \fi\fi
```

With headfoot=lang we can set the language used in heads/foots. For example, in babel/3796 just adds headfoot=english. It misuses \@resetactivechars but seems to work.

```
408 \ifx\bbl@opt@headfoot\@nnil\else
409   \g@addto@macro\@resetactivechars{%
410     \set@typeset@protect
411     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
412     \let\protect\noexpand}
413 \fi
```

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 currently set, but in a future release it will be set to none.

```
414 \ifx\bbl@opt@safe\@undefined
415   \def\bbl@opt@safe{BR}
416   % \let\bbl@opt@safe\@empty % Pending of \cite
417 \fi
```

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

```
418 \bbl@trace{Defining IfBabelLayout}
419 \ifx\bbl@opt@layout\@nnil
420   \newcommand\IfBabelLayout[3]{#3}%
421 \else
422   \newcommand\IfBabelLayout[1]{%
423     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
424     \ifin@
425       \expandafter\@firstoftwo
426     \else
427       \expandafter\@secondoftwo
428     \fi}
429 \fi
430 ⟨/package⟩
431 ⟨∗core⟩
```

6.6 Interlude for Plain

Because of the way docstrip works, we need to insert some code for Plain here. However, the tools provided by the babel installer for literate programming makes this section a short interlude, because the actual code is below, tagged as *Emulate LaTeX*.

```
432 \ifx\ldf@quit\@undefined\else
433 \endinput\fi % Same line!
434 ⟨⟨Make sure ProvidesFile is defined⟩⟩
435 \ProvidesFile{babel.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel common definitions]
436 \ifx\AtBeginDocument\@undefined  % TODO. change test.
437 ⟨⟨Emulate LaTeX⟩⟩
438 \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.

```
439 ⟨/core⟩
440 ⟨∗package | core⟩
```

7 Multiple languages

This is not a separate file (`switch.def`) anymore.

Plain TeX version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```
441 \def\bbl@version{⟨⟨version⟩⟩}
442 \def\bbl@date{⟨⟨date⟩⟩}
443 ⟨⟨Define core switching macros⟩⟩
```

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

```
444 \def\adddialect#1#2{%
445   \global\chardef#1#2\relax
446   \bbl@usehooks{adddialect}{{#1}{#2}}%
447   \begingroup
448     \count@#1\relax
449     \def\bbl@elt##1##2##3##4{%
450       \ifnum\count@=##2\relax
451         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
452         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
453                   set to \expandafter\string\csname l@##1\endcsname\\%
454                   (\string\language\the\count@). Reported}%
455         \def\bbl@elt####1####2####3####4{}%
456       \fi}%
457     \bbl@cs{languages}%
458   \endgroup}
```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an error.

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

```
459 \def\bbl@fixname#1{%
460   \begingroup
461     \def\bbl@tempe{l@}%
462     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
463     \bbl@tempd
464       {\lowercase\expandafter{\bbl@tempd}%
465         {\uppercase\expandafter{\bbl@tempd}%
466           \@empty
467           {\edef\bbl@tempd{\def\noexpand#1{#1}}%
468            \uppercase\expandafter{\bbl@tempd}}}%
469         {\edef\bbl@tempd{\def\noexpand#1{#1}}%
470          \lowercase\expandafter{\bbl@tempd}}}%
471       \@empty
472     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
473   \bbl@tempd
474   \bbl@exp{\\\bbl@usehooks{languagename}{{\languagename}{#1}}}}
475 \def\bbl@iflanguage#1{%
476   \@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`.

71

```
477 \def\bbl@bcpcase#1#2#3#4\@@#5{%
478   \ifx\@empty#3%
479     \uppercase{\def#5{#1#2}}%
480   \else
481     \uppercase{\def#5{#1}}%
482     \lowercase{\edef#5{#5#2#3#4}}%
483   \fi}
484 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
485   \let\bbl@bcp\relax
486   \lowercase{\def\bbl@tempa{#1}}%
487   \ifx\@empty#2%
488     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
489   \else\ifx\@empty#3%
490     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
491     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
492       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
493       {}%
494     \ifx\bbl@bcp\relax
495       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
496     \fi
497   \else
498     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
499     \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
500     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
501       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
502       {}%
503     \ifx\bbl@bcp\relax
504       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
505         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
506         {}%
507     \fi
508     \ifx\bbl@bcp\relax
509       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
510         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
511         {}%
512     \fi
513     \ifx\bbl@bcp\relax
514       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
515     \fi
516   \fi\fi}
517 \let\bbl@initoload\relax
518 \def\bbl@provide@locale{%
519   \ifx\babelprovide\@undefined
520     \bbl@error{For a language to be defined on the fly 'base'\\%
521                 is not enough, and the whole package must be\\%
522                 loaded. Either delete the 'base' option or\\%
523                 request the languages explicitly}%
524                {See the manual for further details.}%
525   \fi
526   \let\bbl@auxname\languagename % Still necessary. TODO
527   \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
528     {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
529   \ifbbl@bcpallowed
530     \expandafter\ifx\csname date\languagename\endcsname\relax
531       \expandafter
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{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
539           \let\bbl@initoload\relax
```

```
540        \fi
541        \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
542      \fi
543    \fi
544  \fi
545  \expandafter\ifx\csname date\languagename\endcsname\relax
546    \IfFileExists{babel-\languagename.tex}%
547      {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
548      {}%
549  \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.

```
550 \def\iflanguage#1{%
551  \bbl@iflanguage{#1}{%
552    \ifnum\csname l@#1\endcsname=\language
553      \expandafter\@firstoftwo
554    \else
555      \expandafter\@secondoftwo
556    \fi}}
```

7.1   Selecting the language

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

```
557 \let\bbl@select@type\z@
558 \edef\selectlanguage{%
559  \noexpand\protect
560  \expandafter\noexpand\csname selectlanguage \endcsname}
```

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.

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

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

```
562 \let\xstring\string
```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

\bbl@pop@language   *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 TEX'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.

\bbl@language@stack   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.

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

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

\bbl@push@language   The stack is simply a list of languagenames, separated with a '+' sign; the push function can be simple:
\bbl@pop@language
```
564 \def\bbl@push@language{%
565  \ifx\languagename\@undefined\else
566    \ifx\currentgrouplevel\@undefined
567      \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
```

```
568       \else
569         \ifnum\currentgrouplevel=\z@
570           \xdef\bbl@language@stack{\languagename+}%
571         \else
572           \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
573         \fi
574       \fi
575     \fi}
```

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

\bbl@pop@lang 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.

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

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is
called in the following way. This means that before \bbl@pop@lang is executed TEX 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).

```
579 \let\bbl@ifrestoring\@secondoftwo
580 \def\bbl@pop@language{%
581   \expandafter\bbl@pop@lang\bbl@language@stack\@@
582   \let\bbl@ifrestoring\@firstoftwo
583   \expandafter\bbl@set@language\expandafter{\languagename}%
584   \let\bbl@ifrestoring\@secondoftwo}
```

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).

```
585 \chardef\localeid\z@
586 \def\bbl@id@last{0}     % No real need for a new counter
587 \def\bbl@id@assign{%
588   \bbl@ifunset{bbl@id@@\languagename}%
589     {\count@\bbl@id@last\relax
590      \advance\count@\@ne
591      \bbl@csarg\chardef{id@@\languagename}\count@
592      \edef\bbl@id@last{\the\count@}%
593      \ifcase\bbl@engine\or
594        \directlua{
595          Babel = Babel or {}
596          Babel.locale_props = Babel.locale_props or {}
597          Babel.locale_props[\bbl@id@last] = {}
598          Babel.locale_props[\bbl@id@last].name = '\languagename'
599        }%
600      \fi}%
601     {}%
602   \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of \selectlanguage.

```
603 \expandafter\def\csname selectlanguage \endcsname#1{%
604   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
605   \bbl@push@language
606   \aftergroup\bbl@pop@language
607   \bbl@set@language{#1}}
```

74

\bbl@set@language The macro \bbl@set@language takes care of switching the language environment *and* of writing entries on the auxiliary files. For historial 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).

```
608 \def\BabelContentsFiles{toc,lof,lot}
609 \def\bbl@set@language#1{% from selectlanguage, pop@
610   % The old buggy way. Preserved for compatibility.
611   \edef\languagename{%
612     \ifnum\escapechar=\expandafter`\string#1\@empty
613     \else\string#1\@empty\fi}%
614   \ifcat\relax\noexpand#1%
615     \expandafter\ifx\csname date\languagename\endcsname\relax
616       \edef\languagename{#1}%
617       \let\localename\languagename
618     \else
619       \bbl@info{Using '\string\language' instead of 'language' is\\%
620                 deprecated. If what you want is to use a\\%
621                 macro containing the actual locale, make\\%
622                 sure it does not not match any language.\\%
623                 Reported}%
624     \ifx\scantokens\@undefined
625        \def\localename{??}%
626     \else
627       \scantokens\expandafter{\expandafter
628         \def\expandafter\localename\expandafter{\languagename}}%
629     \fi
630    \fi
631   \else
632     \def\localename{#1}% This one has the correct catcodes
633   \fi
634   \select@language{\languagename}%
635   % write to auxs
636   \expandafter\ifx\csname date\languagename\endcsname\relax\else
637     \if@filesw
638       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
639         \bbl@savelastskip
640         \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
641         \bbl@restorelastskip
642       \fi
643       \bbl@usehooks{write}{}%
644     \fi
645   \fi}
646 %
647 \let\bbl@restorelastskip\relax
648 \let\bbl@savelastskip\relax
649 %
650 \newif\ifbbl@bcpallowed
651 \bbl@bcpallowedfalse
652 \def\select@language#1{% from set@, babel@aux
653   \ifx\bbl@selectorname\@empty
654     \def\bbl@selectorname{select}%
655   % set hymap
656   \fi
657   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
658   % set name
```

```
659    \edef\languagename{#1}%
660    \bbl@fixname\languagename
661    % TODO. name@map must be here?
662    \bbl@provide@locale
663    \bbl@iflanguage\languagename{%
664      \let\bbl@select@type\z@
665      \expandafter\bbl@switch\expandafter{\languagename}}}
666 \def\babel@aux#1#2{%
667    \select@language{#1}%
668    \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
669      \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
670 \def\babel@toc#1#2{%
671    \select@language{#1}}
```

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 *re*define \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.

```
672 \newif\ifbbl@usedategroup
673 \def\bbl@switch#1{%  from select@, foreign@
674   % make sure there is info for the language if so requested
675   \bbl@ensureinfo{#1}%
676   % restore
677   \originalTeX
678   \expandafter\def\expandafter\originalTeX\expandafter{%
679     \csname noextras#1\endcsname
680     \let\originalTeX\@empty
681     \babel@beginsave}%
682   \bbl@usehooks{afterreset}{}%
683   \languageshorthands{none}%
684   % set the locale id
685   \bbl@id@assign
686   % switch captions, date
687   % No text is supposed to be added here, so we remove any
688   % spurious spaces.
689   \bbl@bsphack
690     \ifcase\bbl@select@type
691       \csname captions#1\endcsname\relax
692       \csname date#1\endcsname\relax
693     \else
694       \bbl@xin@{,captions,}{,\bbl@select@opts,}%
695       \ifin@
696         \csname captions#1\endcsname\relax
697       \fi
698       \bbl@xin@{,date,}{,\bbl@select@opts,}%
699       \ifin@  % if \foreign... within \<lang>date
700         \csname date#1\endcsname\relax
701       \fi
702     \fi
703   \bbl@esphack
704   % switch extras
705   \bbl@usehooks{beforeextras}{}%
706   \csname extras#1\endcsname\relax
707   \bbl@usehooks{afterextras}{}%
708   %  > babel-ensure
```

```
709  %  > babel-sh-<short>
710  %  > babel-bidi
711  %  > babel-fontspec
712  % hyphenation - case mapping
713  \ifcase\bbl@opt@hyphenmap\or
714    \def\BabelLower##1##2{\lccode##1=##2\relax}%
715    \ifnum\bbl@hymapsel>4\else
716      \csname\languagename @bbl@hyphenmap\endcsname
717    \fi
718    \chardef\bbl@opt@hyphenmap\z@
719  \else
720    \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
721      \csname\languagename @bbl@hyphenmap\endcsname
722    \fi
723  \fi
724  \let\bbl@hymapsel\@cclv
725  % hyphenation - select rules
726  \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
727    \edef\bbl@tempa{u}%
728  \else
729    \edef\bbl@tempa{\bbl@cl{lnbrk}}%
730  \fi
731  % linebreaking - handle u, e, k (v in the future)
732  \bbl@xin@{/u}{/\bbl@tempa}%
733  \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
734  \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
735  \ifin@\else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
736  \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
737  \ifin@
738    % unhyphenated/kashida/elongated/padding = allow stretching
739    \language\l@unhyphenated
740    \babel@savevariable\emergencystretch
741    \emergencystretch\maxdimen
742    \babel@savevariable\hbadness
743    \hbadness\@M
744  \else
745    % other = select patterns
746    \bbl@patterns{#1}%
747  \fi
748  % hyphenation - mins
749  \babel@savevariable\lefthyphenmin
750  \babel@savevariable\righthyphenmin
751  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
752    \set@hyphenmins\tw@\thr@@\relax
753  \else
754    \expandafter\expandafter\expandafter\set@hyphenmins
755      \csname #1hyphenmins\endcsname\relax
756  \fi
757  \let\bbl@selectorname\@empty}
```

otherlanguage (*env.*) 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.

```
758  \long\def\otherlanguage#1{%
759    \def\bbl@selectorname{other}%
760    \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
761    \csname selectlanguage \endcsname{#1}%
762    \ignorespaces}
```

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

```
763 \long\def\endotherlanguage{%
764   \global\@ignoretrue\ignorespaces}
```

otherlanguage* (*env.*) 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 \foreign@language.

```
765 \expandafter\def\csname otherlanguage*\endcsname{%
766   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
767 \def\bbl@otherlanguage@s[#1]#2{%
768   \def\bbl@selectorname{other*}%
769   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
770   \def\bbl@select@opts{#1}%
771   \foreign@language{#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".

```
772 \expandafter\let\csname endotherlanguage*\endcsname\relax
```

\foreignlanguage The \foreignlanguage command is another substitute for the \selectlanguage command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.

Unlike \selectlanguage this command doesn't switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the \extras⟨*lang*⟩ command doesn't make any \global changes. The coding is very similar to part of \selectlanguage.

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

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

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

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

```
773 \providecommand\bbl@beforeforeign{}
774 \edef\foreignlanguage{%
775   \noexpand\protect
776   \expandafter\noexpand\csname foreignlanguage \endcsname}
777 \expandafter\def\csname foreignlanguage \endcsname{%
778   \@ifstar\bbl@foreign@s\bbl@foreign@x}
779 \providecommand\bbl@foreign@x[3][]{%
780   \begingroup
781     \def\bbl@selectorname{foreign}%
782     \def\bbl@select@opts{#1}%
783     \let\BabelText\@firstofone
784     \bbl@beforeforeign
785     \foreign@language{#2}%
786     \bbl@usehooks{foreign}{}%
787     \BabelText{#3}% Now in horizontal mode!
788   \endgroup}
789 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
790   \begingroup
791     {\par}%
792     \def\bbl@selectorname{foreign*}%
793     \let\bbl@select@opts\@empty
794     \let\BabelText\@firstofone
795     \foreign@language{#1}%
796     \bbl@usehooks{foreign*}{}%
```

78

```
797    \bbl@dirparastext
798    \BabelText{#2}% Still in vertical mode!
799    {\par}%
800    \endgroup}
```

\foreign@language   This macro does the work for \foreignlanguage and the otherlanguage* 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.

```
801 \def\foreign@language#1{%
802   % set name
803   \edef\languagename{#1}%
804   \ifbbl@usedategroup
805     \bbl@add\bbl@select@opts{,date,}%
806     \bbl@usedategroupfalse
807   \fi
808   \bbl@fixname\languagename
809   % TODO. name@map here?
810   \bbl@provide@locale
811   \bbl@iflanguage\languagename{%
812     \let\bbl@select@type\@ne
813     \expandafter\bbl@switch\expandafter{\languagename}}}
```

The following macro executes conditionally some code based on the selector being used.

```
814 \def\IfBabelSelectorTF#1{%
815   \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
816   \ifin@
817     \expandafter\@firstoftwo
818   \else
819     \expandafter\@secondoftwo
820   \fi}
```

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

```
821 \let\bbl@hyphlist\@empty
822 \let\bbl@hyphenation@\relax
823 \let\bbl@pttnlist\@empty
824 \let\bbl@patterns@\relax
825 \let\bbl@hymapsel=\@cclv
826 \def\bbl@patterns#1{%
827   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
828       \csname l@#1\endcsname
829       \edef\bbl@tempa{#1}%
830     \else
831       \csname l@#1:\f@encoding\endcsname
832       \edef\bbl@tempa{#1:\f@encoding}%
833     \fi
834   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
835   % > luatex
836   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
837     \begingroup
838       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
839       \ifin@\else
840         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
841         \hyphenation{%
842           \bbl@hyphenation@
843           \@ifundefined{bbl@hyphenation@#1}%
844             \@empty
```

79

```
845          {\space\csname bbl@hyphenation@#1\endcsname}}%
846        \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
847      \fi
848    \endgroup}}
```

hyphenrules (*env.*) The environment hyphenrules can be used to select *just* the hyphenation rules. This environment does *not* change \languagename and when the hyphenation rules specified were not loaded it has no effect. Note however, \lccode's and font encodings are not set at all, so in most cases you should use otherlanguage*.

```
849 \def\hyphenrules#1{%
850   \edef\bbl@tempf{#1}%
851   \bbl@fixname\bbl@tempf
852   \bbl@iflanguage\bbl@tempf{%
853     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
854     \ifx\languageshorthands\@undefined\else
855       \languageshorthands{none}%
856     \fi
857     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
858       \set@hyphenmins\tw@\thr@@\relax
859     \else
860       \expandafter\expandafter\expandafter\set@hyphenmins
861       \csname\bbl@tempf hyphenmins\endcsname\relax
862     \fi}}
863 \let\endhyphenrules\@empty
```

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

```
864 \def\providehyphenmins#1#2{%
865   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
866     \@namedef{#1hyphenmins}{#2}%
867   \fi}
```

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

```
868 \def\set@hyphenmins#1#2{%
869   \lefthyphenmin#1\relax
870   \righthyphenmin#2\relax}
```

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

```
871 \ifx\ProvidesFile\@undefined
872  \def\ProvidesLanguage#1[#2 #3 #4]{%
873    \wlog{Language: #1 #4 #3 <#2>}%
874    }
875 \else
876  \def\ProvidesLanguage#1{%
877    \begingroup
878      \catcode`\ 10 %
879      \@makeother\/%
880      \@ifnextchar[%
881        {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
882  \def\@provideslanguage#1[#2]{%
883    \wlog{Language: #1 #2}%
884    \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
885    \endgroup}
886 \fi
```

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

```
887 \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.

```
888 \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':

```
889 \providecommand\setlocale{%
890   \bbl@error
891     {Not yet available}%
892     {Find an armchair, sit down and wait}}
893 \let\uselocale\setlocale
894 \let\locale\setlocale
895 \let\selectlocale\setlocale
896 \let\textlocale\setlocale
897 \let\textlanguage\setlocale
898 \let\languagetext\setlocale
```

7.2   Errors

\@nolanerr The babel package will signal an error when a documents tries to select a language that hasn't been
\@nopatterns 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.

\@noopterr 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 2_ε, 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.

```
899 \edef\bbl@nulllanguage{\string\language=0}
900 \def\bbl@nocaption{\protect\bbl@nocaption@i}
901 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
902   \global\@namedef{#2}{\textbf{?#1?}}%
903   \@nameuse{#2}%
904   \edef\bbl@tempa{#1}%
905   \bbl@sreplace\bbl@tempa{name}{}%
906   \bbl@warning{%
907     \@backslashchar#1 not set for '\languagename'. Please,\\%
908     define it after the language has been loaded\\%
909     (typically in the preamble) with:\\%
910     \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
911     Feel free to contribute on github.com/latex3/babel.\\%
912     Reported}}
913 \def\bbl@tentative{\protect\bbl@tentative@i}
914 \def\bbl@tentative@i#1{%
915   \bbl@warning{%
916     Some functions for '#1' are tentative.\\%
917     They might not work as expected and their behavior\\%
918     could change in the future.\\%
919     Reported}}
920 \def\@nolanerr#1{%
921   \bbl@error
922     {You haven't defined the language '#1' yet.\\%
923      Perhaps you misspelled it or your installation\\%
924      is not complete}%
925     {Your command will be ignored, type <return> to proceed}}
926 \def\@nopatterns#1{%
927   \bbl@warning
928     {No hyphenation patterns were preloaded for\\%
929      the language '#1' into the format.\\%
930      Please, configure your TeX system to add them and\\%
931      rebuild the format. Now I will use the patterns\\%
```

```
932       preloaded for \bbl@nulllanguage\space instead}}
933 \let\bbl@usehooks\@gobbletwo
934 \ifx\bbl@onlyswitch\@empty\endinput\fi
935   % Here ended switch.def
```

Here ended the now discarded switch.def. Here also (currently) ends the base option.

```
936 \ifx\directlua\@undefined\else
937   \ifx\bbl@luapatterns\@undefined
938     \input luababel.def
939   \fi
940 \fi
941 ⟨⟨Basic macros⟩⟩
942 \bbl@trace{Compatibility with language.def}
943 \ifx\bbl@languages\@undefined
944   \ifx\directlua\@undefined
945     \openin1 = language.def % TODO. Remove hardcoded number
946     \ifeof1
947       \closein1
948       \message{I couldn't find the file language.def}
949     \else
950       \closein1
951       \begingroup
952         \def\addlanguage#1#2#3#4#5{%
953           \expandafter\ifx\csname lang@#1\endcsname\relax\else
954             \global\expandafter\let\csname l@#1\expandafter\endcsname
955               \csname lang@#1\endcsname
956           \fi}%
957         \def\uselanguage#1{}%
958         \input language.def
959       \endgroup
960     \fi
961   \fi
962   \chardef\l@english\z@
963 \fi
```

\addto It takes two arguments, a ⟨control sequence⟩ and TeX-code to be added to the ⟨control sequence⟩.
If the ⟨control sequence⟩ has not been defined before it is defined now. The control sequence could
also expand to \relax, in which case a circular definition results. The net result is a stack overflow.
Note there is an inconsistency, because the assignment in the last branch is global.

```
964 \def\addto#1#2{%
965   \ifx#1\@undefined
966     \def#1{#2}%
967   \else
968     \ifx#1\relax
969       \def#1{#2}%
970     \else
971       {\toks@\expandafter{#1#2}%
972        \xdef#1{\the\toks@}}%
973     \fi
974   \fi}
```

The macro \initiate@active@char below takes all the necessary actions to make its argument a
shorthand character. The real work is performed once for each character. But first we define a little
tool.

```
975 \def\bbl@withactive#1#2{%
976   \begingroup
977     \lccode`~=`#2\relax
978     \lowercase{\endgroup#1~}}
```

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

```
979 \def\bbl@redefine#1{%
980   \edef\bbl@tempa{\bbl@stripslash#1}%
981   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
982   \expandafter\def\csname\bbl@tempa\endcsname}
983 \@onlypreamble\bbl@redefine
```

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

```
984 \def\bbl@redefine@long#1{%
985   \edef\bbl@tempa{\bbl@stripslash#1}%
986   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
987   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
988 \@onlypreamble\bbl@redefine@long
```

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

```
989 \def\bbl@redefinerobust#1{%
990   \edef\bbl@tempa{\bbl@stripslash#1}%
991   \bbl@ifunset{\bbl@tempa\space}%
992     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
993      \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
994     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
995   \@namedef{\bbl@tempa\space}}
996 \@onlypreamble\bbl@redefinerobust
```

7.3 Hooks

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

```
997 \bbl@trace{Hooks}
998 \newcommand\AddBabelHook[3][]{%
999   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1000  \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1001  \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1002  \bbl@ifunset{bbl@ev@#2@#3@#1}%
1003    {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1004    {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1005  \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1006 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1007 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1008 \def\bbl@usehooks#1#2{%
1009  \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1010  \def\bbl@elth##1{%
1011    \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1012  \bbl@cs{ev@#1@}%
1013  \ifx\languagename\@undefined\else % Test required for Plain (?)
1014    \ifx\UseHook\@undefined\else\UseHook{babel/\languagename/#1}\fi
1015    \def\bbl@elth##1{%
1016      \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1017    \bbl@cl{ev@#1}%
1018  \fi}
```

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).

```
1019 \def\bbl@evargs{,% <-- don't delete this comma
1020  everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1021  adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1022  beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1023  hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
```

83

```
1024     beforestart=0,languagename=2}
1025 \ifx\NewHook\@undefined\else
1026   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1027   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1028 \fi
```

\babelensure   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 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.

```
1029 \bbl@trace{Defining babelensure}
1030 \newcommand\babelensure[2][]{%
1031   \AddBabelHook{babel-ensure}{afterextras}{%
1032     \ifcase\bbl@select@type
1033       \bbl@cl{e}%
1034     \fi}%
1035   \begingroup
1036     \let\bbl@ens@include\@empty
1037     \let\bbl@ens@exclude\@empty
1038     \def\bbl@ens@fontenc{\relax}%
1039     \def\bbl@tempb##1{%
1040       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1041     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1042     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1043     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1044     \def\bbl@tempc{\bbl@ensure}%
1045     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1046       \expandafter{\bbl@ens@include}}%
1047     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1048       \expandafter{\bbl@ens@exclude}}%
1049     \toks@\expandafter{\bbl@tempc}%
1050     \bbl@exp{%
1051   \endgroup
1052   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
1053 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1054   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1055     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1056       \edef##1{\noexpand\bbl@nocaption
1057         {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1058     \fi
1059     \ifx##1\@empty\else
1060       \in@{##1}{#2}%
1061       \ifin@\else
1062         \bbl@ifunset{bbl@ensure@\languagename}%
1063           {\bbl@exp{%
1064             \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1065               \\\foreignlanguage{\languagename}%
1066               {\ifx\relax#3\else
1067                 \\\fontencoding{#3}\\\selectfont
1068               \fi
1069               ########1}}}}%
1070           {}%
1071         \toks@\expandafter{##1}%
1072         \edef##1{%
1073           \bbl@csarg\noexpand{ensure@\languagename}%
1074           {\the\toks@}}%
1075       \fi
1076       \expandafter\bbl@tempb
```

```
1077     \fi}%
1078  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1079  \def\bbl@tempa##1{% elt for include list
1080     \ifx##1\@empty\else
1081        \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1082        \ifin@\else
1083           \bbl@tempb##1\@empty
1084        \fi
1085        \expandafter\bbl@tempa
1086     \fi}%
1087  \bbl@tempa#1\@empty}
1088 \def\bbl@captionslist{%
1089  \prefacename\refname\abstractname\bibname\chaptername\appendixname
1090  \contentsname\listfigurename\listtablename\indexname\figurename
1091  \tablename\partname\enclname\ccname\headtoname\pagename\seename
1092  \alsoname\proofname\glossaryname}
```

7.4 Setting up language files

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

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

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

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

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

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

Finally we check \originalTeX.

```
1093 \bbl@trace{Macros for setting language files up}
1094 \def\bbl@ldfinit{%
1095  \let\bbl@screset\@empty
1096  \let\BabelStrings\bbl@opt@string
1097  \let\BabelOptions\@empty
1098  \let\BabelLanguages\relax
1099  \ifx\originalTeX\@undefined
1100     \let\originalTeX\@empty
1101  \else
1102     \originalTeX
1103  \fi}
1104 \def\LdfInit#1#2{%
1105  \chardef\atcatcode=\catcode`\@
1106  \catcode`\@=11\relax
1107  \chardef\eqcatcode=\catcode`\=
1108  \catcode`\==12\relax
1109  \expandafter\if\expandafter\@backslashchar
1110                 \expandafter\@car\string#2\@nil
1111     \ifx#2\@undefined\else
1112        \ldf@quit{#1}%
1113     \fi
1114  \else
1115     \expandafter\ifx\csname#2\endcsname\relax\else
1116        \ldf@quit{#1}%
1117     \fi
1118  \fi
```

```
1119    \bbl@ldfinit}
```

\ldf@quit   This macro interrupts the processing of a language definition file.

```
1120 \def\ldf@quit#1{%
1121    \expandafter\main@language\expandafter{#1}%
1122    \catcode`\@=\atcatcode \let\atcatcode\relax
1123    \catcode`\==\eqcatcode \let\eqcatcode\relax
1124    \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.

```
1125 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1126    \bbl@afterlang
1127    \let\bbl@afterlang\relax
1128    \let\BabelModifiers\relax
1129    \let\bbl@screset\relax}%
1130 \def\ldf@finish#1{%
1131    \loadlocalcfg{#1}%
1132    \bbl@afterldf{#1}%
1133    \expandafter\main@language\expandafter{#1}%
1134    \catcode`\@=\atcatcode \let\atcatcode\relax
1135    \catcode`\==\eqcatcode \let\eqcatcode\relax}
```

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

```
1136 \@onlypreamble\LdfInit
1137 \@onlypreamble\ldf@quit
1138 \@onlypreamble\ldf@finish
```

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

```
1139 \def\main@language#1{%
1140    \def\bbl@main@language{#1}%
1141    \let\languagename\bbl@main@language % TODO. Set localename
1142    \bbl@id@assign
1143    \bbl@patterns{\languagename}}
```

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.

```
1144 \def\bbl@beforestart{%
1145    \def\@nolanerr##1{%
1146       \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1147    \bbl@usehooks{beforestart}{}%
1148    \global\let\bbl@beforestart\relax}
1149 \AtBeginDocument{%
1150    {\@nameuse{bbl@beforestart}}%  Group!
1151    \if@filesw
1152       \providecommand\babel@aux[2]{}%
1153       \immediate\write\@mainaux{%
1154          \string\providecommand\string\babel@aux[2]{}}%
1155       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1156    \fi
1157    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1158    \ifbbl@single  % must go after the line above.
1159       \renewcommand\selectlanguage[1]{}%
1160       \renewcommand\foreignlanguage[2]{#2}%
1161       \global\let\babel@aux\@gobbletwo  % Also as flag
1162    \fi
1163    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1164 \def\select@language@x#1{%
1165   \ifcase\bbl@select@type
1166     \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1167   \else
1168     \select@language{#1}%
1169   \fi}
```

7.5 Shorthands

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

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

```
1170 \bbl@trace{Shorhands}
1171 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1172   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1173   \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1174   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1175     \begingroup
1176       \catcode`#1\active
1177       \nfss@catcodes
1178       \ifnum\catcode`#1=\active
1179         \endgroup
1180         \bbl@add\nfss@catcodes{\@makeother#1}%
1181       \else
1182         \endgroup
1183       \fi
1184   \fi}
```

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

```
1185 \def\bbl@remove@special#1{%
1186   \begingroup
1187     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1188               \else\noexpand##1\noexpand##2\fi}%
1189     \def\do{\x\do}%
1190     \def\@makeother{\x\@makeother}%
1191   \edef\x{\endgroup
1192     \def\noexpand\dospecials{\dospecials}%
1193     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1194       \def\noexpand\@sanitize{\@sanitize}%
1195     \fi}%
1196   \x}
```

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

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, \<level>@group, <level>@active and <next-level>@active (except in system).

```
1197 \def\bbl@active@def#1#2#3#4{%
1198   \@namedef{#3#1}{%
1199     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1200       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1201     \else
1202       \bbl@afterfi\csname#2@sh@#1@\endcsname
1203     \fi}%
```

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.

```
1204   \long\@namedef{#3@arg#1}##1{%
1205     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1206       \bbl@afterelse\csname#4#1\endcsname##1%
1207     \else
1208       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1209     \fi}}%
```

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

```
1210 \def\initiate@active@char#1{%
1211   \bbl@ifunset{active@char\string#1}%
1212     {\bbl@withactive
1213       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1214     {}}
```

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

```
1215 \def\@initiate@active@char#1#2#3{%
1216   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1217   \ifx#1\@undefined
1218     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1219   \else
1220     \bbl@csarg\let{oridef@@#2}#1%
1221     \bbl@csarg\edef{oridef@#2}{%
1222       \let\noexpand#1%
1223       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1224   \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*).

```
1225   \ifx#1#3\relax
1226     \expandafter\let\csname normal@char#2\endcsname#3%
1227   \else
1228     \bbl@info{Making #2 an active character}%
1229     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1230       \@namedef{normal@char#2}{%
1231         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1232     \else
1233       \@namedef{normal@char#2}{#3}%
1234     \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).

```
1235    \bbl@restoreactive{#2}%
1236    \AtBeginDocument{%
1237      \catcode`#2\active
1238      \if@filesw
1239        \immediate\write\@mainaux{\catcode`\string#2\active}%
1240      \fi}%
1241    \expandafter\bbl@add@special\csname#2\endcsname
1242    \catcode`#2\active
1243  \fi
```

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

```
1244  \let\bbl@tempa\@firstoftwo
1245  \if\string^#2%
1246    \def\bbl@tempa{\noexpand\textormath}%
1247  \else
1248    \ifx\bbl@mathnormal\@undefined\else
1249      \let\bbl@tempa\bbl@mathnormal
1250    \fi
1251  \fi
1252  \expandafter\edef\csname active@char#2\endcsname{%
1253    \bbl@tempa
1254      {\noexpand\if@safe@actives
1255        \noexpand\expandafter
1256        \expandafter\noexpand\csname normal@char#2\endcsname
1257      \noexpand\else
1258        \noexpand\expandafter
1259        \expandafter\noexpand\csname bbl@doactive#2\endcsname
1260      \noexpand\fi}%
1261    {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1262  \bbl@csarg\edef{doactive#2}{%
1263    \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

$$\text{\active@prefix } \langle char \rangle \text{ \normal@char} \langle char \rangle$$

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

```
1264  \bbl@csarg\edef{active@#2}{%
1265    \noexpand\active@prefix\noexpand#1%
1266    \expandafter\noexpand\csname active@char#2\endcsname}%
1267  \bbl@csarg\edef{normal@#2}{%
1268    \noexpand\active@prefix\noexpand#1%
1269    \expandafter\noexpand\csname normal@char#2\endcsname}%
1270  \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.

```
1271    \bbl@active@def#2\user@group{user@active}{language@active}%
1272    \bbl@active@def#2\language@group{language@active}{system@active}%
1273    \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.

```
1274    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1275      {\expandafter\noexpand\csname normal@char#2\endcsname}%
1276    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1277      {\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@m@s as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```
1278    \if\string'#2%
1279      \let\prim@s\bbl@prim@s
1280      \let\active@math@prime#1%
1281    \fi
1282    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1283  ⟨⟨∗More package options⟩⟩ ≡
1284  \DeclareOption{math=active}{}
1285  \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1286  ⟨⟨/More package options⟩⟩
```

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* and the end of the ldf.

```
1287  \@ifpackagewith{babel}{KeepShorthandsActive}%
1288    {\let\bbl@restoreactive\@gobble}%
1289    {\def\bbl@restoreactive#1{%
1290      \bbl@exp{%
1291        \\\AfterBabelLanguage\\\CurrentOption
1292          {\catcode`#1=\the\catcode`#1\relax}%
1293        \\\AtEndOfPackage
1294          {\catcode`#1=\the\catcode`#1\relax}}}%
1295    \AtEndOfPackage{\let\bbl@restoreactive\@gobble}}
```

\bbl@sh@select This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of \hyphenation.
This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either \bbl@firstcs or \bbl@scndcs. Hence two more arguments need to follow it.

```
1296  \def\bbl@sh@select#1#2{%
1297    \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1298      \bbl@afterelse\bbl@scndcs
1299    \else
1300      \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1301    \fi}
```

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

```
1302  \begingroup
1303  \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1304    {\gdef\active@prefix#1{%
1305      \ifx\protect\@typeset@protect
1306      \else
1307        \ifx\protect\@unexpandable@protect
1308          \noexpand#1%
1309        \else
1310          \protect#1%
1311        \fi
```

90

```
1312        \expandafter\@gobble
1313      \fi}}
1314    {\gdef\active@prefix#1{%
1315      \ifincsname
1316        \string#1%
1317        \expandafter\@gobble
1318      \else
1319        \ifx\protect\@typeset@protect
1320        \else
1321          \ifx\protect\@unexpandable@protect
1322            \noexpand#1%
1323          \else
1324            \protect#1%
1325          \fi
1326          \expandafter\expandafter\expandafter\@gobble
1327        \fi
1328      \fi}}
1329 \endgroup
```

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

```
1330 \newif\if@safe@actives
1331 \@safe@activesfalse
```

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

```
1332 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

\bbl@activate   Both macros take one argument, like \initiate@active@char. The macro is used to change the
\bbl@deactivate   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.

```
1333 \chardef\bbl@activated\z@
1334 \def\bbl@activate#1{%
1335   \chardef\bbl@activated\@ne
1336   \bbl@withactive{\expandafter\let\expandafter}#1%
1337     \csname bbl@active@\string#1\endcsname}
1338 \def\bbl@deactivate#1{%
1339   \chardef\bbl@activated\tw@
1340   \bbl@withactive{\expandafter\let\expandafter}#1%
1341     \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs   These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
```
1342 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1343 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

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

1. a name for the collection of shorthands, i.e. 'system', or 'dutch';

2. the character (sequence) that makes up the shorthand, i.e. ~ or "a;

3. the code to be executed when the shorthand is encountered.

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.

```
1344 \def\babel@texpdf#1#2#3#4{%
1345   \ifx\texorpdfstring\@undefined
1346     \textormath{#1}{#3}%
1347   \else
```

91

```
1348        \texorpdfstring{\textormath{#1}{#3}}{#2}%
1349      % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1350    \fi}
1351 %
1352 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1353 \def\@decl@short#1#2#3\@nil#4{%
1354   \def\bbl@tempa{#3}%
1355   \ifx\bbl@tempa\@empty
1356     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1357     \bbl@ifunset{#1@sh@\string#2@}{}%
1358       {\def\bbl@tempa{#4}%
1359        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1360        \else
1361          \bbl@info
1362            {Redefining #1 shorthand \string#2\\%
1363             in language \CurrentOption}%
1364        \fi}%
1365     \@namedef{#1@sh@\string#2@}{#4}%
1366   \else
1367     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1368     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1369       {\def\bbl@tempa{#4}%
1370        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1371        \else
1372          \bbl@info
1373            {Redefining #1 shorthand \string#2\string#3\\%
1374             in language \CurrentOption}%
1375        \fi}%
1376     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1377   \fi}
```

\textormath Some of the shorthands that will be declared by the language definition files have to be usable in both text and mathmode. To achieve this the helper macro \textormath is provided.

```
1378 \def\textormath{%
1379   \ifmmode
1380     \expandafter\@secondoftwo
1381   \else
1382     \expandafter\@firstoftwo
1383   \fi}
```

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

```
1384 \def\user@group{user}
1385 \def\language@group{english} % TODO. I don't like defaults
1386 \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.

```
1387 \def\useshorthands{%
1388   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1389 \def\bbl@usesh@s#1{%
1390   \bbl@usesh@x
1391     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1392     {#1}}
1393 \def\bbl@usesh@x#1#2{%
1394   \bbl@ifshorthand{#2}%
1395     {\def\user@group{user}%
1396      \initiate@active@char{#2}%
1397      #1%
1398      \bbl@activate{#2}}%
1399     {\bbl@error
```

```
1400          {I can't declare a shorthand turned off (\string#2)}
1401          {Sorry, but you can't use shorthands which have been\\%
1402           turned off in the package options}}}
```

\defineshorthand 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.

```
1403 \def\user@language@group{user@\language@group}
1404 \def\bbl@set@user@generic#1#2{%
1405   \bbl@ifunset{user@generic@active#1}%
1406     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1407      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1408      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1409        \expandafter\noexpand\csname normal@char#1\endcsname}%
1410      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1411        \expandafter\noexpand\csname user@active#1\endcsname}}%
1412   \@empty}
1413 \newcommand\defineshorthand[3][user]{%
1414   \edef\bbl@tempa{\zap@space#1 \@empty}%
1415   \bbl@for\bbl@tempb\bbl@tempa{%
1416     \if*\expandafter\@car\bbl@tempb\@nil
1417       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1418       \@expandtwoargs
1419         \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1420     \fi
1421     \declare@shorthand{\bbl@tempb}{#2}{#3}}}
```

\languageshorthands A user level command to change the language from which shorthands are used. Unfortunately, babel
currently does not keep track of defined groups, and therefore there is no way to catch a possible
change in casing to fix it in the same way languages names are fixed. [TODO].

```
1422 \def\languageshorthands#1{\def\language@group{#1}}
```

\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 lattest to \active@char".

```
1423 \def\aliasshorthand#1#2{%
1424   \bbl@ifshorthand{#2}%
1425     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1426        \ifx\document\@notprerr
1427          \@notshorthand{#2}%
1428        \else
1429          \initiate@active@char{#2}%
1430          \expandafter\let\csname active@char\string#2\expandafter\endcsname
1431            \csname active@char\string#1\endcsname
1432          \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1433            \csname normal@char\string#1\endcsname
1434          \bbl@activate{#2}%
1435        \fi
1436      \fi}%
1437     {\bbl@error
1438       {Cannot declare a shorthand turned off (\string#2)}
1439       {Sorry, but you cannot use shorthands which have been\\%
1440        turned off in the package options}}}
```

\@notshorthand

```
1441 \def\@notshorthand#1{%
1442   \bbl@error{%
1443     The character '\string #1' should be made a shorthand character;\\%
1444     add the command \string\useshorthands\string{#1\string} to
1445     the preamble.\\%
1446     I will ignore your instruction}%
```

93

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

```
1448 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1449 \DeclareRobustCommand*\shorthandoff{%
1450   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1451 \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.

```
1452 \def\bbl@switch@sh#1#2{%
1453   \ifx#2\@nnil\else
1454     \bbl@ifunset{bbl@active@\string#2}%
1455       {\bbl@error
1456         {I can't switch '\string#2' on or off--not a shorthand}%
1457         {This character is not a shorthand. Maybe you made\\%
1458          a typing mistake? I will ignore your instruction.}}%
1459       {\ifcase#1%   off, on, off*
1460         \catcode`#212\relax
1461       \or
1462         \catcode`#2\active
1463         \bbl@ifunset{bbl@shdef@\string#2}%
1464           {}%
1465           {\bbl@withactive{\expandafter\let\expandafter}#2%
1466             \csname bbl@shdef@\string#2\endcsname
1467            \bbl@csarg\let{shdef@\string#2}\relax}%
1468         \ifcase\bbl@activated\or
1469           \bbl@activate{#2}%
1470         \else
1471           \bbl@deactivate{#2}%
1472         \fi
1473       \or
1474         \bbl@ifunset{bbl@shdef@\string#2}%
1475           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1476           {}%
1477         \csname bbl@oricat@\string#2\endcsname
1478         \csname bbl@oridef@\string#2\endcsname
1479       \fi}%
1480     \bbl@afterfi\bbl@switch@sh#1%
1481   \fi}
```

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

```
1482 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1483 \def\bbl@putsh#1{%
1484   \bbl@ifunset{bbl@active@\string#1}%
1485     {\bbl@putsh@i#1\@empty\@nnil}%
1486     {\csname bbl@active@\string#1\endcsname}}
1487 \def\bbl@putsh@i#1#2\@nnil{%
1488   \csname\language@group @sh@\string#1@%
1489     \ifx\@empty#2\else\string#2@\fi\endcsname}
1490 \ifx\bbl@opt@shorthands\@nnil\else
1491   \let\bbl@s@initiate@active@char\initiate@active@char
1492   \def\initiate@active@char#1{%
1493     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1494   \let\bbl@s@switch@sh\bbl@switch@sh
1495   \def\bbl@switch@sh#1#2{%
1496     \ifx#2\@nnil\else
```

```
1497        \bbl@afterfi
1498        \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1499      \fi}
1500    \let\bbl@s@activate\bbl@activate
1501    \def\bbl@activate#1{%
1502      \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1503    \let\bbl@s@deactivate\bbl@deactivate
1504    \def\bbl@deactivate#1{%
1505      \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1506 \fi
```

You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on or off.

```
1507 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}
```

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

```
1508 \def\bbl@prim@s{%
1509   \prime\futurelet\@let@token\bbl@pr@m@s}
1510 \def\bbl@if@primes#1#2{%
1511   \ifx#1\@let@token
1512     \expandafter\@firstoftwo
1513   \else\ifx#2\@let@token
1514     \bbl@afterelse\expandafter\@firstoftwo
1515   \else
1516     \bbl@afterfi\expandafter\@secondoftwo
1517   \fi\fi}
1518 \begingroup
1519   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
1520   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
1521   \lowercase{%
1522     \gdef\bbl@pr@m@s{%
1523       \bbl@if@primes"'%
1524         \pr@@@s
1525       {\bbl@if@primes*^\pr@@@t\egroup}}}
1526 \endgroup
```

Usually the ~ is active and expands to \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).

```
1527 \initiate@active@char{~}
1528 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1529 \bbl@activate{~}
```

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

```
1530 \expandafter\def\csname OT1dqpos\endcsname{127}
1531 \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

```
1532 \ifx\f@encoding\@undefined
1533   \def\f@encoding{OT1}
1534 \fi
```

7.6 Language attributes

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

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```
1535 \bbl@trace{Language attributes}
1536 \newcommand\languageattribute[2]{%
1537   \def\bbl@tempc{#1}%
1538   \bbl@fixname\bbl@tempc
1539   \bbl@iflanguage\bbl@tempc{%
1540     \bbl@vforeach{#2}{%
```

We want to make sure that each attribute is selected only once; therefore we store the already selected attributes in \bbl@known@attribs. When that control sequence is not yet defined this attribute is certainly not selected before.

```
1541       \ifx\bbl@known@attribs\@undefined
1542         \in@false
1543       \else
1544         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1545       \fi
1546       \ifin@
1547         \bbl@warning{%
1548           You have more than once selected the attribute '##1'\\%
1549           for language #1. Reported}%
1550       \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.

```
1551       \bbl@exp{%
1552         \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1553       \edef\bbl@tempa{\bbl@tempc-##1}%
1554       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1555       {\csname\bbl@tempc @attr@##1\endcsname}%
1556       {\@attrerr{\bbl@tempc}{##1}}%
1557     \fi}}}
1558 \@onlypreamble\languageattribute
```

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

```
1559 \newcommand*{\@attrerr}[2]{%
1560   \bbl@error
1561     {The attribute #2 is unknown for language #1.}%
1562     {Your command will be ignored, type <return> to proceed}}
```

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

```
1563 \def\bbl@declare@ttribute#1#2#3{%
1564   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1565   \ifin@
1566     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1567   \fi
1568   \bbl@add@list\bbl@attributes{#1-#2}%
1569   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret TeX code based on whether a certain attribute was set. This command should appear inside the argument to \AtBeginDocument because the attributes are set in the document preamble, *after* babel is loaded.
The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```
1570 \def\bbl@ifattributeset#1#2#3#4{%
1571   \ifx\bbl@known@attribs\@undefined
1572     \in@false
1573   \else
1574     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1575   \fi
1576   \ifin@
1577     \bbl@afterelse#3%
1578   \else
1579     \bbl@afterfi#4%
1580   \fi}
```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the TEX-code to be executed when the attribute is known and the TEX-code to be executed otherwise.
We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```
1581 \def\bbl@ifknown@ttrib#1#2{%
1582   \let\bbl@tempa\@secondoftwo
1583   \bbl@loopx\bbl@tempb{#2}{%
1584     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1585     \ifin@
1586       \let\bbl@tempa\@firstoftwo
1587     \else
1588     \fi}%
1589   \bbl@tempa}
```

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

```
1590 \def\bbl@clear@ttribs{%
1591   \ifx\bbl@attributes\@undefined\else
1592     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1593       \expandafter\bbl@clear@ttrib\bbl@tempa.
1594       }%
1595     \let\bbl@attributes\@undefined
1596   \fi}
1597 \def\bbl@clear@ttrib#1-#2.{%
1598   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1599 \AtBeginDocument{\bbl@clear@ttribs}
```

7.7   Support for saving macro definitions

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

\babel@savecnt The initialization of a new save cycle: reset the counter to zero.
\babel@beginsave
```
1600 \bbl@trace{Macros for saving definitions}
1601 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1602 \newcount\babel@savecnt
1603 \babel@beginsave
```

\babel@save The macro \babel@save⟨csname⟩ saves the current meaning of the control sequence ⟨csname⟩ to
\babel@savevariable \originalTeX[32]. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to \originalTeX and the counter is incremented. The macro

---

[32]\originalTeX has to be expandable, i. e. you shouldn't let it to \relax.

\babel@savevariable⟨*variable*⟩ saves the value of the variable. ⟨*variable*⟩ can be anything allowed after the \the primitive.

```
1604 \def\babel@save#1{%
1605   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1606   \toks@\expandafter{\originalTeX\let#1=}%
1607   \bbl@exp{%
1608     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1609   \advance\babel@savecnt\@ne}
1610 \def\babel@savevariable#1{%
1611   \toks@\expandafter{\originalTeX #1=}%
1612   \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

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

```
1613 \def\bbl@frenchspacing{%
1614   \ifnum\the\sfcode`\.=\@m
1615     \let\bbl@nonfrenchspacing\relax
1616   \else
1617     \frenchspacing
1618     \let\bbl@nonfrenchspacing\nonfrenchspacing
1619   \fi}
1620 \let\bbl@nonfrenchspacing\nonfrenchspacing
1621 \let\bbl@elt\relax
1622 \edef\bbl@fs@chars{%
1623   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1624   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1625   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1626 \def\bbl@pre@fs{%
1627   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1628   \edef\bbl@save@sfcodes{\bbl@fs@chars}}
1629 \def\bbl@post@fs{%
1630   \bbl@save@sfcodes
1631   \edef\bbl@tempa{\bbl@cl{frspc}}%
1632   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1633   \if u\bbl@tempa          % do nothing
1634   \else\if n\bbl@tempa      % non french
1635     \def\bbl@elt##1##2##3{%
1636       \ifnum\sfcode`##1=##2\relax
1637         \babel@savevariable{\sfcode`##1}%
1638         \sfcode`##1=##3\relax
1639       \fi}%
1640     \bbl@fs@chars
1641   \else\if y\bbl@tempa      % french
1642     \def\bbl@elt##1##2##3{%
1643       \ifnum\sfcode`##1=##3\relax
1644         \babel@savevariable{\sfcode`##1}%
1645         \sfcode`##1=##2\relax
1646       \fi}%
1647     \bbl@fs@chars
1648   \fi\fi\fi}
```

7.8 Short tags

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

```
1649 \bbl@trace{Short tags}
1650 \def\babeltags#1{%
1651   \edef\bbl@tempa{\zap@space#1 \@empty}%
```

```
1652    \def\bbl@tempb##1=##2\@@{%
1653      \edef\bbl@tempc{%
1654        \noexpand\newcommand
1655        \expandafter\noexpand\csname ##1\endcsname{%
1656          \noexpand\protect
1657          \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1658        \noexpand\newcommand
1659        \expandafter\noexpand\csname text##1\endcsname{%
1660          \noexpand\foreignlanguage{##2}}}
1661      \bbl@tempc}%
1662    \bbl@for\bbl@tempa\bbl@tempa{%
1663      \expandafter\bbl@tempb\bbl@tempa\@@}}
```

7.9 Hyphens

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

```
1664    \bbl@trace{Hyphens}
1665    \@onlypreamble\babelhyphenation
1666    \AtEndOfPackage{%
1667      \newcommand\babelhyphenation[2][\@empty]{%
1668        \ifx\bbl@hyphenation@\relax
1669          \let\bbl@hyphenation@\@empty
1670        \fi
1671        \ifx\bbl@hyphlist\@empty\else
1672          \bbl@warning{%
1673            You must not intermingle \string\selectlanguage\space and\\%
1674            \string\babelhyphenation\space or some exceptions will not\\%
1675            be taken into account. Reported}%
1676        \fi
1677        \ifx\@empty#1%
1678          \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1679        \else
1680          \bbl@vforeach{#1}{%
1681            \def\bbl@tempa{##1}%
1682            \bbl@fixname\bbl@tempa
1683            \bbl@iflanguage\bbl@tempa{%
1684              \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1685                \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1686                  {}%
1687                  {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1688                #2}}}%
1689        \fi}}
```

\bbl@allowhyphens This macro makes hyphenation possible. Basically its definition is nothing more than \nobreak \hskip 0pt plus 0pt[33].

```
1690    \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1691    \def\bbl@t@one{T1}
1692    \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
```

\babelhyphen Macros to insert common hyphens. Note the space before @ in \babelhyphen. Instead of protecting it with \DeclareRobustCommand, which could insert a \relax, we use the same procedure as shorthands, with \active@prefix.

```
1693    \newcommand\babelnullhyphen{\char\hyphenchar\font}
1694    \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1695    \def\bbl@hyphen{%
1696      \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1697    \def\bbl@hyphen@i#1#2{%
1698      \bbl@ifunset{bbl@hy@#1#2\@empty}%
1699        {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
```

---

[33]TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```
1700      {\csname bbl@hy@#1#2\@empty\endcsname}}
```

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.

```
1701 \def\bbl@usehyphen#1{%
1702   \leavevmode
1703   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1704   \nobreak\hskip\z@skip}
1705 \def\bbl@@usehyphen#1{%
1706   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1707 \def\bbl@hyphenchar{%
1708   \ifnum\hyphenchar\font=\m@ne
1709     \babelnullhyphen
1710   \else
1711     \char\hyphenchar\font
1712   \fi}
```

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 \bbl@hy@nobreak is redundant.

```
1713 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1714 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1715 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1716 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1717 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1718 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1719 \def\bbl@hy@repeat{%
1720   \bbl@usehyphen{%
1721     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1722 \def\bbl@hy@@repeat{%
1723   \bbl@@usehyphen{%
1724     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1725 \def\bbl@hy@empty{\hskip\z@skip}
1726 \def\bbl@hy@@empty{\discretionary{}{}{}}
```

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

```
1727 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

7.10   Multiencoding strings

The aim following commands is to provide a commom 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 tool. It makes global a local variable. This is not the best solution, but it works.

```
1728 \bbl@trace{Multiencoding strings}
1729 \def\bbl@toglobal#1{\global\let#1#1}
```

The 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 \⟨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 uppercasing, we have:

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

and starts over (and similarly when lowercasing).

```
1730 \@ifpackagewith{babel}{nocase}%
1731   {\let\bbl@patchuclc\relax}%
1732   {\def\bbl@patchuclc{%
1733     \global\let\bbl@patchuclc\relax
1734     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1735     \gdef\bbl@uclc##1{%
1736       \let\bbl@encoded\bbl@encoded@uclc
1737       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1738         {##1}%
1739         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1740          \csname\languagename @bbl@uclc\endcsname}%
1741       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1742     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1743     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
1744 % A temporary hack:
1745 \ifx\BabelCaseHack\@undefined
1746 \AtBeginDocument{%
1747   \bbl@exp{%
1748     \\\in@{\string\@uclclist}%
1749          {\expandafter\meaning\csname MakeUppercase \endcsname}}%
1750   \ifin@\else
1751     \expandafter\let\expandafter\bbl@newuc\csname MakeUppercase \endcsname
1752     \protected@namedef{MakeUppercase }#1{{%
1753       \def\reserved@a##1##2{\let##1##2\reserved@a}%
1754       \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b\@gobble}%
1755       \protected@edef\reserved@a{\bbl@newuc{#1}}\reserved@a}}%
1756     \expandafter\let\expandafter\bbl@newlc\csname MakeLowercase \endcsname
1757     \protected@namedef{MakeLowercase }#1{{%
1758       \def\reserved@a##1##2{\let##2##1\reserved@a}%
1759       \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b\@gobble}%
1760       \protected@edef\reserved@a{\bbl@newlc{#1}}\reserved@a}}%
1761   \fi}
1762 \fi
```

```
1763 ⟨⟨*More package options⟩⟩ ≡
1764 \DeclareOption{nocase}{}
1765 ⟨⟨/More package options⟩⟩
```

The following package options control the behavior of \SetString.

```
1766 ⟨⟨*More package options⟩⟩ ≡
1767 \let\bbl@opt@strings\@nnil % accept strings=value
1768 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1769 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1770 \def\BabelStringsDefault{generic}
1771 ⟨⟨/More package options⟩⟩
```

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.

```
1772 \@onlypreamble\StartBabelCommands
1773 \def\StartBabelCommands{%
1774   \begingroup
1775   \@tempcnta="7F
1776   \def\bbl@tempa{%
1777     \ifnum\@tempcnta>"FF\else
1778       \catcode\@tempcnta=11
1779       \advance\@tempcnta\@ne
1780       \expandafter\bbl@tempa
1781     \fi}%
```

```
1782    \bbl@tempa
1783    ⟨⟨Macros local to BabelCommands⟩⟩
1784    \def\bbl@provstring##1##2{%
1785      \providecommand##1{##2}%
1786      \bbl@toglobal##1}%
1787    \global\let\bbl@scafter\@empty
1788    \let\StartBabelCommands\bbl@startcmds
1789    \ifx\BabelLanguages\relax
1790      \let\BabelLanguages\CurrentOption
1791    \fi
1792    \begingroup
1793    \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1794    \StartBabelCommands}
1795 \def\bbl@startcmds{%
1796    \ifx\bbl@screset\@nnil\else
1797      \bbl@usehooks{stopcommands}{}%
1798    \fi
1799    \endgroup
1800    \begingroup
1801    \@ifstar
1802      {\ifx\bbl@opt@strings\@nnil
1803        \let\bbl@opt@strings\BabelStringsDefault
1804      \fi
1805      \bbl@startcmds@i}%
1806      \bbl@startcmds@i}
1807 \def\bbl@startcmds@i#1#2{%
1808    \edef\bbl@L{\zap@space#1 \@empty}%
1809    \edef\bbl@G{\zap@space#2 \@empty}%
1810    \bbl@startcmds@ii}
1811 \let\bbl@startcommands\StartBabelCommands
```

Parse the encoding info to get the label, input, and font parts.

Select the behavior of \SetString. Thre 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.

```
1812 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1813    \let\SetString\@gobbletwo
1814    \let\bbl@stringdef\@gobbletwo
1815    \let\AfterBabelCommands\@gobble
1816    \ifx\@empty#1%
1817      \def\bbl@sc@label{generic}%
1818      \def\bbl@encstring##1##2{%
1819        \ProvideTextCommandDefault##1{##2}%
1820        \bbl@toglobal##1%
1821        \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1822      \let\bbl@sctest\in@true
1823    \else
1824      \let\bbl@sc@charset\space % <- zapped below
1825      \let\bbl@sc@fontenc\space % <-    "        "
1826      \def\bbl@tempa##1=##2\@nil{%
1827        \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1828      \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1829      \def\bbl@tempa##1 ##2{% space -> comma
1830        ##1%
1831        \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1832      \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1833      \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1834      \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
```

```
1835    \def\bbl@encstring##1##2{%
1836      \bbl@foreach\bbl@sc@fontenc{%
1837        \bbl@ifunset{T@####1}%
1838          {}%
1839          {\ProvideTextCommand##1{####1}{##2}%
1840           \bbl@toglobal##1%
1841           \expandafter
1842           \bbl@toglobal\csname####1\string##1\endcsname}}}%
1843    \def\bbl@sctest{%
1844      \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1845    \fi
1846  \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
1847  \else\ifx\bbl@opt@strings\relax    % ie, strings=encoded
1848    \let\AfterBabelCommands\bbl@aftercmds
1849    \let\SetString\bbl@setstring
1850    \let\bbl@stringdef\bbl@encstring
1851  \else        % ie, strings=value
1852  \bbl@sctest
1853  \ifin@
1854    \let\AfterBabelCommands\bbl@aftercmds
1855    \let\SetString\bbl@setstring
1856    \let\bbl@stringdef\bbl@provstring
1857  \fi\fi\fi
1858  \bbl@scswitch
1859  \ifx\bbl@G\@empty
1860    \def\SetString##1##2{%
1861      \bbl@error{Missing group for string \string##1}%
1862        {You must assign strings to some category, typically\\%
1863         captions or extras, but you set none}}%
1864  \fi
1865  \ifx\@empty#1%
1866    \bbl@usehooks{defaultcommands}{}%
1867  \else
1868    \@expandtwoargs
1869    \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1870  \fi}
```

There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes sure \⟨group⟩⟨language⟩ is reset, but only once (\bbl@screset is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing.

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

```
1871 \def\bbl@forlang#1#2{%
1872   \bbl@for#1\bbl@L{%
1873     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1874     \ifin@#2\relax\fi}}
1875 \def\bbl@scswitch{%
1876   \bbl@forlang\bbl@tempa{%
1877     \ifx\bbl@G\@empty\else
1878       \ifx\SetString\@gobbletwo\else
1879         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1880         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1881         \ifin@\else
1882           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1883           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1884         \fi
1885       \fi
1886     \fi}}
1887 \AtEndOfPackage{%
1888   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
```

```
1889    \let\bbl@scswitch\relax}
1890 \@onlypreamble\EndBabelCommands
1891 \def\EndBabelCommands{%
1892    \bbl@usehooks{stopcommands}{}%
1893    \endgroup
1894    \endgroup
1895    \bbl@scafter}
1896 \let\bbl@endcommands\EndBabelCommands
```

Now we define commands to be used inside \StartBabelCommands.

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

```
1897 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1898    \bbl@forlang\bbl@tempa{%
1899       \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1900       \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1901          {\bbl@exp{%
1902             \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1<\bbl@LC>}}}%
1903          {}%
1904       \def\BabelString{#2}%
1905       \bbl@usehooks{stringprocess}{}%
1906       \expandafter\bbl@stringdef
1907          \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}}
```

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

```
1908 \ifx\bbl@opt@strings\relax
1909    \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1910    \bbl@patchuclc
1911    \let\bbl@encoded\relax
1912    \def\bbl@encoded@uclc#1{%
1913       \@inmathwarn#1%
1914       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1915          \expandafter\ifx\csname ?\string#1\endcsname\relax
1916             \TextSymbolUnavailable#1%
1917          \else
1918             \csname ?\string#1\endcsname
1919          \fi
1920       \else
1921          \csname\cf@encoding\string#1\endcsname
1922       \fi}
1923 \else
1924    \def\bbl@scset#1#2{\def#1{#2}}
1925 \fi
```

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

```
1926 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1927 \def\SetStringLoop##1##2{%
1928    \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1929    \count@\z@
1930    \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1931       \advance\count@\@ne
1932       \toks@\expandafter{\bbl@tempa}%
1933       \bbl@exp{%
1934          \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1935          \count@=\the\count@\relax}}}%
1936 ⟨⟨/Macros local to BabelCommands⟩⟩
```

Delaying code Now the definition of \AfterBabelCommands when it is activated.

```
1937 \def\bbl@aftercmds#1{%
1938   \toks@\expandafter{\bbl@scafter#1}%
1939   \xdef\bbl@scafter{\the\toks@}}
```

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

```
1940 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1941   \newcommand\SetCase[3][]{%
1942     \bbl@patchuclc
1943     \bbl@forlang\bbl@tempa{%
1944       \expandafter\bbl@encstring
1945         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1946       \expandafter\bbl@encstring
1947         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1948       \expandafter\bbl@encstring
1949         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1950 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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.

```
1951 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1952   \newcommand\SetHyphenMap[1]{%
1953     \bbl@forlang\bbl@tempa{%
1954       \expandafter\bbl@stringdef
1955         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1956 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1957 \newcommand\BabelLower[2]{% one to one.
1958   \ifnum\lccode#1=#2\else
1959     \babel@savevariable{\lccode#1}%
1960     \lccode#1=#2\relax
1961   \fi}
1962 \newcommand\BabelLowerMM[4]{% many-to-many
1963   \@tempcnta=#1\relax
1964   \@tempcntb=#4\relax
1965   \def\bbl@tempa{%
1966     \ifnum\@tempcnta>#2\else
1967       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1968       \advance\@tempcnta#3\relax
1969       \advance\@tempcntb#3\relax
1970       \expandafter\bbl@tempa
1971     \fi}%
1972   \bbl@tempa}
1973 \newcommand\BabelLowerMO[4]{% many-to-one
1974   \@tempcnta=#1\relax
1975   \def\bbl@tempa{%
1976     \ifnum\@tempcnta>#2\else
1977       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1978       \advance\@tempcnta#3
1979       \expandafter\bbl@tempa
1980     \fi}%
1981   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1982 ⟨⟨*More package options⟩⟩ ≡
1983 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1984 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1985 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
```

```
1986 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1987 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1988 ⟨⟨/More package options⟩⟩
```

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

```
1989 \AtEndOfPackage{%
1990   \ifx\bbl@opt@hyphenmap\@undefined
1991     \bbl@xin@{,}{\bbl@language@opts}%
1992     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1993   \fi}
```

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

```
1994 \newcommand\setlocalecaption{%  TODO. Catch typos.
1995   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1996 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
1997   \bbl@trim@def\bbl@tempa{#2}%
1998   \bbl@xin@{.template}{\bbl@tempa}%
1999   \ifin@
2000     \bbl@ini@captions@template{#3}{#1}%
2001   \else
2002     \edef\bbl@tempd{%
2003       \expandafter\expandafter\expandafter
2004       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2005     \bbl@xin@
2006       {\expandafter\string\csname #2name\endcsname}%
2007       {\bbl@tempd}%
2008     \ifin@ % Renew caption
2009       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2010       \ifin@
2011         \bbl@exp{%
2012           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2013             {\\\bbl@scset\<#2name>\<#1#2name>}%
2014             {}}%
2015       \else % Old way converts to new way
2016         \bbl@ifunset{#1#2name}%
2017           {\bbl@exp{%
2018             \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2019             \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2020               {\def\<#2name>{\<#1#2name>}}%
2021               {}}}%
2022           {}%
2023       \fi
2024     \else
2025       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2026       \ifin@ % New way
2027         \bbl@exp{%
2028           \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2029           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2030             {\\\bbl@scset\<#2name>\<#1#2name>}%
2031             {}}%
2032       \else  % Old way, but defined in the new way
2033         \bbl@exp{%
2034           \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2035           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2036             {\def\<#2name>{\<#1#2name>}}%
2037             {}}%
2038       \fi%
2039     \fi
2040     \@namedef{#1#2name}{#3}%
2041     \toks@\expandafter{\bbl@captionslist}%
2042     \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2043     \ifin@\else
```

```
2044        \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2045        \bbl@toglobal\bbl@captionslist
2046      \fi
2047    \fi}
2048 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')
```

7.11   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.

```
2049 \bbl@trace{Macros related to glyphs}
2050 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2051      \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2052      \setbox\z@\hbox{\lower\dimen\z@ \box\z@}\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.

```
2053 \def\save@sf@q#1{\leavevmode
2054    \begingroup
2055      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2056    \endgroup}
```

7.12   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.

7.12.1   Quotation marks

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

```
2057 \ProvideTextCommand{\quotedblbase}{OT1}{%
2058    \save@sf@q{\set@low@box{\textquotedblright\/}%
2059      \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2060 \ProvideTextCommandDefault{\quotedblbase}{%
2061    \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2062 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2063    \save@sf@q{\set@low@box{\textquoteright\/}%
2064      \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

```
2065 \ProvideTextCommandDefault{\quotesinglbase}{%
2066    \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2067 \ProvideTextCommand{\guillemetleft}{OT1}{%
2068    \ifmmode
2069      \ll
2070    \else
2071      \save@sf@q{\nobreak
2072        \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2073    \fi}
2074 \ProvideTextCommand{\guillemetright}{OT1}{%
2075    \ifmmode
2076      \gg
2077    \else
2078      \save@sf@q{\nobreak
```

```
2079        \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2080    \fi}
2081 \ProvideTextCommand{\guillemotleft}{OT1}{%
2082    \ifmmode
2083       \ll
2084    \else
2085       \save@sf@q{\nobreak
2086          \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2087    \fi}
2088 \ProvideTextCommand{\guillemotright}{OT1}{%
2089    \ifmmode
2090       \gg
2091    \else
2092       \save@sf@q{\nobreak
2093          \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2094    \fi}
```

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

```
2095 \ProvideTextCommandDefault{\guillemetleft}{%
2096    \UseTextSymbol{OT1}{\guillemetleft}}
2097 \ProvideTextCommandDefault{\guillemetright}{%
2098    \UseTextSymbol{OT1}{\guillemetright}}
2099 \ProvideTextCommandDefault{\guillemotleft}{%
2100    \UseTextSymbol{OT1}{\guillemotleft}}
2101 \ProvideTextCommandDefault{\guillemotright}{%
2102    \UseTextSymbol{OT1}{\guillemotright}}
```

`\guilsinglleft`
`\guilsinglright` The single guillemets are not available in OT1 encoding. They are faked.

```
2103 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2104    \ifmmode
2105       <%
2106    \else
2107       \save@sf@q{\nobreak
2108          \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2109    \fi}
2110 \ProvideTextCommand{\guilsinglright}{OT1}{%
2111    \ifmmode
2112       >%
2113    \else
2114       \save@sf@q{\nobreak
2115          \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2116    \fi}
```

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

```
2117 \ProvideTextCommandDefault{\guilsinglleft}{%
2118    \UseTextSymbol{OT1}{\guilsinglleft}}
2119 \ProvideTextCommandDefault{\guilsinglright}{%
2120    \UseTextSymbol{OT1}{\guilsinglright}}
```

7.12.2 Letters

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

```
2121 \DeclareTextCommand{\ij}{OT1}{%
2122    i\kern-0.02em\bbl@allowhyphens j}
2123 \DeclareTextCommand{\IJ}{OT1}{%
2124    I\kern-0.02em\bbl@allowhyphens J}
2125 \DeclareTextCommand{\ij}{T1}{\char188}
2126 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2127 \ProvideTextCommandDefault{\ij}{%
2128    \UseTextSymbol{OT1}{\ij}}
```

```
2129 \ProvideTextCommandDefault{\IJ}{%
2130    \UseTextSymbol{OT1}{\IJ}}
```

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

```
2131 \def\crrtic@{\hrule height0.1ex width0.3em}
2132 \def\crttic@{\hrule height0.1ex width0.33em}
2133 \def\ddj@{%
2134    \setbox0\hbox{d}\dimen@=\ht0
2135    \advance\dimen@1ex
2136    \dimen@.45\dimen@
2137    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2138    \advance\dimen@ii.5ex
2139    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2140 \def\DDJ@{%
2141    \setbox0\hbox{D}\dimen@=.55\ht0
2142    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2143    \advance\dimen@ii.15ex %              correction for the dash position
2144    \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
2145    \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2146    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2147 %
2148 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2149 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2150 \ProvideTextCommandDefault{\dj}{%
2151    \UseTextSymbol{OT1}{\dj}}
2152 \ProvideTextCommandDefault{\DJ}{%
2153    \UseTextSymbol{OT1}{\DJ}}
```

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

```
2154 \DeclareTextCommand{\SS}{OT1}{SS}
2155 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

7.12.3 Shorthands for quotation marks

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

\glq The 'german' single quotes.
\grq
```
2156 \ProvideTextCommandDefault{\glq}{%
2157    \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

The definition of \grq depends on the fontencoding. With T1 encoding no extra kerning is needed.

```
2158 \ProvideTextCommand{\grq}{T1}{%
2159    \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2160 \ProvideTextCommand{\grq}{TU}{%
2161    \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2162 \ProvideTextCommand{\grq}{OT1}{%
2163    \save@sf@q{\kern-.0125em
2164       \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2165       \kern.07em\relax}}
2166 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq The 'german' double quotes.
\grqq
```
2167 \ProvideTextCommandDefault{\glqq}{%
2168    \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

The definition of \grqq depends on the fontencoding. With T1 encoding no extra kerning is needed.

```
2169 \ProvideTextCommand{\grqq}{T1}{%
2170   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2171 \ProvideTextCommand{\grqq}{TU}{%
2172   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2173 \ProvideTextCommand{\grqq}{OT1}{%
2174   \save@sf@q{\kern-.07em
2175     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2176     \kern.07em\relax}}
2177 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq The 'french' single guillemets.
\frq
```
2178 \ProvideTextCommandDefault{\flq}{%
2179   \textormath{\guilsingleft}{\mbox{\guilsingleft}}}
2180 \ProvideTextCommandDefault{\frq}{%
2181   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq The 'french' double guillemets.
\frqq
```
2182 \ProvideTextCommandDefault{\flqq}{%
2183   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2184 \ProvideTextCommandDefault{\frqq}{%
2185   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

7.12.4 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
\umlautlow   default will be \umlauthigh (the normal positioning).

```
2186 \def\umlauthigh{%
2187   \def\bbl@umlauta##1{\leavevmode\bgroup%
2188     \expandafter\accent\csname\f@encoding dqpos\endcsname
2189     ##1\bbl@allowhyphens\egroup}%
2190   \let\bbl@umlaute\bbl@umlauta}
2191 \def\umlautlow{%
2192   \def\bbl@umlauta{\protect\lower@umlaut}}
2193 \def\umlautelow{%
2194   \def\bbl@umlaute{\protect\lower@umlaut}}
2195 \umlauthigh
```

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

```
2196 \expandafter\ifx\csname U@D\endcsname\relax
2197   \csname newdimen\endcsname\U@D
2198 \fi
```

The following code fools TeX's make_accent procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.
Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of .45ex depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \accent primitive, reset the old x-height and insert the base character in the argument.

```
2199 \def\lower@umlaut#1{%
2200   \leavevmode\bgroup
2201     \U@D 1ex%
2202     {\setbox\z@\hbox{%
2203       \expandafter\char\csname\f@encoding dqpos\endcsname}%
```

```
2204     \dimen@ -.45ex\advance\dimen@\ht\z@
2205     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2206   \expandafter\accent\csname\f@encoding dqpos\endcsname
2207   \fontdimen5\font\U@D #1%
2208   \egroup}
```

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).

```
2209 \AtBeginDocument{%
2210   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2211   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2212   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2213   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2214   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2215   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2216   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2217   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2218   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2219   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2220   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}}
```

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

```
2221 \ifx\l@english\@undefined
2222   \chardef\l@english\z@
2223 \fi
2224 % The following is used to cancel rules in ini files (see Amharic).
2225 \ifx\l@unhyphenated\@undefined
2226   \newlanguage\l@unhyphenated
2227 \fi
```

7.13   Layout

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

```
2228 \bbl@trace{Bidi layout}
2229 \providecommand\IfBabelLayout[3]{#3}%
2230 \newcommand\BabelPatchSection[1]{%
2231   \@ifundefined{#1}{}{%
2232     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2233     \@namedef{#1}{%
2234       \@ifstar{\bbl@presec@s{#1}}%
2235               {\@dblarg{\bbl@presec@x{#1}}}}}}
2236 \def\bbl@presec@x#1[#2]#3{%
2237   \bbl@exp{%
2238     \\\select@language@x{\bbl@main@language}%
2239     \\\bbl@cs{sspre@#1}%
2240     \\\bbl@cs{ss@#1}%
2241       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2242       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2243     \\\select@language@x{\languagename}}}
2244 \def\bbl@presec@s#1#2{%
2245   \bbl@exp{%
2246     \\\select@language@x{\bbl@main@language}%
2247     \\\bbl@cs{sspre@#1}%
2248     \\\bbl@cs{ss@#1}*%
2249       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2250     \\\select@language@x{\languagename}}}
2251 \IfBabelLayout{sectioning}%
2252   {\BabelPatchSection{part}%
```

```
2253    \BabelPatchSection{chapter}%
2254    \BabelPatchSection{section}%
2255    \BabelPatchSection{subsection}%
2256    \BabelPatchSection{subsubsection}%
2257    \BabelPatchSection{paragraph}%
2258    \BabelPatchSection{subparagraph}%
2259    \def\babel@toc#1{%
2260       \select@language@x{\bbl@main@language}}}}{}
2261  \IfBabelLayout{captions}%
2262    {\BabelPatchSection{caption}}{}
```

7.14   Load engine specific macros

Some macros are not defined in all engines, so, after loading the files define them if necessary to raise an error.

```
2263  \bbl@trace{Input engine specific macros}
2264  \ifcase\bbl@engine
2265    \input txtbabel.def
2266  \or
2267    \input luababel.def
2268  \or
2269    \input xebabel.def
2270  \fi
2271  \providecommand\babelfont{%
2272    \bbl@error
2273      {This macro is available only in LuaLaTeX and XeLaTeX.}%
2274      {Consider switching to these engines.}}
2275  \providecommand\babelprehyphenation{%
2276    \bbl@error
2277      {This macro is available only in LuaLaTeX.}%
2278      {Consider switching to that engine.}}
2279  \ifx\babelposthyphenation\@undefined
2280    \let\babelposthyphenation\babelprehyphenation
2281    \let\babelpatterns\babelprehyphenation
2282    \let\babelcharproperty\babelprehyphenation
2283  \fi
```

7.15   Creating and modifying languages

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

```
2284  \bbl@trace{Creating languages and reading ini files}
2285  \let\bbl@extend@ini\@gobble
2286  \newcommand\babelprovide[2][]{%
2287    \let\bbl@savelangname\languagename
2288    \edef\bbl@savelocaleid{\the\localeid}%
2289    % Set name and locale id
2290    \edef\languagename{#2}%
2291    \bbl@id@assign
2292    % Initialize keys
2293    \bbl@vforeach{captions,date,import,main,script,language,%
2294       hyphenrules,linebreaking,justification,mapfont,maparabic,%
2295       mapdigits,intraspace,intrapenalty,onchar,transforms,alph,%
2296       Alph,labels,labels*,calendar,date}%
2297      {\bbl@csarg\let{KVP@##1}\@nnil}%
2298    \global\let\bbl@release@transforms\@empty
2299    \let\bbl@calendars\@empty
2300    \global\let\bbl@inidata\@empty
2301    \global\let\bbl@extend@ini\@gobble
2302    \gdef\bbl@key@list{;}%
2303    \bbl@forkv{#1}{%
```

```
2304    \in@{/}{##1}%
2305    \ifin@
2306      \global\let\bbl@extend@ini\bbl@extend@ini@aux
2307      \bbl@renewinikey##1\@@{##2}%
2308    \else
2309      \bbl@csarg\ifx{KVP@##1}\@nnil\else
2310        \bbl@error
2311          {Unknown key '##1' in \string\babelprovide}%
2312          {See the manual for valid keys}%
2313      \fi
2314      \bbl@csarg\def{KVP@##1}{##2}%
2315    \fi}%
2316  \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2317    \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2318  % == init ==
2319  \ifx\bbl@screset\@undefined
2320    \bbl@ldfinit
2321  \fi
2322  % == date (as option) ==
2323  % \ifx\bbl@KVP@date\@nnil\else
2324  % \fi
2325  % ==
2326  \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2327  \ifcase\bbl@howloaded
2328    \let\bbl@lbkflag\@empty % new
2329  \else
2330    \ifx\bbl@KVP@hyphenrules\@nnil\else
2331      \let\bbl@lbkflag\@empty
2332    \fi
2333    \ifx\bbl@KVP@import\@nnil\else
2334      \let\bbl@lbkflag\@empty
2335    \fi
2336  \fi
2337  % == import, captions ==
2338  \ifx\bbl@KVP@import\@nnil\else
2339    \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2340      {\ifx\bbl@initoload\relax
2341        \begingroup
2342          \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2343          \bbl@input@texini{#2}%
2344        \endgroup
2345       \else
2346        \xdef\bbl@KVP@import{\bbl@initoload}%
2347      \fi}%
2348      {}%
2349    \let\bbl@KVP@date\@empty
2350  \fi
2351  \ifx\bbl@KVP@captions\@nnil
2352    \let\bbl@KVP@captions\bbl@KVP@import
2353  \fi
2354  % ==
2355  \ifx\bbl@KVP@transforms\@nnil\else
2356    \bbl@replace\bbl@KVP@transforms{ }{,}%
2357  \fi
2358  % == Load ini ==
2359  \ifcase\bbl@howloaded
2360    \bbl@provide@new{#2}%
2361  \else
2362    \bbl@ifblank{#1}%
2363      {}%  With \bbl@load@basic below
2364      {\bbl@provide@renew{#2}}%
2365  \fi
2366  % Post tasks
```

```
2367  % ----------
2368  % == subsequent calls after the first provide for a locale ==
2369  \ifx\bbl@inidata\@empty\else
2370    \bbl@extend@ini{#2}%
2371  \fi
2372  % == ensure captions ==
2373  \ifx\bbl@KVP@captions\@nnil\else
2374    \bbl@ifunset{bbl@extracaps@#2}%
2375      {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2376      {\bbl@exp{\\\babelensure[exclude=\\\today,
2377                include=\[bbl@extracaps@#2]]{#2}}}%
2378    \bbl@ifunset{bbl@ensure@\languagename}%
2379      {\bbl@exp{%
2380        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2381          \\\foreignlanguage{\languagename}%
2382          {####1}}}}%
2383      {}%
2384    \bbl@exp{%
2385       \\\bbl@toglobal\<bbl@ensure@\languagename>%
2386       \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2387  \fi
2388  % ==
2389  % At this point all parameters are defined if 'import'. Now we
2390  % execute some code depending on them. But what about if nothing was
2391  % imported? We just set the basic parameters, but still loading the
2392  % whole ini file.
2393  \bbl@load@basic{#2}%
2394  % == script, language ==
2395  % Override the values from ini or defines them
2396  \ifx\bbl@KVP@script\@nnil\else
2397    \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2398  \fi
2399  \ifx\bbl@KVP@language\@nnil\else
2400    \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2401  \fi
2402  \ifcase\bbl@engine\or
2403    \bbl@ifunset{bbl@chrng@\languagename}{}%
2404      {\directlua{
2405         Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
2406  \fi
2407   % == onchar ==
2408  \ifx\bbl@KVP@onchar\@nnil\else
2409    \bbl@luahyphenate
2410    \bbl@exp{%
2411      \\\AddToHook{env/document/before}{{\\\select@language{#2}{}}}}%
2412    \directlua{
2413      if Babel.locale_mapped == nil then
2414        Babel.locale_mapped = true
2415        Babel.linebreaking.add_before(Babel.locale_map)
2416        Babel.loc_to_scr = {}
2417        Babel.chr_to_loc = Babel.chr_to_loc or {}
2418      end
2419      Babel.locale_props[\the\localeid].letters = false
2420    }%
2421    \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2422    \ifin@
2423      \directlua{
2424        Babel.locale_props[\the\localeid].letters = true
2425      }%
2426    \fi
2427    \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2428    \ifin@
2429      \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
```

```
2430          \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2431        \fi
2432        \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2433          {\\\bbl@patterns@lua{\languagename}}}%
2434        % TODO - error/warning if no script
2435        \directlua{
2436          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2437            Babel.loc_to_scr[\the\localeid] =
2438              Babel.script_blocks['\bbl@cl{sbcp}']
2439            Babel.locale_props[\the\localeid].lc = \the\localeid\space
2440            Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2441          end
2442        }%
2443      \fi
2444      \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2445      \ifin@
2446        \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2447        \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2448        \directlua{
2449          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2450            Babel.loc_to_scr[\the\localeid] =
2451              Babel.script_blocks['\bbl@cl{sbcp}']
2452          end}%
2453        \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2454          \AtBeginDocument{%
2455            \bbl@patchfont{{\bbl@mapselect}}%
2456            {\selectfont}}%
2457          \def\bbl@mapselect{%
2458            \let\bbl@mapselect\relax
2459            \edef\bbl@prefontid{\fontid\font}}%
2460          \def\bbl@mapdir##1{%
2461            {\def\languagename{##1}%
2462             \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2463             \bbl@switchfont
2464             \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2465               \directlua{
2466                 Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2467                   ['/\bbl@prefontid'] = \fontid\font\space}%
2468            \fi}}%
2469        \fi
2470        \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2471      \fi
2472      % TODO - catch non-valid values
2473    \fi
2474  % == mapfont ==
2475  % For bidi texts, to switch the font based on direction
2476  \ifx\bbl@KVP@mapfont\@nnil\else
2477    \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2478      {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2479                 mapfont. Use 'direction'.%
2480               {See the manual for details.}}}%
2481    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2482    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2483    \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2484      \AtBeginDocument{%
2485        \bbl@patchfont{{\bbl@mapselect}}%
2486        {\selectfont}}%
2487      \def\bbl@mapselect{%
2488        \let\bbl@mapselect\relax
2489        \edef\bbl@prefontid{\fontid\font}}%
2490      \def\bbl@mapdir##1{%
2491        {\def\languagename{##1}%
2492         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
```

```
2493        \bbl@switchfont
2494        \directlua{Babel.fontmap
2495          [\the\csname bbl@wdir@##1\endcsname]%
2496          [\bbl@prefontid]=\fontid\font}}}%
2497    \fi
2498    \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2499  \fi
2500  % == Line breaking: intraspace, intrapenalty ==
2501  % For CJK, East Asian, Southeast Asian, if interspace in ini
2502  \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2503    \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2504  \fi
2505  \bbl@provide@intraspace
2506  % == Line breaking: CJK quotes ==
2507  \ifcase\bbl@engine\or
2508    \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2509    \ifin@
2510      \bbl@ifunset{bbl@quote@\languagename}{}%
2511        {\directlua{
2512          Babel.locale_props[\the\localeid].cjk_quotes = {}
2513          local cs = 'op'
2514          for c in string.utfvalues(%
2515              [[\csname bbl@quote@\languagename\endcsname]]) do
2516            if Babel.cjk_characters[c].c == 'qu' then
2517              Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2518            end
2519            cs = ( cs == 'op') and 'cl' or 'op'
2520          end
2521        }}%
2522    \fi
2523  \fi
2524  % == Line breaking: justification ==
2525  \ifx\bbl@KVP@justification\@nnil\else
2526    \let\bbl@KVP@linebreaking\bbl@KVP@justification
2527  \fi
2528  \ifx\bbl@KVP@linebreaking\@nnil\else
2529    \bbl@xin@{,\bbl@KVP@linebreaking,}%
2530      {,elongated,kashida,cjk,padding,unhyphenated,}%
2531    \ifin@
2532      \bbl@csarg\xdef
2533        {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2534    \fi
2535  \fi
2536  \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2537  \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2538  \ifin@\bbl@arabicjust\fi
2539  \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2540  \ifin@\AtBeginDocument{\bbl@tibetanjust}\fi
2541  % == Line breaking: hyphenate.other.(locale|script) ==
2542  \ifx\bbl@lbkflag\@empty
2543    \bbl@ifunset{bbl@hyotl@\languagename}{}%
2544      {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2545        \bbl@startcommands*{\languagename}{}%
2546          \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2547            \ifcase\bbl@engine
2548              \ifnum##1<257
2549                \SetHyphenMap{\BabelLower{##1}{##1}}%
2550              \fi
2551            \else
2552              \SetHyphenMap{\BabelLower{##1}{##1}}%
2553            \fi}%
2554        \bbl@endcommands}%
2555    \bbl@ifunset{bbl@hyots@\languagename}{}%
```

116

```
2556        {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2557         \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2558           \ifcase\bbl@engine
2559             \ifnum##1<257
2560               \global\lccode##1=##1\relax
2561             \fi
2562           \else
2563             \global\lccode##1=##1\relax
2564           \fi}}%
2565  \fi
2566  % == Counters: maparabic ==
2567  % Native digits, if provided in ini (TeX level, xe and lua)
2568  \ifcase\bbl@engine\else
2569    \bbl@ifunset{bbl@dgnat@\languagename}{}%
2570      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2571        \expandafter\expandafter\expandafter
2572        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2573        \ifx\bbl@KVP@maparabic\@nnil\else
2574          \ifx\bbl@latinarabic\@undefined
2575            \expandafter\let\expandafter\@arabic
2576              \csname bbl@counter@\languagename\endcsname
2577          \else    % ie, if layout=counters, which redefines \@arabic
2578            \expandafter\let\expandafter\bbl@latinarabic
2579              \csname bbl@counter@\languagename\endcsname
2580          \fi
2581        \fi
2582      \fi}%
2583  \fi
2584  % == Counters: mapdigits ==
2585  % Native digits (lua level).
2586  \ifodd\bbl@engine
2587    \ifx\bbl@KVP@mapdigits\@nnil\else
2588      \bbl@ifunset{bbl@dgnat@\languagename}{}%
2589        {\RequirePackage{luatexbase}%
2590         \bbl@activate@preotf
2591         \directlua{
2592           Babel = Babel or {}  %%% -> presets in luababel
2593           Babel.digits_mapped = true
2594           Babel.digits = Babel.digits or {}
2595           Babel.digits[\the\localeid] =
2596             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2597           if not Babel.numbers then
2598             function Babel.numbers(head)
2599               local LOCALE = Babel.attr_locale
2600               local GLYPH = node.id'glyph'
2601               local inmath = false
2602               for item in node.traverse(head) do
2603                 if not inmath and item.id == GLYPH then
2604                   local temp = node.get_attribute(item, LOCALE)
2605                   if Babel.digits[temp] then
2606                     local chr = item.char
2607                     if chr > 47 and chr < 58 then
2608                       item.char = Babel.digits[temp][chr-47]
2609                     end
2610                   end
2611                 elseif item.id == node.id'math' then
2612                   inmath = (item.subtype == 0)
2613                 end
2614               end
2615               return head
2616             end
2617           end
2618        }}%
```

```
2619        \fi
2620      \fi
2621      % == Counters: alph, Alph ==
2622      % What if extras<lang> contains a \babel@save\@alph? It won't be
2623      % restored correctly when exiting the language, so we ignore
2624      % this change with the \bbl@alph@saved trick.
2625      \ifx\bbl@KVP@alph\@nnil\else
2626        \bbl@extras@wrap{\\\bbl@alph@saved}%
2627          {\let\bbl@alph@saved\@alph}%
2628          {\let\@alph\bbl@alph@saved
2629           \babel@save\@alph}%
2630        \bbl@exp{%
2631          \\\bbl@add\<extras\languagename>{%
2632            \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2633      \fi
2634      \ifx\bbl@KVP@Alph\@nnil\else
2635        \bbl@extras@wrap{\\\bbl@Alph@saved}%
2636          {\let\bbl@Alph@saved\@Alph}%
2637          {\let\@Alph\bbl@Alph@saved
2638           \babel@save\@Alph}%
2639        \bbl@exp{%
2640          \\\bbl@add\<extras\languagename>{%
2641            \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2642      \fi
2643      % == Calendars ==
2644      \ifx\bbl@KVP@calendar\@nnil
2645        \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2646      \fi
2647      \def\bbl@tempe##1 ##2\@@{% Get first calendar
2648        \def\bbl@tempa{##1}%
2649        \bbl@exp{\\\bbl@tempe\bbl@KVP@calendar\space\\\@@}}%
2650      \def\bbl@tempe##1.##2.##3\@@{%
2651        \def\bbl@tempc{##1}%
2652        \def\bbl@tempb{##2}}%
2653      \expandafter\bbl@tempe\bbl@tempa..\@@
2654      \bbl@csarg\edef{calpr@\languagename}{%
2655        \ifx\bbl@tempc\@empty\else
2656          calendar=\bbl@tempc
2657        \fi
2658        \ifx\bbl@tempb\@empty\else
2659          ,variant=\bbl@tempb
2660        \fi}%
2661      % == require.babel in ini ==
2662      % To load or reload the babel-*.tex, if require.babel in ini
2663      \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
2664        \bbl@ifunset{bbl@rqtex@\languagename}{}%
2665          {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2666            \let\BabelBeforeIni\@gobbletwo
2667            \chardef\atcatcode=\catcode`\@
2668            \catcode`\@=11\relax
2669            \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2670            \catcode`\@=\atcatcode
2671            \let\atcatcode\relax
2672            \global\bbl@csarg\let{rqtex@\languagename}\relax
2673          \fi}%
2674        \bbl@foreach\bbl@calendars{%
2675          \bbl@ifunset{bbl@ca@##1}{%
2676            \chardef\atcatcode=\catcode`\@
2677            \catcode`\@=11\relax
2678            \InputIfFileExists{babel-ca-##1.tex}{}{}%
2679            \catcode`\@=\atcatcode
2680            \let\atcatcode\relax}%
2681          {}}%
```

```
2682  \fi
2683  % == frenchspacing ==
2684  \ifcase\bbl@howloaded\in@true\else\in@false\fi
2685  \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2686  \ifin@
2687    \bbl@extras@wrap{\\\bbl@pre@fs}%
2688      {\bbl@pre@fs}%
2689      {\bbl@post@fs}%
2690  \fi
2691  % == transforms ==
2692  \ifx\bbl@KVP@transforms\@nnil\else
2693    \def\bbl@elt##1##2##3{%
2694      \in@{$transforms.}{$##1}%
2695      \ifin@
2696        \def\bbl@tempa{##1}%
2697        \bbl@replace\bbl@tempa{transforms.}{}%
2698        \expandafter
2699        \bbl@transforms\csname babel\bbl@tempa\endcsname{##2}{##3}%
2700      \fi}%
2701    \csname bbl@inidata@\languagename\endcsname
2702    \bbl@release@transforms\relax % \relax closes the last item.
2703  \fi
2704  % == main ==
2705  \ifx\bbl@KVP@main\@nnil  % Restore only if not 'main'
2706    \let\languagename\bbl@savelangname
2707    \chardef\localeid\bbl@savelocaleid\relax
2708  \fi}
```

Depending on whether or not the language exists (based on \date<language>), we define two
macros. Remember \bbl@startcommands opens a group.

```
2709  \def\bbl@provide@new#1{%
2710  \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2711  \@namedef{extras#1}{}%
2712  \@namedef{noextras#1}{}%
2713  \bbl@startcommands*{#1}{captions}%
2714    \ifx\bbl@KVP@captions\@nnil %       and also if import, implicit
2715      \def\bbl@tempb##1{%              elt for \bbl@captionslist
2716        \ifx##1\@empty\else
2717          \bbl@exp{%
2718            \\\SetString\\##1{%
2719              \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2720          \expandafter\bbl@tempb
2721        \fi}%
2722      \expandafter\bbl@tempb\bbl@captionslist\@empty
2723    \else
2724      \ifx\bbl@initoload\relax
2725        \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2726      \else
2727        \bbl@read@ini{\bbl@initoload}2%      % Same
2728      \fi
2729    \fi
2730  \StartBabelCommands*{#1}{date}%
2731    \ifx\bbl@KVP@date\@nnil
2732      \bbl@exp{%
2733        \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2734    \else
2735      \bbl@savetoday
2736      \bbl@savedate
2737    \fi
2738  \bbl@endcommands
2739  \bbl@load@basic{#1}%
2740  % == hyphenmins == (only if new)
2741  \bbl@exp{%
```

```
2742        \gdef\<#1hyphenmins>{%
2743          {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2744          {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2745    % == hyphenrules (also in renew) ==
2746    \bbl@provide@hyphens{#1}%
2747    \ifx\bbl@KVP@main\@nnil\else
2748      \expandafter\main@language\expandafter{#1}%
2749    \fi}
2750 %
2751 \def\bbl@provide@renew#1{%
2752    \ifx\bbl@KVP@captions\@nnil\else
2753      \StartBabelCommands*{#1}{captions}%
2754        \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2755      \EndBabelCommands
2756    \fi
2757    \ifx\bbl@KVP@date\@nnil\else
2758      \StartBabelCommands*{#1}{date}%
2759        \bbl@savetoday
2760        \bbl@savedate
2761      \EndBabelCommands
2762    \fi
2763    % == hyphenrules (also in new) ==
2764    \ifx\bbl@lbkflag\@empty
2765      \bbl@provide@hyphens{#1}%
2766    \fi}
```

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.)

```
2767 \def\bbl@load@basic#1{%
2768    \ifcase\bbl@howloaded\or\or
2769      \ifcase\csname bbl@llevel@\languagename\endcsname
2770        \bbl@csarg\let{lname@\languagename}\relax
2771      \fi
2772    \fi
2773    \bbl@ifunset{bbl@lname@#1}%
2774      {\def\BabelBeforeIni##1##2{%
2775         \begingroup
2776           \let\bbl@ini@captions@aux\@gobbletwo
2777           \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2778           \bbl@read@ini{##1}1%
2779           \ifx\bbl@initoload\relax\endinput\fi
2780         \endgroup}%
2781       \begingroup        % boxed, to avoid extra spaces:
2782         \ifx\bbl@initoload\relax
2783           \bbl@input@texini{#1}%
2784         \else
2785           \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2786         \fi
2787       \endgroup}%
2788      {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
2789 \def\bbl@provide@hyphens#1{%
2790    \let\bbl@tempa\relax
2791    \ifx\bbl@KVP@hyphenrules\@nnil\else
2792      \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2793      \bbl@foreach\bbl@KVP@hyphenrules{%
2794        \ifx\bbl@tempa\relax     % if not yet found
2795          \bbl@ifsamestring{##1}{+}%
2796            {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
2797            {}%
2798          \bbl@ifunset{l@##1}%
2799            {}%
```

```
2800              {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
2801        \fi}%
2802    \fi
2803    \ifx\bbl@tempa\relax %        if no opt or no language in opt found
2804      \ifx\bbl@KVP@import\@nnil
2805        \ifx\bbl@initoload\relax\else
2806          \bbl@exp{%              and hyphenrules is not empty
2807            \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2808              {}%
2809              {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2810        \fi
2811      \else % if importing
2812        \bbl@exp{%                 and hyphenrules is not empty
2813          \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2814            {}%
2815            {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2816      \fi
2817    \fi
2818    \bbl@ifunset{bbl@tempa}%       ie, relax or undefined
2819      {\bbl@ifunset{l@#1}%         no hyphenrules found - fallback
2820        {\bbl@exp{\\\adddialect\<l@#1>\language}}%
2821        {}}%                       so, l@<lang> is ok - nothing to do
2822      {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
```

The reader of babel-...tex files. We reset temporarily some catcodes.

```
2823 \def\bbl@input@texini#1{%
2824   \bbl@bsphack
2825     \bbl@exp{%
2826       \catcode`\\\%=14 \catcode`\\\\=0
2827       \catcode`\\\{=1  \catcode`\\\}=2
2828       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
2829       \catcode`\\\%=\the\catcode`\%\relax
2830       \catcode`\\\\=\the\catcode`\\\relax
2831       \catcode`\\\{=\the\catcode`\{\relax
2832       \catcode`\\\}=\the\catcode`\}\relax}%
2833   \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.

```
2834 \def\bbl@iniline#1\bbl@iniline{%
2835   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2836 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2837 \def\bbl@iniskip#1\@@{}%        if starts with ;
2838 \def\bbl@inistore#1=#2\@@{%     full (default)
2839   \bbl@trim@def\bbl@tempa{#1}%
2840   \bbl@trim\toks@{#2}%
2841   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2842   \ifin@\else
2843     \bbl@xin@{,identification/include.}%
2844             {,\bbl@section/\bbl@tempa}%
2845     \ifin@\edef\bbl@required@inis{\the\toks@}\fi
2846     \bbl@exp{%
2847       \\\g@addto@macro\\\bbl@inidata{%
2848         \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2849   \fi}
2850 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2851   \bbl@trim@def\bbl@tempa{#1}%
2852   \bbl@trim\toks@{#2}%
2853   \bbl@xin@{.identification.}{.\bbl@section.}%
2854   \ifin@
2855     \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2856       \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2857   \fi}
```

Now, the 'main loop', which **must be executed inside a group**. At this point, \bbl@inidata may contain data declared in \babelprovide, 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 \babelprovide it's either 1 or 2.

```
2858 \def\bbl@loop@ini{%
2859   \loop
2860     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2861       \endlinechar\m@ne
2862       \read\bbl@readstream to \bbl@line
2863       \endlinechar`\^^M
2864       \ifx\bbl@line\@empty\else
2865         \expandafter\bbl@iniline\bbl@line\bbl@iniline
2866       \fi
2867     \repeat}
2868 \ifx\bbl@readstream\@undefined
2869   \csname newread\endcsname\bbl@readstream
2870 \fi
2871 \def\bbl@read@ini#1#2{%
2872   \global\let\bbl@extend@ini\@gobble
2873   \openin\bbl@readstream=babel-#1.ini
2874   \ifeof\bbl@readstream
2875     \bbl@error
2876       {There is no ini file for the requested language\\%
2877        (#1: \languagename). Perhaps you misspelled it or your\\%
2878        installation is not complete.}%
2879       {Fix the name or reinstall babel.}%
2880   \else
2881     % == Store ini data in \bbl@inidata ==
2882     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2883     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2884     \bbl@info{Importing
2885               \ifcase#2font and identification \or basic \fi
2886                 data for \languagename\\%
2887               from babel-#1.ini. Reported}%
2888     \ifnum#2=\z@
2889       \global\let\bbl@inidata\@empty
2890       \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2891     \fi
2892     \def\bbl@section{identification}%
2893     \let\bbl@required@inis\@empty
2894     \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2895     \bbl@inistore load.level=#2\@@
2896     \bbl@loop@ini
2897     \ifx\bbl@required@inis\@empty\else
2898       \bbl@replace\bbl@required@inis{ }{,}%
2899       \bbl@foreach\bbl@required@inis{%
2900         \openin\bbl@readstream=##1.ini
2901         \bbl@loop@ini}%
2902     \fi
2903     % == Process stored data ==
2904     \bbl@csarg\xdef{lini@\languagename}{#1}%
2905     \bbl@read@ini@aux
2906     % == 'Export' data ==
2907     \bbl@ini@exports{#2}%
2908     \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2909     \global\let\bbl@inidata\@empty
2910     \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
2911     \bbl@toglobal\bbl@ini@loaded
2912   \fi}
2913 \def\bbl@read@ini@aux{%
2914   \let\bbl@savestrings\@empty
```

122

```
2915    \let\bbl@savetoday\@empty
2916    \let\bbl@savedate\@empty
2917    \def\bbl@elt##1##2##3{%
2918      \def\bbl@section{##1}%
2919      \in@{=date.}{=##1}% Find a better place
2920      \ifin@
2921        \bbl@ifunset{bbl@inikv@##1}%
2922          {\bbl@ini@calendar{##1}}%
2923          {}%
2924      \fi
2925      \in@{=identification/extension.}{=##1/##2}%
2926      \ifin@
2927        \bbl@ini@extension{##2}%
2928      \fi
2929      \bbl@ifunset{bbl@inikv@##1}{}%
2930        {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2931    \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.

```
2932  \def\bbl@extend@ini@aux#1{%
2933    \bbl@startcommands*{#1}{captions}%
2934      % Activate captions/... and modify exports
2935      \bbl@csarg\def{inikv@captions.licr}##1##2{%
2936        \setlocalecaption{#1}{##1}{##2}}%
2937      \def\bbl@inikv@captions##1##2{%
2938        \bbl@ini@captions@aux{##1}{##2}}%
2939      \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2940      \def\bbl@exportkey##1##2##3{%
2941        \bbl@ifunset{bbl@@kv@##2}{}%
2942          {\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
2943            \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@@kv@##2>}%
2944          \fi}}%
2945      % As with \bbl@read@ini, but with some changes
2946      \bbl@read@ini@aux
2947      \bbl@ini@exports\tw@
2948      % Update inidata@lang by pretending the ini is read.
2949      \def\bbl@elt##1##2##3{%
2950        \def\bbl@section{##1}%
2951        \bbl@iniline##2=##3\bbl@iniline}%
2952      \csname bbl@inidata@#1\endcsname
2953      \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2954    \StartBabelCommands*{#1}{date}% And from the import stuff
2955      \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2956      \bbl@savetoday
2957      \bbl@savedate
2958    \bbl@endcommands}
```

A somewhat hackish tool to handle calendar sections. TODO. To be improved.

```
2959  \def\bbl@ini@calendar#1{%
2960    \lowercase{\def\bbl@tempa{=#1=}}%
2961    \bbl@replace\bbl@tempa{=date.gregorian}{}%
2962    \bbl@replace\bbl@tempa{=date.}{}%
2963    \in@{.licr=}{#1=}%
2964    \ifin@
2965      \ifcase\bbl@engine
2966        \bbl@replace\bbl@tempa{.licr=}{}%
2967      \else
2968        \let\bbl@tempa\relax
2969      \fi
2970    \fi
2971    \ifx\bbl@tempa\relax\else
2972      \bbl@replace\bbl@tempa{=}{}%
2973      \ifx\bbl@tempa\@empty\else
```

```
2974        \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2975     \fi
2976     \bbl@exp{%
2977       \def\<bbl@inikv@#1>####1####2{%
2978         \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2979  \fi}
```

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

```
2980 \def\bbl@renewinikey#1/#2\@@#3{%
2981   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2982   \edef\bbl@tempb{\zap@space #2 \@empty}%   key
2983   \bbl@trim\toks@{#3}%                       value
2984   \bbl@exp{%
2985     \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2986     \\\g@addto@macro\\\bbl@inidata{%
2987       \\\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}}%
```

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

```
2988 \def\bbl@exportkey#1#2#3{%
2989   \bbl@ifunset{bbl@@kv@#2}%
2990     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2991     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2992       \bbl@csarg\gdef{#1@\languagename}{#3}%
2993      \else
2994       \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
2995     \fi}}
```

Key-value pairs are treated differently depending on the section in the ini file. The following macros
are the readers for `identification` and `typography`. Note `\bbl@ini@exports` is called always (via
`\bbl@inisec`), while `\bbl@after@ini` must be called explicitly after `\bbl@read@ini` if necessary.

```
2996 \def\bbl@iniwarning#1{%
2997   \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2998     {\bbl@warning{%
2999       From babel-\bbl@cs{lini@\languagename}.ini:\\%
3000       \bbl@cs{@kv@identification.warning#1}\\%
3001       Reported }}}
3002 %
3003 \let\bbl@release@transforms\@empty
```

BCP 47 extensions are separated by a single letter (eg, `latin-x-medieval`. The following macro
handles this special case to create correctly the correspondig info.

```
3004 \def\bbl@ini@extension#1{%
3005   \def\bbl@tempa{#1}%
3006   \bbl@replace\bbl@tempa{extension.}{}%
3007   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
3008   \bbl@ifunset{bbl@info@#1}%
3009     {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
3010      \bbl@exp{%
3011        \\\g@addto@macro\\\bbl@moreinfo{%
3012          \\\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}}%
3013     {}}
3014 \let\bbl@moreinfo\@empty
3015 %
3016 \def\bbl@ini@exports#1{%
3017  % Identification always exported
3018   \bbl@iniwarning{}%
3019   \ifcase\bbl@engine
3020     \bbl@iniwarning{.pdflatex}%
3021   \or
3022     \bbl@iniwarning{.lualatex}%
```

```
3023    \or
3024      \bbl@iniwarning{.xelatex}%
3025    \fi%
3026    \bbl@exportkey{llevel}{identification.load.level}{}%
3027    \bbl@exportkey{elname}{identification.name.english}{}%
3028    \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
3029      {\csname bbl@elname@\languagename\endcsname}}%
3030    \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
3031    \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
3032    \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3033    \bbl@exportkey{esname}{identification.script.name}{}%
3034    \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
3035      {\csname bbl@esname@\languagename\endcsname}}%
3036    \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
3037    \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3038    \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
3039    \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
3040    \bbl@moreinfo
3041    % Also maps bcp47 -> languagename
3042    \ifbbl@bcptoname
3043      \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
3044    \fi
3045    % Conditional
3046    \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
3047      \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3048      \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3049      \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3050      \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3051      \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3052      \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3053      \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3054      \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3055      \bbl@exportkey{intsp}{typography.intraspace}{}%
3056      \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3057      \bbl@exportkey{chrng}{characters.ranges}{}%
3058      \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3059      \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3060      \ifnum#1=\tw@           % only (re)new
3061        \bbl@exportkey{rqtex}{identification.require.babel}{}%
3062        \bbl@toglobal\bbl@savetoday
3063        \bbl@toglobal\bbl@savedate
3064        \bbl@savestrings
3065      \fi
3066    \fi}
```

A shared handler for key=val lines to be stored in \bbl@@kv@<section>.<key>.

```
3067 \def\bbl@inikv#1#2{%        key=value
3068   \toks@{#2}%               This hides #'s from ini values
3069   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3070 \let\bbl@inikv@identification\bbl@inikv
3071 \let\bbl@inikv@date\bbl@inikv
3072 \let\bbl@inikv@typography\bbl@inikv
3073 \let\bbl@inikv@characters\bbl@inikv
3074 \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'.

```
3075 \def\bbl@inikv@counters#1#2{%
3076   \bbl@ifsamestring{#1}{digits}%
3077     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3078               decimal digits}%
```

```
3079              {Use another name.}}%
3080      {}%
3081  \def\bbl@tempc{#1}%
3082  \bbl@trim@def{\bbl@tempb*}{#2}%
3083  \in@{.1$}{#1$}%
3084  \ifin@
3085    \bbl@replace\bbl@tempc{.1}{}%
3086    \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3087      \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3088  \fi
3089  \in@{.F.}{#1}%
3090  \ifin@\else\in@{.S.}{#1}\fi
3091  \ifin@
3092    \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3093  \else
3094    \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3095    \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3096    \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3097  \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.

```
3098 \ifcase\bbl@engine
3099   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3100     \bbl@ini@captions@aux{#1}{#2}}
3101 \else
3102   \def\bbl@inikv@captions#1#2{%
3103     \bbl@ini@captions@aux{#1}{#2}}
3104 \fi
```

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

```
3105 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3106   \bbl@replace\bbl@tempa{.template}{}%
3107   \def\bbl@toreplace{#1{}}%
3108   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3109   \bbl@replace\bbl@toreplace{[[}{\csname}%
3110   \bbl@replace\bbl@toreplace{[}{\csname the}%
3111   \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3112   \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3113   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3114   \ifin@
3115     \@nameuse{bbl@patch\bbl@tempa}%
3116     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3117   \fi
3118   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3119   \ifin@
3120     \toks@\expandafter{\bbl@toreplace}%
3121     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3122   \fi}
3123 \def\bbl@ini@captions@aux#1#2{%
3124   \bbl@trim@def\bbl@tempa{#1}%
3125   \bbl@xin@{.template}{\bbl@tempa}%
3126   \ifin@
3127     \bbl@ini@captions@template{#2}\languagename
3128   \else
3129     \bbl@ifblank{#2}%
3130       {\bbl@exp{%
3131         \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3132       {\bbl@trim\toks@{#2}}%
3133     \bbl@exp{%
3134       \\\bbl@add\\\bbl@savestrings{%
3135         \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3136     \toks@\expandafter{\bbl@captionslist}%
```

```
3137     \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3138     \ifin@\else
3139       \bbl@exp{%
3140         \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3141         \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3142     \fi
3143   \fi}
```

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

```
3144 \def\bbl@list@the{%
3145   part,chapter,section,subsection,subsubsection,paragraph,%
3146   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3147   table,page,footnote,mpfootnote,mpfn}
3148 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3149   \bbl@ifunset{bbl@map@#1@\languagename}%
3150     {\@nameuse{#1}}%
3151     {\@nameuse{bbl@map@#1@\languagename}}}
3152 \def\bbl@inikv@labels#1#2{%
3153   \in@{.map}{#1}%
3154   \ifin@
3155     \ifx\bbl@KVP@labels\@nnil\else
3156       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3157       \ifin@
3158         \def\bbl@tempc{#1}%
3159         \bbl@replace\bbl@tempc{.map}{}%
3160         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3161         \bbl@exp{%
3162           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3163             {\ifin@\<#2>\else\\\localcounter{#2}\fi}}%
3164         \bbl@foreach\bbl@list@the{%
3165           \bbl@ifunset{the##1}{}%
3166             {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3167              \bbl@exp{%
3168                \\\bbl@sreplace\<the##1>%
3169                  {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3170                \\\bbl@sreplace\<the##1>%
3171                  {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3172              \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3173                \toks@\expandafter\expandafter\expandafter{%
3174                  \csname the##1\endcsname}%
3175                \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3176             \fi}}%
3177       \fi
3178     \fi
3179 %
3180   \else
3181     %
3182     % The following code is still under study. You can test it and make
3183     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3184     % language dependent.
3185     \in@{enumerate.}{#1}%
3186     \ifin@
3187       \def\bbl@tempa{#1}%
3188       \bbl@replace\bbl@tempa{enumerate.}{}%
3189       \def\bbl@toreplace{#2}%
3190       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3191       \bbl@replace\bbl@toreplace{[}{\csname the}%
3192       \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3193       \toks@\expandafter{\bbl@toreplace}%
3194       % TODO. Execute only once:
3195       \bbl@exp{%
3196         \\\bbl@add\<extras\languagename>{%
3197           \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
```

```
3198          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3199        \\\bbl@toglobal\<extras\languagename>}%
3200    \fi
3201  \fi}
```

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

```
3202 \def\bbl@chaptype{chapter}
3203 \ifx\@makechapterhead\@undefined
3204   \let\bbl@patchchapter\relax
3205 \else\ifx\thechapter\@undefined
3206   \let\bbl@patchchapter\relax
3207 \else\ifx\ps@headings\@undefined
3208   \let\bbl@patchchapter\relax
3209 \else
3210   \def\bbl@patchchapter{%
3211     \global\let\bbl@patchchapter\relax
3212     \gdef\bbl@chfmt{%
3213       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3214         {\@chapapp\space\thechapter}
3215         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}
3216     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3217     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3218     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3219     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3220     \bbl@toglobal\appendix
3221     \bbl@toglobal\ps@headings
3222     \bbl@toglobal\chaptermark
3223     \bbl@toglobal\@makechapterhead}
3224   \let\bbl@patchappendix\bbl@patchchapter
3225 \fi\fi\fi
3226 \ifx\@part\@undefined
3227   \let\bbl@patchpart\relax
3228 \else
3229   \def\bbl@patchpart{%
3230     \global\let\bbl@patchpart\relax
3231     \gdef\bbl@partformat{%
3232       \bbl@ifunset{bbl@partfmt@\languagename}%
3233         {\partname\nobreakspace\thepart}
3234         {\@nameuse{bbl@partfmt@\languagename}}}
3235     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3236     \bbl@toglobal\@part}
3237 \fi
```

Date. Arguments (year, month, day) are *not* protected, on purpose. In \today, arguments are always gregorian, and therefore always converted with other calendars. TODO. Document

```
3238 \let\bbl@calendar\@empty
3239 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3240 \def\bbl@localedate#1#2#3#4{%
3241   \begingroup
3242     \edef\bbl@they{#2}%
3243     \edef\bbl@them{#3}%
3244     \edef\bbl@thed{#4}%
3245     \edef\bbl@tempe{%
3246       \bbl@ifunset{bbl@calpr@\languagename}{}{\bbl@cl{calpr}},%
3247       #1}%
3248     \bbl@replace\bbl@tempe{ }{}%
3249     \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3250     \bbl@replace\bbl@tempe{convert}{convert=}%
3251     \let\bbl@ld@calendar\@empty
3252     \let\bbl@ld@variant\@empty
3253     \let\bbl@ld@convert\relax
```

```
3254      \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3255      \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3256      \bbl@replace\bbl@ld@calendar{gregorian}{}%
3257      \ifx\bbl@ld@calendar\@empty\else
3258        \ifx\bbl@ld@convert\relax\else
3259          \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3260            {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3261        \fi
3262      \fi
3263      \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3264      \edef\bbl@calendar{% Used in \month..., too
3265        \bbl@ld@calendar
3266        \ifx\bbl@ld@variant\@empty\else
3267          .\bbl@ld@variant
3268        \fi}%
3269      \bbl@cased
3270        {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3271          \bbl@they\bbl@them\bbl@thed}%
3272    \endgroup}
3273 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3274 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3275    \bbl@trim@def\bbl@tempa{#1.#2}%
3276    \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3277      {\bbl@trim@def\bbl@tempa{#3}%
3278      \bbl@trim\toks@{#5}%
3279      \@temptokena\expandafter{\bbl@savedate}%
3280      \bbl@exp{%   Reverse order - in ini last wins
3281        \def\\\bbl@savedate{%
3282          \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3283          \the\@temptokena}}}%
3284      {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3285        {\lowercase{\def\bbl@tempb{#6}}%
3286        \bbl@trim@def\bbl@toreplace{#5}%
3287        \bbl@TG@@date
3288        \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3289        \ifx\bbl@savetoday\@empty
3290          \bbl@exp{% TODO. Move to a better place.
3291            \\\AfterBabelCommands{%
3292              \def\<\languagename date>{\\\protect\<\languagename date >}%
3293              \\\newcommand\<\languagename date >[4][]{%
3294                \\\bbl@usedategrouptrue
3295                \<bbl@ensure@\languagename>{%
3296                  \\\localedate[####1]{####2}{####3}{####4}}}}%
3297            \def\\\bbl@savetoday{%
3298              \\\SetString\\\today{%
3299                \<\languagename date>[convert]%
3300                  {\\\the\year}{\\\the\month}{\\\the\day}}}}%
3301        \fi}%
3302      {}}}
```

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).

```
3303 \let\bbl@calendar\@empty
3304 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3305    \@nameuse{bbl@ca@#2}#1\@@}
3306 \newcommand\BabelDateSpace{\nobreakspace}
3307 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3308 \newcommand\BabelDated[1]{{\number#1}}
3309 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3310 \newcommand\BabelDateM[1]{{\number#1}}
```

```
3311 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3312 \newcommand\BabelDateMMMM[1]{{%
3313   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3314 \newcommand\BabelDatey[1]{{\number#1}}%
3315 \newcommand\BabelDateyy[1]{{%
3316   \ifnum#1<10 0\number#1 %
3317   \else\ifnum#1<100 \number#1 %
3318   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3319   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3320   \else
3321     \bbl@error
3322       {Currently two-digit years are restricted to the\\
3323        range 0-9999.}%
3324       {There is little you can do. Sorry.}%
3325   \fi\fi\fi\fi}}
3326 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3327 \def\bbl@replace@finish@iii#1{%
3328   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3329 \def\bbl@TG@@date{%
3330   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3331   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3332   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3333   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3334   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3335   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3336   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3337   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3338   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3339   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3340   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3341   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3342   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3343   \bbl@replace@finish@iii\bbl@toreplace}
3344 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3345 \def\bbl@xdatecntr[#1|#2]{\localnumeral{#2}{#1}}
```

Transforms.

```
3346 \let\bbl@release@transforms\@empty
3347 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3348 \bbl@csarg\let{bbl@inikv@transforms.posthyphenation}\bbl@inikv
3349 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3350   #1[#2]{#3}{#4}{#5}}
3351 \begingroup %  A hack. TODO. Don't require an specific order
3352   \catcode`\%=12
3353   \catcode`\&=14
3354   \gdef\bbl@transforms#1#2#3{&%
3355     \directlua{
3356       local str = [==[#2]==]
3357       str = str:gsub('%.%d+%.%d+$', '')
3358       tex.print([[\def\string\babeltempa{]] .. str .. [[}]])
3359     }&%
3360     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3361     \ifin@
3362       \in@{.0$}{#2$}&%
3363       \ifin@
3364         \directlua{% (\attribute) syntax
3365           local str = string.match([[\bbl@KVP@transforms]],
3366                       '%(([^%(]-)%)[^%]-\babeltempa')
3367           if str == nil then
3368             tex.print([[\def\string\babeltempb{}]])
3369           else
3370             tex.print([[\def\string\babeltempb{,attribute=]] .. str .. [[}]])
3371           end
```

```
3372            }
3373          \toks@{#3}&%
3374          \bbl@exp{&%
3375            \\\g@addto@macro\\\bbl@release@transforms{&%
3376              \relax  &% Closes previous \bbl@transforms@aux
3377              \\\bbl@transforms@aux
3378                \\#1{label=\babeltempa\babeltempb}{\languagename}{\the\toks@}}}&%
3379        \else
3380          \g@addto@macro\bbl@release@transforms{, {#3}}&%
3381        \fi
3382      \fi}
3383 \endgroup
```

Language and Script values to be used when defining a font or setting the direction are set with the following macros.

```
3384 \def\bbl@provide@lsys#1{%
3385   \bbl@ifunset{bbl@lname@#1}%
3386     {\bbl@load@info{#1}}%
3387     {}%
3388   \bbl@csarg\let{lsys@#1}\@empty
3389   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3390   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3391   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3392   \bbl@ifunset{bbl@lname@#1}{}%
3393     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3394   \ifcase\bbl@engine\or\or
3395     \bbl@ifunset{bbl@prehc@#1}{}%
3396       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3397         {}%
3398         {\ifx\bbl@xenohyph\@undefined
3399            \global\let\bbl@xenohyph\bbl@xenohyph@d
3400            \ifx\AtBeginDocument\@notprerr
3401              \expandafter\@secondoftwo  % to execute right now
3402            \fi
3403            \AtBeginDocument{%
3404              \bbl@patchfont{\bbl@xenohyph}%
3405              \expandafter\selectlanguage\expandafter{\languagename}}%
3406         \fi}}%
3407   \fi
3408   \bbl@csarg\bbl@toglobal{lsys@#1}}
3409 \def\bbl@xenohyph@d{%
3410   \bbl@ifset{bbl@prehc@\languagename}%
3411     {\ifnum\hyphenchar\font=\defaulthyphenchar
3412        \iffontchar\font\bbl@cl{prehc}\relax
3413          \hyphenchar\font\bbl@cl{prehc}\relax
3414        \else\iffontchar\font"200B
3415          \hyphenchar\font"200B
3416        \else
3417          \bbl@warning
3418            {Neither 0 nor ZERO WIDTH SPACE are available\\%
3419             in the current font, and therefore the hyphen\\%
3420             will be printed. Try changing the fontspec's\\%
3421             'HyphenChar' to another value, but be aware\\%
3422             this setting is not safe (see the manual)}%
3423          \hyphenchar\font\defaulthyphenchar
3424        \fi\fi
3425     \fi}%
3426     {\hyphenchar\font\defaulthyphenchar}}
3427 % \fi}
```

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

```
3428 \def\bbl@load@info#1{%
3429   \def\BabelBeforeIni##1##2{%
3430     \begingroup
3431       \bbl@read@ini{##1}0%
3432       \endinput        % babel- .tex may contain onlypreamble's
3433     \endgroup}%          boxed, to avoid extra spaces:
3434   {\bbl@input@texini{#1}}}
```

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.

```
3435 \def\bbl@setdigits#1#2#3#4#5{%
3436   \bbl@exp{%
3437     \def\<\languagename digits>####1{%      ie, \langdigits
3438       \<bbl@digits@\languagename>####1\\\@nil}%
3439     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3440     \def\<\languagename counter>####1{%     ie, \langcounter
3441       \\\expandafter\<bbl@counter@\languagename>%
3442       \\\csname c@####1\endcsname}%
3443     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3444       \\\expandafter\<bbl@digits@\languagename>%
3445       \\\number####1\\\@nil}}%
3446   \def\bbl@tempa##1##2##3##4##5{%
3447     \bbl@exp{%    Wow, quite a lot of hashes! :-(
3448       \def\<bbl@digits@\languagename>########1{%
3449        \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
3450        \\\else
3451          \\\ifx0########1#1%
3452          \\\else\\\ifx1########1#2%
3453          \\\else\\\ifx2########1#3%
3454          \\\else\\\ifx3########1#4%
3455          \\\else\\\ifx4########1#5%
3456          \\\else\\\ifx5########1##1%
3457          \\\else\\\ifx6########1##2%
3458          \\\else\\\ifx7########1##3%
3459          \\\else\\\ifx8########1##4%
3460          \\\else\\\ifx9########1##5%
3461          \\\else########1%
3462          \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3463          \\\expandafter\<bbl@digits@\languagename>%
3464        \\\fi}}%
3465   \bbl@tempa}
```

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

```
3466 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3467   \ifx\\#1%                % \\ before, in case #1 is multiletter
3468     \bbl@exp{%
3469       \def\\\bbl@tempa####1{%
3470         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3471   \else
3472     \toks@\expandafter{\the\toks@\or #1}%
3473     \expandafter\bbl@buildifcase
3474   \fi}
```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before `\@@` collects digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey `.F.`, the number after is treated as an special case, for a fixed form (see `babel-he.ini`, for example).

```
3475 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3476 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3477 \newcommand\localecounter[2]{%
3478   \expandafter\bbl@localecntr
3479   \expandafter{\number\csname c@#2\endcsname}{#1}}
```

```
3480 \def\bbl@alphnumeral#1#2{%
3481   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3482 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3483   \ifcase\@car#8\@nil\or   % Currenty <10000, but prepared for bigger
3484     \bbl@alphnumeral@ii{#9}000000#1\or
3485     \bbl@alphnumeral@ii{#9}00000#1#2\or
3486     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3487     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3488     \bbl@alphnum@invalid{>9999}%
3489   \fi}
3490 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3491   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3492     {\bbl@cs{cntr@#1.4@\languagename}#5%
3493      \bbl@cs{cntr@#1.3@\languagename}#6%
3494      \bbl@cs{cntr@#1.2@\languagename}#7%
3495      \bbl@cs{cntr@#1.1@\languagename}#8%
3496      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3497        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3498          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3499      \fi}%
3500     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3501 \def\bbl@alphnum@invalid#1{%
3502   \bbl@error{Alphabetic numeral too large (#1)}%
3503     {Currently this is the limit.}}
```

The information in the identification section can be useful, so the following macro just exposes it with a user command.

```
3504 \def\bbl@localeinfo#1#2{%
3505   \bbl@ifunset{bbl@info@#2}{#1}%
3506     {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
3507       {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3508 \newcommand\localeinfo[1]{%
3509   \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3510     \bbl@afterelse\bbl@localeinfo{}%
3511   \else
3512     \bbl@localeinfo
3513       {\bbl@error{I've found no info for the current locale.\\%
3514                   The corresponding ini file has not been loaded\\%
3515                   Perhaps it doesn't exist}%
3516                  {See the manual for details.}}%
3517       {#1}%
3518   \fi}
3519 % \@namedef{bbl@info@name.locale}{lcname}
3520 \@namedef{bbl@info@tag.ini}{lini}
3521 \@namedef{bbl@info@name.english}{elname}
3522 \@namedef{bbl@info@name.opentype}{lname}
3523 \@namedef{bbl@info@tag.bcp47}{tbcp}
3524 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3525 \@namedef{bbl@info@tag.opentype}{lotf}
3526 \@namedef{bbl@info@script.name}{esname}
3527 \@namedef{bbl@info@script.name.opentype}{sname}
3528 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3529 \@namedef{bbl@info@script.tag.opentype}{sotf}
3530 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3531 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3532 % Extensions are dealt with in a special way
3533 % Now, an internal \LaTeX{} macro:
3534 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}
```

With version 3.75 \BabelEnsureInfo is executed always, but there is an option to disable it.

```
3535 ⟨⟨*More package options⟩⟩ ≡
3536 \DeclareOption{ensureinfo=off}{}
3537 ⟨⟨/More package options⟩⟩
3538 %
```

```
3539 \let\bbl@ensureinfo\@gobble
3540 \newcommand\BabelEnsureInfo{%
3541   \ifx\InputIfFileExists\@undefined\else
3542     \def\bbl@ensureinfo##1{%
3543       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3544   \fi
3545   \bbl@foreach\bbl@loaded{{%
3546     \def\languagename{##1}%
3547     \bbl@ensureinfo{##1}}}}
3548 \@ifpackagewith{babel}{ensureinfo=off}{}%
3549   {\AtEndOfPackage{% Test for plain.
3550     \ifx\@undefined\bbl@loaded\else\BabelEnsureInfo\fi}}
```

More general, but non-expandable, is \getlocaleproperty. To inspect every possible loaded ini, we define \LocaleForEach, where \bbl@ini@loaded is a comma-separated list of locales, built by \bbl@read@ini.

```
3551 \newcommand\getlocaleproperty{%
3552   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3553 \def\bbl@getproperty@s#1#2#3{%
3554   \let#1\relax
3555   \def\bbl@elt##1##2##3{%
3556     \bbl@ifsamestring{##1/##2}{#3}%
3557       {\providecommand#1{##3}%
3558        \def\bbl@elt####1####2####3{}}%
3559       {}}%
3560   \bbl@cs{inidata@#2}}%
3561 \def\bbl@getproperty@x#1#2#3{%
3562   \bbl@getproperty@s{#1}{#2}{#3}%
3563   \ifx#1\relax
3564     \bbl@error
3565       {Unknown key for locale '#2':\\%
3566        #3\\%
3567        \string#1 will be set to \relax}%
3568       {Perhaps you misspelled it.}%
3569   \fi}
3570 \let\bbl@ini@loaded\@empty
3571 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
```

8   Adjusting the Babel bahavior

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

```
3572 \newcommand\babeladjust[1]{%  TODO. Error handling.
3573   \bbl@forkv{#1}{%
3574     \bbl@ifunset{bbl@ADJ@##1@##2}%
3575       {\bbl@cs{ADJ@##1}{##2}}%
3576       {\bbl@cs{ADJ@##1@##2}}}}
3577 %
3578 \def\bbl@adjust@lua#1#2{%
3579   \ifvmode
3580     \ifnum\currentgrouplevel=\z@
3581       \directlua{ Babel.#2 }%
3582       \expandafter\expandafter\expandafter\@gobble
3583     \fi
3584   \fi
3585   {\bbl@error   % The error is gobbled if everything went ok.
3586     {Currently, #1 related features can be adjusted only\\%
3587      in the main vertical list.}%
3588     {Maybe things change in the future, but this is what it is.}}}
3589 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3590   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3591 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3592   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
```

```
3593 \@namedef{bbl@ADJ@bidi.text@on}{%
3594   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3595 \@namedef{bbl@ADJ@bidi.text@off}{%
3596   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3597 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3598   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3599 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3600   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3601 %
3602 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3603   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3604 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3605   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3606 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3607   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3608 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3609   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3610 \@namedef{bbl@ADJ@justify.arabic@on}{%
3611   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3612 \@namedef{bbl@ADJ@justify.arabic@off}{%
3613   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3614 %
3615 \def\bbl@adjust@layout#1{%
3616   \ifvmode
3617     #1%
3618     \expandafter\@gobble
3619   \fi
3620   {\bbl@error   % The error is gobbled if everything went ok.
3621     {Currently, layout related features can be adjusted only\\%
3622      in vertical mode.}%
3623     {Maybe things change in the future, but this is what it is.}}}
3624 \@namedef{bbl@ADJ@layout.tabular@on}{%
3625   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3626 \@namedef{bbl@ADJ@layout.tabular@off}{%
3627   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3628 \@namedef{bbl@ADJ@layout.lists@on}{%
3629   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3630 \@namedef{bbl@ADJ@layout.lists@off}{%
3631   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3632 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3633   \bbl@activateposthyphen}
3634 %
3635 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3636   \bbl@bcpallowedtrue}
3637 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3638   \bbl@bcpallowedfalse}
3639 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3640   \def\bbl@bcp@prefix{#1}}
3641 \def\bbl@bcp@prefix{bcp47-}
3642 \@namedef{bbl@ADJ@autoload.options}#1{%
3643   \def\bbl@autoload@options{#1}}
3644 \let\bbl@autoload@bcpoptions\@empty
3645 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3646   \def\bbl@autoload@bcpoptions{#1}}
3647 \newif\ifbbl@bcptoname
3648 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3649   \bbl@bcptonametrue
3650   \BabelEnsureInfo}
3651 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3652   \bbl@bcptonamefalse}
3653 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3654   \directlua{ Babel.ignore_pre_char = function(node)
3655     return (node.lang == \the\csname l@nohyphenation\endcsname)
```

```
3656      end }}
3657 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3658   \directlua{ Babel.ignore_pre_char = function(node)
3659        return false
3660      end }}
3661 \@namedef{bbl@ADJ@select.write@shift}{%
3662   \let\bbl@restorelastskip\relax
3663   \def\bbl@savelastskip{%
3664      \let\bbl@restorelastskip\relax
3665      \ifvmode
3666        \ifdim\lastskip=\z@
3667          \let\bbl@restorelastskip\nobreak
3668        \else
3669          \bbl@exp{%
3670            \def\\\bbl@restorelastskip{%
3671              \skip@=\the\lastskip
3672              \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3673        \fi
3674      \fi}}
3675 \@namedef{bbl@ADJ@select.write@keep}{%
3676   \let\bbl@restorelastskip\relax
3677   \let\bbl@savelastskip\relax}
3678 \@namedef{bbl@ADJ@select.write@omit}{%
3679   \let\bbl@restorelastskip\relax
3680   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
```

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

```
3681 \ifx\directlua\@undefined\else
3682   \ifx\bbl@luapatterns\@undefined
3683     \input luababel.def
3684   \fi
3685 \fi
```

Continue with LaTeX.

```
3686 ⟨/package | core⟩
3687 ⟨∗package⟩
```

8.1 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.

```
3688 ⟨⟨∗More package options⟩⟩ ≡
3689 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3690 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3691 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3692 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3693 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3694 ⟨⟨/More package options⟩⟩
```

\@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.

```
3695 \bbl@trace{Cross referencing macros}
3696 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3697   \def\@newl@bel#1#2#3{%
```

```
3698    {\@safe@activestrue
3699    \bbl@ifunset{#1@#2}%
3700      \relax
3701      {\gdef\@multiplelabels{%
3702        \@latex@warning@no@line{There were multiply-defined labels}}%
3703      \@latex@warning@no@line{Label `#2' multiply defined}}%
3704    \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.

```
3705    \CheckCommand*\@testdef[3]{%
3706    \def\reserved@a{#3}%
3707    \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3708    \else
3709      \@tempswatrue
3710    \fi}
```

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.

```
3711    \def\@testdef#1#2#3{%  TODO. With @samestring?
3712    \@safe@activestrue
3713    \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3714    \def\bbl@tempb{#3}%
3715    \@safe@activesfalse
3716    \ifx\bbl@tempa\relax
3717    \else
3718      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3719    \fi
3720    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3721    \ifx\bbl@tempa\bbl@tempb
3722    \else
3723      \@tempswatrue
3724    \fi}
3725 \fi
```

\ref   The same holds for the macro \ref that references a label and \pageref to reference a page. We
\pageref make them robust as well (if they weren't already) to prevent problems if they should become
          expanded at the wrong moment.

```
3726 \bbl@xin@{R}\bbl@opt@safe
3727 \ifin@
3728   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3729   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3730     {\expandafter\strip@prefix\meaning\ref}%
3731   \ifin@
3732     \bbl@redefine\@kernel@ref#1{%
3733       \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3734     \bbl@redefine\@kernel@pageref#1{%
3735       \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3736     \bbl@redefine\@kernel@sref#1{%
3737       \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3738     \bbl@redefine\@kernel@spageref#1{%
3739       \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3740   \else
3741     \bbl@redefinerobust\ref#1{%
3742       \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3743     \bbl@redefinerobust\pageref#1{%
3744       \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3745   \fi
3746 \else
3747   \let\org@ref\ref
```

\@citex   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.

```
3750 \bbl@xin@{B}\bbl@opt@safe
3751 \ifin@
3752   \bbl@redefine\@citex[#1]#2{%
3753     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3754     \org@@citex[#1]{\@tempa}}
```

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.

```
3755   \AtBeginDocument{%
3756     \@ifpackageloaded{natbib}{%
```

Notice that we use \def here instead of \bbl@redefine because \org@@citex is already defined and we don't want to overwrite that definition (it would result in parameter stack overflow because of a circular definition).
(Recent versions of natbib change dynamically \@citex, so PR4087 doesn't seem fixable in a simple way. Just load natbib before.)

```
3757     \def\@citex[#1][#2]#3{%
3758       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3759       \org@@citex[#1][#2]{\@tempa}}%
3760     }{}}
```

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

```
3761   \AtBeginDocument{%
3762     \@ifpackageloaded{cite}{%
3763       \def\@citex[#1]#2{%
3764         \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3765       }{}}
```

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

```
3766   \bbl@redefine\nocite#1{%
3767     \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
```

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

```
3768   \bbl@redefine\bibcite{%
3769     \bbl@cite@choice
3770     \bibcite}
```

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

```
3771   \def\bbl@bibcite#1#2{%
3772     \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3773   \def\bbl@cite@choice{%
3774     \global\let\bibcite\bbl@bibcite
3775     \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3776     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3777     \global\let\bbl@cite@choice\relax}
```

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

```
3778    \AtBeginDocument{\bbl@cite@choice}
```

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

```
3779    \bbl@redefine\@bibitem#1{%
3780      \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3781 \else
3782    \let\org@nocite\nocite
3783    \let\org@@citex\@citex
3784    \let\org@bibcite\bibcite
3785    \let\org@@bibitem\@bibitem
3786 \fi
```

8.2 Marks

\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.

```
3787 \bbl@trace{Marks}
3788 \IfBabelLayout{sectioning}
3789   {\ifx\bbl@opt@headfoot\@nnil
3790     \g@addto@macro\@resetactivechars{%
3791       \set@typeset@protect
3792       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3793       \let\protect\noexpand
3794       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3795         \edef\thepage{%
3796           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3797       \fi}%
3798   \fi}
3799 {\ifbbl@single\else
3800   \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3801   \markright#1{%
3802     \bbl@ifblank{#1}%
3803       {\org@markright{}}%
3804       {\toks@{#1}%
3805        \bbl@exp{%
3806          \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3807            {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

\markboth The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token
\@mkboth 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 neeed 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.)

```
3808     \ifx\@mkboth\markboth
3809       \def\bbl@tempc{\let\@mkboth\markboth}
3810     \else
3811       \def\bbl@tempc{}
3812     \fi
3813   \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3814   \markboth#1#2{%
3815     \protected@edef\bbl@tempb##1{%
3816       \protect\foreignlanguage
3817       {\languagename}{\protect\bbl@restore@actives##1}}%
3818     \bbl@ifblank{#1}%
3819       {\toks@{}}%
```

```
3820          {\toks@\expandafter{\bbl@tempb{#1}}}%
3821        \bbl@ifblank{#2}%
3822          {\@temptokena{}}%
3823          {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3824        \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3825        \bbl@tempc
3826      \fi}  % end ifbbl@single, end \IfBabelLayout
```

8.3 Preventing clashes with other packages

8.3.1 `ifthen`

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

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

In order for this to work the argument of \isodd needs to be fully expandable. With the above
redefinition of \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.

```
3827 \bbl@trace{Preventing clashes with other packages}
3828 \ifx\org@ref\@undefined\else
3829   \bbl@xin@{R}\bbl@opt@safe
3830   \ifin@
3831     \AtBeginDocument{%
3832       \@ifpackageloaded{ifthen}{%
3833         \bbl@redefine@long\ifthenelse#1#2#3{%
3834           \let\bbl@temp@pref\pageref
3835           \let\pageref\org@pageref
3836           \let\bbl@temp@ref\ref
3837           \let\ref\org@ref
3838           \@safe@activestrue
3839           \org@ifthenelse{#1}%
3840             {\let\pageref\bbl@temp@pref
3841              \let\ref\bbl@temp@ref
3842              \@safe@activesfalse
3843              #2}%
3844             {\let\pageref\bbl@temp@pref
3845              \let\ref\bbl@temp@ref
3846              \@safe@activesfalse
3847              #3}%
3848         }%
3849       }{}%
3850     }
3851 \fi
```

8.3.2 `varioref`

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

```
3852     \AtBeginDocument{%
3853       \@ifpackageloaded{varioref}{%
3854         \bbl@redefine\@@vpageref#1[#2]#3{%
3855           \@safe@activestrue
```

```
3856          \org@@@vpageref{#1}[#2]{#3}%
3857          \@safe@activesfalse}%
3858        \bbl@redefine\vrefpagenum#1#2{%
3859          \@safe@activestrue
3860          \org@vrefpagenum{#1}{#2}%
3861          \@safe@activesfalse}%
```

The package varioref defines \Ref to be a robust command wich 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.

```
3862          \expandafter\def\csname Ref \endcsname#1{%
3863            \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3864          }{}%
3865      }
3866 \fi
```

8.3.3 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.

```
3867 \AtEndOfPackage{%
3868   \AtBeginDocument{%
3869     \@ifpackageloaded{hhline}%
3870       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3871        \else
3872          \makeatletter
3873          \def\@currname{hhline}\input{hhline.sty}\makeatother
3874        \fi}%
3875      {}}}
```

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

```
3876 \def\substitutefontfamily#1#2#3{%
3877   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3878   \immediate\write15{%
3879     \string\ProvidesFile{#1#2.fd}%
3880     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3881      \space generated font description file]^^J
3882     \string\DeclareFontFamily{#1}{#2}{}^^J
3883     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3884     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3885     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3886     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3887     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3888     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3889     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3890     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3891     }%
3892   \closeout15
3893   }
3894 \@onlypreamble\substitutefontfamily
```

8.4 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of TeX and LaTeX
always come out in the right encoding. There is a list of non-ASCII encodings. Requested encodings
are currently stored in \@fontenc@load@list. If a non-ASCII has been loaded, we define versions of

\TeX and \LaTeX for them using \ensureascii. The default ASCII encoding is set, too (in reverse order): the "main" encoding (when the document begins), the last loaded, or OT1.

\ensureascii

```
3895 \bbl@trace{Encoding and fonts}
3896 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3897 \newcommand\BabelNonText{TS1,T3,TS3}
3898 \let\org@TeX\TeX
3899 \let\org@LaTeX\LaTeX
3900 \let\ensureascii\@firstofone
3901 \AtBeginDocument{%
3902   \def\@elt#1{,#1,}%
3903   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3904   \let\@elt\relax
3905   \let\bbl@tempb\@empty
3906   \def\bbl@tempc{OT1}%
3907   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3908     \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3909   \bbl@foreach\bbl@tempa{%
3910     \bbl@xin@{#1}{\BabelNonASCII}%
3911     \ifin@
3912       \def\bbl@tempb{#1}% Store last non-ascii
3913     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3914       \ifin@\else
3915         \def\bbl@tempc{#1}% Store last ascii
3916       \fi
3917     \fi}%
3918   \ifx\bbl@tempb\@empty\else
3919     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3920     \ifin@\else
3921       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3922     \fi
3923     \edef\ensureascii#1{%
3924       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3925     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3926     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3927   \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.

```
3928 \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.

```
3929 \AtBeginDocument{%
3930   \@ifpackageloaded{fontspec}%
3931     {\xdef\latinencoding{%
3932       \ifx\UTFencname\@undefined
3933         EU\ifcase\bbl@engine\or2\or1\fi
3934       \else
3935         \UTFencname
3936       \fi}}%
3937     {\gdef\latinencoding{OT1}%
3938      \ifx\cf@encoding\bbl@t@one
3939        \xdef\latinencoding{\bbl@t@one}%
3940      \else
3941        \def\@elt#1{,#1,}%
```

142

```
3942        \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3943        \let\@elt\relax
3944        \bbl@xin@{,T1,}\bbl@tempa
3945        \ifin@
3946          \xdef\latinencoding{\bbl@t@one}%
3947        \fi
3948      \fi}}
```

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

```
3949 \DeclareRobustCommand{\latintext}{%
3950   \fontencoding{\latinencoding}\selectfont
3951   \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.

```
3952 \ifx\@undefined\DeclareTextFontCommand
3953   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3954 \else
3955   \DeclareTextFontCommand{\textlatin}{\latintext}
3956 \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).

```
3957 \def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}
```

8.5 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 LuaTeX-ja shows, vertical typesetting is possible, too.

```
3958 \bbl@trace{Loading basic (internal) bidi support}
3959 \ifodd\bbl@engine
3960 \else % TODO. Move to txtbabel
3961   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3962     \bbl@error
3963       {The bidi method 'basic' is available only in\\%
3964        luatex. I'll continue with 'bidi=default', so\\%
3965        expect wrong results}%
3966       {See the manual for further details.}%
3967     \let\bbl@beforeforeign\leavevmode
3968     \AtEndOfPackage{%
3969       \EnableBabelHook{babel-bidi}%
3970       \bbl@xebidipar}
```

```
3971    \fi\fi
3972    \def\bbl@loadxebidi#1{%
3973      \ifx\RTLfootnotetext\@undefined
3974        \AtEndOfPackage{%
3975          \EnableBabelHook{babel-bidi}%
3976          \bbl@loadfontspec % bidi needs fontspec
3977          \usepackage#1{bidi}}%
3978      \fi}
3979    \ifnum\bbl@bidimode>200
3980      \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3981        \bbl@tentative{bidi=bidi}
3982        \bbl@loadxebidi{}
3983      \or
3984        \bbl@loadxebidi{[rldocument]}
3985      \or
3986        \bbl@loadxebidi{}
3987      \fi
3988    \fi
3989  \fi
3990  % TODO? Separate:
3991  \ifnum\bbl@bidimode=\@ne
3992    \let\bbl@beforeforeign\leavevmode
3993    \ifodd\bbl@engine
3994      \newattribute\bbl@attr@dir
3995      \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3996      \bbl@exp{\output{\bodydir\pagedir\the\output}}
3997    \fi
3998    \AtEndOfPackage{%
3999      \EnableBabelHook{babel-bidi}%
4000      \ifodd\bbl@engine\else
4001        \bbl@xebidipar
4002      \fi}
4003  \fi
```

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

```
4004  \bbl@trace{Macros to switch the text direction}
4005  \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4006  \def\bbl@rscripts{% TODO. Base on codes ??
4007    ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4008    Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
4009    Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
4010    Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4011    Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4012    Old South Arabian,}%
4013  \def\bbl@provide@dirs#1{%
4014    \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4015    \ifin@
4016      \global\bbl@csarg\chardef{wdir@#1}\@ne
4017      \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4018      \ifin@
4019        \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
4020      \fi
4021    \else
4022      \global\bbl@csarg\chardef{wdir@#1}\z@
4023    \fi
4024    \ifodd\bbl@engine
4025      \bbl@csarg\ifcase{wdir@#1}%
4026        \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4027      \or
4028        \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4029      \or
4030        \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
```

```
4031      \fi
4032   \fi}
4033 \def\bbl@switchdir{%
4034   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4035   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4036   \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
4037 \def\bbl@setdirs#1{% TODO - math
4038   \ifcase\bbl@select@type % TODO - strictly, not the right test
4039     \bbl@bodydir{#1}%
4040     \bbl@pardir{#1}%
4041   \fi
4042   \bbl@textdir{#1}}
4043 % TODO. Only if \bbl@bidimode > 0?:
4044 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4045 \DisableBabelHook{babel-bidi}
```

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

```
4046 \ifodd\bbl@engine   % luatex=1
4047 \else % pdftex=0, xetex=2
4048   \newcount\bbl@dirlevel
4049   \chardef\bbl@thetextdir\z@
4050   \chardef\bbl@thepardir\z@
4051   \def\bbl@textdir#1{%
4052     \ifcase#1\relax
4053       \chardef\bbl@thetextdir\z@
4054       \bbl@textdir@i\beginL\endL
4055      \else
4056       \chardef\bbl@thetextdir\@ne
4057       \bbl@textdir@i\beginR\endR
4058     \fi}
4059   \def\bbl@textdir@i#1#2{%
4060     \ifhmode
4061       \ifnum\currentgrouplevel>\z@
4062         \ifnum\currentgrouplevel=\bbl@dirlevel
4063           \bbl@error{Multiple bidi settings inside a group}%
4064             {I'll insert a new group, but expect wrong results.}%
4065           \bgroup\aftergroup#2\aftergroup\egroup
4066         \else
4067           \ifcase\currentgrouptype\or % 0 bottom
4068             \aftergroup#2% 1 simple {}
4069           \or
4070             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4071           \or
4072             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4073           \or\or\or % vbox vtop align
4074           \or
4075             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4076           \or\or\or\or\or\or % output math disc insert vcent mathchoice
4077           \or
4078             \aftergroup#2% 14 \begingroup
4079           \else
4080             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4081           \fi
4082         \fi
4083         \bbl@dirlevel\currentgrouplevel
4084       \fi
4085       #1%
4086     \fi}
4087   \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4088   \let\bbl@bodydir\@gobble
4089   \let\bbl@pagedir\@gobble
4090   \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}
```

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).

```
4091 \def\bbl@xebidipar{%
4092   \let\bbl@xebidipar\relax
4093   \TeXXeTstate\@ne
4094   \def\bbl@xeeverypar{%
4095     \ifcase\bbl@thepardir
4096       \ifcase\bbl@thetextdir\else\beginR\fi
4097     \else
4098       {\setbox\z@\lastbox\beginR\box\z@}%
4099     \fi}%
4100   \let\bbl@severypar\everypar
4101   \newtoks\everypar
4102   \everypar=\bbl@severypar
4103   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4104 \ifnum\bbl@bidimode>200
4105   \let\bbl@textdir@i\@gobbletwo
4106   \let\bbl@xebidipar\@empty
4107   \AddBabelHook{bidi}{foreign}{%
4108     \def\bbl@tempa{\def\BabelText####1}%
4109     \ifcase\bbl@thetextdir
4110       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4111     \else
4112       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4113     \fi}
4114   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4115 \fi
4116 \fi
```

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

```
4117 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4118 \AtBeginDocument{%
4119   \ifx\pdfstringdefDisableCommands\@undefined\else
4120     \ifx\pdfstringdefDisableCommands\relax\else
4121       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4122     \fi
4123   \fi}
```

8.6 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.

```
4124 \bbl@trace{Local Language Configuration}
4125 \ifx\loadlocalcfg\@undefined
4126   \@ifpackagewith{babel}{noconfigs}%
4127     {\let\loadlocalcfg\@gobble}%
4128     {\def\loadlocalcfg#1{%
4129       \InputIfFileExists{#1.cfg}%
4130         {\typeout{*************************************^^J%
4131                   * Local config file #1.cfg used^^J%
4132                   *}}%
4133       \@empty}}
4134 \fi
```

8.7 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 catched).

```
4135 \bbl@trace{Language options}
4136 \let\bbl@afterlang\relax
4137 \let\BabelModifiers\relax
4138 \let\bbl@loaded\@empty
4139 \def\bbl@load@language#1{%
4140   \InputIfFileExists{#1.ldf}%
4141     {\edef\bbl@loaded{\CurrentOption
4142       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4143     \expandafter\let\expandafter\bbl@afterlang
4144       \csname\CurrentOption.ldf-h@@k\endcsname
4145     \expandafter\let\expandafter\BabelModifiers
4146       \csname bbl@mod@\CurrentOption\endcsname}%
4147     {\bbl@error{%
4148       Unknown option '\CurrentOption'. Either you misspelled it\\%
4149       or the language definition file \CurrentOption.ldf was not found}{%
4150       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4151       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4152       headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

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.

```
4153 \def\bbl@try@load@lang#1#2#3{%
4154   \IfFileExists{\CurrentOption.ldf}%
4155     {\bbl@load@language{\CurrentOption}}%
4156     {#1\bbl@load@language{#2}#3}}
4157 %
4158 \DeclareOption{hebrew}{%
4159   \input{rlbabel.def}%
4160   \bbl@load@language{hebrew}}
4161 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4162 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4163 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4164 \DeclareOption{polutonikogreek}{%
4165   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4166 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4167 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4168 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}
```

Another way to extend the list of 'known' options for babel was to create the file `bblopts.cfg` in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new `.ldf` file loading the actual one. You can also set the name of the file with the package option config=<name>, which will load <name>.cfg instead.

```
4169 \ifx\bbl@opt@config\@nnil
4170   \@ifpackagewith{babel}{noconfigs}{}%
4171     {\InputIfFileExists{bblopts.cfg}%
4172       {\typeout{*************************************^^J%
4173               * Local config file bblopts.cfg used^^J%
4174               *}}%
4175     {}}%
4176 \else
4177   \InputIfFileExists{\bbl@opt@config.cfg}%
4178     {\typeout{*************************************^^J%
4179             * Local config file \bbl@opt@config.cfg used^^J%
4180             *}}%
4181     {\bbl@error{%
4182       Local config file '\bbl@opt@config.cfg' not found}{%
4183       Perhaps you misspelled it.}}%
4184 \fi
```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be processed they must be defined explicitly. So, package options not yet taken into account and stored in bbl@language@opts are assumed to be languages. If not declared above, the names of the option and the file are the same. We first pre-process the class and package options to determine the main

language, which is processed in the third 'main' pass, *except* if all files are ldf *and* there is no main key. In the latter case (\bbl@opt@main is still \@nnil), the traditional way to set the main language is kept — the last loaded is the main language.

```
4185 \ifx\bbl@opt@main\@nnil
4186   \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
4187     \let\bbl@tempb\@empty
4188     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4189     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4190     \bbl@foreach\bbl@tempb{%    \bbl@tempb is a reversed list
4191       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4192         \ifodd\bbl@iniflag % = *=
4193           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4194         \else % n +=
4195           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4196         \fi
4197       \fi}%
4198   \fi
4199 \else
4200   \bbl@info{Main language set with 'main='. Except if you have\\%
4201            problems, prefer the default mechanism for setting\\%
4202            the main language. Reported}
4203 \fi
```

A few languages are still defined explicitly. They are stored in case they are needed in the 'main' pass (the value can be \relax).

```
4204 \ifx\bbl@opt@main\@nnil\else
4205   \bbl@csarg\let{loadmain\expandafter}\csname ds@\bbl@opt@main\endcsname
4206   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4207 \fi
```

Now define the corresponding loaders. With package options, assume the language exists. With class options, check if the option is a language by checking if the correspondin file exists.

```
4208 \bbl@foreach\bbl@language@opts{%
4209   \def\bbl@tempa{#1}%
4210   \ifx\bbl@tempa\bbl@opt@main\else
4211     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4212       \bbl@ifunset{ds@#1}%
4213         {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4214         {}%
4215     \else                      % + * (other = ini)
4216       \DeclareOption{#1}{%
4217         \bbl@ldfinit
4218         \babelprovide[import]{#1}%
4219         \bbl@afterldf{}}%
4220     \fi
4221   \fi}
4222 \bbl@foreach\@classoptionslist{%
4223   \def\bbl@tempa{#1}%
4224   \ifx\bbl@tempa\bbl@opt@main\else
4225     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4226       \bbl@ifunset{ds@#1}%
4227         {\IfFileExists{#1.ldf}%
4228           {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4229           {}}%
4230         {}%
4231     \else                      % + * (other = ini)
4232       \IfFileExists{babel-#1.tex}%
4233         {\DeclareOption{#1}{%
4234           \bbl@ldfinit
4235           \babelprovide[import]{#1}%
4236           \bbl@afterldf{}}}%
4237         {}%
4238     \fi
```

148

```
4239    \fi}
```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (but remember class options are processes before):

```
4240 \def\AfterBabelLanguage#1{%
4241    \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4242 \DeclareOption*{}
4243 \ProcessOptions*
```

This finished the second pass. Now the third one begins, which loads the main language set with the key `main`. A warning is raised if the main language is not the same as the last named one, or if the value of the key `main` is not a language. With some options in `provide`, the package `luatexbase` is loaded (and immediately used), and therefore `\babelprovide` can't go inside a `\DeclareOption`; this explains why it's executed directly, with a dummy declaration. Then all languages have been loaded, so we deactivate `\AfterBabelLanguage`.

```
4244 \bbl@trace{Option 'main'}
4245 \ifx\bbl@opt@main\@nnil
4246    \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4247    \let\bbl@tempc\@empty
4248    \bbl@for\bbl@tempb\bbl@tempa{%
4249       \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4250       \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4251    \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4252    \expandafter\bbl@tempa\bbl@loaded,\@nnil
4253    \ifx\bbl@tempb\bbl@tempc\else
4254       \bbl@warning{%
4255          Last declared language option is '\bbl@tempc',\\%
4256          but the last processed one was '\bbl@tempb'.\\%
4257          The main language can't be set as both a global\\%
4258          and a package option. Use 'main=\bbl@tempc' as\\%
4259          option. Reported}
4260    \fi
4261 \else
4262    \ifodd\bbl@iniflag  % case 1,3 (main is ini)
4263       \bbl@ldfinit
4264       \let\CurrentOption\bbl@opt@main
4265       \bbl@exp{%  \bbl@opt@provide = empty if *
4266          \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4267       \bbl@afterldf{}
4268       \DeclareOption{\bbl@opt@main}{}
4269    \else % case 0,2 (main is ldf)
4270       \ifx\bbl@loadmain\relax
4271          \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4272       \else
4273          \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4274       \fi
4275       \ExecuteOptions{\bbl@opt@main}
4276       \@namedef{ds@\bbl@opt@main}{}%
4277    \fi
4278    \DeclareOption*{}
4279    \ProcessOptions*
4280 \fi
4281 \def\AfterBabelLanguage{%
4282    \bbl@error
4283       {Too late for \string\AfterBabelLanguage}%
4284       {Languages have been loaded, so I can do nothing}}
```

In order to catch the case where the user didn't specify a language we check whether `\bbl@main@language`, has become defined. If not, the `nil` language is loaded.

```
4285 \ifx\bbl@main@language\@undefined
4286    \bbl@info{%
4287       You haven't specified a language. I'll use 'nil'\\%
```

```
4288      as the main language. Reported}
4289      \bbl@load@language{nil}
4290 \fi
4291 ⟨/package⟩
```

9   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

```
4292 ⟨*kernel⟩
4293 \let\bbl@onlyswitch\@empty
4294 \input babel.def
4295 \let\bbl@onlyswitch\@undefined
4296 ⟨/kernel⟩
4297 ⟨*patterns⟩
```

10   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 `patterns` is used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

```
4298 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4299 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4300 \xdef\bbl@format{\jobname}
4301 \def\bbl@version{⟨⟨version⟩⟩}
4302 \def\bbl@date{⟨⟨date⟩⟩}
4303 \ifx\AtBeginDocument\@undefined
4304   \def\@empty{}
4305 \fi
4306 ⟨⟨Define core switching macros⟩⟩
```

\process@line   Each line in the file `language.dat` is processed by `\process@line` after it is read. The first thing this macro does is to check whether the line starts with =. When the first token of a line is an =, the macro `\process@synonym` is called; otherwise the macro `\process@language` will continue.

```
4307 \def\process@line#1#2 #3 #4 {%
4308   \ifx=#1%
4309     \process@synonym{#2}%
4310   \else
4311     \process@language{#1#2}{#3}{#4}%
4312   \fi
4313   \ignorespaces}
```

\process@synonym   This macro takes care of the lines which start with an =. It needs an empty token register to begin with. `\bbl@languages` is also set to empty.

```
4314 \toks@{}
4315 \def\bbl@languages{}
```

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

We also need to copy the hyphenmin parameters for the synonym.

```
4316 \def\process@synonym#1{%
4317   \ifnum\last@language=\m@ne
4318     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4319   \else
4320     \expandafter\chardef\csname l@#1\endcsname\last@language
4321     \wlog{\string\l@#1=\string\language\the\last@language}%
4322     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4323       \csname\languagename hyphenmins\endcsname
4324     \let\bbl@elt\relax
4325     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4326   \fi}
```

\process@language The macro \process@language is used to process a non-empty line from the 'configuration file'. It has three arguments, each delimited by white space. The first argument is the 'name' of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call \addlanguage to allocate a pattern register and to make that register 'active'. Then the pattern file is read.

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

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

Some pattern files contain changes to the \lccode en \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⟩}{⟨number⟩} {⟨patterns-file⟩}{⟨exceptions-file⟩}. Note the last 2 arguments are empty in 'dialects' defined in language.dat with =. Note also the language name can have encoding info.

Finally, if the counter \language is equal to zero we execute the synonyms stored.

```
4327 \def\process@language#1#2#3{%
4328   \expandafter\addlanguage\csname l@#1\endcsname
4329   \expandafter\language\csname l@#1\endcsname
4330   \edef\languagename{#1}%
4331   \bbl@hook@everylanguage{#1}%
4332   %  > luatex
4333   \bbl@get@enc#1::\@@@
4334   \begingroup
4335     \lefthyphenmin\m@ne
4336     \bbl@hook@loadpatterns{#2}%
4337     %  > luatex
4338     \ifnum\lefthyphenmin=\m@ne
4339     \else
4340       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4341         \the\lefthyphenmin\the\righthyphenmin}%
4342     \fi
4343   \endgroup
4344   \def\bbl@tempa{#3}%
4345   \ifx\bbl@tempa\@empty\else
4346     \bbl@hook@loadexceptions{#3}%
4347     %  > luatex
4348   \fi
4349   \let\bbl@elt\relax
```

```
4350  \edef\bbl@languages{%
4351    \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4352  \ifnum\the\language=\z@
4353    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4354      \set@hyphenmins\tw@\thr@@\relax
4355    \else
4356      \expandafter\expandafter\expandafter\set@hyphenmins
4357        \csname #1hyphenmins\endcsname
4358    \fi
4359    \the\toks@
4360    \toks@{}%
4361  \fi}
```

\bbl@get@enc   The macro \bbl@get@enc extracts the font encoding from the language name and stores it in
\bbl@hyph@enc \bbl@hyph@enc. It uses delimited arguments to achieve this.

```
4362  \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.

```
4363  \def\bbl@hook@everylanguage#1{}
4364  \def\bbl@hook@loadpatterns#1{\input #1\relax}
4365  \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4366  \def\bbl@hook@loadkernel#1{%
4367    \def\addlanguage{\csname newlanguage\endcsname}%
4368    \def\adddialect##1##2{%
4369      \global\chardef##1##2\relax
4370      \wlog{\string##1 = a dialect from \string\language##2}}%
4371    \def\iflanguage##1{%
4372      \expandafter\ifx\csname l@##1\endcsname\relax
4373        \@nolanerr{##1}%
4374      \else
4375        \ifnum\csname l@##1\endcsname=\language
4376          \expandafter\expandafter\expandafter\@firstoftwo
4377        \else
4378          \expandafter\expandafter\expandafter\@secondoftwo
4379        \fi
4380      \fi}%
4381    \def\providehyphenmins##1##2{%
4382      \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4383        \@namedef{##1hyphenmins}{##2}%
4384      \fi}%
4385    \def\set@hyphenmins##1##2{%
4386      \lefthyphenmin##1\relax
4387      \righthyphenmin##2\relax}%
4388    \def\selectlanguage{%
4389      \errhelp{Selecting a language requires a package supporting it}%
4390      \errmessage{Not loaded}}%
4391    \let\foreignlanguage\selectlanguage
4392    \let\otherlanguage\selectlanguage
4393    \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4394    \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4395    \def\setlocale{%
4396      \errhelp{Find an armchair, sit down and wait}%
4397      \errmessage{Not yet available}}%
4398    \let\uselocale\setlocale
4399    \let\locale\setlocale
4400    \let\selectlocale\setlocale
4401    \let\localename\setlocale
4402    \let\textlocale\setlocale
4403    \let\textlanguage\setlocale
4404    \let\languagetext\setlocale}
4405  \begingroup
```

```
4406    \def\AddBabelHook#1#2{%
4407      \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4408        \def\next{\toks1}%
4409      \else
4410        \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4411      \fi
4412      \next}
4413    \ifx\directlua\@undefined
4414      \ifx\XeTeXinputencoding\@undefined\else
4415        \input xebabel.def
4416      \fi
4417    \else
4418      \input luababel.def
4419    \fi
4420    \openin1 = babel-\bbl@format.cfg
4421    \ifeof1
4422    \else
4423      \input babel-\bbl@format.cfg\relax
4424    \fi
4425    \closein1
4426  \endgroup
4427  \bbl@hook@loadkernel{switch.def}
```

\readconfigfile The configuration file can now be opened for reading.

```
4428  \openin1 = language.dat
```

See if the file exists, if not, use the default hyphenation file hyphen.tex. The user will be informed about this.

```
4429  \def\languagename{english}%
4430  \ifeof1
4431    \message{I couldn't find the file language.dat,\space
4432             I will try the file hyphen.tex}
4433    \input hyphen.tex\relax
4434    \chardef\l@english\z@
4435  \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.

```
4436    \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.

```
4437    \loop
4438      \endlinechar\m@ne
4439      \read1 to \bbl@line
4440      \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.

```
4441      \if T\ifeof1F\fi T\relax
4442        \ifx\bbl@line\@empty\else
4443          \edef\bbl@line{\bbl@line\space\space\space}%
4444          \expandafter\process@line\bbl@line\relax
4445        \fi
4446    \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.

```
4447    \begingroup
4448      \def\bbl@elt#1#2#3#4{%
```

153

```
4449        \global\language=#2\relax
4450        \gdef\languagename{#1}%
4451        \def\bbl@elt##1##2##3##4{}}%
4452      \bbl@languages
4453    \endgroup
4454 \fi
4455 \closein1
```

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

```
4456 \if/\the\toks@/\else
4457    \errhelp{language.dat loads no language, only synonyms}
4458    \errmessage{Orphan language synonym}
4459 \fi
```

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

```
4460 \let\bbl@line\@undefined
4461 \let\process@line\@undefined
4462 \let\process@synonym\@undefined
4463 \let\process@language\@undefined
4464 \let\bbl@get@enc\@undefined
4465 \let\bbl@hyph@enc\@undefined
4466 \let\bbl@tempa\@undefined
4467 \let\bbl@hook@loadkernel\@undefined
4468 \let\bbl@hook@everylanguage\@undefined
4469 \let\bbl@hook@loadpatterns\@undefined
4470 \let\bbl@hook@loadexceptions\@undefined
4471 ⟨/patterns⟩
```

Here the code for iniTeX ends.

11   Font handling with fontspec

Add the bidi handler just before luaoftload, 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].

```
4472 ⟨⟨*More package options⟩⟩ ≡
4473 \chardef\bbl@bidimode\z@
4474 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4475 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4476 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4477 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4478 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4479 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4480 ⟨⟨/More package options⟩⟩
```

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 babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading message, which is replaced ba a more explanatory one.

```
4481 ⟨⟨*Font selection⟩⟩ ≡
4482 \bbl@trace{Font handling with fontspec}
4483 \ifx\ExplSyntaxOn\@undefined\else
4484   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4485     \in@{,#1,}{,no-script,language-not-exist,}%
4486     \ifin@\else\bbl@tempfs@nx{#1}{#2}\fi}
4487   \def\bbl@fs@warn@nxx#1#2#3{%
4488     \in@{,#1,}{,no-script,language-not-exist,}%
4489     \ifin@\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4490   \def\bbl@loadfontspec{%
4491     \let\bbl@loadfontspec\relax
```

```
4492        \ifx\fontspec\@undefined
4493          \usepackage{fontspec}%
4494        \fi}%
4495 \fi
4496 \@onlypreamble\babelfont
4497 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4498   \bbl@foreach{#1}{%
4499     \expandafter\ifx\csname date##1\endcsname\relax
4500       \IfFileExists{babel-##1.tex}%
4501         {\babelprovide{##1}}%
4502         {}%
4503     \fi}%
4504   \edef\bbl@tempa{#1}%
4505   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4506   \bbl@loadfontspec
4507   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4508   \bbl@bblfont}
4509 \newcommand\bbl@bblfont[2][]{%  1=features 2=fontname, @font=rm|sf|tt
4510   \bbl@ifunset{\bbl@tempb family}%
4511     {\bbl@providefam{\bbl@tempb}}%
4512     {}%
4513   % For the default font, just in case:
4514   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4515   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4516     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4517      \bbl@exp{%
4518        \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4519        \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4520                        \<bbl@tempb default>\<bbl@tempb family>}}%
4521     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4522        \bbl@csarg\def{\bbl@tempb dflt@##1}{<>{#1}{#2}}}}}%
```

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

```
4523 \def\bbl@providefam#1{%
4524   \bbl@exp{%
4525     \\\newcommand\<#1default>{}% Just define it
4526     \\\bbl@add@list\\\bbl@font@fams{#1}%
4527     \\\DeclareRobustCommand\<#1family>{%
4528       \\\not@math@alphabet\<#1family>\relax
4529       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4530       \\\fontfamily\<#1default>%
4531       \<ifx>\\\UseHooks\\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4532       \\\selectfont}%
4533     \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}}
```

The following macro is activated when the hook babel-fontspec is enabled. But before, we define a macro for a warning, which sets a flag to avoid duplicate them.

```
4534 \def\bbl@nostdfont#1{%
4535   \bbl@ifunset{bbl@WFF@\f@family}%
4536     {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4537      \bbl@infowarn{The current font is not a babel standard family:\\%
4538        #1%
4539        \fontname\font\\%
4540        There is nothing intrinsically wrong with this warning, and\\%
4541        you can ignore it altogether if you do not need these\\%
4542        families. But if they are used in the document, you should be\\%
4543        aware 'babel' will not set Script and Language for them, so\\%
4544        you may consider defining a new family with \string\babelfont.\\%
4545        See the manual for further details about \string\babelfont.\\%
4546        Reported}}%
4547     {}}%
4548 \gdef\bbl@switchfont{%
4549   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4550   \bbl@exp{%  eg Arabic -> arabic
```

```
4551        \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4552    \bbl@foreach\bbl@font@fams{%
4553      \bbl@ifunset{bbl@##1dflt@\languagename}%      (1) language?
4554        {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%      (2) from script?
4555          {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
4556            {}%                                      123=F - nothing!
4557            {\bbl@exp{%                               3=T - from generic
4558              \global\let\<bbl@##1dflt@\languagename>%
4559                          \<bbl@##1dflt@>}}}%
4560          {\bbl@exp{%                                 2=T - from script
4561            \global\let\<bbl@##1dflt@\languagename>%
4562                        \<bbl@##1dflt@*\bbl@tempa>}}}%
4563        {}}%                                          1=T - language, already defined
4564    \def\bbl@tempa{\bbl@nostdfont{}}%
4565    \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4566      \bbl@ifunset{bbl@##1dflt@\languagename}%
4567        {\bbl@cs{famrst@##1}%
4568         \global\bbl@csarg\let{famrst@##1}\relax}%
4569        {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4570          \\\bbl@add\\\originalTeX{%
4571            \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4572                        \<##1default>\<##1family>{##1}}%
4573          \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4574                        \<##1default>\<##1family>}}}%
4575    \bbl@ifrestoring{}{\bbl@tempa}}%
```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```
4576  \ifx\f@family\@undefined\else   % if latex
4577    \ifcase\bbl@engine             % if pdftex
4578      \let\bbl@ckeckstdfonts\relax
4579    \else
4580      \def\bbl@ckeckstdfonts{%
4581        \begingroup
4582          \global\let\bbl@ckeckstdfonts\relax
4583          \let\bbl@tempa\@empty
4584          \bbl@foreach\bbl@font@fams{%
4585            \bbl@ifunset{bbl@##1dflt@}%
4586              {\@nameuse{##1family}%
4587               \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4588               \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4589                 \space\space\fontname\font\\\\}}%
4590               \bbl@csarg\xdef{##1dflt@}{\f@family}%
4591               \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4592              {}}%
4593          \ifx\bbl@tempa\@empty\else
4594            \bbl@infowarn{The following font families will use the default\\%
4595              settings for all or some languages:\\%
4596              \bbl@tempa
4597              There is nothing intrinsically wrong with it, but\\%
4598              'babel' will no set Script and Language, which could\\%
4599              be relevant in some languages. If your document uses\\%
4600              these families, consider redefining them with \string\babelfont.\\%
4601              Reported}%
4602          \fi
4603        \endgroup}
4604    \fi
4605  \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.

```
4606 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4607   \bbl@xin@{<>}{#1}%
4608   \ifin@
4609     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4610   \fi
4611   \bbl@exp{%                'Unprotected' macros return prev values
4612     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4613     \\\bbl@ifsamestring{#2}{\f@family}%
4614     {\\#3%
4615       \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4616      \let\\\bbl@tempa\relax}%
4617     {}}}
4618 %     TODO - next should be global?, but even local does its job. I'm
4619 %     still not sure -- must investigate:
4620 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4621   \let\bbl@tempe\bbl@mapselect
4622   \let\bbl@mapselect\relax
4623   \let\bbl@temp@fam#4%        eg, '\rmfamily', to be restored below
4624   \let#4\@empty     %        Make sure \renewfontfamily is valid
4625   \bbl@exp{%
4626     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4627     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4628       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4629     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4630       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4631     \let\\\bbl@tempfs@nx\<__fontspec_warning:nx>%
4632     \let\<__fontspec_warning:nx>\\\bbl@fs@warn@nx
4633     \let\\\bbl@tempfs@nxx\<__fontspec_warning:nxx>%
4634     \let\<__fontspec_warning:nxx>\\\bbl@fs@warn@nxx
4635     \\\renewfontfamily\\#4%
4636       [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4637   \bbl@exp{%
4638     \let\<__fontspec_warning:nx>\\\bbl@tempfs@nx
4639     \let\<__fontspec_warning:nxx>\\\bbl@tempfs@nxx}%
4640   \begingroup
4641     #4%
4642     \xdef#1{\f@family}%    eg, \bbl@rmdflt@lang{FreeSerif(0)}
4643   \endgroup
4644   \let#4\bbl@temp@fam
4645   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4646   \let\bbl@mapselect\bbl@tempe}%
```

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

```
4647 \def\bbl@font@rst#1#2#3#4{%
4648   \bbl@csarg\def{famrst@#4}{\bbl@font@set{#1}#2#3}}
```

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

```
4649 \def\bbl@font@fams{rm,sf,tt}
4650 ⟨⟨/Font selection⟩⟩
```

12 Hooks for XeTeX and LuaTeX

12.1 XeTeX

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

```
4651 ⟨⟨*Footnote changes⟩⟩ ≡
4652 \bbl@trace{Bidi footnotes}
4653 \ifnum\bbl@bidimode>\z@
4654   \def\bbl@footnote#1#2#3{%
4655     \@ifnextchar[%
```

```
4656        {\bbl@footnote@o{#1}{#2}{#3}}%
4657        {\bbl@footnote@x{#1}{#2}{#3}}}
4658    \long\def\bbl@footnote@x#1#2#3#4{%
4659      \bgroup
4660        \select@language@x{\bbl@main@language}%
4661        \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4662      \egroup}
4663    \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4664      \bgroup
4665        \select@language@x{\bbl@main@language}%
4666        \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4667      \egroup}
4668    \def\bbl@footnotetext#1#2#3{%
4669      \@ifnextchar[%
4670        {\bbl@footnotetext@o{#1}{#2}{#3}}%
4671        {\bbl@footnotetext@x{#1}{#2}{#3}}}
4672    \long\def\bbl@footnotetext@x#1#2#3#4{%
4673      \bgroup
4674        \select@language@x{\bbl@main@language}%
4675        \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4676      \egroup}
4677    \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4678      \bgroup
4679        \select@language@x{\bbl@main@language}%
4680        \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4681      \egroup}
4682    \def\BabelFootnote#1#2#3#4{%
4683      \ifx\bbl@fn@footnote\@undefined
4684        \let\bbl@fn@footnote\footnote
4685      \fi
4686      \ifx\bbl@fn@footnotetext\@undefined
4687        \let\bbl@fn@footnotetext\footnotetext
4688      \fi
4689      \bbl@ifblank{#2}%
4690        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}%
4691         \@namedef{\bbl@stripslash#1text}%
4692           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4693        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4694         \@namedef{\bbl@stripslash#1text}%
4695           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4696 \fi
4697 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4698 ⟨*xetex⟩
4699 \def\BabelStringsDefault{unicode}
4700 \let\xebbl@stop\relax
4701 \AddBabelHook{xetex}{encodedcommands}{%
4702   \def\bbl@tempa{#1}%
4703   \ifx\bbl@tempa\@empty
4704     \XeTeXinputencoding"bytes"%
4705   \else
4706     \XeTeXinputencoding"#1"%
4707   \fi
4708   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4709 \AddBabelHook{xetex}{stopcommands}{%
4710   \xebbl@stop
4711   \let\xebbl@stop\relax}
4712 \def\bbl@intraspace#1 #2 #3\@@{%
4713   \bbl@csarg\gdef{xeisp@\languagename}%
4714     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4715 \def\bbl@intrapenalty#1\@@{%
4716   \bbl@csarg\gdef{xeipn@\languagename}%
```

```
4717        {\XeTeXlinebreakpenalty #1\relax}}
4718 \def\bbl@provide@intraspace{%
4719   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4720   \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4721   \ifin@
4722     \bbl@ifunset{bbl@intsp@\languagename}{}%
4723       {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4724         \ifx\bbl@KVP@intraspace\@nnil
4725           \bbl@exp{%
4726             \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4727         \fi
4728         \ifx\bbl@KVP@intrapenalty\@nnil
4729           \bbl@intrapenalty0\@@
4730         \fi
4731       \fi
4732       \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4733         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4734       \fi
4735       \ifx\bbl@KVP@intrapenalty\@nnil\else
4736         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4737       \fi
4738       \bbl@exp{%
4739         % TODO. Execute only once (but redundant):
4740         \\\bbl@add\<extras\languagename>{%
4741           \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4742           \<bbl@xeisp@\languagename>%
4743           \<bbl@xeipn@\languagename>}%
4744         \\\bbl@toglobal\<extras\languagename>%
4745         \\\bbl@add\<noextras\languagename>{%
4746           \XeTeXlinebreaklocale "en"}%
4747         \\\bbl@toglobal\<noextras\languagename>}%
4748       \ifx\bbl@ispacesize\@undefined
4749         \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4750         \ifx\AtBeginDocument\@notprerr
4751           \expandafter\@secondoftwo  % to execute right now
4752         \fi
4753         \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4754       \fi}%
4755   \fi}
4756 \ifx\DisableBabelHook\@undefined\endinput\fi
4757 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4758 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4759 \DisableBabelHook{babel-fontspec}
4760 ⟨⟨Font selection⟩⟩
4761 \input txtbabel.def
4762 ⟨/xetex⟩
```

12.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the TeX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex–xet babel*, which is the bidi model in both pdftex and xetex.

```
4763 ⟨*texxet⟩
4764 \providecommand\bbl@provide@intraspace{}
4765 \bbl@trace{Redefinitions for bidi layout}
4766 \def\bbl@sspre@caption{%
4767   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4768 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4769 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
```

```
4770 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4771 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4772   \def\@hangfrom#1{%
4773     \setbox\@tempboxa\hbox{{#1}}%
4774     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4775     \noindent\box\@tempboxa}
4776   \def\raggedright{%
4777     \let\\\@centercr
4778     \bbl@startskip\z@skip
4779     \@rightskip\@flushglue
4780     \bbl@endskip\@rightskip
4781     \parindent\z@
4782     \parfillskip\bbl@startskip}
4783   \def\raggedleft{%
4784     \let\\\@centercr
4785     \bbl@startskip\@flushglue
4786     \bbl@endskip\z@skip
4787     \parindent\z@
4788     \parfillskip\bbl@endskip}
4789 \fi
4790 \IfBabelLayout{lists}
4791   {\bbl@sreplace\list
4792     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4793   \def\bbl@listleftmargin{%
4794     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4795   \ifcase\bbl@engine
4796     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4797     \def\p@enumiii{\p@enumii)\theenumii(}%
4798   \fi
4799   \bbl@sreplace\@verbatim
4800     {\leftskip\@totalleftmargin}%
4801     {\bbl@startskip\textwidth
4802      \advance\bbl@startskip-\linewidth}%
4803   \bbl@sreplace\@verbatim
4804     {\rightskip\z@skip}%
4805     {\bbl@endskip\z@skip}}%
4806   {}
4807 \IfBabelLayout{contents}
4808   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4809    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4810   {}
4811 \IfBabelLayout{columns}
4812   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4813   \def\bbl@outputhbox#1{%
4814     \hb@xt@\textwidth{%
4815       \hskip\columnwidth
4816       \hfil
4817       {\normalcolor\vrule \@width\columnseprule}%
4818       \hfil
4819       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4820       \hskip-\textwidth
4821       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4822       \hskip\columnsep
4823       \hskip\columnwidth}}}%
4824   {}
4825 ⟨⟨Footnote changes⟩⟩
4826 \IfBabelLayout{footnotes}%
4827   {\BabelFootnote\footnote\languagename{}{}%
4828    \BabelFootnote\localfootnote\languagename{}{}%
4829    \BabelFootnote\mainfootnote{}{}{}}
4830   {}
```

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.

```
4831 \IfBabelLayout{counters}%
4832   {\let\bbl@latinarabic=\@arabic
4833    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4834    \let\bbl@asciiroman=\@roman
4835    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4836    \let\bbl@asciiRoman=\@Roman
4837    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4838 ⟨/texxet⟩
```

12.3   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`).

```
4839 ⟨∗luatex⟩
4840 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4841 \bbl@trace{Read language.dat}
4842 \ifx\bbl@readstream\@undefined
4843   \csname newread\endcsname\bbl@readstream
4844 \fi
4845 \begingroup
4846   \toks@{}
4847   \count@\z@ % 0=start, 1=0th, 2=normal
4848   \def\bbl@process@line#1#2 #3 #4 {%
4849     \ifx=#1%
4850       \bbl@process@synonym{#2}%
4851     \else
4852       \bbl@process@language{#1#2}{#3}{#4}%
4853     \fi
4854     \ignorespaces}
4855   \def\bbl@manylang{%
4856     \ifnum\bbl@last>\@ne
```

```
4857        \bbl@info{Non-standard hyphenation setup}%
4858      \fi
4859      \let\bbl@manylang\relax}
4860  \def\bbl@process@language#1#2#3{%
4861      \ifcase\count@
4862        \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4863      \or
4864        \count@\tw@
4865      \fi
4866      \ifnum\count@=\tw@
4867        \expandafter\addlanguage\csname l@#1\endcsname
4868        \language\allocationnumber
4869        \chardef\bbl@last\allocationnumber
4870        \bbl@manylang
4871        \let\bbl@elt\relax
4872        \xdef\bbl@languages{%
4873          \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4874      \fi
4875      \the\toks@
4876      \toks@{}}
4877  \def\bbl@process@synonym@aux#1#2{%
4878      \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4879      \let\bbl@elt\relax
4880      \xdef\bbl@languages{%
4881        \bbl@languages\bbl@elt{#1}{#2}{}{}}%
4882  \def\bbl@process@synonym#1{%
4883      \ifcase\count@
4884        \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4885      \or
4886        \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4887      \else
4888        \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4889      \fi}
4890  \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4891      \chardef\l@english\z@
4892      \chardef\l@USenglish\z@
4893      \chardef\bbl@last\z@
4894      \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4895      \gdef\bbl@languages{%
4896        \bbl@elt{english}{0}{hyphen.tex}{}%
4897        \bbl@elt{USenglish}{0}{}{}}
4898  \else
4899      \global\let\bbl@languages@format\bbl@languages
4900      \def\bbl@elt#1#2#3#4{% Remove all except language 0
4901        \ifnum#2>\z@\else
4902          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4903        \fi}%
4904      \xdef\bbl@languages{\bbl@languages}%
4905  \fi
4906  \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4907  \bbl@languages
4908  \openin\bbl@readstream=language.dat
4909  \ifeof\bbl@readstream
4910      \bbl@warning{I couldn't find language.dat. No additional\\%
4911                    patterns loaded. Reported}%
4912  \else
4913      \loop
4914        \endlinechar\m@ne
4915        \read\bbl@readstream to \bbl@line
4916        \endlinechar`\^^M
4917        \if T\ifeof\bbl@readstream F\fi T\relax
4918          \ifx\bbl@line\@empty\else
4919            \edef\bbl@line{\bbl@line\space\space\space}%
```

162

```
4920          \expandafter\bbl@process@line\bbl@line\relax
4921        \fi
4922      \repeat
4923    \fi
4924 \endgroup
4925 \bbl@trace{Macros for reading patterns files}
4926 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4927 \ifx\babelcatcodetablenum\@undefined
4928    \ifx\newcatcodetable\@undefined
4929      \def\babelcatcodetablenum{5211}
4930      \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4931    \else
4932      \newcatcodetable\babelcatcodetablenum
4933      \newcatcodetable\bbl@pattcodes
4934    \fi
4935 \else
4936    \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4937 \fi
4938 \def\bbl@luapatterns#1#2{%
4939    \bbl@get@enc#1::\@@@
4940    \setbox\z@\hbox\bgroup
4941      \begingroup
4942        \savecatcodetable\babelcatcodetablenum\relax
4943        \initcatcodetable\bbl@pattcodes\relax
4944        \catcodetable\bbl@pattcodes\relax
4945          \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4946          \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4947          \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4948          \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4949          \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4950          \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4951          \input #1\relax
4952        \catcodetable\babelcatcodetablenum\relax
4953      \endgroup
4954      \def\bbl@tempa{#2}%
4955      \ifx\bbl@tempa\@empty\else
4956        \input #2\relax
4957      \fi
4958    \egroup}%
4959 \def\bbl@patterns@lua#1{%
4960    \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4961      \csname l@#1\endcsname
4962      \edef\bbl@tempa{#1}%
4963    \else
4964      \csname l@#1:\f@encoding\endcsname
4965      \edef\bbl@tempa{#1:\f@encoding}%
4966    \fi\relax
4967    \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4968    \@ifundefined{bbl@hyphendata@\the\language}%
4969      {\def\bbl@elt##1##2##3##4{%
4970        \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4971          \def\bbl@tempb{##3}%
4972          \ifx\bbl@tempb\@empty\else % if not a synonymous
4973            \def\bbl@tempc{{##3}{##4}}%
4974          \fi
4975          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4976        \fi}%
4977      \bbl@languages
4978      \@ifundefined{bbl@hyphendata@\the\language}%
4979        {\bbl@info{No hyphenation patterns were set for\\%
4980                  language '\bbl@tempa'. Reported}}%
4981        {\expandafter\expandafter\expandafter\bbl@luapatterns
4982          \csname bbl@hyphendata@\the\language\endcsname}}{}}
```

```
4983 \endinput\fi
4984   % Here ends \ifx\AddBabelHook\@undefined
4985   % A few lines are only read by hyphen.cfg
4986 \ifx\DisableBabelHook\@undefined
4987   \AddBabelHook{luatex}{everylanguage}{%
4988     \def\process@language##1##2##3{%
4989       \def\process@line####1####2 ####3 ####4 {}}}
4990   \AddBabelHook{luatex}{loadpatterns}{%
4991     \input #1\relax
4992     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4993       {{#1}{}}}
4994   \AddBabelHook{luatex}{loadexceptions}{%
4995     \input #1\relax
4996     \def\bbl@tempb##1##2{{##1}{#1}}%
4997     \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4998       {\expandafter\expandafter\expandafter\bbl@tempb
4999        \csname bbl@hyphendata@\the\language\endcsname}}
5000 \endinput\fi
5001   % Here stops reading code for hyphen.cfg
5002   % The following is read the 2nd time it's loaded
5003 \begingroup  % TODO - to a lua file
5004 \catcode`\%=12
5005 \catcode`\'=12
5006 \catcode`\"=12
5007 \catcode`\:=12
5008 \directlua{
5009   Babel = Babel or {}
5010   function Babel.bytes(line)
5011     return line:gsub("(.)",
5012       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5013   end
5014   function Babel.begin_process_input()
5015     if luatexbase and luatexbase.add_to_callback then
5016       luatexbase.add_to_callback('process_input_buffer',
5017                                  Babel.bytes,'Babel.bytes')
5018     else
5019       Babel.callback = callback.find('process_input_buffer')
5020       callback.register('process_input_buffer',Babel.bytes)
5021     end
5022   end
5023   function Babel.end_process_input ()
5024     if luatexbase and luatexbase.remove_from_callback then
5025       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5026     else
5027       callback.register('process_input_buffer',Babel.callback)
5028     end
5029   end
5030   function Babel.addpatterns(pp, lg)
5031     local lg = lang.new(lg)
5032     local pats = lang.patterns(lg) or ''
5033     lang.clear_patterns(lg)
5034     for p in pp:gmatch('[^%s]+') do
5035       ss = ''
5036       for i in string.utfcharacters(p:gsub('%d', '')) do
5037         ss = ss .. '%d?' .. i
5038       end
5039       ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5040       ss = ss:gsub('%.%%d%?$', '%%.')
5041       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5042       if n == 0 then
5043         tex.sprint(
5044           [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5045           .. p .. [[}]])
```

```
5046        pats = pats .. ' ' .. p
5047      else
5048        tex.sprint(
5049          [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5050          .. p .. [[}]])
5051      end
5052    end
5053    lang.patterns(lg, pats)
5054  end
5055  Babel.characters = Babel.characters or {}
5056  Babel.ranges = Babel.ranges or {}
5057  function Babel.hlist_has_bidi(head)
5058    local has_bidi = false
5059    local ranges = Babel.ranges
5060    for item in node.traverse(head) do
5061      if item.id == node.id'glyph' then
5062        local itemchar = item.char
5063        local chardata = Babel.characters[itemchar]
5064        local dir = chardata and chardata.d or nil
5065        if not dir then
5066          for nn, et in ipairs(ranges) do
5067            if itemchar < et[1] then
5068              break
5069            elseif itemchar <= et[2] then
5070              dir = et[3]
5071              break
5072            end
5073          end
5074        end
5075        if dir and (dir == 'al' or dir == 'r') then
5076          has_bidi = true
5077        end
5078      end
5079    end
5080    return has_bidi
5081  end
5082  function Babel.set_chranges_b (script, chrng)
5083    if chrng == '' then return end
5084    texio.write('Replacing ' .. script .. ' script ranges')
5085    Babel.script_blocks[script] = {}
5086    for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5087      table.insert(
5088        Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5089    end
5090  end
5091 }
5092 \endgroup
5093 \ifx\newattribute\@undefined\else
5094   \newattribute\bbl@attr@locale
5095   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5096   \AddBabelHook{luatex}{beforeextras}{%
5097     \setattribute\bbl@attr@locale\localeid}
5098 \fi
5099 \def\BabelStringsDefault{unicode}
5100 \let\luabbl@stop\relax
5101 \AddBabelHook{luatex}{encodedcommands}{%
5102   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5103   \ifx\bbl@tempa\bbl@tempb\else
5104     \directlua{Babel.begin_process_input()}%
5105     \def\luabbl@stop{%
5106       \directlua{Babel.end_process_input()}}%
5107   \fi}%
5108 \AddBabelHook{luatex}{stopcommands}{%
```

165

```
5109    \luabbl@stop
5110    \let\luabbl@stop\relax}
5111  \AddBabelHook{luatex}{patterns}{%
5112    \@ifundefined{bbl@hyphendata@\the\language}%
5113      {\def\bbl@elt##1##2##3##4{%
5114        \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5115          \def\bbl@tempb{##3}%
5116          \ifx\bbl@tempb\@empty\else % if not a synonymous
5117            \def\bbl@tempc{{##3}{##4}}%
5118          \fi
5119          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5120        \fi}%
5121      \bbl@languages
5122      \@ifundefined{bbl@hyphendata@\the\language}%
5123        {\bbl@info{No hyphenation patterns were set for\\%
5124                  language '#2'. Reported}}%
5125        {\expandafter\expandafter\expandafter\bbl@luapatterns
5126          \csname bbl@hyphendata@\the\language\endcsname}}{}%
5127    \@ifundefined{bbl@patterns@}{}{%
5128      \begingroup
5129        \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5130        \ifin@\else
5131          \ifx\bbl@patterns@\@empty\else
5132            \directlua{ Babel.addpatterns(
5133              [[\bbl@patterns@]], \number\language) }%
5134          \fi
5135          \@ifundefined{bbl@patterns@#1}%
5136            \@empty
5137            {\directlua{ Babel.addpatterns(
5138                [[\space\csname bbl@patterns@#1\endcsname]],
5139                \number\language) }}%
5140          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5141        \fi
5142      \endgroup}%
5143    \bbl@exp{%
5144      \bbl@ifunset{bbl@prehc@\languagename}{}%
5145        {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5146          {\prehyphenchar=\bbl@cl{prehc}\relax}}}}
```

\babelpatterns   This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the global ones and \bbl@patterns@<lang> for language ones. We make sure there is a space between words when multiple commands are used.

```
5147  \@onlypreamble\babelpatterns
5148  \AtEndOfPackage{%
5149    \newcommand\babelpatterns[2][\@empty]{%
5150      \ifx\bbl@patterns@\relax
5151        \let\bbl@patterns@\@empty
5152      \fi
5153      \ifx\bbl@pttnlist\@empty\else
5154        \bbl@warning{%
5155          You must not intermingle \string\selectlanguage\space and\\%
5156          \string\babelpatterns\space or some patterns will not\\%
5157          be taken into account. Reported}%
5158      \fi
5159      \ifx\@empty#1%
5160        \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5161      \else
5162        \edef\bbl@tempb{\zap@space#1 \@empty}%
5163        \bbl@for\bbl@tempa\bbl@tempb{%
5164          \bbl@fixname\bbl@tempa
5165          \bbl@iflanguage\bbl@tempa{%
5166            \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5167              \@ifundefined{bbl@patterns@\bbl@tempa}%
```

166

```
5168            \@empty
5169              {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5170          #2}}}%
5171    \fi}}
```

12.4   Southeast Asian scripts

First, some general code for line breaking, used by \babelposthyphenation.
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.

```
5172 % TODO - to a lua file
5173 \directlua{
5174   Babel = Babel or {}
5175   Babel.linebreaking = Babel.linebreaking or {}
5176   Babel.linebreaking.before = {}
5177   Babel.linebreaking.after = {}
5178   Babel.locale = {} % Free to use, indexed by \localeid
5179   function Babel.linebreaking.add_before(func)
5180     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5181     table.insert(Babel.linebreaking.before, func)
5182   end
5183   function Babel.linebreaking.add_after(func)
5184     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5185     table.insert(Babel.linebreaking.after, func)
5186   end
5187 }
5188 \def\bbl@intraspace#1 #2 #3\@@{%
5189   \directlua{
5190     Babel = Babel or {}
5191     Babel.intraspaces = Babel.intraspaces or {}
5192     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5193       {b = #1, p = #2, m = #3}
5194     Babel.locale_props[\the\localeid].intraspace = %
5195       {b = #1, p = #2, m = #3}
5196 }}
5197 \def\bbl@intrapenalty#1\@@{%
5198   \directlua{
5199     Babel = Babel or {}
5200     Babel.intrapenalties = Babel.intrapenalties or {}
5201     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5202     Babel.locale_props[\the\localeid].intrapenalty = #1
5203 }}
5204 \begingroup
5205 \catcode`\%=12
5206 \catcode`\^=14
5207 \catcode`\'=12
5208 \catcode`\~=12
5209 \gdef\bbl@seaintraspace{^
5210   \let\bbl@seaintraspace\relax
5211   \directlua{
5212     Babel = Babel or {}
5213     Babel.sea_enabled = true
5214     Babel.sea_ranges = Babel.sea_ranges or {}
5215     function Babel.set_chranges (script, chrng)
5216       local c = 0
5217       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5218         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5219         c = c + 1
5220       end
5221     end
5222     function Babel.sea_disc_to_space (head)
5223       local sea_ranges = Babel.sea_ranges
```

```
5224        local last_char = nil
5225        local quad = 655360       ^% 10 pt = 655360 = 10 * 65536
5226        for item in node.traverse(head) do
5227          local i = item.id
5228          if i == node.id'glyph' then
5229            last_char = item
5230          elseif i == 7 and item.subtype == 3 and last_char
5231              and last_char.char > 0x0C99 then
5232            quad = font.getfont(last_char.font).size
5233            for lg, rg in pairs(sea_ranges) do
5234              if last_char.char > rg[1] and last_char.char < rg[2] then
5235                lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5236                local intraspace = Babel.intraspaces[lg]
5237                local intrapenalty = Babel.intrapenalties[lg]
5238                local n
5239                if intrapenalty ~= 0 then
5240                  n = node.new(14, 0)     ^% penalty
5241                  n.penalty = intrapenalty
5242                  node.insert_before(head, item, n)
5243                end
5244                n = node.new(12, 13)      ^% (glue, spaceskip)
5245                node.setglue(n, intraspace.b * quad,
5246                                intraspace.p * quad,
5247                                intraspace.m * quad)
5248                node.insert_before(head, item, n)
5249                node.remove(head, item)
5250              end
5251            end
5252          end
5253        end
5254      end
5255    }^^
5256    \bbl@luahyphenate}
```

12.5   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.

```
5257 \catcode`\%=14
5258 \gdef\bbl@cjkintraspace{%
5259   \let\bbl@cjkintraspace\relax
5260   \directlua{
5261     Babel = Babel or {}
5262     require('babel-data-cjk.lua')
5263     Babel.cjk_enabled = true
5264     function Babel.cjk_linebreak(head)
5265       local GLYPH = node.id'glyph'
5266       local last_char = nil
5267       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5268       local last_class = nil
5269       local last_lang = nil
5270
5271       for item in node.traverse(head) do
5272         if item.id == GLYPH then
5273
5274           local lang = item.lang
5275
5276           local LOCALE = node.get_attribute(item,
5277                   Babel.attr_locale)
```

```
5278            local props = Babel.locale_props[LOCALE]
5279
5280            local class = Babel.cjk_class[item.char].c
5281
5282            if props.cjk_quotes and props.cjk_quotes[item.char] then
5283              class = props.cjk_quotes[item.char]
5284            end
5285
5286            if class == 'cp' then class = 'cl' end % )] as CL
5287            if class == 'id' then class = 'I' end
5288
5289            local br = 0
5290            if class and last_class and Babel.cjk_breaks[last_class][class] then
5291              br = Babel.cjk_breaks[last_class][class]
5292            end
5293
5294            if br == 1 and props.linebreak == 'c' and
5295                lang ~= \the\l@nohyphenation\space and
5296                last_lang ~= \the\l@nohyphenation then
5297              local intrapenalty = props.intrapenalty
5298              if intrapenalty ~= 0 then
5299                local n = node.new(14, 0)      % penalty
5300                n.penalty = intrapenalty
5301                node.insert_before(head, item, n)
5302              end
5303              local intraspace = props.intraspace
5304              local n = node.new(12, 13)      % (glue, spaceskip)
5305              node.setglue(n, intraspace.b * quad,
5306                              intraspace.p * quad,
5307                              intraspace.m * quad)
5308              node.insert_before(head, item, n)
5309            end
5310
5311            if font.getfont(item.font) then
5312              quad = font.getfont(item.font).size
5313            end
5314            last_class = class
5315            last_lang = lang
5316          else % if penalty, glue or anything else
5317            last_class = nil
5318          end
5319        end
5320      lang.hyphenate(head)
5321    end
5322  }%
5323  \bbl@luahyphenate}
5324 \gdef\bbl@luahyphenate{%
5325   \let\bbl@luahyphenate\relax
5326   \directlua{
5327     luatexbase.add_to_callback('hyphenate',
5328     function (head, tail)
5329       if Babel.linebreaking.before then
5330         for k, func in ipairs(Babel.linebreaking.before)  do
5331           func(head)
5332         end
5333       end
5334       if Babel.cjk_enabled then
5335         Babel.cjk_linebreak(head)
5336       end
5337       lang.hyphenate(head)
5338       if Babel.linebreaking.after then
5339         for k, func in ipairs(Babel.linebreaking.after)  do
5340           func(head)
```

```
5341                end
5342              end
5343            if Babel.sea_enabled then
5344              Babel.sea_disc_to_space(head)
5345            end
5346          end,
5347        'Babel.hyphenate')
5348    }
5349  }
5350  \endgroup
5351  \def\bbl@provide@intraspace{%
5352    \bbl@ifunset{bbl@intsp@\languagename}{}%
5353      {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5354        \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5355        \ifin@           % cjk
5356          \bbl@cjkintraspace
5357          \directlua{
5358              Babel = Babel or {}
5359              Babel.locale_props = Babel.locale_props or {}
5360              Babel.locale_props[\the\localeid].linebreak = 'c'
5361          }%
5362          \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5363          \ifx\bbl@KVP@intrapenalty\@nnil
5364            \bbl@intrapenalty0\@@
5365          \fi
5366        \else            % sea
5367          \bbl@seaintraspace
5368          \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5369          \directlua{
5370            Babel = Babel or {}
5371            Babel.sea_ranges = Babel.sea_ranges or {}
5372            Babel.set_chranges('\bbl@cl{sbcp}',
5373                                '\bbl@cl{chrng}')
5374          }%
5375          \ifx\bbl@KVP@intrapenalty\@nnil
5376            \bbl@intrapenalty0\@@
5377          \fi
5378        \fi
5379      \fi
5380      \ifx\bbl@KVP@intrapenalty\@nnil\else
5381        \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5382      \fi}}
```

12.6 Arabic justification

```
5383  \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5384  \def\bblar@chars{%
5385    0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5386    0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5387    0640,0641,0642,0643,0644,0645,0646,0647,0649}
5388  \def\bblar@elongated{%
5389    0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5390    063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5391    0649,064A}
5392  \begingroup
5393    \catcode`\_=11 \catcode`\:=11
5394    \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5395  \endgroup
5396  \gdef\bbl@arabicjust{%
5397    \let\bbl@arabicjust\relax
5398    \newattribute\bblar@kashida
5399    \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5400    \bblar@kashida=\z@
```

```
5401  \bbl@patchfont{{\bbl@parsejalt}}%
5402  \directlua{
5403    Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5404    Babel.arabic.elong_map[\the\localeid]   = {}
5405    luatexbase.add_to_callback('post_linebreak_filter',
5406      Babel.arabic.justify, 'Babel.arabic.justify')
5407    luatexbase.add_to_callback('hpack_filter',
5408      Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5409  }}%
5410 % Save both node lists to make replacement. TODO. Save also widths to
5411 % make computations
5412 \def\bblar@fetchjalt#1#2#3#4{%
5413  \bbl@exp{\\\bbl@foreach{#1}}{%
5414    \bbl@ifunset{bblar@JE@##1}%
5415      {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5416      {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5417    \directlua{%
5418      local last = nil
5419      for item in node.traverse(tex.box[0].head) do
5420        if item.id == node.id'glyph' and item.char > 0x600 and
5421           not (item.char == 0x200D) then
5422          last = item
5423        end
5424      end
5425      Babel.arabic.#3['##1#4'] = last.char
5426    }}}
5427 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5428 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5429 % positioning?
5430 \gdef\bbl@parsejalt{%
5431  \ifx\addfontfeature\@undefined\else
5432    \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5433    \ifin@
5434      \directlua{%
5435        if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5436          Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5437          tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5438        end
5439      }%
5440    \fi
5441  \fi}
5442 \gdef\bbl@parsejalti{%
5443  \begingroup
5444    \let\bbl@parsejalt\relax     % To avoid infinite loop
5445    \edef\bbl@tempb{\fontid\font}%
5446    \bblar@nofswarn
5447    \bblar@fetchjalt\bblar@elongated{}{from}{}%
5448    \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5449    \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5450    \addfontfeature{RawFeature=+jalt}%
5451  % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5452    \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5453    \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5454    \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5455      \directlua{%
5456        for k, v in pairs(Babel.arabic.from) do
5457          if Babel.arabic.dest[k] and
5458             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5459            Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5460              [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5461          end
5462        end
5463      }%
```

```
5464    \endgroup}
5465 %
5466 \begingroup
5467 \catcode`#=11
5468 \catcode`~=11
5469 \directlua{
5470
5471 Babel.arabic = Babel.arabic or {}
5472 Babel.arabic.from = {}
5473 Babel.arabic.dest = {}
5474 Babel.arabic.justify_factor = 0.95
5475 Babel.arabic.justify_enabled = true
5476
5477 function Babel.arabic.justify(head)
5478   if not Babel.arabic.justify_enabled then return head end
5479   for line in node.traverse_id(node.id'hlist', head) do
5480     Babel.arabic.justify_hlist(head, line)
5481   end
5482   return head
5483 end
5484
5485 function Babel.arabic.justify_hbox(head, gc, size, pack)
5486   local has_inf = false
5487   if Babel.arabic.justify_enabled and pack == 'exactly' then
5488     for n in node.traverse_id(12, head) do
5489       if n.stretch_order > 0 then has_inf = true end
5490     end
5491     if not has_inf then
5492       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5493     end
5494   end
5495   return head
5496 end
5497
5498 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5499   local d, new
5500   local k_list, k_item, pos_inline
5501   local width, width_new, full, k_curr, wt_pos, goal, shift
5502   local subst_done = false
5503   local elong_map = Babel.arabic.elong_map
5504   local last_line
5505   local GLYPH = node.id'glyph'
5506   local KASHIDA = Babel.attr_kashida
5507   local LOCALE = Babel.attr_locale
5508
5509   if line == nil then
5510     line = {}
5511     line.glue_sign = 1
5512     line.glue_order = 0
5513     line.head = head
5514     line.shift = 0
5515     line.width = size
5516   end
5517
5518   % Exclude last line. todo. But-- it discards one-word lines, too!
5519   % ? Look for glue = 12:15
5520   if (line.glue_sign == 1 and line.glue_order == 0) then
5521     elongs = {}     % Stores elongated candidates of each line
5522     k_list = {}     % And all letters with kashida
5523     pos_inline = 0  % Not yet used
5524
5525     for n in node.traverse_id(GLYPH, line.head) do
5526       pos_inline = pos_inline + 1 % To find where it is. Not used.
```

```
5527
5528      % Elongated glyphs
5529      if elong_map then
5530        local locale = node.get_attribute(n, LOCALE)
5531        if elong_map[locale] and elong_map[locale][n.font] and
5532            elong_map[locale][n.font][n.char] then
5533          table.insert(elongs, {node = n, locale = locale} )
5534          node.set_attribute(n.prev, KASHIDA, 0)
5535        end
5536      end
5537
5538      % Tatwil
5539      if Babel.kashida_wts then
5540        local k_wt = node.get_attribute(n, KASHIDA)
5541        if k_wt > 0 then % todo. parameter for multi inserts
5542          table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5543        end
5544      end
5545
5546    end % of node.traverse_id
5547
5548    if #elongs == 0 and #k_list == 0 then goto next_line end
5549    full  = line.width
5550    shift = line.shift
5551    goal  = full * Babel.arabic.justify_factor % A bit crude
5552    width = node.dimensions(line.head)    % The 'natural' width
5553
5554    % == Elongated ==
5555    % Original idea taken from 'chikenize'
5556    while (#elongs > 0 and width < goal) do
5557      subst_done = true
5558      local x = #elongs
5559      local curr = elongs[x].node
5560      local oldchar = curr.char
5561      curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5562      width = node.dimensions(line.head)  % Check if the line is too wide
5563      % Substitute back if the line would be too wide and break:
5564      if width > goal then
5565        curr.char = oldchar
5566        break
5567      end
5568      % If continue, pop the just substituted node from the list:
5569      table.remove(elongs, x)
5570    end
5571
5572    % == Tatwil ==
5573    if #k_list == 0 then goto next_line end
5574
5575    width = node.dimensions(line.head)     % The 'natural' width
5576    k_curr = #k_list
5577    wt_pos = 1
5578
5579    while width < goal do
5580      subst_done = true
5581      k_item = k_list[k_curr].node
5582      if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5583        d = node.copy(k_item)
5584        d.char = 0x0640
5585        line.head, new = node.insert_after(line.head, k_item, d)
5586        width_new = node.dimensions(line.head)
5587        if width > goal or width == width_new then
5588          node.remove(line.head, new) % Better compute before
5589          break
```

173

```
5590            end
5591            width = width_new
5592          end
5593          if k_curr == 1 then
5594            k_curr = #k_list
5595            wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5596          else
5597            k_curr = k_curr - 1
5598          end
5599        end
5600
5601        ::next_line::
5602
5603        % Must take into account marks and ins, see luatex manual.
5604        % Have to be executed only if there are changes. Investigate
5605        % what's going on exactly.
5606        if subst_done and not gc then
5607          d = node.hpack(line.head, full, 'exactly')
5608          d.shift = shift
5609          node.insert_before(head, line, d)
5610          node.remove(head, line)
5611        end
5612    end % if process line
5613 end
5614 }
5615 \endgroup
5616 \fi\fi % Arabic just block
```

12.7 Common stuff

```
5617 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5618 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5619 \DisableBabelHook{babel-fontspec}
5620 ⟨⟨Font selection⟩⟩
```

12.8 Automatic fonts and ids switching

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

```
5621 % TODO - to a lua file
5622 \directlua{
5623 Babel.script_blocks = {
5624   ['dflt'] = {},
5625   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5626               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5627   ['Armn'] = {{0x0530, 0x058F}},
5628   ['Beng'] = {{0x0980, 0x09FF}},
5629   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5630   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5631   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5632               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5633   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5634   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5635               {0xAB00, 0xAB2F}},
5636   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5637 % Don't follow strictly Unicode, which places some Coptic letters in
5638 % the 'Greek and Coptic' block
5639   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5640   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
```

```
5641                {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5642                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5643                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5644                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5645                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5646   ['Hebr'] = {{0x0590, 0x05FF}},
5647   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5648                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5649   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5650   ['Knda'] = {{0x0C80, 0x0CFF}},
5651   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5652                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5653                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5654   ['Laoo'] = {{0x0E80, 0x0EFF}},
5655   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5656                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5657                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5658   ['Mahj'] = {{0x11150, 0x1117F}},
5659   ['Mlym'] = {{0x0D00, 0x0D7F}},
5660   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5661   ['Orya'] = {{0x0B00, 0x0B7F}},
5662   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5663   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5664   ['Taml'] = {{0x0B80, 0x0BFF}},
5665   ['Telu'] = {{0x0C00, 0x0C7F}},
5666   ['Tfng'] = {{0x2D30, 0x2D7F}},
5667   ['Thai'] = {{0x0E00, 0x0E7F}},
5668   ['Tibt'] = {{0x0F00, 0x0FFF}},
5669   ['Vaii'] = {{0xA500, 0xA63F}},
5670   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5671 }
5672
5673 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5674 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5675 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5676
5677 function Babel.locale_map(head)
5678   if not Babel.locale_mapped then return head end
5679
5680   local LOCALE = Babel.attr_locale
5681   local GLYPH = node.id('glyph')
5682   local inmath = false
5683   local toloc_save
5684   for item in node.traverse(head) do
5685     local toloc
5686     if not inmath and item.id == GLYPH then
5687       % Optimization: build a table with the chars found
5688       if Babel.chr_to_loc[item.char] then
5689         toloc = Babel.chr_to_loc[item.char]
5690       else
5691         for lc, maps in pairs(Babel.loc_to_scr) do
5692           for _, rg in pairs(maps) do
5693             if item.char >= rg[1] and item.char <= rg[2] then
5694               Babel.chr_to_loc[item.char] = lc
5695               toloc = lc
5696               break
5697             end
5698           end
5699         end
5700       end
5701       % Now, take action, but treat composite chars in a different
5702       % fashion, because they 'inherit' the previous locale. Not yet
5703       % optimized.
```

175

```
5704        if not toloc and
5705            (item.char >= 0x0300 and item.char <= 0x036F) or
5706            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5707            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5708          toloc = toloc_save
5709        end
5710        if toloc and Babel.locale_props[toloc] and
5711            Babel.locale_props[toloc].letters and
5712            tex.getcatcode(item.char) \string~= 11 then
5713          toloc = nil
5714        end
5715        if toloc and toloc > -1 then
5716          if Babel.locale_props[toloc].lg then
5717            item.lang = Babel.locale_props[toloc].lg
5718            node.set_attribute(item, LOCALE, toloc)
5719          end
5720          if Babel.locale_props[toloc]['/'..item.font] then
5721            item.font = Babel.locale_props[toloc]['/'..item.font]
5722          end
5723          toloc_save = toloc
5724        end
5725      elseif not inmath and item.id == 7 then % Apply recursively
5726        item.replace = item.replace and Babel.locale_map(item.replace)
5727        item.pre     = item.pre and Babel.locale_map(item.pre)
5728        item.post    = item.post and Babel.locale_map(item.post)
5729      elseif item.id == node.id'math' then
5730        inmath = (item.subtype == 0)
5731      end
5732    end
5733    return head
5734 end
5735 }
```

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

```
5736 \newcommand\babelcharproperty[1]{%
5737   \count@=#1\relax
5738   \ifvmode
5739     \expandafter\bbl@chprop
5740   \else
5741     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5742                 vertical mode (preamble or between paragraphs)}%
5743                {See the manual for futher info}%
5744   \fi}
5745 \newcommand\bbl@chprop[3][\the\count@]{%
5746   \@tempcnta=#1\relax
5747   \bbl@ifunset{bbl@chprop@#2}%
5748     {\bbl@error{No property named '#2'. Allowed values are\\%
5749                 direction (bc), mirror (bmg), and linebreak (lb)}%
5750                {See the manual for futher info}}%
5751     {}%
5752   \loop
5753     \bbl@cs{chprop@#2}{#3}%
5754   \ifnum\count@<\@tempcnta
5755     \advance\count@\@ne
5756   \repeat}
5757 \def\bbl@chprop@direction#1{%
5758   \directlua{
5759     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5760     Babel.characters[\the\count@]['d'] = '#1'
5761   }}
5762 \let\bbl@chprop@bc\bbl@chprop@direction
5763 \def\bbl@chprop@mirror#1{%
```

```
5764  \directlua{
5765    Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5766    Babel.characters[\the\count@]['m'] = '\number#1'
5767  }}
5768 \let\bbl@chprop@bmg\bbl@chprop@mirror
5769 \def\bbl@chprop@linebreak#1{%
5770    \directlua{
5771      Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5772      Babel.cjk_characters[\the\count@]['c'] = '#1'
5773  }}
5774 \let\bbl@chprop@lb\bbl@chprop@linebreak
5775 \def\bbl@chprop@locale#1{%
5776    \directlua{
5777      Babel.chr_to_loc = Babel.chr_to_loc or {}
5778      Babel.chr_to_loc[\the\count@] =
5779        \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5780  }}
```

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.

```
5781 \directlua{
5782    Babel.nohyphenation = \the\l@nohyphenation
5783 }
```

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).

```
5784 \begingroup
5785 \catcode`\~=12
5786 \catcode`\%=12
5787 \catcode`\&=14
5788 \catcode`\|=12
5789 \gdef\babelprehyphenation{&%
5790    \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5791 \gdef\babelposthyphenation{&%
5792    \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5793 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} &% WIP
5794 \gdef\bbl@settransform#1[#2]#3#4#5{&%
5795    \ifcase#1
5796      \bbl@activateprehyphen
5797    \or
5798      \bbl@activateposthyphen
5799    \fi
5800    \begingroup
5801      \def\babeltempa{\bbl@add@list\babeltempb}&%
5802      \let\babeltempb\@empty
5803      \def\bbl@tempa{#5}&%
5804      \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5805      \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5806        \bbl@ifsamestring{##1}{remove}&%
5807          {\bbl@add@list\babeltempb{nil}}&%
5808          {\directlua{
5809             local rep = [=[##1]=]
5810             rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5811             rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5812             rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5813             if #1 == 0 or #1 == 2 then
5814               rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
```

```
5815                    'space = {' .. '%2, %3, %4' .. '}')
5816                  rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5817                     'spacefactor = {' .. '%2, %3, %4' .. '}')
5818                  rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5819                else
5820                  rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5821                  rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5822                  rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5823                end
5824                tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5825            }}}&%
5826      \bbl@foreach\babeltempb{&%
5827        \bbl@forkv{{##1}}{&%
5828          \in@{,####1,}{,nil,step,data,remove,insert,string,no,pre,&%
5829             no,post,penalty,kashida,space,spacefactor,}&%
5830          \ifin@\else
5831            \bbl@error
5832            {Bad option '####1' in a transform.\\&%
5833             I'll ignore it but expect more errors}&%
5834            {See the manual for further info.}&%
5835          \fi}}&%
5836      \let\bbl@kv@attribute\relax
5837      \let\bbl@kv@label\relax
5838      \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5839      \ifx\bbl@kv@attribute\relax\else
5840        \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5841      \fi
5842      \directlua{
5843        local lbkr = Babel.linebreaking.replacements[#1]
5844        local u = unicode.utf8
5845        local id, attr, label
5846        if #1 == 0 or #1 == 2 then
5847          id = \the\csname bbl@id@@#3\endcsname\space
5848        else
5849          id = \the\csname l@#3\endcsname\space
5850        end
5851        \ifx\bbl@kv@attribute\relax
5852          attr = -1
5853        \else
5854          attr = luatexbase.registernumber'\bbl@kv@attribute'
5855        \fi
5856        \ifx\bbl@kv@label\relax\else  &% Same refs:
5857          label = [==[\bbl@kv@label]==]
5858        \fi
5859        &% Convert pattern:
5860        local patt = string.gsub([==[#4]==], '%s', '')
5861        if #1 == 0 or #1 == 2 then
5862          patt = string.gsub(patt, '|', ' ')
5863        end
5864        if not u.find(patt, '()', nil, true) then
5865          patt = '()' .. patt .. '()'
5866        end
5867        if #1 == 1 then
5868          patt = string.gsub(patt, '%(%)%^', '^()')
5869          patt = string.gsub(patt, '%$%(%)', '()$')
5870        end
5871        patt = u.gsub(patt, '{(.)}',
5872                function (n)
5873                  return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5874                end)
5875        patt = u.gsub(patt, '{(%x%x%x%x+)}',
5876                function (n)
5877                  return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
```

178

```
5878            end)
5879        lbkr[id] = lbkr[id] or {}
5880        table.insert(lbkr[id],
5881          { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5882      }&%
5883    \endgroup}
5884 \endgroup
5885 \def\bbl@activateposthyphen{%
5886    \let\bbl@activateposthyphen\relax
5887    \directlua{
5888      require('babel-transforms.lua')
5889      Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5890    }}
5891 \def\bbl@activateprehyphen{%
5892    \let\bbl@activateprehyphen\relax
5893    \directlua{
5894      require('babel-transforms.lua')
5895      Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5896    }}
```

12.9 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.

```
5897 \def\bbl@activate@preotf{%
5898    \let\bbl@activate@preotf\relax  % only once
5899    \directlua{
5900      Babel = Babel or {}
5901      %
5902      function Babel.pre_otfload_v(head)
5903        if Babel.numbers and Babel.digits_mapped then
5904          head = Babel.numbers(head)
5905        end
5906        if Babel.bidi_enabled then
5907          head = Babel.bidi(head, false, dir)
5908        end
5909        return head
5910      end
5911      %
5912      function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5913        if Babel.numbers and Babel.digits_mapped then
5914          head = Babel.numbers(head)
5915        end
5916        if Babel.bidi_enabled then
5917          head = Babel.bidi(head, false, dir)
5918        end
5919        return head
5920      end
5921      %
5922      luatexbase.add_to_callback('pre_linebreak_filter',
5923        Babel.pre_otfload_v,
5924        'Babel.pre_otfload_v',
5925        luatexbase.priority_in_callback('pre_linebreak_filter',
5926          'luaotfload.node_processor') or nil)
5927      %
5928      luatexbase.add_to_callback('hpack_filter',
5929        Babel.pre_otfload_h,
5930        'Babel.pre_otfload_h',
5931        luatexbase.priority_in_callback('hpack_filter',
5932          'luaotfload.node_processor') or nil)
5933    }}
```

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=.

```
5934 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5935   \let\bbl@beforeforeign\leavevmode
5936   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5937   \RequirePackage{luatexbase}
5938   \bbl@activate@preotf
5939   \directlua{
5940     require('babel-data-bidi.lua')
5941     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5942       require('babel-bidi-basic.lua')
5943     \or
5944       require('babel-bidi-basic-r.lua')
5945     \fi}
5946   % TODO - to locale_props, not as separate attribute
5947   \newattribute\bbl@attr@dir
5948   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5949   % TODO. I don't like it, hackish:
5950   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5951   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5952 \fi\fi
5953 \chardef\bbl@thetextdir\z@
5954 \chardef\bbl@thepardir\z@
5955 \def\bbl@getluadir#1{%
5956   \directlua{
5957     if tex.#1dir == 'TLT' then
5958       tex.sprint('0')
5959     elseif tex.#1dir == 'TRT' then
5960       tex.sprint('1')
5961     end}}
5962 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5963   \ifcase#3\relax
5964     \ifcase\bbl@getluadir{#1}\relax\else
5965       #2 TLT\relax
5966     \fi
5967   \else
5968     \ifcase\bbl@getluadir{#1}\relax
5969       #2 TRT\relax
5970     \fi
5971   \fi}
5972 \def\bbl@thedir{0}
5973 \def\bbl@textdir#1{%
5974   \bbl@setluadir{text}\textdir{#1}%
5975   \chardef\bbl@thetextdir#1\relax
5976   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5977   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5978 \def\bbl@pardir#1{%
5979   \bbl@setluadir{par}\pardir{#1}%
5980   \chardef\bbl@thepardir#1\relax}
5981 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5982 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5983 \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
5984 %
5985 \ifnum\bbl@bidimode>\z@
5986   \def\bbl@insidemath{0}%
5987   \def\bbl@everymath{\def\bbl@insidemath{1}}
5988   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
5989   \frozen@everymath\expandafter{%
5990     \expandafter\bbl@everymath\the\frozen@everymath}
5991   \frozen@everydisplay\expandafter{%
5992     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
5993   \AtBeginDocument{
```

```
5994    \directlua{
5995      function Babel.math_box_dir(head)
5996        if not (token.get_macro('bbl@insidemath') == '0') then
5997          if Babel.hlist_has_bidi(head) then
5998            local d = node.new(node.id'dir')
5999            d.dir = '+TRT'
6000            node.insert_before(head, node.has_glyph(head), d)
6001            for item in node.traverse(head) do
6002              node.set_attribute(item,
6003                Babel.attr_dir, token.get_macro('bbl@thedir'))
6004            end
6005          end
6006        end
6007        return head
6008      end
6009      luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6010        "Babel.math_box_dir", 0)
6011  }}%
6012 \fi
```

12.10 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.

```
6013 \bbl@trace{Redefinitions for bidi layout}
6014 %
6015 ⟨⟨*More package options⟩⟩ ≡
6016 \chardef\bbl@eqnpos\z@
6017 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6018 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6019 ⟨⟨/More package options⟩⟩
6020 %
6021 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
6022 \ifnum\bbl@bidimode>\z@
6023   \ifx\matheqdirmode\@undefined\else
6024     \matheqdirmode\@ne
6025   \fi
6026   \let\bbl@eqnodir\relax
6027   \def\bbl@eqdel{()}
6028   \def\bbl@eqnum{%
6029     {\normalfont\normalcolor
6030      \expandafter\@firstoftwo\bbl@eqdel
6031      \theequation
6032      \expandafter\@secondoftwo\bbl@eqdel}}
6033   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6034   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6035   \def\bbl@eqno@flip#1{%
6036     \ifdim\predisplaysize=-\maxdimen
6037       \eqno
6038       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6039     \else
6040       \leqno\hbox{#1}%
6041     \fi}
```

181

```
6042    \def\bbl@leqno@flip#1{%
6043      \ifdim\predisplaysize=-\maxdimen
6044        \leqno
6045        \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6046      \else
6047        \eqno\hbox{#1}%
6048      \fi}
6049    \AtBeginDocument{%
6050      \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6051        \AddToHook{env/equation/begin}{%
6052          \ifnum\bbl@thetextdir>\z@
6053            \let\@eqnnum\bbl@eqnum
6054            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6055            \chardef\bbl@thetextdir\z@
6056            \bbl@add\normalfont{\bbl@eqnodir}%
6057            \ifcase\bbl@eqnpos
6058              \let\bbl@puteqno\bbl@eqno@flip
6059            \or
6060              \let\bbl@puteqno\bbl@leqno@flip
6061            \fi
6062          \fi}%
6063        \ifnum\bbl@eqnpos=\tw@\else
6064          \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6065        \fi
6066        \AddToHook{env/eqnarray/begin}{%
6067          \ifnum\bbl@thetextdir>\z@
6068            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6069            \chardef\bbl@thetextdir\z@
6070            \bbl@add\normalfont{\bbl@eqnodir}%
6071            \ifnum\bbl@eqnpos=\@ne
6072              \def\@eqnnum{%
6073                \setbox\z@\hbox{\bbl@eqnum}%
6074                \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6075            \else
6076              \let\@eqnnum\bbl@eqnum
6077            \fi
6078          \fi}
6079        % Hack. YA luatex bug?:
6080        \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6081      \else % amstex
6082        \ifx\bbl@noamsmath\@undefined
6083          \ifnum\bbl@eqnpos=\@ne
6084            \let\bbl@ams@lap\hbox
6085          \else
6086            \let\bbl@ams@lap\llap
6087          \fi
6088          \ExplSyntaxOn
6089          \bbl@sreplace\intertext@{\normalbaselines}%
6090            {\normalbaselines
6091             \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6092          \ExplSyntaxOff
6093          \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6094          \ifx\bbl@ams@lap\hbox % leqno
6095            \def\bbl@ams@flip#1{%
6096              \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6097          \else % eqno
6098            \def\bbl@ams@flip#1{%
6099              \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6100          \fi
6101          \def\bbl@ams@preset#1{%
6102            \ifnum\bbl@thetextdir>\z@
6103              \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6104              \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
```

```
6105            \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6106          \fi}%
6107        \ifnum\bbl@eqnpos=\tw@\else
6108          \def\bbl@ams@equation{%
6109            \ifnum\bbl@thetextdir>\z@
6110              \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6111              \chardef\bbl@thetextdir\z@
6112              \bbl@add\normalfont{\bbl@eqnodir}%
6113              \ifcase\bbl@eqnpos
6114                \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6115              \or
6116                \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6117              \fi
6118            \fi}%
6119          \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6120          \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6121        \fi
6122        \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6123        \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6124        \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6125        \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6126        \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6127        \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6128        \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6129        % Hackish, for proper alignment. Don't ask me why it works!:
6130        \bbl@exp{% Avoid a 'visible' conditional
6131          \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
6132        \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6133        \AddToHook{env/split/before}{%
6134          \ifnum\bbl@thetextdir>\z@
6135            \bbl@ifsamestring\@currenvir{equation}%
6136              {\ifx\bbl@ams@lap\hbox % leqno
6137                \def\bbl@ams@flip#1{%
6138                  \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6139              \else
6140                \def\bbl@ams@flip#1{%
6141                  \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6142              \fi}%
6143            {}%
6144          \fi}%
6145      \fi
6146    \fi}
6147 \fi
6148 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
6149 \ifnum\bbl@bidimode>\z@
6150   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6151     \bbl@exp{%
6152       \def\\\bbl@insidemath{0}%
6153       \mathdir\the\bodydir
6154       #1%              Once entered in math, set boxes to restore values
6155       \<ifmmode>%
6156         \everyvbox{%
6157           \the\everyvbox
6158           \bodydir\the\bodydir
6159           \mathdir\the\mathdir
6160           \everyhbox{\the\everyhbox}%
6161           \everyvbox{\the\everyvbox}}%
6162         \everyhbox{%
6163           \the\everyhbox
6164           \bodydir\the\bodydir
6165           \mathdir\the\mathdir
6166           \everyhbox{\the\everyhbox}%
6167           \everyvbox{\the\everyvbox}}%
```

183

```
6168         \<fi>}}%
6169     \def\@hangfrom#1{%
6170       \setbox\@tempboxa\hbox{{#1}}%
6171       \hangindent\wd\@tempboxa
6172       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6173         \shapemode\@ne
6174       \fi
6175       \noindent\box\@tempboxa}
6176   \fi
6177   \IfBabelLayout{tabular}
6178     {\let\bbl@OL@@tabular\@tabular
6179      \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6180      \let\bbl@NL@@tabular\@tabular
6181      \AtBeginDocument{%
6182        \ifx\bbl@NL@@tabular\@tabular\else
6183          \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6184          \let\bbl@NL@@tabular\@tabular
6185        \fi}}
6186     {}
6187   \IfBabelLayout{lists}
6188     {\let\bbl@OL@list\list
6189      \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6190      \let\bbl@NL@list\list
6191      \def\bbl@listparshape#1#2#3{%
6192        \parshape #1 #2 #3 %
6193        \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6194          \shapemode\tw@
6195        \fi}}
6196     {}
6197   \IfBabelLayout{graphics}
6198     {\let\bbl@pictresetdir\relax
6199      \def\bbl@pictsetdir#1{%
6200        \ifcase\bbl@thetextdir
6201          \let\bbl@pictresetdir\relax
6202        \else
6203          \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6204            \or\textdir TLT
6205            \else\bodydir TLT \textdir TLT
6206          \fi
6207          % \(text|par)dir required in pgf:
6208          \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6209        \fi}%
6210      \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6211      \directlua{
6212        Babel.get_picture_dir = true
6213        Babel.picture_has_bidi = 0
6214        %
6215        function Babel.picture_dir (head)
6216          if not Babel.get_picture_dir then return head end
6217          if Babel.hlist_has_bidi(head) then
6218            Babel.picture_has_bidi = 1
6219          end
6220          return head
6221        end
6222        luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6223          "Babel.picture_dir")
6224      }%
6225      \AtBeginDocument{%
6226        \long\def\put(#1,#2)#3{%
6227          \@killglue
6228          % Try:
6229          \ifx\bbl@pictresetdir\relax
6230            \def\bbl@tempc{0}%
```

184

```
6231        \else
6232          \directlua{
6233            Babel.get_picture_dir = true
6234            Babel.picture_has_bidi = 0
6235          }%
6236          \setbox\z@\hb@xt@\z@{%
6237            \@defaultunitsset\@tempdimc{#1}\unitlength
6238            \kern\@tempdimc
6239            #3\hss}% TODO: #3 executed twice (below). That's bad.
6240          \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6241        \fi
6242        % Do:
6243        \@defaultunitsset\@tempdimc{#2}\unitlength
6244        \raise\@tempdimc\hb@xt@\z@{%
6245          \@defaultunitsset\@tempdimc{#1}\unitlength
6246          \kern\@tempdimc
6247          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6248        \ignorespaces}%
6249      \MakeRobust\put}%
6250    \AtBeginDocument
6251      {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6252       \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6253         \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6254         \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6255         \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6256       \fi
6257       \ifx\tikzpicture\@undefined\else
6258         \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6259         \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6260         \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6261       \fi
6262       \ifx\tcolorbox\@undefined\else
6263         \def\tcb@drawing@env@begin{%
6264         \csname tcb@before@\tcb@split@state\endcsname
6265         \bbl@pictsetdir\tw@
6266         \begin{kvtcb@graphenv}%
6267         \tcb@bbdraw%
6268         \tcb@apply@graph@patches
6269         }%
6270         \def\tcb@drawing@env@end{%
6271         \end{kvtcb@graphenv}%
6272         \bbl@pictresetdir
6273         \csname tcb@after@\tcb@split@state\endcsname
6274         }%
6275       \fi
6276    }}
6277  {}
```

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.

```
6278 \IfBabelLayout{counters}%
6279  {\let\bbl@OL@@textsuperscript\@textsuperscript
6280   \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6281   \let\bbl@latinarabic=\@arabic
6282   \let\bbl@OL@@arabic\@arabic
6283   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6284   \@ifpackagewith{babel}{bidi=default}%
6285     {\let\bbl@asciiroman=\@roman
6286      \let\bbl@OL@@roman\@roman
6287      \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6288      \let\bbl@asciiRoman=\@Roman
6289      \let\bbl@OL@@roman\@Roman
```

```
6290        \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6291        \let\bbl@OL@labelenumii\labelenumii
6292        \def\labelenumii{)\theenumii(}%
6293        \let\bbl@OL@p@enumiii\p@enumiii
6294        \def\p@enumiii{\p@enumii)\theenumii(}}{}{}
6295 ⟨⟨Footnote changes⟩⟩
6296 \IfBabelLayout{footnotes}%
6297   {\let\bbl@OL@footnote\footnote
6298    \BabelFootnote\footnote\languagename{}{}%
6299    \BabelFootnote\localfootnote\languagename{}{}%
6300    \BabelFootnote\mainfootnote{}{}{}}
6301   {}
```

Some LATEX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```
6302 \IfBabelLayout{extras}%
6303   {\let\bbl@OL@underline\underline
6304    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6305    \let\bbl@OL@LaTeX2e\LaTeX2e
6306    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6307       \if b\expandafter\@car\f@series\@nil\boldmath\fi
6308       \babelsublr{%
6309          \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6310   {}
6311 ⟨/luatex⟩
```

12.11   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 discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).
post_hyphenate_replace is the callback applied after lang.hyphenate. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the luatex manual), we must convert it to a utf8 position. With first, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With last we must take into account the capture position points to the next character. Here word_head points to the starting node of the text to be matched.

```
6312 ⟨*transforms⟩
6313 Babel.linebreaking.replacements = {}
6314 Babel.linebreaking.replacements[0] = {}  -- pre
6315 Babel.linebreaking.replacements[1] = {}  -- post
6316 Babel.linebreaking.replacements[2] = {}  -- post-line WIP
6317
6318 -- Discretionaries contain strings as nodes
6319 function Babel.str_to_nodes(fn, matches, base)
6320   local n, head, last
6321   if fn == nil then return nil end
6322   for s in string.utfvalues(fn(matches)) do
6323     if base.id == 7 then
6324       base = base.replace
6325     end
6326     n = node.copy(base)
6327     n.char     = s
6328     if not head then
6329       head = n
6330     else
6331       last.next = n
6332     end
6333     last = n
6334   end
6335   return head
```

```lua
6336 end
6337
6338 Babel.fetch_subtext = {}
6339
6340 Babel.ignore_pre_char = function(node)
6341   return (node.lang == Babel.nohyphenation)
6342 end
6343
6344 -- Merging both functions doesn't seen feasible, because there are too
6345 -- many differences.
6346 Babel.fetch_subtext[0] = function(head)
6347   local word_string = ''
6348   local word_nodes = {}
6349   local lang
6350   local item = head
6351   local inmath = false
6352
6353   while item do
6354
6355     if item.id == 11 then
6356       inmath = (item.subtype == 0)
6357     end
6358
6359     if inmath then
6360       -- pass
6361
6362     elseif item.id == 29 then
6363       local locale = node.get_attribute(item, Babel.attr_locale)
6364
6365       if lang == locale or lang == nil then
6366         lang = lang or locale
6367         if Babel.ignore_pre_char(item) then
6368           word_string = word_string .. Babel.us_char
6369         else
6370           word_string = word_string .. unicode.utf8.char(item.char)
6371         end
6372         word_nodes[#word_nodes+1] = item
6373       else
6374         break
6375       end
6376
6377     elseif item.id == 12 and item.subtype == 13 then
6378       word_string = word_string .. ' '
6379       word_nodes[#word_nodes+1] = item
6380
6381     -- Ignore leading unrecognized nodes, too.
6382     elseif word_string ~= '' then
6383       word_string = word_string .. Babel.us_char
6384       word_nodes[#word_nodes+1] = item  -- Will be ignored
6385     end
6386
6387     item = item.next
6388   end
6389
6390   -- Here and above we remove some trailing chars but not the
6391   -- corresponding nodes. But they aren't accessed.
6392   if word_string:sub(-1) == ' ' then
6393     word_string = word_string:sub(1,-2)
6394   end
6395   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6396   return word_string, word_nodes, item, lang
6397 end
6398
```

```
6399 Babel.fetch_subtext[1] = function(head)
6400   local word_string = ''
6401   local word_nodes = {}
6402   local lang
6403   local item = head
6404   local inmath = false
6405
6406   while item do
6407
6408     if item.id == 11 then
6409       inmath = (item.subtype == 0)
6410     end
6411
6412     if inmath then
6413       -- pass
6414
6415     elseif item.id == 29 then
6416       if item.lang == lang or lang == nil then
6417         if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6418           lang = lang or item.lang
6419           word_string = word_string .. unicode.utf8.char(item.char)
6420           word_nodes[#word_nodes+1] = item
6421         end
6422       else
6423         break
6424       end
6425
6426     elseif item.id == 7 and item.subtype == 2 then
6427       word_string = word_string .. '='
6428       word_nodes[#word_nodes+1] = item
6429
6430     elseif item.id == 7 and item.subtype == 3 then
6431       word_string = word_string .. '|'
6432       word_nodes[#word_nodes+1] = item
6433
6434     -- (1) Go to next word if nothing was found, and (2) implicitly
6435     -- remove leading USs.
6436     elseif word_string == '' then
6437       -- pass
6438
6439     -- This is the responsible for splitting by words.
6440     elseif (item.id == 12 and item.subtype == 13) then
6441       break
6442
6443     else
6444       word_string = word_string .. Babel.us_char
6445       word_nodes[#word_nodes+1] = item  -- Will be ignored
6446     end
6447
6448     item = item.next
6449   end
6450
6451   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6452   return word_string, word_nodes, item, lang
6453 end
6454
6455 function Babel.pre_hyphenate_replace(head)
6456   Babel.hyphenate_replace(head, 0)
6457 end
6458
6459 function Babel.post_hyphenate_replace(head)
6460   Babel.hyphenate_replace(head, 1)
6461 end
```

```
6462
6463 Babel.us_char = string.char(31)
6464
6465 function Babel.hyphenate_replace(head, mode)
6466   local u = unicode.utf8
6467   local lbkr = Babel.linebreaking.replacements[mode]
6468   if mode == 2 then mode = 0 end -- WIP
6469
6470   local word_head = head
6471
6472   while true do  -- for each subtext block
6473
6474     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6475
6476     if Babel.debug then
6477       print()
6478       print((mode == 0) and '@@@@<' or '@@@@>', w)
6479     end
6480
6481     if nw == nil and w == '' then break end
6482
6483     if not lang then goto next end
6484     if not lbkr[lang] then goto next end
6485
6486     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6487     -- loops are nested.
6488     for k=1, #lbkr[lang] do
6489       local p = lbkr[lang][k].pattern
6490       local r = lbkr[lang][k].replace
6491       local attr = lbkr[lang][k].attr or -1
6492
6493       if Babel.debug then
6494         print('*****', p, mode)
6495       end
6496
6497       -- This variable is set in some cases below to the first *byte*
6498       -- after the match, either as found by u.match (faster) or the
6499       -- computed position based on sc if w has changed.
6500       local last_match = 0
6501       local step = 0
6502
6503       -- For every match.
6504       while true do
6505         if Babel.debug then
6506           print('=====')
6507         end
6508         local new  -- used when inserting and removing nodes
6509
6510         local matches = { u.match(w, p, last_match) }
6511
6512         if #matches < 2 then break end
6513
6514         -- Get and remove empty captures (with ()'s, which return a
6515         -- number with the position), and keep actual captures
6516         -- (from (...)), if any, in matches.
6517         local first = table.remove(matches, 1)
6518         local last  = table.remove(matches, #matches)
6519         -- Non re-fetched substrings may contain \31, which separates
6520         -- subsubstrings.
6521         if string.find(w:sub(first, last-1), Babel.us_char) then break end
6522
6523         local save_last = last -- with A()BC()D, points to D
6524
```

```
6525        -- Fix offsets, from bytes to unicode. Explained above.
6526        first = u.len(w:sub(1, first-1)) + 1
6527        last  = u.len(w:sub(1, last-1)) -- now last points to C
6528
6529        -- This loop stores in a small table the nodes
6530        -- corresponding to the pattern. Used by 'data' to provide a
6531        -- predictable behavior with 'insert' (w_nodes is modified on
6532        -- the fly), and also access to 'remove'd nodes.
6533        local sc = first-1            -- Used below, too
6534        local data_nodes = {}
6535
6536        local enabled = true
6537        for q = 1, last-first+1 do
6538          data_nodes[q] = w_nodes[sc+q]
6539          if enabled
6540              and attr > -1
6541              and not node.has_attribute(data_nodes[q], attr)
6542            then
6543            enabled = false
6544          end
6545        end
6546
6547        -- This loop traverses the matched substring and takes the
6548        -- corresponding action stored in the replacement list.
6549        -- sc = the position in substr nodes / string
6550        -- rc = the replacement table index
6551        local rc = 0
6552
6553        while rc < last-first+1 do -- for each replacement
6554          if Babel.debug then
6555            print('.....', rc + 1)
6556          end
6557          sc = sc + 1
6558          rc = rc + 1
6559
6560          if Babel.debug then
6561            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6562            local ss = ''
6563            for itt in node.traverse(head) do
6564             if itt.id == 29 then
6565               ss = ss .. unicode.utf8.char(itt.char)
6566             else
6567               ss = ss .. '{' .. itt.id .. '}'
6568             end
6569            end
6570            print('*****************', ss)
6571
6572          end
6573
6574          local crep = r[rc]
6575          local item = w_nodes[sc]
6576          local item_base = item
6577          local placeholder = Babel.us_char
6578          local d
6579
6580          if crep and crep.data then
6581            item_base = data_nodes[crep.data]
6582          end
6583
6584          if crep then
6585            step = crep.step or 0
6586          end
6587
```

```lua
6588            if (not enabled) or (crep and next(crep) == nil) then -- = {}
6589              last_match = save_last    -- Optimization
6590              goto next
6591
6592            elseif crep == nil or crep.remove then
6593              node.remove(head, item)
6594              table.remove(w_nodes, sc)
6595              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6596              sc = sc - 1  -- Nothing has been inserted.
6597              last_match = utf8.offset(w, sc+1+step)
6598              goto next
6599
6600            elseif crep and crep.kashida then -- Experimental
6601              node.set_attribute(item,
6602                Babel.attr_kashida,
6603                crep.kashida)
6604              last_match = utf8.offset(w, sc+1+step)
6605              goto next
6606
6607            elseif crep and crep.string then
6608              local str = crep.string(matches)
6609              if str == '' then  -- Gather with nil
6610                node.remove(head, item)
6611                table.remove(w_nodes, sc)
6612                w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6613                sc = sc - 1  -- Nothing has been inserted.
6614              else
6615                local loop_first = true
6616                for s in string.utfvalues(str) do
6617                  d = node.copy(item_base)
6618                  d.char = s
6619                  if loop_first then
6620                    loop_first = false
6621                    head, new = node.insert_before(head, item, d)
6622                    if sc == 1 then
6623                      word_head = head
6624                    end
6625                    w_nodes[sc] = d
6626                    w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6627                  else
6628                    sc = sc + 1
6629                    head, new = node.insert_before(head, item, d)
6630                    table.insert(w_nodes, sc, new)
6631                    w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6632                  end
6633                  if Babel.debug then
6634                    print('.....', 'str')
6635                    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6636                  end
6637                end  -- for
6638                node.remove(head, item)
6639              end  -- if ''
6640              last_match = utf8.offset(w, sc+1+step)
6641              goto next
6642
6643            elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6644              d = node.new(7, 0)   -- (disc, discretionary)
6645              d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6646              d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6647              d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6648              d.attr = item_base.attr
6649              if crep.pre == nil then  -- TeXbook p96
6650                d.penalty = crep.penalty or tex.hyphenpenalty
```

```
6651              else
6652                d.penalty = crep.penalty or tex.exhyphenpenalty
6653              end
6654              placeholder = '|'
6655              head, new = node.insert_before(head, item, d)
6656
6657          elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6658              -- ERROR
6659
6660          elseif crep and crep.penalty then
6661              d = node.new(14, 0)   -- (penalty, userpenalty)
6662              d.attr = item_base.attr
6663              d.penalty = crep.penalty
6664              head, new = node.insert_before(head, item, d)
6665
6666          elseif crep and crep.space then
6667              -- 655360 = 10 pt = 10 * 65536 sp
6668              d = node.new(12, 13)      -- (glue, spaceskip)
6669              local quad = font.getfont(item_base.font).size or 655360
6670              node.setglue(d, crep.space[1] * quad,
6671                              crep.space[2] * quad,
6672                              crep.space[3] * quad)
6673              if mode == 0 then
6674                placeholder = ' '
6675              end
6676              head, new = node.insert_before(head, item, d)
6677
6678          elseif crep and crep.spacefactor then
6679              d = node.new(12, 13)       -- (glue, spaceskip)
6680              local base_font = font.getfont(item_base.font)
6681              node.setglue(d,
6682                crep.spacefactor[1] * base_font.parameters['space'],
6683                crep.spacefactor[2] * base_font.parameters['space_stretch'],
6684                crep.spacefactor[3] * base_font.parameters['space_shrink'])
6685              if mode == 0 then
6686                placeholder = ' '
6687              end
6688              head, new = node.insert_before(head, item, d)
6689
6690          elseif mode == 0 and crep and crep.space then
6691              -- ERROR
6692
6693          end  -- ie replacement cases
6694
6695          -- Shared by disc, space and penalty.
6696          if sc == 1 then
6697            word_head = head
6698          end
6699          if crep.insert then
6700            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6701            table.insert(w_nodes, sc, new)
6702            last = last + 1
6703          else
6704            w_nodes[sc] = d
6705            node.remove(head, item)
6706            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6707          end
6708
6709          last_match = utf8.offset(w, sc+1+step)
6710
6711          ::next::
6712
6713      end  -- for each replacement
```

```
6714
6715        if Babel.debug then
6716            print('.....', '/')
6717            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6718        end
6719
6720      end  -- for match
6721
6722    end  -- for patterns
6723
6724    ::next::
6725    word_head = nw
6726  end  -- for substring
6727  return head
6728 end
6729
6730 -- This table stores capture maps, numbered consecutively
6731 Babel.capture_maps = {}
6732
6733 -- The following functions belong to the next macro
6734 function Babel.capture_func(key, cap)
6735   local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6736   local cnt
6737   local u = unicode.utf8
6738   ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6739   if cnt == 0 then
6740     ret = u.gsub(ret, '{(%x%x%x%x+)}',
6741           function (n)
6742             return u.char(tonumber(n, 16))
6743           end)
6744   end
6745   ret = ret:gsub("%[%[%]%]%.%.", '')
6746   ret = ret:gsub("%.%.%[%[%]%]", '')
6747   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6748 end
6749
6750 function Babel.capt_map(from, mapno)
6751   return Babel.capture_maps[mapno][from] or from
6752 end
6753
6754 -- Handle the {n|abc|ABC} syntax in captures
6755 function Babel.capture_func_map(capno, from, to)
6756   local u = unicode.utf8
6757   from = u.gsub(from, '{(%x%x%x%x+)}',
6758       function (n)
6759         return u.char(tonumber(n, 16))
6760       end)
6761   to = u.gsub(to, '{(%x%x%x%x+)}',
6762       function (n)
6763         return u.char(tonumber(n, 16))
6764       end)
6765   local froms = {}
6766   for s in string.utfcharacters(from) do
6767     table.insert(froms, s)
6768   end
6769   local cnt = 1
6770   table.insert(Babel.capture_maps, {})
6771   local mlen = table.getn(Babel.capture_maps)
6772   for s in string.utfcharacters(to) do
6773     Babel.capture_maps[mlen][froms[cnt]] = s
6774     cnt = cnt + 1
6775   end
6776   return "]]..Babel.capt_map(m[" .. capno .. "]," ..
```

```
6777        (mlen) .. ")..". .. "[["
6778 end
6779
6780 -- Create/Extend reversed sorted list of kashida weights:
6781 function Babel.capture_kashida(key, wt)
6782   wt = tonumber(wt)
6783   if Babel.kashida_wts then
6784     for p, q in ipairs(Babel.kashida_wts) do
6785       if wt  == q then
6786         break
6787       elseif wt > q then
6788         table.insert(Babel.kashida_wts, p, wt)
6789         break
6790       elseif table.getn(Babel.kashida_wts) == p then
6791         table.insert(Babel.kashida_wts, wt)
6792       end
6793     end
6794   else
6795     Babel.kashida_wts = { wt }
6796   end
6797   return 'kashida = ' .. wt
6798 end
6799 ⟨/transforms⟩
```

12.12   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:

```
[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.

```
6800 ⟨∗basic-r⟩
```

194

```
6801 Babel = Babel or {}
6802
6803 Babel.bidi_enabled = true
6804
6805 require('babel-data-bidi.lua')
6806
6807 local characters = Babel.characters
6808 local ranges = Babel.ranges
6809
6810 local DIR = node.id("dir")
6811
6812 local function dir_mark(head, from, to, outer)
6813   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6814   local d = node.new(DIR)
6815   d.dir = '+' .. dir
6816   node.insert_before(head, from, d)
6817   d = node.new(DIR)
6818   d.dir = '-' .. dir
6819   node.insert_after(head, to, d)
6820 end
6821
6822 function Babel.bidi(head, ispar)
6823   local first_n, last_n          -- first and last char with nums
6824   local last_es                  -- an auxiliary 'last' used with nums
6825   local first_d, last_d          -- first and last char in L/R block
6826   local dir, dir_real
```

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

```
6827   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6828   local strong_lr = (strong == 'l') and 'l' or 'r'
6829   local outer = strong
6830
6831   local new_dir = false
6832   local first_dir = false
6833   local inmath = false
6834
6835   local last_lr
6836
6837   local type_n = ''
6838
6839   for item in node.traverse(head) do
6840
6841     -- three cases: glyph, dir, otherwise
6842     if item.id == node.id'glyph'
6843       or (item.id == 7 and item.subtype == 2) then
6844
6845       local itemchar
6846       if item.id == 7 and item.subtype == 2 then
6847         itemchar = item.replace.char
6848       else
6849         itemchar = item.char
6850       end
6851       local chardata = characters[itemchar]
6852       dir = chardata and chardata.d or nil
6853       if not dir then
6854         for nn, et in ipairs(ranges) do
6855           if itemchar < et[1] then
6856             break
6857           elseif itemchar <= et[2] then
6858             dir = et[3]
6859             break
```

```
6860            end
6861          end
6862        end
6863        dir = dir or 'l'
6864        if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end
```

Next is based on the assumption babel sets the language AND switches the script with its dir. We treat a language block as a separate Unicode sequence. The following piece of code is executed at the first glyph after a 'dir' node. We don't know the current language until then. This is not exactly true, as the math mode may insert explicit dirs in the node list, so, for the moment there is a hack by brute force (just above).

```
6865        if new_dir then
6866          attr_dir = 0
6867          for at in node.traverse(item.attr) do
6868            if at.number == Babel.attr_dir then
6869              attr_dir = at.value % 3
6870            end
6871          end
6872          if attr_dir == 1 then
6873            strong = 'r'
6874          elseif attr_dir == 2 then
6875            strong = 'al'
6876          else
6877            strong = 'l'
6878          end
6879          strong_lr = (strong == 'l') and 'l' or 'r'
6880          outer = strong_lr
6881          new_dir = false
6882        end
6883
6884        if dir == 'nsm' then dir = strong end          -- W1
```

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

```
6885        dir_real = dir              -- We need dir_real to set strong below
6886        if dir == 'al' then dir = 'r' end -- W3
```

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

```
6887        if strong == 'al' then
6888          if dir == 'en' then dir = 'an' end            -- W2
6889          if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6890          strong_lr = 'r'                               -- W3
6891        end
```

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

```
6892      elseif item.id == node.id'dir' and not inmath then
6893        new_dir = true
6894        dir = nil
6895      elseif item.id == node.id'math' then
6896        inmath = (item.subtype == 0)
6897      else
6898        dir = nil          -- Not a char
6899      end
```

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

```
6900      if dir == 'en' or dir == 'an' or dir == 'et' then
6901        if dir ~= 'et' then
6902          type_n = dir
6903        end
6904        first_n = first_n or item
```

```
6905       last_n = last_es or item
6906       last_es = nil
6907     elseif dir == 'es' and last_n then  -- W3+W6
6908       last_es = item
6909     elseif dir == 'cs' then            -- it's right - do nothing
6910     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6911       if strong_lr == 'r' and type_n ~= '' then
6912         dir_mark(head, first_n, last_n, 'r')
6913       elseif strong_lr == 'l' and first_d and type_n == 'an' then
6914         dir_mark(head, first_n, last_n, 'r')
6915         dir_mark(head, first_d, last_d, outer)
6916         first_d, last_d = nil, nil
6917       elseif strong_lr == 'l' and type_n ~= '' then
6918         last_d = last_n
6919       end
6920       type_n = ''
6921       first_n, last_n = nil, nil
6922     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:

```
6923     if dir == 'l' or dir == 'r' then
6924       if dir ~= outer then
6925         first_d = first_d or item
6926         last_d = item
6927       elseif first_d and dir ~= strong_lr then
6928         dir_mark(head, first_d, last_d, outer)
6929         first_d, last_d = nil, nil
6930       end
6931     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>, resptly, 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.

```
6932     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6933       item.char = characters[item.char] and
6934                   characters[item.char].m or item.char
6935     elseif (dir or new_dir) and last_lr ~= item then
6936       local mir = outer .. strong_lr .. (dir or outer)
6937       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6938         for ch in node.traverse(node.next(last_lr)) do
6939           if ch == item then break end
6940           if ch.id == node.id'glyph' and characters[ch.char] then
6941             ch.char = characters[ch.char].m or ch.char
6942           end
6943         end
6944       end
6945     end
```

Save some values for the next iteration. If the current node is 'dir', open a new sequence. Since dir could be changed, strong is set with its real value (dir_real).

```
6946     if dir == 'l' or dir == 'r' then
6947       last_lr = item
6948       strong = dir_real          -- Don't search back - best save now
6949       strong_lr = (strong == 'l') and 'l' or 'r'
6950     elseif new_dir then
6951       last_lr = nil
6952     end
6953   end
```

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

```
6954  if last_lr and outer == 'r' then
6955    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6956      if characters[ch.char] then
6957        ch.char = characters[ch.char].m or ch.char
6958      end
6959    end
6960  end
6961  if first_n then
6962    dir_mark(head, first_n, last_n, outer)
6963  end
6964  if first_d then
6965    dir_mark(head, first_d, last_d, outer)
6966  end
```

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

```
6967  return node.prev(head) or head
6968 end
6969 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6970 ⟨∗basic⟩
6971 Babel = Babel or {}
6972
6973 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6974
6975 Babel.fontmap = Babel.fontmap or {}
6976 Babel.fontmap[0] = {}       -- l
6977 Babel.fontmap[1] = {}       -- r
6978 Babel.fontmap[2] = {}       -- al/an
6979
6980 Babel.bidi_enabled = true
6981 Babel.mirroring_enabled = true
6982
6983 require('babel-data-bidi.lua')
6984
6985 local characters = Babel.characters
6986 local ranges = Babel.ranges
6987
6988 local DIR = node.id('dir')
6989 local GLYPH = node.id('glyph')
6990
6991 local function insert_implicit(head, state, outer)
6992   local new_state = state
6993   if state.sim and state.eim and state.sim ~= state.eim then
6994     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6995     local d = node.new(DIR)
6996     d.dir = '+' .. dir
6997     node.insert_before(head, state.sim, d)
6998     local d = node.new(DIR)
6999     d.dir = '-' .. dir
7000     node.insert_after(head, state.eim, d)
7001   end
7002   new_state.sim, new_state.eim = nil, nil
7003   return head, new_state
7004 end
7005
7006 local function insert_numeric(head, state)
7007   local new
7008   local new_state = state
7009   if state.san and state.ean and state.san ~= state.ean then
7010     local d = node.new(DIR)
```

```
7011      d.dir = '+TLT'
7012      _, new = node.insert_before(head, state.san, d)
7013      if state.san == state.sim then state.sim = new end
7014      local d = node.new(DIR)
7015      d.dir = '-TLT'
7016      _, new = node.insert_after(head, state.ean, d)
7017      if state.ean == state.eim then state.eim = new end
7018    end
7019    new_state.san, new_state.ean = nil, nil
7020    return head, new_state
7021 end
7022
7023 -- TODO - \hbox with an explicit dir can lead to wrong results
7024 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7025 -- was s made to improve the situation, but the problem is the 3-dir
7026 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7027 -- well.
7028
7029 function Babel.bidi(head, ispar, hdir)
7030    local d    -- d is used mainly for computations in a loop
7031    local prev_d = ''
7032    local new_d = false
7033
7034    local nodes = {}
7035    local outer_first = nil
7036    local inmath = false
7037
7038    local glue_d = nil
7039    local glue_i = nil
7040
7041    local has_en = false
7042    local first_et = nil
7043
7044    local ATDIR = Babel.attr_dir
7045
7046    local save_outer
7047    local temp = node.get_attribute(head, ATDIR)
7048    if temp then
7049      temp = temp % 3
7050      save_outer = (temp == 0 and 'l') or
7051                   (temp == 1 and 'r') or
7052                   (temp == 2 and 'al')
7053    elseif ispar then           -- Or error? Shouldn't happen
7054      save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7055    else                        -- Or error? Shouldn't happen
7056      save_outer = ('TRT' == hdir) and 'r' or 'l'
7057    end
7058      -- when the callback is called, we are just _after_ the box,
7059      -- and the textdir is that of the surrounding text
7060    -- if not ispar and hdir ~= tex.textdir then
7061    --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7062    -- end
7063    local outer = save_outer
7064    local last = outer
7065    -- 'al' is only taken into account in the first, current loop
7066    if save_outer == 'al' then save_outer = 'r' end
7067
7068    local fontmap = Babel.fontmap
7069
7070    for item in node.traverse(head) do
7071
7072      -- In what follows, #node is the last (previous) node, because the
7073      -- current one is not added until we start processing the neutrals.
```

```
7074
7075     -- three cases: glyph, dir, otherwise
7076     if item.id == GLYPH
7077         or (item.id == 7 and item.subtype == 2) then
7078
7079       local d_font = nil
7080       local item_r
7081       if item.id == 7 and item.subtype == 2 then
7082         item_r = item.replace    -- automatic discs have just 1 glyph
7083       else
7084         item_r = item
7085       end
7086       local chardata = characters[item_r.char]
7087       d = chardata and chardata.d or nil
7088       if not d or d == 'nsm' then
7089         for nn, et in ipairs(ranges) do
7090           if item_r.char < et[1] then
7091             break
7092           elseif item_r.char <= et[2] then
7093             if not d then d = et[3]
7094             elseif d == 'nsm' then d_font = et[3]
7095             end
7096             break
7097           end
7098         end
7099       end
7100       d = d or 'l'
7101
7102       -- A short 'pause' in bidi for mapfont
7103       d_font = d_font or d
7104       d_font = (d_font == 'l' and 0) or
7105                (d_font == 'nsm' and 0) or
7106                (d_font == 'r' and 1) or
7107                (d_font == 'al' and 2) or
7108                (d_font == 'an' and 2) or nil
7109       if d_font and fontmap and fontmap[d_font][item_r.font] then
7110         item_r.font = fontmap[d_font][item_r.font]
7111       end
7112
7113       if new_d then
7114         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7115         if inmath then
7116           attr_d = 0
7117         else
7118           attr_d = node.get_attribute(item, ATDIR)
7119           attr_d = attr_d % 3
7120         end
7121         if attr_d == 1 then
7122           outer_first = 'r'
7123           last = 'r'
7124         elseif attr_d == 2 then
7125           outer_first = 'r'
7126           last = 'al'
7127         else
7128           outer_first = 'l'
7129           last = 'l'
7130         end
7131         outer = last
7132         has_en = false
7133         first_et = nil
7134         new_d = false
7135       end
7136
```

```
7137         if glue_d then
7138           if (d == 'l' and 'l' or 'r') ~= glue_d then
7139             table.insert(nodes, {glue_i, 'on', nil})
7140           end
7141           glue_d = nil
7142           glue_i = nil
7143         end
7144
7145       elseif item.id == DIR then
7146         d = nil
7147         if head ~= item then new_d = true end
7148
7149       elseif item.id == node.id'glue' and item.subtype == 13 then
7150         glue_d = d
7151         glue_i = item
7152         d = nil
7153
7154       elseif item.id == node.id'math' then
7155         inmath = (item.subtype == 0)
7156
7157       else
7158         d = nil
7159       end
7160
7161       -- AL <= EN/ET/ES     -- W2 + W3 + W6
7162       if last == 'al' and d == 'en' then
7163         d = 'an'              -- W3
7164       elseif last == 'al' and (d == 'et' or d == 'es') then
7165         d = 'on'              -- W6
7166       end
7167
7168       -- EN + CS/ES + EN     -- W4
7169       if d == 'en' and #nodes >= 2 then
7170         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7171             and nodes[#nodes-1][2] == 'en' then
7172           nodes[#nodes][2] = 'en'
7173         end
7174       end
7175
7176       -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
7177       if d == 'an' and #nodes >= 2 then
7178         if (nodes[#nodes][2] == 'cs')
7179             and nodes[#nodes-1][2] == 'an' then
7180           nodes[#nodes][2] = 'an'
7181         end
7182       end
7183
7184       -- ET/EN              -- W5 + W7->l / W6->on
7185       if d == 'et' then
7186         first_et = first_et or (#nodes + 1)
7187       elseif d == 'en' then
7188         has_en = true
7189         first_et = first_et or (#nodes + 1)
7190       elseif first_et then       -- d may be nil here !
7191         if has_en then
7192           if last == 'l' then
7193             temp = 'l'     -- W7
7194           else
7195             temp = 'en'    -- W5
7196           end
7197         else
7198           temp = 'on'      -- W6
7199         end
```

```
7200        for e = first_et, #nodes do
7201          if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7202        end
7203        first_et = nil
7204        has_en = false
7205      end
7206
7207      -- Force mathdir in math if ON (currently works as expected only
7208      -- with 'l')
7209      if inmath and d == 'on' then
7210        d = ('TRT' == tex.mathdir) and 'r' or 'l'
7211      end
7212
7213      if d then
7214        if d == 'al' then
7215          d = 'r'
7216          last = 'al'
7217        elseif d == 'l' or d == 'r' then
7218          last = d
7219        end
7220        prev_d = d
7221        table.insert(nodes, {item, d, outer_first})
7222      end
7223
7224    outer_first = nil
7225
7226  end
7227
7228  -- TODO -- repeated here in case EN/ET is the last node. Find a
7229  -- better way of doing things:
7230  if first_et then        -- dir may be nil here !
7231    if has_en then
7232      if last == 'l' then
7233        temp = 'l'     -- W7
7234      else
7235        temp = 'en'    -- W5
7236      end
7237    else
7238      temp = 'on'      -- W6
7239    end
7240    for e = first_et, #nodes do
7241      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7242    end
7243  end
7244
7245  -- dummy node, to close things
7246  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7247
7248  --------------  NEUTRAL -----------------
7249
7250  outer = save_outer
7251  last = outer
7252
7253  local first_on = nil
7254
7255  for q = 1, #nodes do
7256    local item
7257
7258    local outer_first = nodes[q][3]
7259    outer = outer_first or outer
7260    last = outer_first or last
7261
7262    local d = nodes[q][2]
```

```
7263    if d == 'an' or d == 'en' then d = 'r' end
7264    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7265
7266    if d == 'on' then
7267      first_on = first_on or q
7268    elseif first_on then
7269      if last == d then
7270        temp = d
7271      else
7272        temp = outer
7273      end
7274      for r = first_on, q - 1 do
7275        nodes[r][2] = temp
7276        item = nodes[r][1]    -- MIRRORING
7277        if Babel.mirroring_enabled and item.id == GLYPH
7278            and temp == 'r' and characters[item.char] then
7279          local font_mode = ''
7280          if font.fonts[item.font].properties then
7281            font_mode = font.fonts[item.font].properties.mode
7282          end
7283          if font_mode ~= 'harf' and font_mode ~= 'plug' then
7284            item.char = characters[item.char].m or item.char
7285          end
7286        end
7287      end
7288      first_on = nil
7289    end
7290
7291    if d == 'r' or d == 'l' then last = d end
7292  end
7293
7294  --------------  IMPLICIT, REORDER ----------------
7295
7296  outer = save_outer
7297  last = outer
7298
7299  local state = {}
7300  state.has_r = false
7301
7302  for q = 1, #nodes do
7303
7304    local item = nodes[q][1]
7305
7306    outer = nodes[q][3] or outer
7307
7308    local d = nodes[q][2]
7309
7310    if d == 'nsm' then d = last end              -- W1
7311    if d == 'en' then d = 'an' end
7312    local isdir = (d == 'r' or d == 'l')
7313
7314    if outer == 'l' and d == 'an' then
7315      state.san = state.san or item
7316      state.ean = item
7317    elseif state.san then
7318      head, state = insert_numeric(head, state)
7319    end
7320
7321    if outer == 'l' then
7322      if d == 'an' or d == 'r' then      -- im -> implicit
7323        if d == 'r' then state.has_r = true end
7324        state.sim = state.sim or item
7325        state.eim = item
```

203

```
7326        elseif d == 'l' and state.sim and state.has_r then
7327          head, state = insert_implicit(head, state, outer)
7328        elseif d == 'l' then
7329          state.sim, state.eim, state.has_r = nil, nil, false
7330        end
7331      else
7332        if d == 'an' or d == 'l' then
7333          if nodes[q][3] then -- nil except after an explicit dir
7334            state.sim = item  -- so we move sim 'inside' the group
7335          else
7336            state.sim = state.sim or item
7337          end
7338          state.eim = item
7339        elseif d == 'r' and state.sim then
7340          head, state = insert_implicit(head, state, outer)
7341        elseif d == 'r' then
7342          state.sim, state.eim = nil, nil
7343        end
7344      end
7345
7346      if isdir then
7347        last = d              -- Don't search back - best save now
7348      elseif d == 'on' and state.san  then
7349        state.san = state.san or item
7350        state.ean = item
7351      end
7352
7353    end
7354
7355    return node.prev(head) or head
7356 end
7357 ⟨/basic⟩
```

13   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.

14   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.

```
7358 ⟨∗nil⟩
7359 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7360 \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.

```
7361 \ifx\l@nil\@undefined
7362   \newlanguage\l@nil
```

```
7363    \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7364    \let\bbl@elt\relax
7365    \edef\bbl@languages{%  Add it to the list of languages
7366      \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7367 \fi
```

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

```
7368 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
\datenil

```
7369 \let\captionsnil\@empty
7370 \let\datenil\@empty
```

There is no locale file for this pseudo-language, so the corresponding fields are defined here.

```
7371 \def\bbl@inidata@nil{%
7372    \bbl@elt{identification}{tag.ini}{und}%
7373    \bbl@elt{identification}{load.level}{0}%
7374    \bbl@elt{identification}{charset}{utf8}%
7375    \bbl@elt{identification}{version}{1.0}%
7376    \bbl@elt{identification}{date}{2022-05-16}%
7377    \bbl@elt{identification}{name.local}{nil}%
7378    \bbl@elt{identification}{name.english}{nil}%
7379    \bbl@elt{identification}{name.babel}{nil}%
7380    \bbl@elt{identification}{tag.bcp47}{und}%
7381    \bbl@elt{identification}{language.tag.bcp47}{und}%
7382    \bbl@elt{identification}{tag.opentype}{dflt}%
7383    \bbl@elt{identification}{script.name}{Latin}%
7384    \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7385    \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7386    \bbl@elt{identification}{level}{1}%
7387    \bbl@elt{identification}{encodings}{}%
7388    \bbl@elt{identification}{derivate}{no}}
7389 \@namedef{bbl@tbcp@nil}{und}
7390 \@namedef{bbl@lbcp@nil}{und}
7391 \@namedef{bbl@lotf@nil}{dflt}
7392 \@namedef{bbl@elname@nil}{nil}
7393 \@namedef{bbl@lname@nil}{nil}
7394 \@namedef{bbl@esname@nil}{Latin}
7395 \@namedef{bbl@sname@nil}{Latin}
7396 \@namedef{bbl@sbcp@nil}{Latn}
7397 \@namedef{bbl@sotf@nil}{Latn}
```

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.

```
7398 \ldf@finish{nil}
7399 ⟨/nil⟩
```

15   Calendars

The code for specific calendars are placed in the specific files, loaded when requested by an ini file in the identification section with require.calendars.
Start with function to compute the Julian day. It's based on the little library calendar.js, by John Walker, in the public domain.

```
7400 ⟨⟨*Compute Julian day⟩⟩ ≡
7401 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
7402 \def\bbl@cs@gregleap#1{%
7403    (\bbl@fpmod{#1}{4} == 0) &&
7404      (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
7405 \def\bbl@cs@jd#1#2#3{% year, month, day
7406    \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
```

```
7407    floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
7408    floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7409    ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7410 ⟨⟨/Compute Julian day⟩⟩
```

15.1 Islamic

The code for the Civil calendar is based on it, too.

```
7411 ⟨∗ca-islamic⟩
7412 \ExplSyntaxOn
7413 ⟨⟨Compute Julian day⟩⟩
7414 % == islamic (default)
7415 % Not yet implemented
7416 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
7417 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7418    ((#3 + ceil(29.5 * (#2 - 1)) +
7419    (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7420    1948439.5) - 1) }
7421 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7422 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7423 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7424 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7425 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7426 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7427    \edef\bbl@tempa{%
7428       \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7429    \edef#5{%
7430       \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7431    \edef#6{\fp_eval:n{
7432       min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7433    \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}
```

The Umm al-Qura calendar, used mainly in Saudi Arabia, is based on moment-hijri, by Abdullah Alsigar (license MIT).

Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers Hijri ∼1435/∼1460 (Gregorian ∼2014/∼2038).

```
7434 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7435    56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7436    57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7437    57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7438    57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7439    58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7440    58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7441    58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7442    58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7443    59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7444    59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7445    59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7446    60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7447    60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7448    60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7449    60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7450    61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7451    61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7452    61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7453    62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7454    62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7455    62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7456    63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7457    63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7458    63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
```

```
7459    63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7460    64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7461    64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7462    64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7463    65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7464    65401,65431,65460,65490,65520}
7465 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7466 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7467 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7468 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
7469    \ifnum#2>2014 \ifnum#2<2038
7470        \bbl@afterfi\expandafter\@gobble
7471    \fi\fi
7472        {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
7473    \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7474        \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7475    \count@\@ne
7476    \bbl@foreach\bbl@cs@umalqura@data{%
7477        \advance\count@\@ne
7478        \ifnum##1>\bbl@tempd\else
7479            \edef\bbl@tempe{\the\count@}%
7480            \edef\bbl@tempb{##1}%
7481        \fi}%
7482    \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7483    \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7484    \edef#5{\fp_eval:n{ \bbl@tempa + 1  }}%
7485    \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7486    \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}}
7487 \ExplSyntaxOff
7488 \bbl@add\bbl@precalendar{%
7489    \bbl@replace\bbl@ld@calendar{-civil}{}%
7490    \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7491    \bbl@replace\bbl@ld@calendar{+}{}%
7492    \bbl@replace\bbl@ld@calendar{-}{}}}
7493 ⟨/ca-islamic⟩
```

16 Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptions by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in `hebcal.sty`

```
7494 ⟨*ca-hebrew⟩
7495 \newcount\bbl@cntcommon
7496 \def\bbl@remainder#1#2#3{%
7497    #3=#1\relax
7498    \divide #3 by #2\relax
7499    \multiply #3 by -#2\relax
7500    \advance #3 by #1\relax}%
7501 \newif\ifbbl@divisible
7502 \def\bbl@checkifdivisible#1#2{%
7503    {\countdef\tmp=0
7504    \bbl@remainder{#1}{#2}{\tmp}%
7505    \ifnum \tmp=0
7506        \global\bbl@divisibletrue
7507    \else
7508        \global\bbl@divisiblefalse
7509    \fi}}
7510 \newif\ifbbl@gregleap
7511 \def\bbl@ifgregleap#1{%
7512    \bbl@checkifdivisible{#1}{4}%
7513    \ifbbl@divisible
7514        \bbl@checkifdivisible{#1}{100}%
```

```
7515      \ifbbl@divisible
7516          \bbl@checkifdivisible{#1}{400}%
7517            \ifbbl@divisible
7518                \bbl@gregleaptrue
7519            \else
7520                \bbl@gregleapfalse
7521            \fi
7522        \else
7523            \bbl@gregleaptrue
7524        \fi
7525    \else
7526        \bbl@gregleapfalse
7527    \fi
7528    \ifbbl@gregleap}
7529 \def\bbl@gregdayspriormonths#1#2#3{%
7530      {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7531          181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7532      \bbl@ifgregleap{#2}%
7533          \ifnum #1 > 2
7534              \advance #3 by 1
7535          \fi
7536      \fi
7537      \global\bbl@cntcommon=#3}%
7538      #3=\bbl@cntcommon}
7539 \def\bbl@gregdaysprioryears#1#2{%
7540    {\countdef\tmpc=4
7541    \countdef\tmpb=2
7542    \tmpb=#1\relax
7543    \advance \tmpb by -1
7544    \tmpc=\tmpb
7545    \multiply \tmpc by 365
7546    #2=\tmpc
7547    \tmpc=\tmpb
7548    \divide \tmpc by 4
7549    \advance #2 by \tmpc
7550    \tmpc=\tmpb
7551    \divide \tmpc by 100
7552    \advance #2 by -\tmpc
7553    \tmpc=\tmpb
7554    \divide \tmpc by 400
7555    \advance #2 by \tmpc
7556    \global\bbl@cntcommon=#2\relax}%
7557    #2=\bbl@cntcommon}
7558 \def\bbl@absfromgreg#1#2#3#4{%
7559    {\countdef\tmpd=0
7560    #4=#1\relax
7561    \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7562    \advance #4 by \tmpd
7563    \bbl@gregdaysprioryears{#3}{\tmpd}%
7564    \advance #4 by \tmpd
7565    \global\bbl@cntcommon=#4\relax}%
7566    #4=\bbl@cntcommon}
7567 \newif\ifbbl@hebrleap
7568 \def\bbl@checkleaphebryear#1{%
7569    {\countdef\tmpa=0
7570    \countdef\tmpb=1
7571    \tmpa=#1\relax
7572    \multiply \tmpa by 7
7573    \advance \tmpa by 1
7574    \bbl@remainder{\tmpa}{19}{\tmpb}%
7575    \ifnum \tmpb < 7
7576        \global\bbl@hebrleaptrue
7577    \else
```

```
7578        \global\bbl@hebrleapfalse
7579      \fi}}
7580  \def\bbl@hebrelapsedmonths#1#2{%
7581    {\countdef\tmpa=0
7582     \countdef\tmpb=1
7583     \countdef\tmpc=2
7584     \tmpa=#1\relax
7585     \advance \tmpa by -1
7586     #2=\tmpa
7587     \divide #2 by 19
7588     \multiply #2 by 235
7589     \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7590     \tmpc=\tmpb
7591     \multiply \tmpb by 12
7592     \advance #2 by \tmpb
7593     \multiply \tmpc by 7
7594     \advance \tmpc by 1
7595     \divide \tmpc by 19
7596     \advance #2 by \tmpc
7597     \global\bbl@cntcommon=#2}%
7598    #2=\bbl@cntcommon}
7599  \def\bbl@hebrelapseddays#1#2{%
7600    {\countdef\tmpa=0
7601     \countdef\tmpb=1
7602     \countdef\tmpc=2
7603     \bbl@hebrelapsedmonths{#1}{#2}%
7604     \tmpa=#2\relax
7605     \multiply \tmpa by 13753
7606     \advance \tmpa by 5604
7607     \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7608     \divide \tmpa by 25920
7609     \multiply #2 by 29
7610     \advance #2 by 1
7611     \advance #2 by \tmpa
7612     \bbl@remainder{#2}{7}{\tmpa}%
7613     \ifnum \tmpc < 19440
7614        \ifnum \tmpc < 9924
7615        \else
7616           \ifnum \tmpa=2
7617              \bbl@checkleaphebryear{#1}% of a common year
7618              \ifbbl@hebrleap
7619              \else
7620                 \advance #2 by 1
7621              \fi
7622           \fi
7623        \fi
7624        \ifnum \tmpc < 16789
7625        \else
7626           \ifnum \tmpa=1
7627              \advance #1 by -1
7628              \bbl@checkleaphebryear{#1}% at the end of leap year
7629              \ifbbl@hebrleap
7630                 \advance #2 by 1
7631              \fi
7632           \fi
7633        \fi
7634     \else
7635        \advance #2 by 1
7636     \fi
7637     \bbl@remainder{#2}{7}{\tmpa}%
7638     \ifnum \tmpa=0
7639        \advance #2 by 1
7640     \else
```

```
7641        \ifnum \tmpa=3
7642            \advance #2 by 1
7643        \else
7644            \ifnum \tmpa=5
7645                \advance #2 by 1
7646            \fi
7647        \fi
7648    \fi
7649    \global\bbl@cntcommon=#2\relax}%
7650  #2=\bbl@cntcommon}
7651 \def\bbl@daysinhebryear#1#2{%
7652  {\countdef\tmpe=12
7653    \bbl@hebrelapseddays{#1}{\tmpe}%
7654    \advance #1 by 1
7655    \bbl@hebrelapseddays{#1}{#2}%
7656    \advance #2 by -\tmpe
7657    \global\bbl@cntcommon=#2}%
7658  #2=\bbl@cntcommon}
7659 \def\bbl@hebrdayspriormonths#1#2#3{%
7660  {\countdef\tmpf= 14
7661    #3=\ifcase #1\relax
7662          0 \or
7663          0 \or
7664         30 \or
7665         59 \or
7666         89 \or
7667        118 \or
7668        148 \or
7669        148 \or
7670        177 \or
7671        207 \or
7672        236 \or
7673        266 \or
7674        295 \or
7675        325 \or
7676        400
7677    \fi
7678    \bbl@checkleaphebryear{#2}%
7679    \ifbbl@hebrleap
7680        \ifnum #1 > 6
7681            \advance #3 by 30
7682        \fi
7683    \fi
7684    \bbl@daysinhebryear{#2}{\tmpf}%
7685    \ifnum #1 > 3
7686        \ifnum \tmpf=353
7687            \advance #3 by -1
7688        \fi
7689        \ifnum \tmpf=383
7690            \advance #3 by -1
7691        \fi
7692    \fi
7693    \ifnum #1 > 2
7694        \ifnum \tmpf=355
7695            \advance #3 by 1
7696        \fi
7697        \ifnum \tmpf=385
7698            \advance #3 by 1
7699        \fi
7700    \fi
7701    \global\bbl@cntcommon=#3\relax}%
7702  #3=\bbl@cntcommon}
7703 \def\bbl@absfromhebr#1#2#3#4{%
```

```
7704    {#4=#1\relax
7705      \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7706      \advance #4 by #1\relax
7707      \bbl@hebrelapseddays{#3}{#1}%
7708      \advance #4 by #1\relax
7709      \advance #4 by -1373429
7710      \global\bbl@cntcommon=#4\relax}%
7711    #4=\bbl@cntcommon}
7712  \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7713    {\countdef\tmpx= 17
7714      \countdef\tmpy= 18
7715      \countdef\tmpz= 19
7716      #6=#3\relax
7717      \global\advance #6 by 3761
7718      \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7719      \tmpz=1  \tmpy=1
7720      \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7721      \ifnum \tmpx > #4\relax
7722          \global\advance #6 by -1
7723          \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7724      \fi
7725      \advance #4 by -\tmpx
7726      \advance #4 by 1
7727      #5=#4\relax
7728      \divide #5 by 30
7729      \loop
7730          \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7731          \ifnum \tmpx < #4\relax
7732              \advance #5 by 1
7733              \tmpy=\tmpx
7734      \repeat
7735      \global\advance #5 by -1
7736      \global\advance #4 by -\tmpy}}
7737  \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
7738  \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7739  \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
7740    \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7741    \bbl@hebrfromgreg
7742      {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7743      {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7744    \edef#4{\the\bbl@hebryear}%
7745    \edef#5{\the\bbl@hebrmonth}%
7746    \edef#6{\the\bbl@hebrday}}
7747  ⟨/ca-hebrew⟩
```

17 Persian

There is an algorithm written in TeX by Jabri, Abolhassani, Pournader and Esfahbod, created for the first versions of the FarsiTeX system (no longer available), but the original license is GPL, so its use with LPPL is problematic. The code here follows loosely that by John Walker, which is free and accurate, but sadly very complex, so the relevant data for the years 2013-2050 have been pre-calculated and stored. Actually, all we need is the first day (either March 20 or March 21).

```
7748  ⟨∗ca-persian⟩
7749  \ExplSyntaxOn
7750  ⟨⟨Compute Julian day⟩⟩
7751  \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7752    2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7753  \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
7754    \edef\bbl@tempa{#1}%  20XX-03-\bbl@tempe = 1 farvardin:
7755    \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7756      \bbl@afterfi\expandafter\@gobble
7757    \fi\fi
```

```
7758        {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
7759    \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7760    \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7761    \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7762    \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7763    \ifnum\bbl@tempc<\bbl@tempb
7764        \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7765        \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7766        \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7767        \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
7768    \fi
7769    \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7770    \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7771    \edef#5{\fp_eval:n{% set Jalali month
7772        (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7773    \edef#6{\fp_eval:n{% set Jalali day
7774        (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}}}
7775 \ExplSyntaxOff
7776 ⟨/ca-persian⟩
```

18   Coptic and Ethiopic

Adapted from jquery.calendars.package-1.1.4, written by Keith Wood, 2010. Dual license: GPL and MIT. The only difference is the epoch.

```
7777 ⟨∗ca-coptic⟩
7778 \ExplSyntaxOn
7779 ⟨⟨Compute Julian day⟩⟩
7780 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
7781    \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7782    \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
7783    \edef#4{\fp_eval:n{%
7784        floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7785    \edef\bbl@tempc{\fp_eval:n{%
7786        \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
7787    \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7788    \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
7789 \ExplSyntaxOff
7790 ⟨/ca-coptic⟩
7791 ⟨∗ca-ethiopic⟩
7792 \ExplSyntaxOn
7793 ⟨⟨Compute Julian day⟩⟩
7794 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
7795    \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7796    \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
7797    \edef#4{\fp_eval:n{%
7798        floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7799    \edef\bbl@tempc{\fp_eval:n{%
7800        \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
7801    \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7802    \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
7803 \ExplSyntaxOff
7804 ⟨/ca-ethiopic⟩
```

19   Buddhist

That's very simple.

```
7805 ⟨∗ca-buddhist⟩
7806 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
7807    \edef#4{\number\numexpr#1+543\relax}%
7808    \edef#5{#2}%
7809    \edef#6{#3}}
```

20   Support for Plain TₑX (`plain.def`)

20.1   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 iniTₑX, 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 iniTₑX sees, we need to set some category codes just to be able to change the definition of `\input`.

```
7811 ⟨∗bplain | blplain⟩
7812 \catcode`\{=1 % left brace is begin-group character
7813 \catcode`\}=2 % right brace is end-group character
7814 \catcode`\#=6 % hash mark is macro parameter character
```

If a file called `hyphen.cfg` can be found, we make sure that *it* will be read instead of the file `hyphen.tex`. We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
7815 \openin 0 hyphen.cfg
7816 \ifeof0
7817 \else
7818   \let\a\input
```

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.

```
7819   \def\input #1 {%
7820     \let\input\a
7821     \a hyphen.cfg
7822     \let\a\undefined
7823   }
7824 \fi
7825 ⟨/bplain | blplain⟩
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
7826 ⟨bplain⟩\a plain.tex
7827 ⟨blplain⟩\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
7828 ⟨bplain⟩\def\fmtname{babel-plain}
7829 ⟨blplain⟩\def\fmtname{babel-lplain}
```

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.

20.2   Emulating some LᴬTₑX features

The file `babel.def` expects some definitions made in the LᴬTₑX 2_ε style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore and alternative mechanism is provided. For

the moment, only \baleloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```
7830 ⟨⟨*Emulate LaTeX⟩⟩ ≡
7831 \def\@empty{}
7832 \def\loadlocalcfg#1{%
7833   \openin0#1.cfg
7834   \ifeof0
7835     \closein0
7836   \else
7837     \closein0
7838     {\immediate\write16{**********************************}%
7839      \immediate\write16{* Local config file #1.cfg used}%
7840      \immediate\write16{*}%
7841      }
7842     \input #1.cfg\relax
7843   \fi
7844   \@endofldf}
```

20.3   General tools

A number of LaTeX macro's that are needed later on.

```
7845 \long\def\@firstofone#1{#1}
7846 \long\def\@firstoftwo#1#2{#1}
7847 \long\def\@secondoftwo#1#2{#2}
7848 \def\@nnil{\@nil}
7849 \def\@gobbletwo#1#2{}
7850 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7851 \def\@star@or@long#1{%
7852   \@ifstar
7853   {\let\l@ngrel@x\relax#1}%
7854   {\let\l@ngrel@x\long#1}}
7855 \let\l@ngrel@x\relax
7856 \def\@car#1#2\@nil{#1}
7857 \def\@cdr#1#2\@nil{#2}
7858 \let\@typeset@protect\relax
7859 \let\protected@edef\edef
7860 \long\def\@gobble#1{}
7861 \edef\@backslashchar{\expandafter\@gobble\string\\}
7862 \def\strip@prefix#1>{}
7863 \def\g@addto@macro#1#2{{%
7864     \toks@\expandafter{#1#2}%
7865     \xdef#1{\the\toks@}}}
7866 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7867 \def\@nameuse#1{\csname #1\endcsname}
7868 \def\@ifundefined#1{%
7869   \expandafter\ifx\csname#1\endcsname\relax
7870     \expandafter\@firstoftwo
7871   \else
7872     \expandafter\@secondoftwo
7873   \fi}
7874 \def\@expandtwoargs#1#2#3{%
7875   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7876 \def\zap@space#1 #2{%
7877   #1%
7878   \ifx#2\@empty\else\expandafter\zap@space\fi
7879   #2}
7880 \let\bbl@trace\@gobble
7881 \def\bbl@error#1#2{%
7882   \begingroup
7883     \newlinechar=`\^^J
7884     \def\\{^^J(babel) }%
7885     \errhelp{#2}\errmessage{\\#1}%
```

```
7886    \endgroup}
7887 \def\bbl@warning#1{%
7888    \begingroup
7889      \newlinechar=`\^^J
7890      \def\\{^^J(babel) }%
7891      \message{\\#1}%
7892    \endgroup}
7893 \let\bbl@infowarn\bbl@warning
7894 \def\bbl@info#1{%
7895    \begingroup
7896      \newlinechar=`\^^J
7897      \def\\{^^J}%
7898      \wlog{#1}%
7899    \endgroup}
```

LaTeX 2ε has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.

```
7900 \ifx\@preamblecmds\@undefined
7901    \def\@preamblecmds{}
7902 \fi
7903 \def\@onlypreamble#1{%
7904    \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7905      \@preamblecmds\do#1}}
7906 \@onlypreamble\@onlypreamble
```

Mimick LaTeX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```
7907 \def\begindocument{%
7908    \@begindocumenthook
7909    \global\let\@begindocumenthook\@undefined
7910    \def\do##1{\global\let##1\@undefined}%
7911    \@preamblecmds
7912    \global\let\do\noexpand}
7913 \ifx\@begindocumenthook\@undefined
7914    \def\@begindocumenthook{}
7915 \fi
7916 \@onlypreamble\@begindocumenthook
7917 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
```

We also have to mimick LaTeX's \AtEndOfPackage. Our replacement macro is much simpler; it stores its argument in \@endofldf.

```
7918 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7919 \@onlypreamble\AtEndOfPackage
7920 \def\@endofldf{}
7921 \@onlypreamble\@endofldf
7922 \let\bbl@afterlang\@empty
7923 \chardef\bbl@opt@hyphenmap\z@
```

LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer \ifx. The same trick is applied below.

```
7924 \catcode`\&=\z@
7925 \ifx&if@filesw\@undefined
7926    \expandafter\let\csname if@filesw\expandafter\endcsname
7927      \csname iffalse\endcsname
7928 \fi
7929 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
7930 \def\newcommand{\@star@or@long\new@command}
7931 \def\new@command#1{%
7932    \@testopt{\@newcommand#1}0}
7933 \def\@newcommand#1[#2]{%
7934    \@ifnextchar [{\@xargdef#1[#2]}%
```

```
7935                  {\@argdef#1[#2]}}
7936 \long\def\@argdef#1[#2]#3{%
7937   \@yargdef#1\@ne{#2}{#3}}
7938 \long\def\@xargdef#1[#2][#3]#4{%
7939   \expandafter\def\expandafter#1\expandafter{%
7940     \expandafter\@protected@testopt\expandafter #1%
7941     \csname\string#1\expandafter\endcsname{#3}}%
7942   \expandafter\@yargdef \csname\string#1\endcsname
7943   \tw@{#2}{#4}}
7944 \long\def\@yargdef#1#2#3{%
7945   \@tempcnta#3\relax
7946   \advance \@tempcnta \@ne
7947   \let\@hash@\relax
7948   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7949   \@tempcntb #2%
7950   \@whilenum\@tempcntb <\@tempcnta
7951   \do{%
7952     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7953     \advance\@tempcntb \@ne}%
7954   \let\@hash@##%
7955   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7956 \def\providecommand{\@star@or@long\provide@command}
7957 \def\provide@command#1{%
7958   \begingroup
7959     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7960   \endgroup
7961   \expandafter\@ifundefined\@gtempa
7962     {\def\reserved@a{\new@command#1}}%
7963     {\let\reserved@a\relax
7964      \def\reserved@a{\new@command\reserved@a}}%
7965   \reserved@a}%

7966 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7967 \def\declare@robustcommand#1{%
7968   \edef\reserved@a{\string#1}%
7969   \def\reserved@b{#1}%
7970   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7971   \edef#1{%
7972     \ifx\reserved@a\reserved@b
7973       \noexpand\x@protect
7974       \noexpand#1%
7975     \fi
7976     \noexpand\protect
7977     \expandafter\noexpand\csname
7978       \expandafter\@gobble\string#1 \endcsname
7979   }%
7980   \expandafter\new@command\csname
7981     \expandafter\@gobble\string#1 \endcsname
7982 }
7983 \def\x@protect#1{%
7984   \ifx\protect\@typeset@protect\else
7985     \@x@protect#1%
7986   \fi
7987 }
7988 \catcode`\&=\z@  % Trick to hide conditionals
7989   \def\@x@protect#1&fi#2#3{&fi\protect#1}
```

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.

```
7990   \def\bbl@tempa{\csname newif\endcsname&ifin@}
7991 \catcode`\&=4
7992 \ifx\in@\@undefined
7993   \def\in@#1#2{%
```

```
7994      \def\in@@##1#1##2##3\in@@{%
7995        \ifx\in@##2\in@false\else\in@true\fi}%
7996      \in@@#2#1\in@\in@@}
7997 \else
7998   \let\bbl@tempa\@empty
7999 \fi
8000 \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).

```
8001 \def\@ifpackagewith#1#2#3#4{#3}
```

The LaTeX macro \@ifl@aded checks whether a file was loaded. This functionality is not needed for plain TeX but we need the macro to be defined as a no-op.

```
8002 \def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands \newcommand and \providecommand exist with some sensible definition. They are not fully equivalent to their LaTeX 2ε versions; just enough to make things work in plain TeXenvironments.

```
8003 \ifx\@tempcnta\@undefined
8004   \csname newcount\endcsname\@tempcnta\relax
8005 \fi
8006 \ifx\@tempcntb\@undefined
8007   \csname newcount\endcsname\@tempcntb\relax
8008 \fi
```

To prevent wasting two counters in LaTeX (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (\count10).

```
8009 \ifx\bye\@undefined
8010   \advance\count10 by -2\relax
8011 \fi
8012 \ifx\@ifnextchar\@undefined
8013   \def\@ifnextchar#1#2#3{%
8014     \let\reserved@d=#1%
8015     \def\reserved@a{#2}\def\reserved@b{#3}%
8016     \futurelet\@let@token\@ifnch}
8017   \def\@ifnch{%
8018     \ifx\@let@token\@sptoken
8019       \let\reserved@c\@xifnch
8020     \else
8021       \ifx\@let@token\reserved@d
8022         \let\reserved@c\reserved@a
8023       \else
8024         \let\reserved@c\reserved@b
8025       \fi
8026     \fi
8027     \reserved@c}
8028   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
8029   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
8030 \fi
8031 \def\@testopt#1#2{%
8032   \@ifnextchar[{#1}{#1[#2]}}
8033 \def\@protected@testopt#1{%
8034   \ifx\protect\@typeset@protect
8035     \expandafter\@testopt
8036   \else
8037     \@x@protect#1%
8038   \fi}
8039 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8040       #2\relax}\fi}
8041 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8042        \else\expandafter\@gobble\fi{#1}}
```

217

20.4 Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain TeX environment.

```
8043 \def\DeclareTextCommand{%
8044     \@dec@text@cmd\providecommand
8045 }
8046 \def\ProvideTextCommand{%
8047     \@dec@text@cmd\providecommand
8048 }
8049 \def\DeclareTextSymbol#1#2#3{%
8050     \@dec@text@cmd\chardef#1{#2}#3\relax
8051 }
8052 \def\@dec@text@cmd#1#2#3{%
8053     \expandafter\def\expandafter#2%
8054         \expandafter{%
8055             \csname#3-cmd\expandafter\endcsname
8056             \expandafter#2%
8057             \csname#3\string#2\endcsname
8058         }%
8059 %   \let\@ifdefinable\@rc@ifdefinable
8060     \expandafter#1\csname#3\string#2\endcsname
8061 }
8062 \def\@current@cmd#1{%
8063   \ifx\protect\@typeset@protect\else
8064       \noexpand#1\expandafter\@gobble
8065   \fi
8066 }
8067 \def\@changed@cmd#1#2{%
8068     \ifx\protect\@typeset@protect
8069         \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8070             \expandafter\ifx\csname ?\string#1\endcsname\relax
8071               \expandafter\def\csname ?\string#1\endcsname{%
8072                   \@changed@x@err{#1}%
8073               }%
8074             \fi
8075             \global\expandafter\let
8076               \csname\cf@encoding \string#1\expandafter\endcsname
8077               \csname ?\string#1\endcsname
8078         \fi
8079         \csname\cf@encoding\string#1%
8080           \expandafter\endcsname
8081     \else
8082         \noexpand#1%
8083     \fi
8084 }
8085 \def\@changed@x@err#1{%
8086     \errhelp{Your command will be ignored, type <return> to proceed}%
8087     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8088 \def\DeclareTextCommandDefault#1{%
8089   \DeclareTextCommand#1?%
8090 }
8091 \def\ProvideTextCommandDefault#1{%
8092   \ProvideTextCommand#1?%
8093 }
8094 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8095 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8096 \def\DeclareTextAccent#1#2#3{%
8097   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8098 }
8099 \def\DeclareTextCompositeCommand#1#2#3#4{%
8100     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8101     \edef\reserved@b{\string##1}%
8102     \edef\reserved@c{%
```

```
8103        \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8104    \ifx\reserved@b\reserved@c
8105        \expandafter\expandafter\expandafter\ifx
8106            \expandafter\@car\reserved@a\relax\relax\@nil
8107            \@text@composite
8108        \else
8109            \edef\reserved@b##1{%
8110                \def\expandafter\noexpand
8111                    \csname#2\string#1\endcsname####1{%
8112                    \noexpand\@text@composite
8113                        \expandafter\noexpand\csname#2\string#1\endcsname
8114                        ####1\noexpand\@empty\noexpand\@text@composite
8115                        {##1}%
8116                }%
8117            }%
8118            \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8119        \fi
8120        \expandafter\def\csname\expandafter\string\csname
8121            #2\endcsname\string#1-\string#3\endcsname{#4}
8122    \else
8123        \errhelp{Your command will be ignored, type <return> to proceed}%
8124        \errmessage{\string\DeclareTextCompositeCommand\space used on
8125            inappropriate command \protect#1}
8126    \fi
8127 }
8128 \def\@text@composite#1#2#3\@text@composite{%
8129    \expandafter\@text@composite@x
8130        \csname\string#1-\string#2\endcsname
8131 }
8132 \def\@text@composite@x#1#2{%
8133    \ifx#1\relax
8134        #2%
8135    \else
8136        #1%
8137    \fi
8138 }
8139 %
8140 \def\@strip@args#1:#2-#3\@strip@args{#2}
8141 \def\DeclareTextComposite#1#2#3#4{%
8142    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8143    \bgroup
8144        \lccode`\@=#4%
8145        \lowercase{%
8146    \egroup
8147        \reserved@a @%
8148    }%
8149 }
8150 %
8151 \def\UseTextSymbol#1#2{#2}
8152 \def\UseTextAccent#1#2#3{}
8153 \def\@use@text@encoding#1{}
8154 \def\DeclareTextSymbolDefault#1#2{%
8155    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8156 }
8157 \def\DeclareTextAccentDefault#1#2{%
8158    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8159 }
8160 \def\cf@encoding{OT1}
```

Currently we only use the LaTeX 2_ε method for accents for those that are known to be made active in *some* language definition file.

```
8161 \DeclareTextAccent{\"}{OT1}{127}
8162 \DeclareTextAccent{\'}{OT1}{19}
```

```
8163 \DeclareTextAccent{\^}{OT1}{94}
8164 \DeclareTextAccent{\`}{OT1}{18}
8165 \DeclareTextAccent{\~}{OT1}{126}
```

The following control sequences are used in babel.def but are not defined for PLAIN TEX.

```
8166 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8167 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8168 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
8169 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8170 \DeclareTextSymbol{\i}{OT1}{16}
8171 \DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the LATEX-control sequence \scriptsize to be available. Because plain TEX doesn't have such a sofisticated font mechanism as LATEX has, we just \let it to \sevenrm.

```
8172 \ifx\scriptsize\@undefined
8173   \let\scriptsize\sevenrm
8174 \fi
```

And a few more "dummy" definitions.

```
8175 \def\languagename{english}%
8176 \let\bbl@opt@shorthands\@nnil
8177 \def\bbl@ifshorthand#1#2#3{#2}%
8178 \let\bbl@language@opts\@empty
8179 \ifx\babeloptionstrings\@undefined
8180   \let\bbl@opt@strings\@nnil
8181 \else
8182   \let\bbl@opt@strings\babeloptionstrings
8183 \fi
8184 \def\BabelStringsDefault{generic}
8185 \def\bbl@tempa{normal}
8186 \ifx\babeloptionmath\bbl@tempa
8187   \def\bbl@mathnormal{\noexpand\textormath}
8188 \fi
8189 \def\AfterBabelLanguage#1#2{}
8190 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8191 \let\bbl@afterlang\relax
8192 \def\bbl@opt@safe{BR}
8193 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8194 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8195 \expandafter\newif\csname ifbbl@single\endcsname
8196 \chardef\bbl@bidimode\z@
8197 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
8198 ⟨*plain⟩
8199 \input babel.def
8200 ⟨/plain⟩
```

21   Acknowledgements

I would like to thank all who volunteered as β-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.
During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

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

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

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

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

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

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

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

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

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

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

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

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

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